Claude Code 日本語ドキュメント

リリース *1.0.0*

code.claude.com

2026 年 10 月 06 日

目次

第 I 部　はじめに **1**

第 1 章　概要 **3**
1.1 はじめに . . . 3
1.2 できること . . . 6
1.3 Claude Code をどこでも使用する . . . 8
1.4 次のステップ . . . 9

第 2 章　クイックスタート **11**
2.1 始める前に . . . 11
2.2 ステップ 1：Claude Code をインストールする . . . 11
2.3 ステップ 2：アカウントにログインする . . . 13
2.4 ステップ 3：最初のセッションを開始する . . . 14
2.5 ステップ 4：最初の質問をする . . . 14
2.6 ステップ 5：最初のコード変更を行う . . . 14
2.7 ステップ 6：Claude Code で Git を使用する . . . 15
2.8 ステップ 7：バグを修正するか機能を追加する . . . 15
2.9 ステップ 8：他の一般的なワークフローを試す . . . 16
2.10 必須コマンド . . . 16
2.11 初心者向けのプロのヒント . . . 17
2.12 次のステップ . . . 18
2.13 ヘルプを取得する . . . 18

第 3 章　高度なセットアップ **19**
3.1 システム要件 . . . 19
3.2 Claude Code をインストール . . . 20
3.3 インストールを確認 . . . 23
3.4 認証 . . . 23
3.5 Claude Code を更新 . . . 24
3.6 高度なインストールオプション . . . 27
3.7 Claude Code をアンインストール . . . 33

第 4 章　トラブルシューティング **37**
4.1 パフォーマンスと安定性 . . . 38
4.2 さらにヘルプを得る . . . 42

第 II 部　基本操作 **43**

第 5 章　CLI リファレンス **45**
5.1 CLI コマンド . . . 45

5.2 CLI フラグ . . . 51
5.3 関連項目 . . . 74

第 6 章 インタラクティブモード 75
6.1 キーボードショートカット . . . 75
6.2 コマンド . . . 83
6.3 Vim エディタモード . . . 84
6.4 コマンド履歴 . . . 87
6.5 バックグラウンド Bash コマンド . . . 88
6.6 Claude が作業中にメッセージをキューに入れる . . . 90
6.7 プロンプト提案 . . . 92
6.8 絵文字ショートコード . . . 93
6.9 入力中にスペルチェックを実行する . . . 93
6.10 プロンプト内の見えない文字 . . . 96
6.11 /diff で変更内容を確認する . . . 96
6.12 /btw で補足質問をする . . . 98
6.13 タスクリスト . . . 100
6.14 セッションの要約 . . . 100
6.15 使用制限がリセットされるまで待機する . . . 100
6.16 PR レビューステータス . . . 102
6.17 Issue 参照リンク . . . 104
6.18 関連項目 . . . 104

第 7 章 一般的なワークフロー 105
7.1 プロンプトレシピ . . . 105
7.2 以前の会話を再開する . . . 113
7.3 worktree を使用して並列セッションを実行する . . . 113
7.4 編集前に計画する . . . 113
7.5 研究を subagent に委譲する . . . 114
7.6 Claude をスクリプトにパイプする . . . 114
7.7 次のステップ . . . 114

第 8 章 Claude Code のベストプラクティス 115
8.1 Claude に自分の作業を検証する方法を与える . . . 115
8.2 最初に探索し、次に計画し、その後コーディングする . . . 117
8.3 プロンプトで具体的なコンテキストを提供する . . . 118
8.4 環境を設定する . . . 119
8.5 効果的にコミュニケーションする . . . 124
8.6 セッションを管理する . . . 126
8.7 自動化とスケール . . . 128
8.8 一般的な失敗パターンを避ける . . . 132
8.9 直感を開発する . . . 132
8.10 関連リソース . . . 133

第 III 部 統合環境 135

第 9 章 Desktop application 137

9.1 セッションを開始する . . . 138
9.2 コードの操作 . . . 138
9.3 ワークスペースを配置する . . . 143
9.4 Claude にコンピュータを使用させる . . . 146
9.5 セッションを管理する . . . 148
9.6 Claude Code を拡張する . . . 151
9.7 環境設定 . . . 158
9.8 エンタープライズ設定 . . . 160
9.9 CLI から移行する場合 . . . 166
9.10 トラブルシューティング . . . 170

第 10 章 VS Code で Claude Code を使用する **173**
10.1 前提条件 . . . 173
10.2 拡張機能をインストールする . . . 174
10.3 はじめに . . . 174
10.4 プロンプトボックスを使用する . . . 177
10.5 ワークフローをカスタマイズする . . . 184
10.6 プラグインを管理する . . . 186
10.7 Chrome でブラウザタスクを自動化する . . . 189
10.8 VS Code コマンドとショートカット . . . 190
10.9 設定を構成する . . . 193
10.10 スクリーンリーダーを使用する . . . 195
10.11 VS Code 拡張機能と Claude Code CLI . . . 196
10.12 git で作業する . . . 199
10.13 サードパーティプロバイダーを使用する . . . 199
10.14 セキュリティとプライバシー . . . 200
10.15 一般的な問題を解決する . . . 202
10.16 拡張機能をアンインストールする . . . 203
10.17 次のステップ . . . 203

第 11 章 JetBrains IDEs **205**
11.1 サポートされている IDE . . . 205
11.2 機能 . . . 205
11.3 インストール . . . 206
11.4 使用方法 . . . 206
11.5 設定 . . . 206
11.6 特別な設定 . . . 208
11.7 トラブルシューティング . . . 209
11.8 セキュリティに関する考慮事項 . . . 209

第 12 章 Chrome で Claude Code を使用する **213**
12.1 機能 . . . 213
12.2 前提条件 . . . 214
12.3 CLI で開始する . . . 214
12.4 ワークフロー例 . . . 217
12.5 トラブルシューティング . . . 219
12.6 関連項目 . . . 221

第 13 章 クラウドで Claude Code を使用する **223**
13.1 クラウド環境 . . . 224
13.2 GitHub 認証オプション . . . 224
13.3 ターミナルとクラウド間でタスクを移動 . . . 225
13.4 セッションの操作 . . . 231
13.5 プルリクエストの自動修正 . . . 234
13.6 セキュリティと分離 . . . 235
13.7 トラブルシューティング . . . 235
13.8 制限事項 . . . 236
13.9 関連リソース . . . 237

第 14 章 Slack での Claude Code **239**
14.1 ユースケース . . . 239
14.2 前提条件 . . . 240
14.3 Slack での Claude Code のセットアップ . . . 240
14.4 仕組み . . . 241
14.5 ユーザーインターフェース要素 . . . 242
14.6 アクセスと権限 . . . 242
14.7 どこでアクセスできるか . . . 243
14.8 ベストプラクティス . . . 243
14.9 トラブルシューティング . . . 244
14.10 現在の制限事項 . . . 245
14.11 関連リソース . . . 245

第 15 章 Claude Code GitHub Actions **247**
15.1 セットアップ . . . 247
15.2 インタラクティブモードと自動化モード . . . 251
15.3 使用例 . . . 252
15.4 ベストプラクティス . . . 255
15.5 クラウドプロバイダーを使用 . . . 256
15.6 トラブルシューティング . . . 257
15.7 高度な設定 . . . 257
15.8 ベータ版からアップグレード . . . 259
15.9 次のステップ . . . 259

第 16 章 Claude Code GitLab CI/CD **261**
16.1 GitLab で Claude Code を使用する理由 . . . 261
16.2 仕組み . . . 261
16.3 Claude にはどのようなことができますか？ . . . 262
16.4 セットアップ . . . 262
16.5 使用例 . . . 264
16.6 Amazon Bedrock と Google Cloud での使用 . . . 264
16.7 設定例 . . . 266
16.8 ベストプラクティス . . . 269
16.9 トラブルシューティング . . . 270
16.10 高度な設定 . . . 271

第 IV 部　拡張機能 **273**

第 17 章 Claude をスキルで拡張する **275**
17.1 バンドルされたスキル . . . 275
17.2 はじめに . . . 278
17.3 スキルの読み込み場所を選択する . . . 279
17.4 スキルを設定する . . . 287
17.5 高度なパターン . . . 300
17.6 スキルを評価して反復する . . . 308
17.7 スキルを共有する . . . 309
17.8 トラブルシューティング . . . 314
17.9 関連リソース . . . 316

第 18 章 カスタムサブエージェントの作成 **317**
18.1 組み込みサブエージェント . . . 318
18.2 クイックスタート：最初のサブエージェントを作成する . . . 320
18.3 サブエージェントの設定 . . . 321
18.4 サブエージェントを使用する . . . 343
18.5 現在の会話をフォークする . . . 354
18.6 サブエージェントの例 . . . 356
18.7 次のステップ . . . 361

第 19 章 MCP を使用して Claude Code をツールに接続する **363**
19.1 MCP でできること . . . 363
19.2 MCP サーバーを検索してビルドする . . . 364
19.3 MCP サーバーのインストール . . . 364
19.4 MCP インストールスコープ . . . 380
19.5 実践的な例 . . . 385
19.6 リモート MCP サーバーで認証する . . . 386
19.7 JSON 設定から MCP サーバーを追加する . . . 395
19.8 Claude Desktop から MCP サーバーをインポートする . . . 396
19.9 claude.ai から MCP サーバーを使用する . . . 396
19.10 Claude Code を MCP サーバーとして使用する . . . 400
19.11 MCP 出力制限と警告 . . . 401
19.12 ルートレベルのコンビネータを持つツール入力スキーマ . . . 403
19.13 無効な入力スキーマを持つツール . . . 403
19.14 特定のツールに対して承認を要求する . . . 404
19.15 MCP エリシテーション要求に応答する . . . 405
19.16 MCP リソースを使用する . . . 405
19.17 MCP ツール検索でスケーリング . . . 406
19.18 MCP プロンプトをコマンドとして使用する . . . 409
19.19 管理対象 MCP 設定 . . . 410

第 20 章 hooks でアクションを自動化する **411**
20.1 最初の hook をセットアップする . . . 411
20.2 自動化できるもの . . . 413
20.3 hooks の仕組み . . . 423
20.4 プロンプトベースの hooks . . . 434

20.5 エージェントベースの hooks . . . 435
20.6 HTTP hooks . . . 436
20.7 制限事項とトラブルシューティング . . . 437
20.8 詳細を学ぶ . . . 441

第 21 章 Hooks リファレンス **443**
21.1 フック ライフサイクル . . . 443
21.2 設定 . . . 450
21.3 フック入出力 . . . 466
21.4 フックイベント . . . 480
21.5 プロンプト ベースのフック . . . 552
21.6 エージェント ベースのフック . . . 556
21.7 バックグラウンドでフックを実行 . . . 558
21.8 セキュリティに関する考慮事項 . . . 560
21.9 Windows PowerShell ツール . . . 561
21.10 フックをデバッグ . . . 562

第 22 章 プラグインを作成する **565**
22.1 プラグインとスタンドアロン設定を使い分ける . . . 565
22.2 クイックスタート . . . 565
22.3 スキルディレクトリでプラグインを開発する . . . 568
22.4 プラグイン構造の概要 . . . 569
22.5 より複雑なプラグインを開発する . . . 570
22.6 既存の設定をプラグインに変換する . . . 575
22.7 次のステップ . . . 577

第 23 章 プラグインリファレンス **579**
23.1 プラグインコンポーネントリファレンス . . . 579
23.2 プラグインのインストールスコープ . . . 589
23.3 スキルディレクトリプラグイン . . . 589
23.4 claude.ai から同期されたプラグイン . . . 590
23.5 プラグインマニフェストスキーマ . . . 592
23.6 プラグインのキャッシングとファイル解決 . . . 605
23.7 プラグインディレクトリ構造 . . . 608
23.8 CLI コマンドリファレンス . . . 610
23.9 デバッグと開発ツール . . . 622
23.10 配布とバージョン管理リファレンス . . . 624
23.11 関連項目 . . . 626

第 24 章 プラグインマーケットプレイスの作成と配布 **627**
24.1 概要 . . . 627
24.2 チュートリアル：ローカルマーケットプレイスの作成 . . . 627
24.3 マーケットプレイスファイルの作成 . . . 630
24.4 マーケットプレイススキーマ . . . 631
24.5 プラグインエントリ . . . 633
24.6 プラグインソース . . . 636
24.7 マーケットプレイスのホストと配布 . . . 652
24.8 検証とテスト . . . 665

24.9 CLI からマーケットプレイスを管理する . . . 666
24.10 トラブルシューティング . . . 669
24.11 関連項目 . . . 676

第 25 章 マーケットプレイスから事前構築されたプラグインを発見してインストールする 677
25.1 マーケットプレイスの仕組み . . . 677
25.2 公式 Anthropic マーケットプレイス . . . 677
25.3 コミュニティ マーケットプレイス . . . 680
25.4 試してみる: デモマーケットプレイスを追加する . . . 680
25.5 マーケットプレイスを追加する . . . 682
25.6 プラグインをインストールする . . . 685
25.7 インストール済みプラグインを管理する . . . 687
25.8 マーケットプレイスを管理する . . . 690
25.9 チーム マーケットプレイスを構成する . . . 691
25.10 セキュリティ . . . 692
25.11 トラブルシューティング . . . 692
25.12 次のステップ . . . 693

第 26 章 Claude Code をプログラムで実行する 695
26.1 基本的な使用方法 . . . 695
26.2 例 . . . 698
26.3 次のステップ . . . 707

第 V 部 設定 709

第 27 章 設定ファイルと優先順位 711
27.1 設定ファイルと誰に影響するか . . . 711
27.2 設定を変更する . . . 715
27.3 設定の優先順位 . . . 719
27.4 クラウドセッションの設定 . . . 724
27.5 次は何か . . . 724

第 28 章 モデル設定 725
28.1 利用可能なモデル . . . 725
28.2 モデル選択を制限する . . . 732
28.3 組織のデフォルトモデル . . . 740
28.4 組織の effort 上限 . . . 741
28.5 特殊なモデルの動作 . . . 742
28.6 コンテキストウィンドウと自動圧縮 . . . 754
28.7 現在のモデルの確認 . . . 756
28.8 カスタムモデルオプションを追加する . . . 756
28.9 環境変数 . . . 757
28.10 バージョン履歴 . . . 763

第 29 章 キーボードショートカットのカスタマイズ 765
29.1 設定ファイル . . . 765
29.2 コンテキスト . . . 766
29.3 利用可能なアクション . . . 766

29.4 キーストロークシンタックス . . . 778
29.5 デフォルトのショートカットのバインドを解除する . . . 780
29.6 予約済みショートカット . . . 782
29.7 ターミナルの競合 . . . 782
29.8 テキストフィールド . . . 782
29.9 Vim モードの相互作用 . . . 783
29.10 検証 . . . 783

第 30 章 ステータスラインをカスタマイズする 785
30.1 ステータスラインを設定する . . . 786
30.2 ステータスラインをステップバイステップで構築する . . . 787
30.3 ステータスラインの仕組み . . . 788
30.4 利用可能なデータ . . . 789
30.5 例 . . . 799
30.6 サブエージェントステータスライン . . . 815
30.7 ヒント . . . 816
30.8 トラブルシューティング . . . 817

第 31 章 出力スタイル 821
31.1 組み込み出力スタイル . . . 821
31.2 出力スタイルを変更する . . . 824
31.3 カスタム出力スタイルを作成する . . . 824
31.4 出力スタイルと他の機能の選択 . . . 826
31.5 出力スタイルの仕組み . . . 827
31.6 関連リソース . . . 827

第 32 章 Claude Code 用にターミナルを設定する 829
32.1 複数行のプロンプトを入力する . . . 829
32.2 macOS で Option キーショートカットを有効にする . . . 830
32.3 ターミナルベルまたは通知を取得する . . . 831
32.4 tmux を設定する . . . 832
32.5 Windows で Backspace がワード全体を削除する問題を修正する . . . 832
32.6 カラーテーマを合わせる . . . 833
32.7 フルスクリーンレンダリングに切り替える . . . 837
32.8 幅広いターミナルでレスポンス幅を制限する . . . 838
32.9 大量のコンテンツを貼り付ける . . . 838
32.10 Vim キーバインディングでプロンプトを編集する . . . 839
32.11 関連リソース . . . 839

第 33 章 その他の LLM gateway 841
33.1 ゲートウェイが提供するもの . . . 841
33.2 ゲートウェイをロールアウトする . . . 842
33.3 サブスクリプションとゲートウェイ . . . 842
33.4 関連ページ . . . 843

第 VI 部 エンタープライズ 845

第 34 章 エンタープライズデプロイメント概要 847

34.1 デプロイメントオプションの比較 . . . 847
34.2 プロキシとゲートウェイの構成 . . . 848
34.3 組織のベストプラクティス . . . 849
34.4 次のステップ . . . 850

第 35 章 認証 **851**
35.1 Claude Code にログインする . . . 851
35.2 チーム認証を設定する . . . 852
35.3 認証情報管理 . . . 857

第 36 章 Amazon Bedrock 上の Claude Code **863**
36.1 前提条件 . . . 863
36.2 Bedrock でサインイン . . . 863
36.3 手動でセットアップする . . . 864
36.4 スタートアップモデルチェック . . . 871
36.5 クロスリージョン推論プロファイルプレフィックス . . . 872
36.6 IAM 設定 . . . 873
36.7 1M トークンコンテキストウィンドウ . . . 874
36.8 サービスティア . . . 875
36.9 AWS Guardrails . . . 875
36.10 Mantle エンドポイントを使用する . . . 875
36.11 トラブルシューティング . . . 877
36.12 追加リソース . . . 880

第 37 章 Google Cloud の Agent Platform 上の Claude Code **881**
37.1 前提条件 . . . 881
37.2 Agent Platform でサインイン . . . 881
37.3 リージョン設定 . . . 882
37.4 手動でセットアップする . . . 882
37.5 起動時のモデルチェック . . . 886
37.6 IAM 設定 . . . 887
37.7 100 万トークンコンテキストウィンドウ . . . 888
37.8 トラブルシューティング . . . 888
37.9 追加リソース . . . 889

第 38 章 Claude Code on Microsoft Foundry **891**
38.1 前提条件 . . . 891
38.2 セットアップ . . . 891
38.3 Azure RBAC 構成 . . . 894
38.4 トラブルシューティング . . . 894
38.5 その他のリソース . . . 895

第 39 章 エンタープライズネットワーク設定 **897**
39.1 プロキシ設定 . . . 897
39.2 CA 証明書ストア . . . 898
39.3 カスタム CA 証明書 . . . 899
39.4 mTLS 認証 . . . 899
39.5 設定を確認する . . . 900

39.6 バックグラウンドエージェントにネットワーク設定を適用する . . . 901
39.7 ストリーミングアイドルウォッチドッグ . . . 902
39.8 ネットワークアクセス要件 . . . 904
39.9 追加リソース . . . 908

第 VII 部　運用 **909**

第 40 章 コストを効果的に管理する **911**
40.1 コストを追跡する . . . 911
40.2 組織のコストを管理する . . . 915
40.3 トークン使用量を削減する . . . 919
40.4 バックグラウンドトークン使用量 . . . 923
40.5 長いセッションで使用量が増加する理由 . . . 923
40.6 Claude Code の動作の変更を理解する . . . 924

第 41 章 チームの使用状況を分析で追跡する **927**
41.1 Team と Enterprise の分析にアクセスする . . . 927
41.2 API カスタマー向けの分析にアクセスする . . . 932
41.3 関連リソース . . . 933

第 42 章 監視 **935**
42.1 クイックスタート . . . 935
42.2 管理者設定 . . . 936
42.3 設定の詳細 . . . 938
42.4 クラウドセッションと Claude Tag からのテレメトリ . . . 955
42.5 利用可能なメトリクスとイベント . . . 956
42.6 メトリクスとイベントデータの解釈 . . . 988
42.7 監査セキュリティイベント . . . 990
42.8 バックエンドに関する考慮事項 . . . 993
42.9 サービス情報 . . . 993
42.10 ROI 測定リソース . . . 994
42.11 セキュリティとプライバシー . . . 994
42.12 Amazon Bedrock での Claude Code の監視 . . . 996

第 43 章 Claude があなたのプロジェクトを記憶する方法 **997**
43.1 CLAUDE.md と自動メモリ . . . 997
43.2 CLAUDE.md ファイル . . . 998
43.3 AGENTS.md . . . 1008
43.4 自動メモリ . . . 1013
43.5 `/memory` で表示および編集する . . . 1015
43.6 メモリの問題をトラブルシューティングする . . . 1016
43.7 関連リソース . . . 1018

第 44 章 チェックポイント **1019**
44.1 チェックポイントの仕組み . . . 1019
44.2 一般的なユースケース . . . 1021
44.3 制限事項 . . . 1021
44.4 関連項目 . . . 1022

第 45 章 開発コンテナ **1023**
45.1 開発コンテナに Claude Code を追加する 1024
45.2 再構築時に認証と設定を保持する 1025
45.3 組織ポリシーを適用する 1026
45.4 ネットワークエグレスを制限する 1027
45.5 権限プロンプトなしで実行する 1027
45.6 リファレンスコンテナを試す 1028
45.7 次のステップ 1028

第 VIII 部 セキュリティ・プライバシー **1031**

第 46 章 セキュリティ **1033**
46.1 セキュリティへのアプローチ方法 1033
46.2 プロンプトインジェクションから保護する 1034
46.3 MCP セキュリティ 1036
46.4 IDE セキュリティ 1036
46.5 クラウド実行セキュリティ 1036
46.6 セキュリティベストプラクティス 1037
46.7 関連リソース 1038

第 47 章 権限を設定する **1039**
47.1 権限システム 1039
47.2 権限を管理する 1041
47.3 権限モード 1042
47.4 権限ルール構文 1043
47.5 ツール固有の権限ルール 1047
47.6 フックで権限を拡張する 1059
47.7 作業ディレクトリ 1060
47.8 権限とサンドボックス化がどのように相互作用するか 1063
47.9 管理設定 1063
47.10 設定の優先順位 1064
47.11 プロジェクトの許可ルールとワークスペーストラスト 1064
47.12 設定例 1067
47.13 関連項目 1067

第 48 章 サンドボックス化された Bash ツールを設定する **1069**
48.1 サンドボックスが制限する内容 1069
48.2 はじめに 1071
48.3 サンドボックスを設定する 1077
48.4 サンドボックス化の仕組み 1086
48.5 サンドボックスが権限と権限モードにどのように関連するか 1091
48.6 組織のサンドボックスを設定する 1093
48.7 トラブルシューティング 1097
48.8 制限事項 1102
48.9 関連項目 1104

第 49 章 データ使用 **1105**

49.1 データポリシー 1105
49.2 データアクセス 1107
49.3 ローカル Claude Code：データフローと依存関係 1107
49.4 テレメトリサービス 1109
49.5 API プロバイダーのデフォルト動作 1110

第 50 章 法的および規制対応 **1113**
50.1 法的契約 1113
50.2 規制対応 1114
50.3 使用ポリシー 1114
50.4 セキュリティと信頼 1115

用語集 **1117**
A 1117
B 1118
C 1119
D 1120
E 1121
F 1121
H 1121
M 1122
N 1122
O 1123
P 1123
R 1124
S 1124
T 1126
V 1127
W 1127
非推奨および名前変更された用語 1127

この文書について

本書は、Claude Code 公式ドキュメントの日本語版を PDF 向けに組み直したものです。内容は変換元のままで、加筆や要約はしていません。

項目	内容
変換元	claude-code-docs-ja の `docs-ja/index.md`
変換元の版	2026-10-04 時点（コミット `bfaa3e1`）
公式サイト	https://code.claude.com/docs/ja/
収録範囲	`index.md` からリンクされているページのうち 48 ページと、下記の 3 ページ。並び順は `index.md` と同じです。

Web 版との違いは次のとおりです。

Web 版の要素	本書での扱い
タブ、折りたたみ	見出し語と字下げで、すべての内容を順に掲載しています。
対話型の図	省略し、その位置に注釈を置いています。
収録していないページへのリンク	公式サイトへのリンクにしています。
収録しているページへのリンク	文書内の参照にし、ページ番号を添えています。
コード例の中の絵文字	フォントに字形が無いものは空白になります。

`index.md` に載っている「IAM」は、変換元にファイルがありません。公式サイトで 2 ページに分割されたため、本書では次のように収録しています。

ページ	収録した部	位置
権限を設定する	セキュリティ・プライバシー	「サンドボックス」の章の直前
認証	エンタープライズ	「概要」の章の直後

配置は、英語版の公式サイトのナビゲーションに合わせています。

このほか、`index.md` から次の 2 点を変えています。

- 「リリース」の部（CHANGELOG）は収録していません。
- `index.md` に無い「用語集」を、巻末に章番号なしで収録しています。

第I部

はじめに

第1章　概要

> Claude Code は agentic coding ツールで、コードベースを読み取り、ファイルを編集し、コマンドを実行し、開発ツールと統合します。ターミナル、IDE、デスクトップアプリ、ブラウザで利用できます。

Claude Code は AI を活用したコーディングアシスタントで、機能の構築、バグの修正、開発タスクの自動化を支援します。コードベース全体を理解し、複数のファイルとツール間で作業して目標を達成できます。

1.1 はじめに

Claude Code は複数のサーフェスで実行されます。ターミナル、IDE 拡張機能、デスクトップアプリ、Web です。下のタブから 1 つを選択してはじめましょう。ほとんどのサーフェスには Claude サブスクリプションまたは Anthropic Console アカウントが必要です。Terminal CLI、VS Code、JetBrains も **サードパーティプロバイダー** (ページ 847) をサポートしています。

Terminal

ターミナルで Claude Code を直接操作するための機能豊富な CLI です。ファイルを編集し、コマンドを実行し、コマンドラインからプロジェクト全体を管理できます。

Claude Code をインストールするには、ターミナルを開いてシステムのコマンドを実行してください。ターミナルを使用したことがない場合は、ターミナルガイドでターミナルを開いてコマンドを貼り付ける方法を確認できます。

ネイティブインストール（推奨）

macOS、Linux、WSL：

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell：

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD：

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd &&␣
↪del install.cmd
```

インストーラーが完了したら、新しいターミナルウィンドウを開いて `claude --version` を実行してください。インストールが正常に完了すると、バージョン番号が表示されます。シェルが `claude` が見つからない、または認識されていないと表示される場合は、インストールディレクトリがまだ PATH に含まれていません。PATH を修正するを参照してください。

`The token '&&' is not a valid statement separator` というエラーが表示される場合は、CMD ではなく PowerShell を使用しています。`'irm' is not recognized as an internal or external command` というエラーが表示される場合は、PowerShell ではなく CMD を使用しています。PowerShell を使用している場合、プロンプトに `PS C:\` と表示され、CMD を使用している場合は `PS` なしで `C:\` と表示されます。

インストールコマンドが `syntax error near unexpected token '<'`、`403`、またはその他の curl エラーで失敗する場合は、インストールのトラブルシューティングを参照して、エラーを修正方法に照合し、代替インストール方法を確認してください。

Git for Windows は、Claude Code が Bash ツールを使用できるようにネイティブ Windows で推奨されます。Git for Windows がインストールされていない場合、Claude Code はシェルツールとして PowerShell を代わりに使用します。WSL セットアップは Git for Windows を必要としません。

注釈:

ネイティブインストールは、最新バージョンに保つために自動的にバックグラウンドで更新されます。

Homebrew

```
brew install --cask claude-code
```

Homebrew は 2 つの cask を提供しています。`claude-code` は安定リリースチャネルを追跡しており、通常は約 1 週間遅れており、大きな回帰を伴うリリースをスキップします。`claude-code@latest` は最新チャネルを追跡し、新しいバージョンが出荷されるとすぐに受け取ります。

注釈:

Homebrew インストールは自動更新されません。インストールした cask に応じて、`brew upgrade claude-code` または `brew upgrade claude-code@latest` を実行して、最新の機能とセキュリティ修正を取得してください。

WinGet

```
winget install Anthropic.ClaudeCode
```

注釈:

WinGet インストールは自動更新されません。最新の機能とセキュリティ修正を取得するために、定期的に `winget upgrade Anthropic.ClaudeCode` を実行してください。

また、Debian、Fedora、RHEL、Alpine で *apt*、*dnf*、**または** *apk* (ページ 28) を使用してインストールすることもできます。

その後、任意のプロジェクトで Claude Code を開始します。`your-project` をマシン上のプロジェク

トディレクトリへのパスに置き換えてください：

```
cd your-project
claude
```

初回使用時にログインするよう促されます。ANTHROPIC_API_KEY 環境変数を設定している場合、Claude Code はログインプロンプトをスキップし、代わりにキーを承認するよう求めます。これで完了です！**クイックスタートに進む** → (ページ 11)

Tip:

インストールオプション、手動更新、またはアンインストール手順については **高度なセットアップ** (ページ 19) を参照してください。問題が発生した場合は インストールのトラブルシューティング にアクセスしてください。

VS Code

VS Code 拡張機能は、インラインの差分表示、@-メンション、プラン確認、会話履歴をエディター内で直接提供します。

- VS Code 用にインストール
- Cursor 用にインストール

または、拡張機能ビュー（Mac では Cmd+Shift+X、Windows/Linux では Ctrl+Shift+X）で「Claude Code」を検索してください。インストール後、コマンドパレット（Cmd+Shift+P / Ctrl+Shift+P）を開き、「Claude Code」と入力して、**新しいタブで開く** を選択します。

VS Code **ではじめる** → (ページ 174)

Desktop app

IDE またはターミナルの外で Claude Code を実行するためのスタンドアロンアプリです。差分を視覚的に確認し、複数のセッションを並行実行し、定期的なタスクをスケジュール設定し、クラウドセッションを開始できます。

ダウンロードしてインストール：

- macOS（Intel および Apple Silicon）
- Windows（x64）
- Windows ARM64
- Ubuntu または Debian では、アプリはベータ版です。Linux インストール手順に従って apt でインストールしてください。

インストール後、Claude を起動し、サインインして、**Code** タブをクリックしてコーディングを開始します。アプリには Claude Code が含まれているため、CLI を別途インストールする必要はありません。有料サブスクリプションが必要です。

デスクトップアプリについて詳しく →

Web

ローカルセットアップなしでブラウザで Claude Code を実行します。長時間実行されるタスクを開始して完了を待つ、ローカルにないリポジトリで作業する、または複数のタスクを並行実行できます。

長期的な作業の場合は、プロジェクトを作成して、Claude に複数のセッションを調整させることができます。デスクトップブラウザと iOS および Android 用 Claude アプリで利用できます。

claude.ai/code でコーディングを開始します。

Web ではじめる →

JetBrains

IntelliJ IDEA、PyCharm、WebStorm、その他の JetBrains IDE 用のプラグインで、インタラクティブな差分表示と選択コンテキスト共有機能があります。

JetBrains Marketplace から Claude Code プラグイン をインストールして IDE を再起動します。プラグインには Claude Code CLI が必要で、別途インストールしてください。*JetBrains* **セットアップステップ** (ページ 206) を参照してください。

JetBrains **ではじめる** → (ページ 205)

1.2 できること

Claude Code を使用できるいくつかの方法を紹介します：

先延ばしにしている作業を自動化する

Claude Code は、1 日を費やす退屈なタスクを処理します：テストされていないコードのテスト作成、プロジェクト全体のリントエラー修正、マージコンフリクト解決、依存関係の更新、リリースノートの作成。

```
claude "write tests for the auth module, run them, and fix any failures"
```

機能を構築し、バグを修正する

プレーンテキストで実現したいことを説明します。Claude Code はアプローチを計画し、複数のファイル間でコードを作成し、動作を検証します。

バグの場合は、エラーメッセージを貼り付けるか、症状を説明します。Claude Code はコードベース全体で問題をトレースし、根本原因を特定し、修正を実装します。詳細な例については **一般的なワークフロー** (ページ 105) を参照してください。

コミットとプルリクエストを作成する

Claude Code は git と直接連携します。変更をステージングし、コミットメッセージを作成し、ブランチを作成し、プルリクエストを開きます。

```
claude "commit my changes with a descriptive message"
```

CI では、*GitHub Actions* (ページ 247) または *GitLab CI/CD* (ページ 261) でコードレビューと問題トリアージを自動化できます。

MCP でツールを接続する

Model Context Protocol（MCP） (ページ 363) は、AI ツールを外部データソースに接続するためのオープンスタンダードです。MCP を使用すると、Claude Code は Google Drive のデザインドキュメントを読み取り、Jira のチケットを更新し、Slack からデータをプルするか、独自のカスタムツーリングを使用できます。MCP クイックスタート は、最初のサーバーをエンドツーエンドで接続します。

指示、スキル、フックでカスタマイズする

CLAUDE.md (ページ 997) はプロジェクトルートに追加するマークダウンファイルで、Claude Code はすべてのセッションの開始時に読み取ります。コーディング標準、アーキテクチャの決定、推奨ライブラリ、レビューチェックリストを設定するために使用します。リポジトリに他のコーディングエージェント用の `AGENTS.md` が既にある場合、Claude Code は **それを読み取ることができます** (ページ 1008)。Claude は **自動メモリ** (ページ 1013) も構築し、セッション間で学習内容を保存し、何も書かずに共有します。

スキル (ページ 275) を作成して、チームが共有できる反復可能なワークフローをパッケージ化します (`/review-pr` や `/deploy-staging` など)。

フック (ページ 443) を使用すると、ファイル編集後の自動フォーマットやコミット前のリント実行など、Claude Code アクション前後にシェルコマンドを実行できます。

エージェントを並行実行し、カスタムエージェントを構築する

複数の *Claude Code* エージェント (ページ 317) を生成して、タスクの異なる部分に同時に取り組みます。リードエージェントが作業を調整し、サブタスクを割り当て、結果をマージします。

複数の完全なセッションを並行して実行し、1 つの画面から監視するには、バックグラウンドエージェント を使用します。完全にカスタムなワークフローの場合、Agent SDK を使用すると、Claude Code のツールと機能を活用した独自のエージェントを構築でき、オーケストレーション、ツールアクセス、権限を完全に制御できます。

CLI でパイプ、スクリプト、自動化する

Claude Code は構成可能で Unix 哲学に従います。ログをパイプで渡し、CI で実行するか、他のツールと連鎖させます：

```
# 最近のログ出力を分析する
tail -200 app.log | claude -p "Slack me if you see any anomalies"

# CI で翻訳を自動化する
claude -p "translate new strings into French and raise a PR for review"

# ファイル全体でバルク操作
git diff main --name-only | claude -p "review these changed files for security␣
↪issues"
```

すべてのコマンドとフラグのセットについては *CLI* **リファレンス** (ページ 45) を参照してください。

定期的なタスクをスケジュール設定する

繰り返される作業を自動化するためにスケジュールで Claude を実行します：朝の PR レビュー、夜間の CI 障害分析、週次の依存関係監査、または PR マージ後のドキュメント同期。

- ルーティン はクラウドで実行されるため、コンピューターがオフの場合でも実行し続けます。API 呼び出しまたは GitHub イベントでトリガーすることもできます。Web、デスクトップアプリ、または CLI で `/schedule` を実行して作成します。
- デスクトップスケジュール済みタスク はマシン上で実行され、ローカルファイルとツールに直接アクセスできます
- `/loop` は CLI セッション内でプロンプトを繰り返し、クイックポーリングを行います

どこからでも作業する

セッションは単一のサーフェスに限定されません。コンテキストが変わるにつれて、環境間で作業を移動します：

- デスクから離れて、電話または リモートコントロール を使用した任意のブラウザから作業を続けます
- *Dispatch* (ページ 151) にメッセージを送信して、電話からタスクを送信し、作成されたデスクトップセッションを開きます
- *Web* (ページ 223) または Claude モバイルアプリ で長時間実行されるタスクを開始し、`claude --teleport` でターミナルにプルします。Teleport には claude.ai サブスクリプションが必要です。
- `/desktop` を実行して、現在のターミナルセッションを **デスクトップアプリ** (ページ 137) に続行し、差分を視覚的に確認できます。`/desktop` ハンドオフには claude.ai サブスクリプションが必要です。macOS と x64 Windows で利用可能です。
- チームチャットからタスクをルーティング：*Slack* (ページ 239) で `@Claude` にメンションしてバグレポートを送信し、プルリクエストを取得します

1.3 Claude Code をどこでも使用する

各 **サーフェス** は同じ基盤となる Claude Code エンジンに接続するため、repo の CLAUDE.md ファイル、設定、MCP サーバーはすべてのサーフェスで機能します。

上記の *Terminal* (ページ 11)、*VS Code* (ページ 173)、*JetBrains* (ページ 205)、*Desktop* (ページ 137)、*Web* (ページ 223) サーフェスを超えて、Claude Code は CI/CD、チャット、ブラウザワークフローと統合します：

実現したいこと	最適なオプション
ローカルセッションを電話または別のデバイスから続行する	リモートコントロール
Telegram、Discord、iMessage、または独自の webhook からセッションにイベントをプッシュする	チャネル
ローカルでタスクを開始し、モバイルで続行する	`claude --cloud` (ページ 225)、その後 Claude モバイルアプリ
定期的なスケジュールで Claude を実行する	ルーティン または デスクトップスケジュール済みタスク
PR レビューと問題トリアージを自動化する	*GitHub Actions* (ページ 247) または *GitLab CI/CD* (ページ 261)
すべての PR で自動コードレビューを取得する	GitHub Code Review
Slack からプルリクエストへバグレポートをルーティングする	*Slack* (ページ 239)
ライブ Web アプリケーションをデバッグする	*Chrome* (ページ 213)
独自のワークフロー用のカスタムエージェントを構築する	Agent SDK

1.4 次のステップ

Claude Code をインストールしたら、これらのガイドでさらに詳しく学べます。

- **クイックスタート** (ページ 11)：コードベースの探索から修正のコミットまで、最初の実際のタスクを実行します
- **指示とメモリを保存する** (ページ 997)：CLAUDE.md ファイルと自動メモリで Claude に永続的な指示を与えます
- **一般的なワークフロー** (ページ 105) と **ベストプラクティス** (ページ 115)：Claude Code から最大限の価値を得るためのパターン
- Claude Academy：無料の自習コース。Claude Code 101 と Claude Code in Action を含みます
- すべてのタスクのためのハーネス：Claude Code チームが 動的ワークフロー を使用して大規模にサブエージェントを調整する方法
- **設定** (ページ 711)：ワークフローに合わせて Claude Code をカスタマイズします
- **トラブルシューティング** (ページ 37)：一般的な問題の解決策
- code.claude.com：デモ、価格設定、製品の詳細

第2章　クイックスタート

Claude Code へようこそ！

このクイックスタートガイドを使用すれば、数分で AI を活用したコーディング支援を利用できます。このガイドを終了する頃には、一般的な開発タスクに Claude Code を使用する方法を理解できるようになります。

2.1 始める前に

以下を確認してください：

- ターミナルまたはコマンドプロンプトが開いている
  - ターミナルを使用したことがない場合は、ターミナルガイドをご覧ください
- 作業するコードプロジェクトがある
- Claude サブスクリプション（Pro、Max、Team、または Enterprise）、Claude Console アカウント、または**サポートされているクラウドプロバイダー** (ページ 847) 経由のアクセスがある

注釈:

このガイドはターミナル CLI について説明しています。Claude Code はウェブ、**デスクトップアプリ** (ページ 137)、*VS Code* (ページ 173) および *JetBrains IDE* (ページ 205)、*Slack* (ページ 239)、および *GitHub Actions* (ページ 247) と *GitLab* (ページ 261) を使用した CI/CD でも利用できます。**すべてのインターフェース** (ページ 8) を参照してください。

2.2 ステップ 1：Claude Code をインストールする

Claude Code をインストールするには、ターミナルを開いてシステムのコマンドを実行してください。ターミナルを使用したことがない場合は、ターミナルガイドでターミナルを開いてコマンドを貼り付ける方法を確認できます。

ネイティブインストール（推奨）

macOS、Linux、WSL：

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell：

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD：

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del␣
↪install.cmd
```

インストーラーが完了したら、新しいターミナルウィンドウを開いて `claude --version` を実行してください。インストールが正常に完了すると、バージョン番号が表示されます。シェルが `claude` が見つからない、または認識されていないと表示される場合は、インストールディレクトリがまだ PATH に含まれていません。PATH を修正するを参照してください。

`The token '&&' is not a valid statement separator` というエラーが表示される場合は、CMD ではなく PowerShell を使用しています。`'irm' is not recognized as an internal or external command` というエラーが表示される場合は、PowerShell ではなく CMD を使用しています。PowerShell を使用している場合、プロンプトに `PS C:\` と表示され、CMD を使用している場合は `PS` なしで `C:\` と表示されます。

インストールコマンドが `syntax error near unexpected token '<'`、`403`、またはその他の curl エラーで失敗する場合は、インストールのトラブルシューティングを参照して、エラーを修正方法に照合し、代替インストール方法を確認してください。

Git for Windows は、Claude Code が Bash ツールを使用できるようにネイティブ Windows で推奨されます。Git for Windows がインストールされていない場合、Claude Code はシェルツールとして PowerShell を代わりに使用します。WSL セットアップは Git for Windows を必要としません。

注釈:

ネイティブインストールは、最新バージョンに保つために自動的にバックグラウンドで更新されます。

Homebrew

```
brew install --cask claude-code
```

Homebrew は 2 つの cask を提供しています。`claude-code` は安定リリースチャネルを追跡しており、通常は約 1 週間遅れており、大きな回帰を伴うリリースをスキップします。`claude-code@latest` は最新チャネルを追跡し、新しいバージョンが出荷されるとすぐに受け取ります。

注釈:

Homebrew インストールは自動更新されません。インストールした cask に応じて、`brew upgrade claude-code` または `brew upgrade claude-code@latest` を実行して、最新の機能とセキュリティ修正を取得してください。

WinGet

```
winget install Anthropic.ClaudeCode
```

注釈:

WinGet インストールは自動更新されません。最新の機能とセキュリティ修正を取得するために、定期的に `winget upgrade Anthropic.ClaudeCode` を実行してください。

また、Debian、Fedora、RHEL、Alpine で *apt、dnf、または apk* (ページ 28) を使用してインストールすることもできます。

インストールが正常に機能したことを確認するには、以下を実行してください：

```
claude --version
```

このコマンドは、バージョン番号の後に `(Claude Code)` を出力します。

2.3 ステップ 2：アカウントにログインする

Claude Code を使用するにはアカウントが必要です。`claude` コマンドでインタラクティブセッションを開始すると、初回使用時にログインするよう求められます：

```
claude
```

Claude サブスクリプションまたは Console アカウントの場合は、プロンプトに従ってブラウザで認証を完了してください。`ANTHROPIC_API_KEY` 環境変数を設定している場合、Claude Code はログインプロンプトをスキップし、代わりにキーを承認するよう求めます。後でアカウントを切り替えるか再認証するには、実行中のセッション内で `/login` と入力します：

```
/login
```

以下のいずれかのアカウントタイプを使用してログインできます：

- Claude Pro、Max、Team、または Enterprise（推奨）
- Claude Console（プリペイドクレジット付き API アクセス）。初回ログイン時に、コスト追跡を一元化するために「Claude Code」ワークスペースが Console に自動的に作成されます。
- *Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry* (ページ 847)（エンタープライズクラウドプロバイダー）
- 組織が実行している自己ホスト型の Claude apps gateway：管理者がゲートウェイ URL を事前に設定し、`/login` で **Cloud gateway** 画面が直接開き、企業 SSO でサインインできます

ログイン後、認証情報が保存され、再度ログインする必要はありません。詳細は **認証情報管理** (ページ 857) をご覧ください。

2.4 ステップ 3：最初のセッションを開始する

任意のプロジェクトディレクトリでターミナルを開き、Claude Code を開始します：

```
cd /path/to/your/project
claude
```

/path/to/your/project を、作業したいプロジェクトのパスに置き換えてください。

Claude Code プロンプトが表示され、バージョン、現在のモデル、および作業ディレクトリが上に表示されます。利用可能なコマンドについては /help を入力するか、前の会話を続行するには /resume を入力します。

2.5 ステップ 4：最初の質問をする

コードベースを理解することから始めましょう。以下のコマンドのいずれかを試してください：

```
what does this project do?
```

Claude はファイルを分析して概要を提供します。より具体的な質問をすることもできます：

```
what technologies does this project use?
```

```
where is the main entry point?
```

```
explain the folder structure
```

Claude 自体の機能について質問することもできます：

```
what can Claude Code do?
```

```
how do I create custom skills in Claude Code?
```

```
can Claude Code work with Docker?
```

注釈:

Claude Code は必要に応じてプロジェクトファイルを読み込みます。コンテキストを手動で追加する必要はありません。

2.6 ステップ 5：最初のコード変更を行う

次に、Claude Code に実際のコーディングを行わせましょう。簡単なタスクを試してください：

```
メインファイルに hello world 関数を追加してください
```

Claude Code は適切なファイルを見つけて、変更内容を表示します。変更を行う前に確認を求める場合は、**Yes** を選択して承認してください。

Claude Code v2.1.283 以降では、auto モードはインタラクティブターミナルセッションの組み込みの開始権限モードです。分類器があなたの代わりにアクションをレビューし、Claude はほとんどのファイルを編集し、ほとんどのコマンドをあなたに尋ねることなく実行します。それより前のバージョンでは、auto モードは Pro、Max、Team プランのみで組み込みの開始権限モードです。インストール直後に開始するセッションについては、インストールまたはアップグレード後の最初のセッションを参照してください。

注釈:

設定またはお客様の組織が異なる開始権限モードを設定できます。セッションが開始する権限モードに、その内容が記載されています。いつでも `Shift+Tab` を押して、現在のセッションの権限モードを切り替えることができます。

2.7 ステップ 6：Claude Code で Git を使用する

Claude Code は Git 操作を会話形式にします：

```
どのファイルを変更しましたか？
```

```
説明的なメッセージで変更をコミットしてください
```

より複雑な Git 操作を求めることもできます：

```
feature/quickstart という名前の新しいブランチを作成してください
```

```
最後の 5 つのコミットを表示してください
```

```
マージコンフリクトの解決を手伝ってください
```

2.8 ステップ 7：バグを修正するか機能を追加する

Claude はデバッグと機能実装に長けています。

自然言語で実現したいことを説明します：

```
ユーザー登録フォームに入力検証を追加してください
```

または既存の問題を修正します：

```
ユーザーが空のフォームを送信できるバグがあります。修正してください
```

Claude Code は以下を実行します：

- 関連するコードを見つける
- コンテキストを理解する
- ソリューションを実装する
- 利用可能な場合はテストを実行する

2.9 ステップ 8：他の一般的なワークフローを試す

Claude と連携する方法は多数あります：

コードをリファクタリングする

```
認証モジュールをリファクタリングして、コールバックの代わりに async/await を使用するようにしてください
```

テストを書く

```
計算機関数のユニットテストを書いてください
```

ドキュメントを更新する

```
インストール手順で README を更新してください
```

コードレビュー

```
変更をレビューして改善を提案してください
```

Tip:

有能な同僚と話すように Claude と話してください。実現したいことを説明すれば、それを実現するのに役立ちます。

2.10 必須コマンド

日常的に使用する最も重要なコマンドは以下の通りです。シェルコマンドはターミナルから実行して Claude Code を開始または再開します。セッションコマンドは Claude Code 起動後に実行します。

シェルコマンド

コマンド	機能	例
`claude`	インタラクティブモードを開始する	`claude`
`claude "task"`	初期プロンプト付きでインタラクティブモードを開始する	`claude "fix the build error"`

次のページに続く

表 1 – 前のページからの続き

コマンド	機能	例
`claude -p "query"`	1 回限りのクエリを実行してから終了する	`claude -p "explain this function"`
`claude -c`	現在のディレクトリで最新の会話を続行する	`claude -c`
`claude -r`	前の会話を再開する	`claude -r`

セッションコマンド

コマンド	機能	例
`/clear`	会話履歴をクリアする	`/clear`
`/help`	利用可能なコマンドを表示する	`/help`
`/exit` または Ctrl+D 2 回	Claude Code を終了する	`/exit`

シェルコマンドの完全なリストについては *CLI* **リファレンス** (ページ 45) を、セッションコマンドの完全なリストについては コマンドリファレンスを参照してください。

2.11 初心者向けのプロのヒント

詳細については、**ベストプラクティス** (ページ 115) と**一般的なワークフロー** (ページ 105) を参照してください。

リクエストを具体的にする

代わりに：' バグを修正してください'

試してください：' ユーザーが間違った認証情報を入力した後に空白の画面が表示されるログインバグを修正してください'

段階的な指示を使用する

複雑なタスクをステップに分割します：

```
1. ユーザープロファイル用の新しいデータベーステーブルを作成する
2. ユーザープロファイルを取得および更新するための API エンドポイントを作成する
3. ユーザーが自分の情報を表示および編集できるウェブページを構築する
```

Claude に最初に探索させる

変更を加える前に、Claude にコードを理解させます：

```
データベーススキーマを分析する
```

```
英国の顧客によって最も頻繁に返品される製品を表示するダッシュボードを構築する
```

ショートカットで時間を節約する

- / を入力してすべてのコマンドとスキルを表示する
- Tab キーでコマンド補完を使用する

- ↑ キーでコマンド履歴を表示する
- `Shift+Tab` を押して権限モードをサイクルさせる

2.12 次のステップ

基本を学習したので、より高度な機能を探索してください：

- Claude Code の仕組み

  agentic ループ、組み込みツール、および Claude Code がプロジェクトと相互作用する方法を理解する

- **ベストプラクティス** (ページ 115)

  効果的なプロンプティングとプロジェクト設定でより良い結果を得る

- **一般的なワークフロー** (ページ 105)

  一般的なタスクのステップバイステップガイド

- Claude Code を拡張する

  CLAUDE.md、スキル、フック、MCP などでカスタマイズする

2.13 ヘルプを取得する

- **Claude Code 内**：`/help` を入力するか、「how do I」という質問をする
- **ドキュメント**：ここにいます！他のガイドを参照してください
- **コース**：Claude Code 101 と Claude Academy の他の無料のセルフペースコースを受講する
- **コミュニティ**：Discord に参加してヒントとサポートを得る

第3章　高度なセットアップ

> Claude Code のシステム要件、プラットフォーム固有のインストール、バージョン管理、およびアンインストール。

このページでは、システム要件、プラットフォーム固有のインストール詳細、更新、およびアンインストールについて説明します。初回セッションのガイド付きウォークスルーについては、**クイックスタート** (ページ 11) を参照してください。ターミナルを使用したことがない場合は、ターミナルガイドを参照してください。

3.1 システム要件

Claude Code は以下のプラットフォームと構成で実行されます。

- **オペレーティングシステム**:
  - macOS 13.0 以上
  - Windows 10 1809 以上または Windows Server 2019 以上
  - Ubuntu 20.04 以上
  - Debian 10 以上
  - Alpine Linux 3.19 以上
- **ハードウェア**: 4 GB 以上の RAM、x64 または ARM64 プロセッサ
- **ネットワーク**: インターネット接続が必要です。**ネットワーク構成** (ページ 904) を参照してください。
- **シェル**: Bash、Zsh、PowerShell、または CMD。
- **場所**: Anthropic サポート対象国

3.1.1 追加の依存関係

- **ripgrep**: 通常は Claude Code に含まれています。検索が失敗する場合は、**検索トラブルシューティング** (ページ 41) を参照してください。

3.2 Claude Code をインストール

Tip:

グラフィカルインターフェースをお好みですか？Desktop app を使用すると、ターミナルなしで Claude Code を使用できます。macOS、Windows、または Linux でダウンロードしてください。

Claude Code をインストールするには、ターミナルを開いてシステムのコマンドを実行してください。ターミナルを使用したことがない場合は、ターミナルガイドでターミナルを開いてコマンドを貼り付ける方法を確認できます。

ネイティブインストール（推奨）

macOS、Linux、WSL：

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell：

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD：

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del␣
↪install.cmd
```

インストーラーが完了したら、新しいターミナルウィンドウを開いて `claude --version` を実行してください。インストールが正常に完了すると、バージョン番号が表示されます。シェルが `claude` が見つからない、または認識されていないと表示される場合は、インストールディレクトリがまだ PATH に含まれていません。PATH を修正するを参照してください。

`The token '&&' is not a valid statement separator` というエラーが表示される場合は、CMD ではなく PowerShell を使用しています。`'irm' is not recognized as an internal or external command` というエラーが表示される場合は、PowerShell ではなく CMD を使用しています。PowerShell を使用している場合、プロンプトに `PS C:\` と表示され、CMD を使用している場合は `PS` なしで `C:\` と表示されます。

インストールコマンドが `syntax error near unexpected token '<'`、`403`、またはその他の curl エラーで失敗する場合は、インストールのトラブルシューティングを参照して、エラーを修正方法に照合し、代替インストール方法を確認してください。

Git for Windows は、Claude Code が Bash ツールを使用できるようにネイティブ Windows で推奨されます。Git for Windows がインストールされていない場合、Claude Code はシェルツールとして PowerShell を代わりに使用します。WSL セットアップは Git for Windows を必要としません。

注釈:

ネイティブインストールは、最新バージョンに保つために自動的にバックグラウンドで更新されます。

Homebrew

```
brew install --cask claude-code
```

Homebrew は 2 つの cask を提供しています。`claude-code` は安定リリースチャネルを追跡しており、通常は約 1 週間遅れており、大きな回帰を伴うリリースをスキップします。`claude-code@latest` は最新チャネルを追跡し、新しいバージョンが出荷されるとすぐに受け取ります。

注釈:

Homebrew インストールは自動更新されません。インストールした cask に応じて、`brew upgrade claude-code` または `brew upgrade claude-code@latest` を実行して、最新の機能とセキュリティ修正を取得してください。

WinGet

```
winget install Anthropic.ClaudeCode
```

注釈:

WinGet インストールは自動更新されません。最新の機能とセキュリティ修正を取得するために、定期的に `winget upgrade Anthropic.ClaudeCode` を実行してください。

また、Debian、Fedora、RHEL、Alpine で *apt、dnf、または apk* (ページ 28) を使用してインストールすることもできます。

インストールが完了したら、作業するプロジェクトでターミナルを開き、Claude Code を起動します。

```
claude
```

Claude Code はターミナルで対話的なセッションを開きます。

インストール中に問題が発生した場合は、インストールとログインのトラブルシューティングを参照してください。

3.2.1 Windows でのセットアップ

Claude Code をネイティブに Windows で実行することも、WSL 内で実行することもできます。プロジェクトの場所と必要な機能に基づいて選択してください。

オプション	必須	サンドボックス (ページ 1069)	使用時期
ネイティブ Windows	なし；Git for Windows はオプション	サポートされていません	Windows ネイティブプロジェクトとツール
WSL 2	WSL 2 有効	サポートされています	Linux ツールチェーンまたはサンドボックス化されたコマンド実行
WSL 1	WSL 1 有効	サポートされていません	WSL 2 が利用できない場合

オプション 1: ネイティブ Windows

PowerShell または CMD からインストールコマンドを実行します。管理者として実行する必要はありません。Git for Windows をインストールすることはオプションです。これにより、Bash ツールと Monitor ツールに必要な Git Bash が提供されます。

PowerShell または CMD からインストールするかどうかは、実行するインストールコマンドにのみ影響します。プロンプトは PowerShell では `PS C:\Users\YourName>` と表示され、CMD では `PS` なしで `C:\Users\YourName>` と表示されます。ターミナルが初めての場合は、ターミナルガイドで各ステップを説明しています。

インストール後、任意のターミナルから `claude` を起動します。

- **Git for Windows がない場合**、Claude Code は PowerShell ツール経由でシェルコマンドを実行します。
- **Git for Windows がある場合**、Claude Code は Bash ツールに Git Bash を使用します。Claude Code が Git Bash を見つけられない場合は、*settings.json* **ファイル** (ページ 711) でパスを設定します。

```
{
  "env": {
    "CLAUDE_CODE_GIT_BASH_PATH": "C:\\Program Files\\Git\\bin\\bash.exe"
  }
}
```

Git for Windows がインストールされている場合、PowerShell ツールは Bash と並行して利用可能です。claude.ai と Console アカウントではデフォルトで有効になっており、Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry セッションでは `CLAUDE_CODE_USE_POWERSHELL_TOOL=1` で有効になります。`0` に設定してツールをオフにします。セットアップと制限については、PowerShell ツールを参照してください。

オプション 2: WSL

WSL ディストリビューションを開き、上記の**インストール手順** (ページ 20) から Linux インストーラーを実行します。PowerShell または CMD からではなく、WSL ターミナル内で `claude` をインストールして起動します。

3.2.2 Alpine Linux と musl ベースのディストリビューション

Alpine およびその他の musl/uClibc ベースのディストリビューション上に Claude Code をインストールするには、インストールコマンド用に `bash` と `curl` が必要であり、実行時に `libgcc`、`libstdc++`、および `ripgrep` が必要です。Alpine はデフォルトで `bash` と `curl` を含まないため、ドキュメント化されたイン

ストールコマンドは not found エラーで失敗します。これらをインストールするまで、ディストリビューションのパッケージマネージャーを使用してこれらのパッケージをインストールしてから、USE_BUILTIN_RIPGREP=0 を設定します。

この例は Alpine で必要なパッケージをインストールします。

```
apk add bash curl libgcc libstdc++ ripgrep
```

Alpine では、ripgrep はコミュニティリポジトリにあります。apk がパッケージが見つからないと報告する場合は、Alpine バージョンを使用して /etc/apk/repositories にコミュニティリポジトリを追加します。

```
echo "https://dl-cdn.alpinelinux.org/alpine/v3.22/community" >> /etc/apk/repositories
```

apk update を実行してパッケージインデックスを更新し、apk add コマンドを再試行します。

次に、settings.json ファイルで USE_BUILTIN_RIPGREP を 0 に設定します。

```
{
  "env": {
    "USE_BUILTIN_RIPGREP": "0"
  }
}
```

3.3 インストールを確認

インストール後、Claude Code が機能していることを確認します。

```
claude --version
```

正常にインストールされている場合は、2.1.211 (Claude Code) のようなバージョン番号が表示されます。

これが command not found または別のエラーで失敗する場合は、インストールとログインのトラブルシューティングを参照してください。

インストールと構成をより詳しく確認するには、*claude doctor* (ページ 42) を実行します。

```
claude doctor
```

claude doctor は、セッションを開始せずに読み取り専用のインストールと設定の診断を出力します。これには、インストールの正常性、設定ファイルの検証エラー、および推奨される修正を含む警告が含まれます。

3.4 認証

Claude Code には、Pro、Max、Team、Enterprise、または Console アカウントが必要です。無料の claude.ai プランには Claude Code アクセスは含まれていません。*Amazon Bedrock* (ページ 863)、*Google Cloud の Agent Platform* (ページ 881)、または *Microsoft Foundry* (ページ 891) などのサードパーティ API プロバイダーで Claude Code を使用することもできます。

インストール後、claude を実行してブラウザーのプロンプトに従ってログインします。ANTHROPIC_API_KEY 環境変数が設定されている場合、Claude Code はブラウザーを開く代わりに、キーを承認するよう 1 回プロンプトを表示します。すべてのアカウントタイプとチームセットアップオプションについては、**認証** (ページ 851) を参照してください。

3.5 Claude Code を更新

ネイティブインストールは自動的にバックグラウンドで更新されます。**リリースチャネルを構成** (ページ 25) して、更新をすぐに受け取るか遅延安定スケジュールで受け取るかを制御することも、**自動更新を完全に無効にする** (ページ 26) こともできます。Homebrew、WinGet、および *Linux* **パッケージマネージャー** (ページ 28) インストールは手動更新が必要です。

3.5.1 自動更新

Claude Code は起動時と実行中に定期的に更新をチェックします。更新はバックグラウンドでダウンロードおよびインストールされ、次に Claude Code を起動するときに有効になります。

claude doctor を実行して、最新の更新試行の結果を確認します。

macOS と Linux では、ネイティブインストーラーは ~/.local/bin/claude のランチャーを ~/.local/share/claude/versions/ へのシンボリックリンクとして管理します。そのランチャーを独自のスクリプトまたはシンボリックリンクに置き換えた場合、自動更新と claude update はそれをそのまま保持します。新しいバージョンは引き続き versions/ ディレクトリの下にインストールされ、ランチャーはどのバージョンを実行するかを決定します。v2.1.207 より前では、自動アップデーターは毎回の更新時にそのパスのカスタムランチャーを独自のシンボリックリンクに置き換えていました。

カスタムランチャーを使用する場合、Claude Code はランチャーがどのバージョンを必要とするかを判断できないため、インストール済みのすべてのバージョンをディスク上に保持します。claude doctor はネイティブインストーラーが作成しなかったランチャーを報告します。

Claude Code にランチャーを再度管理させるには、~/.local/bin/claude を削除して claude update を実行します。

npm グローバルインストールが npm グローバルディレクトリが書き込み可能でないため自動更新できない場合、Claude Code は起動時に 1 回限りの通知を表示し、claude doctor は利用可能な修正を一覧表示します。詳細については、インストール中の権限エラーを参照してください。

注釈:

Homebrew、WinGet、apt、dnf、および apk インストールはデフォルトでは自動更新されません。Homebrew と WinGet でオプトインするには、以下を参照してください。Homebrew を手動でアップグレードするには、`brew upgrade claude-code` または `brew upgrade claude-code@latest` を実行します (インストールした cask によって異なります)。WinGet の場合は、`winget upgrade Anthropic.ClaudeCode` を実行します。Linux パッケージマネージャーの場合は、*Linux* **パッケージマネージャーでインストール** (ページ 28) のアップグレードコマンドを参照してください。

Claude Code が Homebrew または WinGet で自動的にアップグレードコマンドを実行するようにするには、`CLAUDE_CODE_PACKAGE_MANAGER_AUTO_UPDATE` を 1 に設定します。Claude Code は新しいバージョンが利用可能な場合、バックグラウンドでアップグレードを実行し、成功時に再起動プロンプトを表示します。アップグレードは Claude Code パッケージのみを対象とし、インストール済みの他のソフトウェアには影響しません。

WinGet では、Claude Code の実行中にアップグレードが失敗する場合があります。これは Windows が実行可能ファイルをロックするためです。その場合、Claude Code は代わりに手動コマンドを表示します。apt、dnf、および apk は、これらのコマンドが昇格された権限を必要とするため、手動アップグレードが必要です。

既知の問題: Claude Code は、新しいバージョンがこれらのパッケージマネージャーで利用可能になる前に更新を通知する場合があります。アップグレードが失敗した場合は、しばらく待ってからもう一度試してください。

Homebrew はアップグレード後、古いバージョンをディスク上に保持します。`brew cleanup` を定期的に実行してディスク容量を回収します。

3.5.2 リリースチャネルを構成

`autoUpdatesChannel` 設定を使用して、Claude Code が自動更新と `claude update` に従うリリースチャネルを制御します。

- `"latest"`、デフォルト: リリースされるとすぐに新機能を受け取ります
- `"stable"`: 通常約 1 週間前のバージョンを使用し、大きな回帰を伴うリリースをスキップします

これを `/config` → **自動更新チャネル**経由で構成するか、*settings.json* **ファイル** (ページ 711) に追加します。

```
{
  "autoUpdatesChannel": "stable"
}
```

新しくリリースされたモデルでは、stable チャネルが提供するバージョンよりも新しい Claude Code バージョンが必要になる場合があります。そのモデルをすぐに使用するには、latest チャネルに移行してください。

エンタープライズデプロイメントの場合、管理設定を使用して、組織全体で一貫したリリースチャネルを適用できます。

Homebrew インストールは、この設定ではなく cask 名でチャネルを選択します。`claude-code` は安定版を追跡し、`claude-code@latest` は最新版を追跡します。

apt、dnf、および apk リポジトリからのインストールは、この設定ではなくリポジトリでチャネルを選択します。チャネルを切り替えるには、*Linux* **パッケージマネージャーでインストール** (ページ 28) の手順に従ってください。

3.5.3 最小バージョンをピン留め

minimumVersion 設定は下限を確立します。バックグラウンド自動更新と claude update は、この値より下のバージョンのインストールを拒否するため、"stable" チャネルに移動しても、既に新しい "latest" ビルドを使用している場合はダウングレードされません。

/config 経由で "latest" から "stable" に切り替えると、現在のバージョンに留まるか、ダウングレードを許可するかを選択するよう求められます。留まることを選択すると、minimumVersion がそのバージョンに設定されます。"latest" に戻すと、それがクリアされます。

settings.json **ファイル** (ページ 711) に追加して、下限を明示的にピン留めします。

```
{
  "autoUpdatesChannel": "stable",
  "minimumVersion": "2.1.100"
}
```

管理設定では、これはユーザーおよびプロジェクト設定がオーバーライドできない組織全体の最小値を適用します。

minimumVersion ピンは更新のみを制約します。Claude Code がバージョン範囲外で起動することを拒否するようにするには、代わりに管理設定の requiredMinimumVersion と requiredMaximumVersion を使用します。更新は requiredMaximumVersion の上限も尊重します。requiredMinimumVersion と requiredMaximumVersion を参照してください。

3.5.4 自動更新を無効にする

settings.json ファイルの env キーで DISABLE_AUTOUPDATER を "1" に設定します。

```
{
  "env": {
    "DISABLE_AUTOUPDATER": "1"
  }
}
```

ネイティブまたは npm インストールで、claude doctor を実行して変更が有効になったことを確認し、Auto-updates 行が enabled ではなく disabled (set by env: DISABLE_AUTOUPDATER) を表示していることを確認します。

DISABLE_AUTOUPDATER はバックグラウンドチェックのみを停止します。claude update と claude install は引き続き機能します。手動更新を含むすべての更新パスをブロックするには、代わりに DISABLE_UPDATES を設定します。独自のチャネルを通じて Claude Code を配布し、ユーザーが提供するバージョンに留まる必要がある場合に使用します。

3.5.5 手動で更新

次のバックグラウンドチェックを待たずに更新をすぐに適用するには、以下を実行します。

```
claude update
```

更新がインストールされると、コマンドは `Successfully updated from <old version> to version <new version>` を報告します。既に最新バージョンを使用している場合は、`Claude Code is up to date (<version>)` を報告します。Homebrew、WinGet、または apk で管理されるインストールは、代わりに `Claude is up to date!` を報告します。

3.6 高度なインストールオプション

これらのオプションは、バージョンピニング、Linux パッケージマネージャー、npm、ネットワークストレージ、およびバイナリ整合性の検証用です。

3.6.1 特定のバージョンをインストール

ネイティブインストーラーは、特定のバージョン番号またはリリースチャネル（`latest` または `stable`）を受け入れます。インストール時に選択したチャネルが自動更新のデフォルトになります。詳細については、**リリースチャネルを構成** (ページ 25) を参照してください。

最新バージョンをインストールするには（デフォルト）：

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash
```

Windows PowerShell

```
irm https://claude.ai/install.ps1 | iex
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del␣
↪install.cmd
```

安定版をインストールするには:

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash -s stable
```

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) stable
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd stable &&␣
↪del install.cmd
```

特定のバージョン番号をインストールするには:

macOS, Linux, WSL

```
curl -fsSL https://claude.ai/install.sh | bash -s 2.1.89
```

Windows PowerShell

```
& ([scriptblock]::Create((irm https://claude.ai/install.ps1))) 2.1.89
```

Windows CMD

```
curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd 2.1.89 &&␣
↪del install.cmd
```

インストールされたバージョンを確認するには、`claude --version` を実行します。このコマンドは、`2.1.89 (Claude Code)` など、渡した正確なバージョンを出力します。

3.6.2 Linux パッケージマネージャーでのインストール

Claude Code は署名付き apt、dnf、および apk リポジトリを公開しています。各リポジトリは 2 つのチャネルを提供します。`stable` は通常約 1 週間前のバージョンを提供し、大きな回帰を伴うリリースをスキップします。`latest` はリリースが出荷されるとすぐにすべてのリリースを提供します。以下のコマンドは `stable` チャネルを構成します。これはほとんどのユーザーに適しています。各タブは `latest` リポジトリ URL も表示します。パッケージマネージャーのインストールは Claude Code を通じて自動更新されません。更新は通常のシステムアップグレードワークフローを通じて提供されます。

すべてのリポジトリは *Claude Code* **リリース署名キー** (ページ 31) で署名されています。キーを信頼する前に、各タブで説明されているとおりに検証してください。

apt

Debian および Ubuntu 用です。以下のインストールコマンドは `curl` で署名キーをダウンロードし、`gpg` で検証します。新しい Debian および Ubuntu インストールには `curl` と `gpg` が含まれていない場合があります。いずれかのコマンドが `command not found` を報告する場合は、まず両方をインストールしてください:

```
sudo apt install curl gnupg
```

署名キーをダウンロードします:

```
sudo install -d -m 0755 /etc/apt/keyrings
sudo curl -fsSL https://downloads.claude.ai/keys/claude-code.asc \
  -o /etc/apt/keyrings/claude-code.asc
```

このダウンロードが失敗した場合、後で `apt update` が `NO_PUBKEY BAA929FF1A7ECACE` で失敗します。キーがダウンロードされ、Anthropic に属していることを確認してから続行します:

```
gpg --show-keys /etc/apt/keyrings/claude-code.asc
```

gpg が出力するフィンガープリントは 31DDDE24DDFAB679F42D7BD2BAA929FF1A7ECACE である必要があります。gpg がファイルを開けない、または有効な OpenPGP データが含まれていないと報告する場合、ダウンロードが失敗したか、間違ったコンテンツが返されました。ネットワークが downloads.claude.ai に到達できることを確認してから、ダウンロードコマンドを再実行します。

stable チャネルでリポジトリを登録してインストールします:

```
echo "deb [signed-by=/etc/apt/keyrings/claude-code.asc] https://downloads.claude.
↪ai/claude-code/apt/stable stable main" \
  | sudo tee /etc/apt/sources.list.d/claude-code.list
sudo apt update
sudo apt install claude-code
```

代わりに latest チャネルを使用するには、URL パスとスイート名の両方が変わります。この deb 行を使用します:

```
echo "deb [signed-by=/etc/apt/keyrings/claude-code.asc] https://downloads.claude.
↪ai/claude-code/apt/latest latest main" \
  | sudo tee /etc/apt/sources.list.d/claude-code.list
```

後で更新するには、sudo apt update && sudo apt upgrade claude-code を実行します。

dnf

Fedora および RHEL 用です。以下のコマンドは stable チャネルを構成します:

```
sudo tee /etc/yum.repos.d/claude-code.repo <<'EOF'
[claude-code]
name=Claude Code
baseurl=https://downloads.claude.ai/claude-code/rpm/stable
enabled=1
gpgcheck=1
gpgkey=https://downloads.claude.ai/keys/claude-code.asc
EOF
sudo dnf install claude-code
```

代わりに latest チャネルを使用するには、baseurl を latest リポジトリに設定します:

```
baseurl=https://downloads.claude.ai/claude-code/rpm/latest
```

dnf は最初のインストール時にキーをダウンロードし、フィンガープリントを確認するよう求めます。受け入れる前に 31DD DE24 DDFA B679 F42D 7BD2 BAA9 29FF 1A7E CACE と一致することを確認してください。

後で更新するには、sudo dnf upgrade claude-code を実行します。

apk

Alpine Linux 用です。以下のコマンドは stable チャネルを構成します:

```
wget -O /etc/apk/keys/claude-code.rsa.pub \
  https://downloads.claude.ai/keys/claude-code.rsa.pub
echo "https://downloads.claude.ai/claude-code/apk/stable" >> /etc/apk/
↪repositories
apk add claude-code
```

latest チャネルに切り替えるには、stable リポジトリ行を削除し、latest リポジトリを追加します:

```
sed -i '\|downloads.claude.ai/claude-code/apk/stable|d' /etc/apk/repositories
echo "https://downloads.claude.ai/claude-code/apk/latest" >> /etc/apk/
↪repositories
```

sha256sum /etc/apk/keys/claude-code.rsa.pub でダウンロードされたキーを検証します。これは 395759c1f7449ef4cdef305a42e820f3c766d6090d142634ebdb049f113168b6 を報告する必要があります。

後で更新するには、apk update && apk upgrade claude-code を実行します。

3.6.3 npm でのインストール

Claude Code をグローバル npm パッケージとしてインストールすることもできます。npm パッケージには Node.js 22 以上が必要です。古い Node.js バージョンでは、npm はインストール中に失敗するのではなく EBADENGINE 警告を出力します。インストールは完了し、パッケージがランタイムで Node.js を使用しないネイティブバイナリをダウンロードするため、claude は引き続き実行されます。

```
npm install -g @anthropic-ai/claude-code
```

npm パッケージは、スタンドアロンインストーラーと同じネイティブバイナリをインストールします。npm は @anthropic-ai/claude-code-darwin-arm64 などのプラットフォーム固有のオプション依存関係を通じてバイナリをプルし、postinstall ステップがそれを所定の位置にリンクします。インストールされた claude バイナリ自体は Node を呼び出しません。

サポートされている npm インストールプラットフォームは darwin-arm64、darwin-x64、linux-x64、linux-arm64、linux-x64-musl、linux-arm64-musl、win32-x64、および win32-arm64 です。パッケージマネージャーはオプション依存関係を許可する必要があります。インストール後にバイナリが見つからない場合は、トラブルシューティングを参照してください。

npm インストールをアップグレードするには、npm install -g @anthropic-ai/claude-code@latest を実行します。npm update -g は避けてください。これは元のインストールからの semver 範囲を尊重し、最新リリースに移動しない可能性があります。

警告:

sudo npm install -g を使用しないでください。これはアクセス許可の問題とセキュリティリスクにつながる可能性があります。アクセス許可エラーが発生した場合は、トラブルシューティングアクセス許可エラーを参照してください。

3.6.4 ネットワークストレージへのインストール

実行中のセッションは、起動時だけでなく動作中にも Claude Code 実行ファイルの一部をディスクから読み取ります。ネットワークストレージ上でファイルが切り詰められたり削除されたりするなどして、セッションの途中でファイルが読み取れなくなると、セッションはクラッシュします。Linux では、シェルはこれを `Bus error` として報告します。

複数のマシンにマウントされた NFS ホームなど、ホームディレクトリがネットワークストレージ上にある場合は、各セッションの実行ファイルがセッション終了まで読み取り可能な状態を保つようにインストールを配置してください:

- **ローカルディスクにインストールする**: *Linux* **パッケージマネージャー** (ページ 28) や独自のデプロイツールなどを使用して、各マシンのローカルファイルシステムにバイナリを配置します。ユーザーごとの npm プレフィックスとネイティブインストーラーのデフォルトの `~/.local/share/claude/versions/` ディレクトリは、どちらもホームディレクトリ内にあります。
- **各バージョンを専用のディレクトリに保持する**: `npm install -g` で npm インストールをその場でアップグレードすると、以前のバイナリが削除されます。複数のマシンが共有するストレージでは、他のマシン上のセッションがまだ実行中のファイルが削除されることになります。新しいバージョンはそれぞれ古いバージョンの隣にインストールし、ユーザーをそちらに移行してください。
- **古いバージョンは、どのマシンでも実行されている可能性がなくなってから削除する**: マシンは他のマシンで実行中のプロセスを確認できないため、削除前に実行中のプロセスを確認するだけでは不十分です。
- **Claude Code 自体の更新をオフにする**: `DISABLE_UPDATES` を設定し、独自のツールで新しいバージョンをインストールします。そうしないと、あるマシンでの npm インストールの自動更新が同じインプレースアップグレードを実行し、他のマシン上のセッションが実行中のバイナリを削除してしまいます。ユーザーは引き続き `claude update` や `claude install` を実行できるため、`DISABLE_AUTOUPDATER` を設定するだけでは不十分です。**自動更新を無効にする** (ページ 26) を参照してください。

ネイティブインストーラーは `~/.local/share/claude/versions/` から古いバージョンを自動的に削除します。このディレクトリが共有ストレージ上にある場合、この点が問題になります。ランチャーが指しているバージョンと、同じマシン上のセッションが実行中のバージョンに加えて、最新の 2 つのバージョンを保持し、残りを削除します。削除されたバージョンを実行している別のマシン上のセッションは、そのバイナリを失います。**カスタムランチャー** (ページ 24) を使用する場合、Claude Code はインストールされたすべてのバージョンを保持し、クリーンアップはユーザーに任されます。

3.6.5 バイナリ整合性とコード署名

各リリースは、すべてのプラットフォームバイナリの SHA256 チェックサムを含む `manifest.json` を公開します。マニフェストは Anthropic GPG キーで署名されているため、マニフェスト上の署名を検証することで、それが列挙するすべてのバイナリを推移的に検証します。

マニフェスト署名を検証

ステップ 1〜3 には、gpg と curl を備えた POSIX シェルが必要です。Windows では、Git Bash または WSL で実行します。ステップ 4 には PowerShell オプションが含まれています。

1. **公開鍵をダウンロードしてインポート**

   リリース署名キーは固定 URL で公開されています。

   ```
   curl -fsSL https://downloads.claude.ai/keys/claude-code.asc | gpg --import
   ```

   インポートされたキーのフィンガープリントを表示します。

   ```
   gpg --fingerprint security@anthropic.com
   ```

   出力にこのフィンガープリントが含まれていることを確認します:

   ```
   31DD DE24 DDFA B679 F42D  7BD2 BAA9 29FF 1A7E CACE
   ```

2. **マニフェストと署名をダウンロード**

   VERSION を検証するリリースに設定します。

   ```
   REPO=https://downloads.claude.ai/claude-code-releases
   VERSION=2.1.89
   curl -fsSLO "$REPO/$VERSION/manifest.json"
   curl -fsSLO "$REPO/$VERSION/manifest.json.sig"
   ```

3. **署名を検証**

   マニフェストに対して分離された署名を検証します。

   ```
   gpg --verify manifest.json.sig manifest.json
   ```

   有効な結果は `Good signature from "Anthropic Claude Code Release Signing <security@anthropic.com>"` を報告します。

   gpg は新しくインポートされたキーに対して `WARNING: This key is not certified with a trusted signature!` も出力します。これは予想されています。`Good signature` 行は暗号化チェックが成功したことを確認します。ステップ 1 のフィンガープリント比較はキー自体が本物であることを確認します。

4. **バイナリをマニフェストに対して確認**

   バイナリの SHA256 チェックサムを `manifest.json` の `platforms.<platform>.checksum` の下にリストされている値と比較します。以下のコマンドは、現在のディレクトリに `claude` バイナリがあることを想定しています。代わりにインストールされたネイティブバイナリを検証するには、`~/.local/share/claude/versions/VERSION` に対してコマンドを実行します。VERSION をステップ 2 で設定したリリースに置き換えます。

   Linux

```
sha256sum claude
```

macOS

```
shasum -a 256 claude
```

Windows PowerShell

```
(Get-FileHash claude.exe -Algorithm SHA256).Hash.ToLower()
```

注釈:

マニフェスト署名は `2.1.89` 以降のリリースで利用可能です。以前のリリースは分離された署名なしで `manifest.json` にチェックサムを公開します。

プラットフォームコード署名

署名付きマニフェストに加えて、個別のバイナリはサポートされている場所でプラットフォーム固有のコード署名を実行します。

- **macOS**: "Anthropic PBC" によって署名され、Apple によって公証されています。`codesign --verify --verbose ./claude` で検証します。
- **Windows**: "Anthropic, PBC" によって署名されています。`Get-AuthenticodeSignature .\claude.exe` で検証します。
- **Linux**: バイナリは個別にコード署名されていません。`claude-code-releases` バケットから直接ダウンロードするか、ネイティブインストーラーを使用する場合は、上記のマニフェスト署名で整合性を検証します。*apt、dnf、または apk* (ページ 28) でインストールする場合、パッケージマネージャーはリポジトリ署名キーを使用して署名を自動的に検証します。

3.7 Claude Code をアンインストール

Claude Code を削除するには、インストール方法の指示に従ってください。アンインストール後も `claude` が実行される場合は、2 番目のインストールまたは古いインストーラーからの残存シェルエイリアスがある可能性があります。競合するインストールを確認を参照して、それを見つけて削除してください。

3.7.1 ネイティブインストール

Claude Code バイナリとバージョンファイルを削除します：

macOS, Linux, WSL

```
rm -f ~/.local/bin/claude
rm -rf ~/.local/share/claude
```

Windows PowerShell

```
Remove-Item -Path "$env:USERPROFILE\.local\bin\claude.exe" -Force
Remove-Item -Path "$env:USERPROFILE\.local\share\claude" -Recurse -Force
```

3.7.2 Homebrew インストール

インストールした Homebrew cask を削除します。安定版 cask をインストールした場合：

```
brew uninstall --cask claude-code
```

最新版 cask をインストールした場合：

```
brew uninstall --cask claude-code@latest
```

3.7.3 WinGet インストール

WinGet パッケージを削除します：

```
winget uninstall Anthropic.ClaudeCode
```

3.7.4 apt / dnf / apk

パッケージとリポジトリ構成を削除します：

apt

```
sudo apt remove claude-code
sudo rm /etc/apt/sources.list.d/claude-code.list /etc/apt/keyrings/claude-code.
↪asc
```

dnf

```
sudo dnf remove claude-code
sudo rm /etc/yum.repos.d/claude-code.repo
```

apk

```
apk del claude-code
sed -i '\|downloads.claude.ai/claude-code/apk|d' /etc/apk/repositories
rm /etc/apk/keys/claude-code.rsa.pub
```

3.7.5 npm

グローバル npm パッケージを削除します：

```
npm uninstall -g @anthropic-ai/claude-code
```

3.7.6 構成ファイルを削除

警告:

構成ファイルを削除すると、すべての設定、許可されたツール、MCP サーバー構成、およびセッション履歴が削除されます。

VS Code 拡張機能、JetBrains プラグイン、および Desktop アプリも `~/.claude/` に書き込みます。それらのいずれかがまだインストールされている場合、ディレクトリは次回実行時に再作成されます。Claude Code を完全に削除するには、これらのファイルを削除する前に、*VS Code* **拡張機能** (ページ 203)、JetBrains プラグイン、および Desktop アプリをアンインストールしてください。

Claude Code の設定とキャッシュされたデータを削除するには：

macOS, Linux, WSL

```
# ユーザー設定と状態を削除
rm -rf ~/.claude
rm ~/.claude.json

# プロジェクト固有の設定を削除（プロジェクトディレクトリから実行）
rm -rf .claude
rm -f .mcp.json
```

Windows PowerShell

```
# ユーザー設定と状態を削除
Remove-Item -Path "$env:USERPROFILE\.claude" -Recurse -Force
Remove-Item -Path "$env:USERPROFILE\.claude.json" -Force

# プロジェクト固有の設定を削除（プロジェクトディレクトリから実行）
Remove-Item -Path ".claude" -Recurse -Force
Remove-Item -Path ".mcp.json" -Force
```

第4章　トラブルシューティング

Claude Code の高い CPU またはメモリ使用量、ハング、auto-compact スラッシング、検索の問題を修正し、その他の問題に対応する適切なページを見つけます。

このページでは、Claude Code が実行中のパフォーマンス、安定性、検索の問題について説明します。その他の問題については、問題が発生している場所に一致するページから始めてください：

症状	移動先
`command not found`、インストール失敗、PATH の問題、`EACCES`、TLS エラー	インストールとログインのトラブルシューティング
更新またはインストールダウンロードが `The connection dropped while downloading the update` または `aborted` で失敗する	エラーリファレンス
ログインループ、OAuth エラー、`403 Forbidden`、「organization disabled」、Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry 認証情報	インストールとログインのトラブルシューティング
設定が適用されない、hooks が実行されない、MCP サーバーがロードされない	設定をデバッグする
セッションが auto モードで開始された、または Claude がファイルを編集してコマンドを実行する（確認なし）	セッションが開始するモード
`API Error: 5xx`、`529 Overloaded`、`429`、リクエスト検証エラー	エラーリファレンス
`model not found` または `you may not have access to it`	エラーリファレンス
Claude が実行するコマンドが `Your disk quota is full`、`is full (ENOSPC)`、または `Command output was lost` で失敗する	エラーリファレンス
VS Code 拡張機能が接続されていない、または Claude を検出していない	*VS Code* **統合** (ページ 202)
VS Code または SDK アプリで `Claude Code process exited with code 1`	エラーリファレンス
JetBrains プラグインまたは IDE が検出されない	*JetBrains* **統合** (ページ 209)
CPU またはメモリ使用量が多い、応答が遅い、ハング、検索がファイルを見つけられない	**パフォーマンスと安定性** (ページ 38)（下記）

どれが当てはまるかわからない場合は、Claude Code 内で `/doctor` を実行して、インストール、設定、拡張機能、コンテキスト使用量の自動チェックを実行してください。確認後に適用できる修正を提案します。`claude` がまったく起動しない場合は、代わりにシェルから `claude doctor` を実行してください。MCP サーバーのステータスを確認するには `/mcp` を実行してください。

4.1 パフォーマンスと安定性

これらのセクションでは、リソース使用量、応答性、検索動作に関連する問題について説明します。

4.1.1 CPU またはメモリ使用量が多い

Claude Code はほとんどの開発環境で動作するように設計されていますが、大規模なコードベースを処理する場合、かなりのリソースを消費する可能性があります。パフォーマンスの問題が発生している場合：

1. `/compact` を定期的に使用してコンテキストサイズを削減します。`Not enough messages to compact.` が返される場合、カンバセーションのターン数が少なすぎて要約できません。これは、単一の大規模なペーストがコンテキストを満杯にした場合でも、コンテキストが満杯でも発生する可能性があります
2. 主要なタスク間で Claude Code を閉じて再起動します
3. 大規模なビルドディレクトリを `.gitignore` ファイルに追加することを検討してください
4. *`claude --safe-mode`* (ページ 51) で再起動して、プラグイン、MCP サーバー、またはフックが原因かどうかを確認します。セッション中のすべてのカスタマイズが無効になります。使用量が低下した場合は、設定をデバッグするを参照して、どれが原因かを特定します

セッションのヒープメモリが 2.5GB を超える場合、重大なメモリ使用量警告が表示されます。メモリを解放するには、Claude Code を再起動して *`claude --continue`* (ページ 51) を実行し、新しいプロセスでカンバセーションを再開します。

フルスクリーンレンダリングの外では、`/compact` を実行するとメモリも解放されます。メモリ使用量が 2.5GB を下回ると、警告は消えます。

これらのステップ後もメモリ使用量が高いままの場合は、`/heapdump` を実行して 2 つのファイルを `~/Desktop` に書き込みます。`<session-id>.heapsnapshot` という名前の JavaScript ヒープスナップショットと、`<session-id>-diagnostics.json` という名前のメモリ分析です。Claude Code はコマンドメニューからコマンドを非表示にします。完全に入力してください。Linux でデスクトップフォルダがない場合、ファイルはホームディレクトリに書き込まれます。

警告:

`.heapsnapshot` ファイルには、プロセス内のすべての文字列が含まれています。これには、完全なカンバセーションと認証情報が含まれます。公開の issue に添付したり、共有したりしないでください。

コマンドはカンバセーションに要約も出力します。プロセスの合計メモリ、JS ヒープ内のメモリ量、ヒープ外のメモリ量を表示します。また、高いメモリ増加率や異常に多くのオープンハンドルなど、検出されたリーク指標も表示します。要約は、ほとんどのメモリが JS ヒープ（スナップショットがキャプチャする）にあるか、ネイティブメモリ（キャプチャしない）にあるかを示します。

出力に対して次の 2 つのいずれかを実行します：

- **報告する**：GitHub issue を開き、`-diagnostics.json` ファイルのみを添付します。このファイルには、出力された要約の背後にある統計が含まれており、カンバセーション内容や認証情報は含まれていません

- **自分で調査する**：要約がほとんどのメモリが JS ヒープにあると示している場合、Chrome DevTools の Memory → Load で `.heapsnapshot` ファイルを開き、保持されたサイズでソートして、メモリを保持しているものを確認します

要約がほとんどのメモリがネイティブにあると示している場合、スナップショットはそれを表示できません。代わりに、要約のリーク指標をレポートに含めてください。

4.1.2 ターミナルで大きなテーブルが切り取られる

200 行以上の Markdown テーブルは、最初の 200 行とそれに続く … `N more rows not shown` 行をレンダリングします。表示のみが制限されます。完全なテーブルはカンバセーションに残り、`/copy` はすべての行をコピーします。ターミナルで読むには大きすぎるテーブルの場合は、Claude にファイルに書き込むよう依頼してください。v2.1.208 より前では、Claude Code はすべての行をレンダリングしていたため、非常に大きなテーブルを含むセッションを再開すると、再レンダリング中にスタールする可能性がありました。

4.1.3 自動コンパクションがスラッシングエラーで停止する

`Autocompact is thrashing: the context refilled to the limit...` が表示される場合、自動コンパクションは成功しましたが、ファイルまたはツール出力がコンテキストウィンドウを数回連続で満杯に戻しました。Claude Code は進捗を遂行していないループで API 呼び出しを無駄にするのを避けるために再試行を停止します。

回復するには：

1. Claude に、ファイル全体ではなく、特定の行範囲または関数など、より小さなチャンクで大きなファイルを読むよう依頼します
2. `/compact` を実行して、大きな出力を削除するフォーカスを使用します（例：`/compact keep only the plan and the diff`）
3. 大規模ファイルの作業を *subagent* (ページ 317) に移動して、別のコンテキストウィンドウで実行されるようにします
4. 以前のカンバセーションがもう必要ない場合は `/clear` を実行します

`/clear` の後にエラーが再発する場合は、`/context` を実行し、`Messages` 行とその上の行を比較します：

- `Messages` **が最大の行である場合**：新しい会話内のファイルまたはツール出力がウィンドウを再び満杯にしているため、手順 1〜3 をもう一度実行します
- **他の行の合計のほうが大きい場合**：セッション開始時に読み込まれるものによって作業の余地がほとんど残っていないため、起動時に読み込まれるものを削減します

4.1.4 コマンドがハングまたはフリーズする

Claude Code が応答しないように見える場合：

1. Ctrl+C を押して現在の操作をキャンセルしてみます
2. 応答しない場合は、ターミナルを閉じて再起動する必要があります

再起動してもカンバセーションは失われません。同じディレクトリで `claude --resume` を実行してセッションを再開してください。

4.1.5 エディタの統合ターミナルでのテキストの文字化けまたは破損

VS Code、Cursor、または Devin Desktop の統合ターミナルで Claude Code を実行する場合、文字がボックス、スミア、または間違ったグリフとしてレンダリングされる場合、ターミナルの GPU レンダラーが原因である可能性があります。Claude Code 内で `/terminal-setup` を実行して、`terminal.integrated.gpuAcceleration` を `"off"` に設定するか、エディタの設定で手動で設定してウィンドウをリロードします。**ターミナル設定** (ページ 829) で、`/terminal-setup` が書き込む他の設定を参照してください。

4.1.6 フルスクリーンレンダリングでマウスホイールが一度に 1 行スクロールする

フルスクリーンレンダリングでは、Claude Code はカンバセーション自体をスクロールし、ターミナルに任せません。各ホイールノッチが希望より少ない行を移動する場合、`/scroll-speed` を実行してノッチあたりの行数を増やして保存するか、`CLAUDE_CODE_SCROLL_SPEED` 環境変数を設定します。ただし、JetBrains IDE ターミナルでは、Claude Code は独自のスクロール処理を適用し、どちらも有効になりません。マウスホイールスクロールで、各値が受け入れるものを参照してください。

速度を変更せずにより速く移動するには、`PgUp` と `PgDn` を押して一度に半画面スクロールします。スクロールをターミナルのネイティブスクロールバックに戻すには、`/tui default` を実行してクラシックレンダラーに切り替えます。

4.1.7 サンドボックス内で pbcopy などのクリップボードコマンドが失敗する

サンドボックス (ページ 1069) がオンの場合、`pbcopy`、`xclip`、`wl-copy` などのクリップボードユーティリティは、サンドボックス化された Bash コマンド内からシステムクリップボードに到達できず、Claude がテキストをパイプした後、クリップボードが変更されないままになる可能性があります。

Claude の出力をクリップボードに配置するには、Claude に応答でコンテンツを出力するよう依頼してから、`/copy` を実行します。`/copy` はサンドボックス化されたコマンドではなく Claude Code プロセス自体からクリップボードに書き込むため、サンドボックスはそれをブロックしません。応答全体ではなく単一のコードブロックをコピーでき、コピーしたものをファイルに書き込んで、パスを出力します。これにより、クリップボード書き込みがターミナルに到達しない場合（例えば SSH 経由）のフォールバックが提供されます。

Claude がテキストをこれらのツールにパイプする場合、`pbcopy *`、`wl-copy *`、または `xclip *` を `excludedCommands` に追加しても、そのコマンドをサンドボックスの外で実行することはできません。

4.1.8 SSH 経由でコピーされたテキストがローカルクリップボードに到達しない

Claude Code がリモートマシンで SSH 経由で実行されている場合、ローカルマシンでクリップボードツールを実行できません。tmux の外では、フルスクリーンレンダリングでテキストを選択するか `/copy` を実行すると、Claude Code はテキストを OSC 52 エスケープシーケンスとしてターミナルに送信します。ターミナルがそれをクリップボードに配置するかどうかを決定します。`/copy` は、テキストが到達したかどうかに関わらず `Copied to clipboard` を報告し、tmux の外では選択通知は `sent N chars via OSC 52` と表示されます。

一部のターミナルは OSC 52 に対応していません。iTerm2 は **Settings > General > Selection > Applications in terminal may access clipboard** をオンにするまで無視し、macOS Terminal.app はそれをサポートしていません。

OSC 52 なしでテキストを取得するには：

- ターミナルのネイティブ選択キーを押しながらドラッグしてから、ターミナルの通常のショートカット（Cmd+C など）でコピーします。キーは Terminal.app では Fn、iTerm2 では Option です。ネイティブテキスト選択を保持で他のターミナルのキーを一覧表示します。
- リモートマシンで CLAUDE_CODE_DISABLE_MOUSE=1 を設定して、ターミナルがセッション全体の選択を処理するようにします。

4.1.9 検索と発見の問題

Search ツール、@file メンション、カスタムエージェント、またはカスタムスキルがファイルを見つけられない場合、バンドルされた ripgrep バイナリがシステムで実行されない可能性があります。プラットフォームの ripgrep パッケージをインストールして、Claude Code にそれを使用するよう指示します：

macOS

```
brew install ripgrep
```

Ubuntu/Debian

```
sudo apt install ripgrep
```

Alpine

```
apk add ripgrep
```

ripgrep は Alpine のコミュニティリポジトリにあります。apk がパッケージが見つからないと報告する場合は、*Alpine Linux* **セットアップ** (ページ 22) を参照してください。

Arch

```
pacman -S ripgrep
```

Windows

```
winget install BurntSushi.ripgrep.MSVC
```

その後、USE_BUILTIN_RIPGREP を 0 に設定します。シェル環境または settings.json の env ブロックで設定します：

```
{
  "env": {
    "USE_BUILTIN_RIPGREP": "0"
  }
}
```

切り替えが有効になったことを確認するには、ターミナルで claude doctor を実行して、Search 行がシステム ripgrep のパスを表示していることを確認します。OK (bundled) ではなく。

4.1.10 WSL での遅い、または不完全な検索結果

WSL でファイルシステム間で作業する場合のディスク読み取りパフォーマンスペナルティにより、WSL で Claude Code を使用する場合、Search ツール使用時に予想より少ないマッチが返される可能性があります。検索は機能しますが、ネイティブファイルシステムより少ない結果を返します。

注釈:

`claude doctor` はこの場合、Search を OK として表示します。

解決策：

1. **より具体的な検索を送信する**：検索するファイル数を減らすために、ディレクトリまたはファイルタイプを指定します：「auth-service パッケージで JWT 検証ロジックを検索」または「JS ファイルで md5 ハッシュの使用を見つける」。
2. **プロジェクトを Linux ファイルシステムに移動する**：可能であれば、プロジェクトが Windows ファイルシステム（`/mnt/c/`）ではなく Linux ファイルシステム（`/home/`）に配置されていることを確認します。
3. **ネイティブ Windows を使用する**：WSL ではなく Windows でネイティブに Claude Code を実行することを検討して、ファイルシステムのパフォーマンスを向上させます。

4.2 さらにヘルプを得る

ここで説明されていない問題が発生している場合：

1. `/doctor` を実行してセットアップをチェックし、`/mcp` を実行して MCP サーバーのステータスを確認します
2. Claude Code 内で `/feedback` コマンドを使用して、Anthropic に問題を直接報告します
3. GitHub リポジトリで既知の問題を確認します
4. Claude に直接その機能と機能について質問します。Claude はドキュメントへの組み込みアクセスを持っています。

アカウント、請求、またはサブスクリプションの問題については、代わりに Anthropic サポートにお問い合わせください。claude.ai にサインインし（Console ユーザーの場合：platform.claude.com）、左下のイニシャルをクリックして、**Get help** を選択します。各プランで人間のエージェントに連絡できるユーザーを含む完全なフローについては、How to get support を参照してください。

第II部

基本操作

第5章　CLI リファレンス

Claude Code コマンドラインインターフェースの完全なリファレンス。コマンドとフラグを含みます。

5.1 CLI コマンド

これらのコマンドを使用して、セッションを開始し、コンテンツをパイプし、会話を再開し、更新を管理できます。

コマンド	説明	例
`claude`	インタラクティブセッションを開始	`claude`
`claude "query"`	初期プロンプト付きでインタラクティブセッションを開始	`claude "explain this project"`
`claude -p "query"`	SDK 経由でクエリを実行してから終了	`claude -p "explain this function"`
`cat file | claude -p "query"`	パイプされたコンテンツを処理	`cat logs.txt | claude -p "explain"`
`claude -c`	現在のディレクトリで最新の会話を続行	`claude -c`
`claude -c -p "query"`	SDK 経由で続行	`claude -c -p "Check for type errors"`
`claude -r "<session>" "query"`	セッション ID または名前でセッションを再開	`claude -r "auth-refactor" "Finish this PR"`
`claude update`	最新バージョンに更新	`claude update`
`claude gateway`	自己ホスト型 Claude apps gateway サーバーを開始します。Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry 上の Claude Code の前に SSO とポリシーをデプロイする管理者向けです。`--config` で `gateway.yaml` を指す必要があります。	`claude gateway --config gateway.yaml`

次のページに続く

表 1 – 前のページからの続き

コマンド	説明	例
`claude install [version]`	ネイティブバイナリをインストールまたは再インストールします。`2.1.118` のようなバージョン、または `stable` または `latest` を受け入れます。**特定のバージョンをインストール** (ページ 27) を参照してください	`claude install stable`
`claude auth login`	Anthropic アカウントにサインインします。`--email` を使用してメールアドレスを事前入力し、`--sso` を使用して SSO 認証を強制し、`--console` を使用して Claude サブスクリプションの代わりに Anthropic Console で API 使用料金をサインインできます	`claude auth login --console`
`claude auth logout`	Anthropic アカウントからログアウト	`claude auth logout`
`claude auth status`	認証ステータスを JSON として表示します。`--text` を使用して人間が読める形式で表示できます。ログイン済みの場合はコード 0 で終了し、ログインしていない場合は 1 で終了します。JSON には、CLI が使用する 設定ディレクトリ の名前を付ける `configDirectory` フィールドが含まれます。このフィールドには Claude Code v2.1.268 以降が必要です。JSON の `authMethod` フィールドは、`none`、`claude.ai`、`oauth_token`、`api_key`、`api_key_helper`、または `third_party` のいずれかです	`claude auth status`

次のページに続く

表 1 – 前のページからの続き

コマンド	説明	例
`claude agents`	エージェントビュー を開いて、並列バックグラウンドセッションを監視およびディスパッチします。`--cwd <path>` を使用して、そのディレクトリの下で開始されたセッションのみを表示するか、`--json` を使用してアクティブなセッションを JSON 配列として出力してスクリプト作成用にします（`--json --all` は完了したバックグラウンドセッションも含みます）。`--permission-mode`、`--model`、`--effort`、または `--agent` を渡して、ディスパッチされたセッションのデフォルト を設定します。トップレベルの `claude` コマンドと同様に `--settings`、`--add-dir`、`--plugin-dir`、および `--mcp-config` を受け入れます。エージェントビューを開くにはインタラクティブターミナルが必要です	`claude agents --json`
`claude attach <id>`	このターミナルで バックグラウンドセッション に接続します	`claude attach 7c5dcf5d`
`claude auto-mode defaults`	組み込み auto モード 分類器ルールを JSON として出力します。`claude auto-mode config` を使用して、設定が適用された有効な設定を確認してください。`--label <prefix>` は、ラベルがそのプレフィックスで始まるルールのみを出力します。大文字と小文字を区別しません。Claude Code v2.1.208 以降が必要です	`claude auto-mode defaults --label 'Git Destructive'`
`claude auto-mode reset`	ユーザー設定ファイルから `autoMode` セクションを削除して、デフォルト auto モード 設定を復元します。書き込み前に確認を求めます。`-y`/`--yes` を渡してプロンプトをスキップします。管理設定 または `--settings` フラグからのルールは引き続き適用されます。Claude Code v2.1.212 以降が必要です。デフォルトと有効な設定を検査 を参照してください	`claude auto-mode reset --yes`

次のページに続く

表 1 – 前のページからの続き

コマンド	説明	例
`claude daemon status`	バックグラウンドセッション スーパーバイザー の状態、バージョン、ソケットディレクトリ、および診断用のワーカー数を出力します。スーパーバイザーが実行されていない場合は 1 で終了します	`claude daemon status`
`claude daemon stop --any`	バックグラウンドセッション スーパーバイザー とそれがホストするセッションを停止します。`--keep-workers` を渡して、バックグラウンドセッションを実行したままにして、次のスーパーバイザーが再接続できるようにします。`--any` はオンデマンドスーパーバイザーの停止を確認します。これはデフォルトです。これを使用して、応答しないスーパーバイザー から回復します	`claude daemon stop --any --keep-workers`
`claude doctor`	セッションを開始せずにターミナルから読み取り専用のインストールおよび設定診断を出力します。インストール正常性、設定ファイル検証エラー、および Remote Control 適格性を含みます。セッション内のセットアップチェックアップで修正を適用することもできます。`/doctor` を実行してください	`claude doctor`
`claude import [source]`	他のコーディングエージェントからの設定を Claude Code に取り込むために `/import` を実行するインタラクティブセッションを開始します。コマンドと同じ `--dry-run` および `--yes` オプションを受け入れます。Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry、または AWS 上の Claude Platform では利用できません。機能フラグ取得 をオフにした場合も利用できません。Claude Code v2.1.213 以降が必要です	`claude import codex --dry-run`
`claude logs <id>`	バックグラウンドセッション からの最近の出力を出力します	`claude logs 7c5dcf5d`
`claude mcp`	Model Context Protocol（MCP）サーバーを設定	*Claude Code MCP* **ドキュメント** (ページ 363) を参照してください。

次のページに続く

表 1 – 前のページからの続き

コマンド	説明	例
`claude mcp login <name>`	設定済み MCP サーバーの OAuth フローを実行します。インタラクティブな `/mcp` パネルを開きません。HTTP、SSE、および claude.ai コネクタサーバーで機能します。SSH 経由で `--no-browser` を追加して、ブラウザを開く代わりに認可 URL を出力し、リダイレクト URL をプロンプトに貼り付けます。**コマンドラインから認証** (ページ 388) を参照してください	`claude mcp login sentry`
`claude mcp logout <name>`	MCP サーバーの保存された OAuth 認証情報をクリアします	`claude mcp logout sentry`
`claude plugin`	Claude Code plugins を管理します。エイリアス：`claude plugins`。サブコマンドについては plugin reference を参照してください	`claude plugin install code-review@claude-plugins-official`
`claude purge [path]`	プロジェクトのすべてのローカル Claude Code 状態を削除します：トランスクリプト、タスクリスト、デバッグログ、ファイル編集履歴、プロンプト履歴行、および `~/.claude.json` 内のプロジェクトエントリ。`[path]` を省略して、インタラクティブリストから選択します。フラグ：`--dry-run` でプレビュー、`-y`/`--yes` で確認をスキップ、`-i`/`--interactive` で各項目を確認、`--all` ですべてのプロジェクト。ローカルデータをクリア を参照してください	`claude purge ~/work/repo --dry-run`
`claude remote-control`	Remote Control サーバーを開始して、Claude.ai または Claude アプリから Claude Code を制御します。サーバーモード（ローカルインタラクティブセッションなし）で実行されます。サーバーモードフラグ を参照してください。サーバーを停止した後、サーバーが提供していたセッションを復元できます。サーバー停止後のセッション再開 を参照してください	`claude remote-control --name "My Project"`

次のページに続く

表 1 – 前のページからの続き

コマンド	説明	例
`claude respawn <id>`	会話を保持したまま、バックグラウンドセッション を再開します。実行中または停止中。`--all` を使用してすべての実行中セッションを再開します。たとえば、更新された Claude Code バイナリを取得するため	`claude respawn 7c5dcf5d`
`claude rm <id>`	バックグラウンドセッション をリストから削除します。削除が セッションの worktree を理由に拒否 され、2 回目の `claude rm` で解決できる場合、拒否は渡すべき正確なフラグと値を出力します：`--discard-unpushed <commit>@<worktree-id>` はプッシュされていないコミットを持つ worktree をそれらのコミットとともに破棄し、`--force-remove-worktree <worktree-id>` は git または `WorktreeRemove` フックが削除できなかった worktree ディレクトリを削除します。`--discard-unpushed` には Claude Code v2.1.260 以降が必要です。また、`--force-remove-worktree` には v2.1.268 以降が必要です。会話トランスクリプトはローカルマシンに残り、`claude --resume` を通じて利用可能です	`claude rm 7c5dcf5d`
`claude self-hosted-runner`	このマシンまたはコンテナを 自己ホスト環境 に登録し、Claude Code クラウドセッションをインフラストラクチャ上でホストするランナープロセスを開始します。`claude self-hosted-runner setup` でガイド付きオペレーターウォークスルーを実行し、`claude self-hosted-runner doctor` で デプロイされたランナーを診断 し、`claude self-hosted-runner orchestrator` で オンデマンドランナー をスポーンします。Claude Code v2.1.224 以降が必要です	`claude self-hosted-runner setup`

次のページに続く

表 1 – 前のページからの続き

コマンド	説明	例
`claude setup-token`	CI とスクリプト用の長期間有効な OAuth トークンを生成します。ターミナルにトークンを出力し、保存しません。Claude サブスクリプションが必要です。**長期間有効なトークンを生成** (ページ 860) を参照してください	`claude setup-token`
`claude stop <id>`	バックグラウンドセッション を停止します。`claude kill` も受け入れます	`claude stop 7c5dcf5d`
`claude ultrareview [target]`	ultrareview を非対話的に実行します。結果を stdout に出力し、成功時は 0 で終了し、失敗時は 1 で終了します。`--json` を使用して生のペイロードを取得し、`--timeout <minutes>` を使用して 45 分のデフォルトを上書きします。`github.com` プルリクエストターゲットで `--post` を使用して、完了した結果を GitHub アカウントから PR に 1 つのプレーンコメントとして投稿します。`--no-post` がデフォルトです。`--post` と `--no-post` には Claude Code v2.1.227 以降が必要です。プルリクエストに結果を投稿 を参照してください	`claude ultrareview 1234 --json`

サブコマンドを誤入力した場合、Claude Code は最も近い一致を提案して、セッションを開始せずに終了します。たとえば、`claude udpate` は `Did you mean claude update?` と出力します。

`claude --dangerously-skip-permissions daemon <subcommand>` は `daemon` サブコマンドを実行するため、`claude` がフラグを含むようにエイリアスされている場合でもサブコマンドは機能します。先頭の `--dangerously-skip-permissions` または `--allow-dangerously-skip-permissions` のみがこの方法で `daemon` にルーティングされます。`daemon` の前にそれ以外のフラグがある場合、サブコマンドは実行されません。

5.2 CLI フラグ

これらのコマンドラインフラグを使用して Claude Code の動作をカスタマイズします。`claude --help` はすべてのフラグをリストアップしていないため、フラグが `--help` に表示されていないことは、そのフラグが利用できないことを意味しません。

フラグ	説明	例
`--add-dir`	Claude がファイルを読み書きするための追加の作業ディレクトリを追加します。ファイルアクセスを許可します。Claude Code は**これらのディレクトリから *.claude/* 設定のほとんどを検出しません** (ページ 1061)。各パスがディレクトリとして存在することを検証します。`\\server\share` などのネットワークパスのほとんどを追加することはできません。これらのディレクトリをセッション間で保持するには、設定で `permissions.additionalDirectories` を設定します	`claude --add-dir ../apps ../lib`
`--advisor <model>`	このセッションのサーバー側アドバイザーツールを有効にします。モデルエイリアス `fable`、`opus`、`sonnet`、または完全なモデル ID を使用します。`advisorModel` 設定よりも優先されます。`fable` には Fable アクセスが必要です	`claude --advisor opus`
`--agent`	現在のセッションのエージェントを指定します（`agent` 設定をオーバーライドします）	`claude --agent my-custom-agent`
`--agents`	JSON を使用してカスタムサブエージェントを動的に定義します。***CLI* で定義されたサブエージェントのリストされたフィールド** (ページ 321) を受け入れます。`--print` を使用する場合、値は代わりに JSON オブジェクトを保持するファイルへのパスになります。ファイル形式には Claude Code v2.1.281 以降が必要です。Claude Code は起動時に値を検証し、無効な値で終了します。メッセージについては `Invalid --agents configuration` を参照してください。また、検証をスキップするフラグと環境変数についても参照してください。検証には Claude Code v2.1.242 以降が必要です	`claude --agents '{"reviewer":{"description": "Reviews code", "prompt":"You are a code reviewer"}}'`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--allow-dangerously-skip-permissions`	Shift+Tab モードサイクルに bypassPermissions を追加します。ただし、それで開始しません。plan などの別のモードで開始し、後で bypassPermissions に切り替えることができます。権限モードを参照してください	`claude --permission-mode plan --allow-dangerously-skip-permissions`
`--allowedTools, --allowed-tools`	権限を求めずに実行するツール。パターンマッチングについては権限ルール構文を参照してください。利用可能なツールを制限するには、代わりに `--tools` を使用してください。タスク追跡ツールの 1 つをここで名前を付けた場合、Claude Code もセッションをオプトインします	`"Bash(git log *)" "Bash(git diff *)" "Read"`
`--append-subagent-system-prompt`	すべての**サブエージェント** (ページ 317) のシステムプロンプトの末尾にカスタムテキストを追加します。ネストされたサブエージェントを含みますが、**フォークされたサブエージェント** (ページ 354) は除きます。フォークされたサブエージェントは会話独自のプロンプトを再利用します。-p を使用した非対話モードでのみ適用されます。Claude Code v2.1.205 以降が必要です	`claude -p --append-subagent-system-prompt "Cite file paths in every answer" "query"`
`--append-subagent-system-prompt-file`	ファイルからテキストを読み込み、**サブエージェント** (ページ 317) システムプロンプトに追加します。コマンドラインで渡すには長すぎるテキストの場合、`--append-subagent-system-prompt` の代替です。2 つのフラグを組み合わせることはできません。-p を使用した非対話モードでのみ適用されます。Claude Code v2.1.261 以降が必要です	`claude -p --append-subagent-system-prompt-file ./subagent-rules.txt "query"`
`--append-system-prompt`	デフォルトシステムプロンプトの末尾にカスタムテキストを追加します	`claude --append-system-prompt "Always use TypeScript"`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--append-system-prompt-file`	ファイルから追加のシステムプロンプトテキストを読み込み、デフォルトプロンプトに追加します	`claude --append-system-prompt-file ./extra-rules.txt`
`--autocompact <auto|tokens>`	保存された設定を変更せずに、このセッションの**自動コンパクトウィンドウ** (ページ 754) を設定します。`/autocompact` と同じ値を受け入れます。そのセクションでは値の形式とフラグをオーバーライドするものについて説明しています。Claude Code v2.1.221 以降が必要です	`claude --autocompact 500k`
`--ax-screen-reader`	スクリーンリーダーフレンドリーな出力をレンダリングします。装飾的なボーダーやアニメーションのないフラットテキスト。クラシックレンダラーを強制するため、`tui` 設定は効果がありません。接続されたバックグラウンドセッションは引き続きフルスクリーンでレンダリングされます。`CLAUDE_AX_SCREEN_READER` と `axScreenReader` 設定よりも優先されます。Claude Code v2.1.181 以降が必要です	`claude --ax-screen-reader`
`--bare`	最小限モード。フック、スキル、カスタムコマンド、サブエージェント、インストール済みプラグイン、MCP サーバー、自動メモリ、CLAUDE.md の自動検出をスキップして、スクリプト化された呼び出しをより高速に開始します。`--add-dir` で渡すディレクトリ内のスキルは引き続き読み込まれます。Claude は Bash、ファイル読み取り、ファイル編集ツールにアクセスできます。`CLAUDE_CODE_SIMPLE` を設定します。**ベアモード** (ページ 696) を参照してください	`claude --bare -p "query"`
`--betas`	API リクエストに含める Beta ヘッダー（API キーユーザーのみ）	`claude --betas interleaved-thinking`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
--bg, --background	セッションをバックグラウンドエージェントとして開始し、すぐに戻ります。セッション ID と管理コマンドを出力します。--exec と組み合わせて、Claude セッションの代わりにシェルコマンドをバックグラウンドジョブとして実行するか、--agent と組み合わせて特定のサブエージェントを実行します。ディレクトリの**ワークスペーストラスト** (ページ 1064) を確認してから開始します。-p/--print と組み合わせることはできません。エラーリファレンスを参照してください	claude --bg "investigate the flaky test"
--channels	（研究プレビュー）Claude がこのセッションでリッスンすべきチャネル通知を持つ MCP サーバー。plugin:<name>@<marketplace> エントリのスペース区切りリスト。claude.ai または Console API キーを通じた Anthropic 認証が必要です	claude --channels plugin:my-notifier@my-marketplace
--chrome	*Chrome* **ブラウザ統合** (ページ 213) を有効にして、Web 自動化とテストを行います	claude --chrome
--cloud	タスク説明を使用して、新しい**クラウドセッション** (ページ 223) を作成します。セッション ID（session_... または cse_...）または claude.ai/code URL を使用して、-p でそのセッションに代わりにメッセージをキューに入れます。**フォローアップメッセージを送信** (ページ 228) を参照してください。	claude --cloud "Fix the login bug"

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
--continue, -c	現在のディレクトリで最新の会話を読み込みます。終了したバックグラウンドセッションを含みます。終了したバックグラウンドセッションを開くには Claude Code v2.1.257 以降が必要です。claude -p またはエージェント SDK で作成されたセッション、および最初のプロンプトが /loop だったセッションをスキップします。claude -p --continue には -p、SDK、/loop セッションが含まれます。このディレクトリを /add-dir で追加したセッションを含みます	claude --continue
--dangerously-load-development-channels	承認されたアローリストにないチャネルを有効にして、ローカル開発を行います。plugin:<name>@<marketplace> および server:<name> エントリを受け入れます。確認を求めます	claude --dangerously-load-development-channels server:webhook
--dangerously-skip-permissions	権限プロンプトをスキップします。--permission-mode bypassPermissions と同等です。権限モードでこれが何をスキップし、何をスキップしないかを参照してください。--bg で開始されたセッションの場合、モードはスーパーバイザーがセッションを再開するときに保持されます	claude --dangerously-skip-permissions
--debug	デバッグモードを有効にします。オプションのカテゴリフィルタリング (--debug='mcp,startup' または --debug='!1p' など) を使用します。フィルターは = 形式でのみバインドされます。スペース区切りフィルターはフィルタリングなしでデバッグモードを有効にします	claude --debug='mcp,startup'
--debug-file <path>	デバッグログを特定のファイルパスに書き込みます。暗黙的にデバッグモードを有効にします。CLAUDE_CODE_DEBUG_LOGS_DIR よりも優先されます	claude --debug-file /tmp/claude-debug.log

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--desktop`	*Claude Desktop* **アプリ** (ページ 137) を現在のディレクトリで開き、ターミナルでセッションを開始せずに終了します。`--continue` を追加するか、セッション ID を使用して `--resume` を追加して、**代わりに** *Desktop* **でそのセッションを開く** (ページ 166) ことができます。ここで `--resume` はセッション ID のみを取り、名前またはトランスクリプトパスは取りません。プロンプトと `--verbose` および `--debug` フラグ以外の他のフラグは取りません。アプリがセッション自体を開始するため。macOS および x64 Windows で Claude サブスクリプションでサインインしている場合に利用可能です。Claude Code v2.1.285 以降が必要です	`claude --desktop`
`--disable-slash-commands`	このセッションのすべてのスキルとコマンドを無効にします	`claude --disable-slash-commands`
`--disallowedTools`, `--disallowed-tools`	拒否ルール。ベアツール名は一致するツールを Claude のコンテキストから削除します。`"Edit"` は Edit を削除し、`"*"` はすべてのツールを削除し、`"mcp__*"` はすべての MCP ツールを削除します。`Bash(rm *)` などのスコープ付きルールはツールを利用可能なままにし、**書かれたとおりに** (ページ 1048) 一致する呼び出しのみを拒否します。`End Conversation` という名前のルールは、他のツールが残っている間はそれを削除できません	`"Bash(git log *)" "Bash(git diff *)" "Edit"`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
--effort	現在のセッションの**努力レベル** (ページ 747) を設定します。オプション：low、medium、high、xhigh、max、または ultracode。利用可能なレベルはモデルによって異なります。ultracode は ultracode をオンにして xhigh 努力をリクエストし、Claude Code v2.1.203 以降が必要です。このセッションの model Settings と effortLevel 設定をオーバーライドし、保持されません	claude --effort high
--enable-auto-mode	v2.1.111 で削除されました。オートモードはデフォルトで Shift+Tab サイクルに含まれるようになりました。--permission-mode auto を使用して開始します	claude --permission-mode auto
--environment <environment-id>	指定された ID の自己ホスト環境で実行される新しいクラウドセッションを作成します。環境 ID は ccpool_ で始まります。ディスパッチ動作とそれが拒否するフラグの組み合わせについては、--environment ディスパッチ動作を参照してください。Claude Code v2.1.224 以降が必要です	claude -p "Fix the login bug" --environment ccpool_abc123
--exclude-dynamic-system-prompt-sections	ユーザーごとのコンテキスト（自動メモリの場所など) をシステムプロンプトから最初のユーザーメッセージに移動します。異なるユーザーとマシンで同じタスクを実行する場合、プロンプトキャッシュの再利用を改善します。デフォルトシステムプロンプトでのみ適用されます。--system-prompt または --system-prompt-file が設定されている場合は無視されます。スクリプト化された、マルチユーザーワークロードで -p を使用します	claude -p --exclude-dynamic-system-prompt-sections "query"
--exec	Claude セッションを開始する代わりに、PTY バックアップシェルコマンドをバックグラウンドジョブとして実行します。--bg と組み合わせてシェルから起動します	claude --bg --exec 'pytest -x'

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--fallback-model`	プライマリモデルがオーバーロードされているか利用できない場合（例：廃止されたモデル）に、指定されたモデルへの自動フォールバックを有効にします。順序で試行されるコンマ区切りリストを受け入れます。**フォールバックモデルチェーン** (ページ 743) を参照してください。セッション間でチェーンを保持するには、このフラグがオーバーライドする fallbackModel 設定を使用します	`claude --fallback-model sonnet,haiku`
`--fork-session`	再開するときに、元のセッション ID を再利用する代わりに新しいセッション ID を作成します（`--resume` または `--continue` で使用）	`claude --resume abc123 --fork-session`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--forward-subagent-text`	**サブエージェント** (ページ 317) テキストと思考ブロックを出力ストリームに `parent_tool_use_id` が設定された `assistant` および `user` メッセージとして出力して、各サブエージェントのトランスクリプトを再構築できるようにします。このフラグがない場合、Claude Code は**フォアグラウンド** (ページ 344) で実行されるサブエージェントのテキストと思考ブロックを省略します。`--print` と `--output-format stream-json` が必要です。Claude Code は**ネストされたサブエージェント** (ページ 349) からのメッセージも転送し、各サブエージェントを生成したエージェントツール呼び出しの ID に `parent_tool_use_id` を設定します。これには Claude Code v2.1.219 以降が必要です。フォークされたスキルが生成するサブエージェントのメッセージ、およびネストされたフォークされたスキルのメッセージには v2.1.275 以降が必要です。`CLAUDE_CODE_FORWARD_SUBAGENT_TEXT` 環境変数は同じ動作を有効にします。Claude Code v2.1.211 以降が必要です	`claude -p --output-format stream-json --verbose --forward-subagent-text "query"`
`--from-pr`	セッションピッカーを特定のプルリクエストにリンクされたセッションにフィルタリングして開きます。PR 番号、GitHub または GitHub Enterprise PR URL、GitLab マージリクエスト URL、または Bitbucket プルリクエスト URL を受け入れます。Claude がプルリクエストを作成すると、セッションは自動的にリンクされます	`claude --from-pr 123`
`--ide`	起動時に、正確に 1 つの有効な IDE が利用可能な場合、自動的に IDE に接続します	`claude --ide`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--init`	セッションの前に init マッチャーで**セットアップフック** (ページ 485) を実行します（プリントモードのみ）	`claude -p --init "query"`
`--init-only`	**セットアップ** (ページ 485) および SessionStart フックを実行してから、会話を開始せずに終了します	`claude --init-only`
`--include-hook-events`	フックライフサイクルイベントを出力ストリームに含めます。SessionStart および Setup フックイベントは常に含まれ、このフラグは必要ありません。Notification、SessionEnd、PreCompact、PostCompact などの一部のフックイベントは、このフラグを使用しても hook_started イベントを生成しません。これらのイベントについては、Claude Code は 1 秒以上実行されるコマンドフックが出力を生成している間は hook_progress を出力し、**バックグラウンドで実行されるフック** (ページ 558) が終了したときのみ hook_response を出力します。--output-format stream-json が必要です	`claude -p --output-format stream-json --verbose --include-hook-events "query"`
`--include-partial-messages`	出力に部分的なストリーミングイベントを含めます。--print と --output-format stream-json が必要です	`claude -p --output-format stream-json --verbose --include-partial-messages "query"`
`--input-format`	プリントモードの入力形式を指定します(オプション:text、stream-json)	`claude -p --output-format json --input-format stream-json`
`--json-schema`	エージェントがワークフローを完了した後、JSON スキーマに一致する検証済み JSON 出力を取得します（プリントモードのみ）。構造化出力を参照してください。Claude Code は無効なスキーマでエラーで終了し、format キーワードをクライアント側検証なしの注釈として受け入れます	`claude -p --json-schema '{"type":"object", "properties":{...}}' "query"`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--maintenance`	セッションの前に `maintenance` マッチャーで**セットアップフック** (ページ 485) を実行します (プリントモードのみ)	`claude -p --maintenance "query"`
`--max-budget-usd`	API 呼び出しで支出する最大ドル額。停止する前に支出します (プリントモードのみ)。**サブエージェント** (ページ 317) からの支出はキャップにカウントされます。`--continue` または `--resume` で会話に戻るときに、以前の実行から復元された合計はそれに対してカウントされません。支出がキャップに達すると、別のサブエージェントを生成すると `Budget limit reached` で失敗し、Claude Code は引き続き実行中のバックグラウンドサブエージェントを停止します。キャップ実行動作には Claude Code v2.1.217 以降が必要です	`claude -p --max-budget-usd 5.00 "query"`
`--max-turns`	エージェンティックターンの数を制限します (プリントモードのみ)。制限に達するとエラーで終了します。デフォルトでは制限がありません。`--input-format stream-json` を使用する場合、制限がターンを終了するときにキューに入れられたメッセージは引き続きキューに入れられ、独自の制限で新しいターンを開始します	`claude -p --max-turns 3 "query"`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
--mcp-config	JSON ファイルまたは文字列から MCP サーバーを読み込みます（スペース区切り）。このフラグを -p で渡すと、Claude Code は最初のターンを実行する前に、まだ保留中のサーバーが接続されるまで待機します。デフォルトでは MCP_TIMEOUT スタートアップタイムアウト 30 秒まで待機します。**キャッシュされたツールリスト** (ページ 369) を持つサーバーは待機をスキップし、最初の使用時に接続します。待機には Claude Code v2.1.221 以降が必要です	claude --mcp-config ./mcp.json
--model	sonnet、opus、haiku、fable などの**モデルエイリアス** (ページ 725) またはモデルの完全な名前を使用して、現在のセッションのモデルを設定します。model 設定と *ANTHROPIC_MODEL* (ページ 757) をオーバーライドします	claude --model claude-sonnet-5
--name, -n	セッションの表示名を設定します。/resume とターミナルタイトルに表示されます。claude --resume <name> で名前付きセッションを再開できます。対話型セッションで、このマシン上の別のライブセッションが既に名前を使用している場合、Claude Code はその変種を適用します。 /rename はセッション中に名前を変更し、プロンプトバーにも表示します	claude -n "my-feature-work"
--no-chrome	このセッションの *Chrome* **ブラウザ統合** (ページ 213) を無効にします	claude --no-chrome
--no-session-persistence	セッション永続性を無効にして、セッションがディスクに保存されず、再開できないようにします。プリントモードのみ。CLAUDE_CODE_SKIP_PROMPT_HISTORY 環境変数は任意のモードで同じことを行います	claude -p --no-session-persistence "query"

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--output-format`	プリントモードの出力形式を指定します（オプション：`text`、`json`、`stream-json`）	`claude -p "query" --output-format json`
`--permission-mode`	指定された権限モードで開始します。`default`、`acceptEdits`、`plan`、`auto`、`dontAsk`、`bypassPermissions`、または `manual` を `default` のエイリアスとして受け入れます。`manual` エイリアスは UI が Manual とラベル付けする権限モードを選択し、Claude Code v2.1.200 以降が必要です。`claude --help` は `default` の代わりにそれをリストアップし、両方の値が機能します。設定ファイルから `defaultMode` をオーバーライドします。このフラグまたは `--dangerously-skip-permissions` がない場合、新しいセッションはセッションが開始される権限モードで説明されている権限モードで開始されます。`-p` の場合、何も設定されていないときは `default` です	`claude --permission-mode plan`
`--permission-prompt-tool`	非対話モードで権限プロンプトを処理する MCP ツールを指定します。Claude Code は、最初のターンを実行する前に、そのツールの MCP サーバーが接続されるまで待機します。デフォルトでは `MCP_TIMEOUT` スタートアップタイムアウト 30 秒まで待機します。プロンプトツールは、**ユーザーインタラクションが必要** (ページ 404) とマークされた MCP ツールを承認できません。Claude Code は 1 つに対する `allow` 結果を拒否に変換します。この制限には Claude Code v2.1.199 以降が必要です	`claude -p --permission-prompt-tool mcp_auth_tool "query"`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--permission-prompts`	プリントモードで権限プロンプトに誰が答えるかを設定します。デフォルトの `host` を使用すると、Claude Code はそれらをエージェント SDK ホストまたは `--permission-prompt-tool` ツールに送信します。誰も答えられない場合は none を渡し、Claude Code は代わりにそれらを拒否します。**無人実行で権限プロンプトをオフにする** (ページ 705) を参照してください。Claude Code v2.1.259 以降が必要です	`claude -p --permission-prompts none "query"`
`--plugin-dir`	ディレクトリまたは `.zip` アーカイブからプラグインを読み込むか、プラグインのフォルダから複数を読み込みます。このセッションのみ。各フラグは 1 つのパスを取ります。より多くのパスについてはフラグを繰り返します。`--plugin-dir A --plugin-dir B.zip`。プラグインのフォルダを渡すには Claude Code v2.1.265 以降が必要です	`claude --plugin-dir ./my-plugin`
`--plugin-url`	URL からプラグイン `.zip` アーカイブをフェッチします。このセッションのみ。複数のプラグインについてはフラグを繰り返すか、単一の引用値でスペース区切り URL を渡します	`claude --plugin-url https://example.com/plugin.zip`
`--print, -p`	対話モードなしで応答を出力します (プログラム的な使用の詳細については Agent SDK ドキュメントを参照してください)。まだ実行中のバックグラウンドセッションに対する `--resume` については、実行中のバックグラウンドセッションを再開するを参照してください	`claude -p "query"`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
--prompt-suggestions	各ターンの後に予測される次のユーザープロンプトを含む prompt_suggestion メッセージを出力します。非常に短い会話は何も生成しない可能性があります。--print、--output-format stream-json、--verbose が必要です。**プロンプト提案** (ページ 92) を参照してください	claude -p --prompt-suggestions --output-format stream-json --verbose "query"
--ref <branch>	--environment を使用して、新しいセッションのチェックアウトをローカル HEAD の代わりに名前付きリファレンスに基づかせます	claude -p "Run the smoke test" --environment ccpool_abc123 --ref main
--remote	--cloud の非推奨エイリアス。既存セッション形式を含みます	claude --remote "Fix the login bug"
--remote-control, --rc	リモートコントロールを有効にして対話型セッションを開始して、claude.ai または Claude アプリからも制御できるようにします。オプションでセッションの名前を渡します	claude --remote-control "My Project"
--remote-control-session-name-prefix <prefix>	明示的な名前が設定されていない場合、自動生成されたリモートコントロールセッション名のプレフィックス。デフォルトはマシンのホスト名で、myhost-graceful-unicorn などの名前を生成します。CLAUDE_REMOTE_CONTROL_SESSION_NAME_PREFIX を設定して同じ効果を得ます	claude remote-control --remote-control-session-name-prefix dev-box
--replay-user-messages	stdin からのユーザーメッセージを stdout に再出力して確認します。--input-format stream-json と --output-format stream-json が必要です	claude -p --input-format stream-json --output-format stream-json --verbose --replay-user-messages

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--restricted`	制限モードで開始します。評価ハーネスが共有マシン上で `claude` を駆動し、Claude Code がそのマシンのコマンドを実行したり、ユーザーおよびプロジェクト設定を読み取ったりしてはいけない場合に使用します。Claude Code は、`--tools` で個別に名前を付けない限り、`default` プリセットを通じて、コマンドまたはコードを実行するビルトインツールと WebFetch を削除します。また、ビルトインファイルツールを**作業ディレクトリ** (ページ 1060) に限定し、管理設定と `--settings` のみを読み込み、`bypassPermissions` を拒否し、クラウドセッションの作成を拒否します。Claude Code v2.1.248 以降が必要です	`claude --restricted -p "query"`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--resume`, `-r`	ID または名前で特定のセッションを再開するか、セッションを選択するための対話型ピッカーを表示します。ID の代わりに、セッションの `.jsonl` トランスクリプトファイルへの絶対パスを渡すことができます。ピッカーと名前検索には、このディレクトリを `/add-dir` で追加したセッションが含まれます。セッション ID を渡すと、Claude Code は現在のプロジェクトディレクトリとその git ワークツリーを検索してから、このマシン上の他のすべてのプロジェクトを検索します。v2.1.223 より前は、ID 検索は現在のプロジェクトディレクトリとその git ワークツリーのみをカバーしていました。バックグラウンドセッションはピッカーに `bg` でマークされて表示されます。まだ実行中のセッションを再開すると、そのセッションを `claude attach` を通じてこのターミナルで開き、コマンドラインで渡すプロンプトはそれに次のターンとして送信されます。v2.1.285 より前は、Claude Code は拒否し、代わりに実行する `claude attach` コマンドを出力しました	`claude --resume auth-refactor`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--safe-mode`	壊れた設定をトラブルシューティングするためにすべてのカスタマイズを無効にして開始します。CLAUDE.md、スキル、プラグイン、フック、MCP サーバー、カスタムコマンドとエージェント、出力スタイル、ワークフロー、カスタムテーマ、カスタムキーバインディング、ステータスラインとファイル提案コマンド、LSP サーバー、自動メモリは読み込まれません。認証、モデル選択、ビルトインツール、権限は通常どおり機能します。これは *--bare* (ページ 696) と異なります。管理設定ポリシーは引き続き適用されます。ポリシー設定フック、ステータスライン、ファイル提案コマンドを含みます。管理プラグイン、管理スキル、管理 CLAUDE.md、ポリシー設定 MCP サーバーは含まれません。**自動モデルフォールバック** (ページ 744) をトリガーするカスタマイズかどうかを確認するのに役立ちます。`CLAUDE_CODE_SAFE_MODE` を設定します	`claude --safe-mode`
`--session-id`	会話に特定のセッション ID を使用します（有効な UUID である必要があります）	`claude --session-id "550e8400-e29b-41d4-a716-446655440000"`
`--setting-sources`	読み込む設定ソースのコンマ区切りリスト（`user`、`project`、`local`）。エージェントビューとエージェントチームを参照してください。このセッションから開始するセッションがリストを継承します	`claude --setting-sources user,project`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--settings`	設定 JSON ファイルまたはインライン JSON 文字列へのパス。ここで設定した値は、このセッションの `settings.json` ファイルの同じキーをオーバーライドします。省略したキーはファイルベースの値を保持します。ファイルは 2 MiB 以下の通常ファイルである必要があります。**設定の優先順位** (ページ 719) を参照してください	`claude --settings ./settings.json`
`--strict-mcp-config`	`--mcp-config` からのみ MCP サーバーを使用し、他のすべての MCP 設定を無視します。フラグが管理 MCP ファイルの下で何を行うかについては、管理 mcp.json での排他的制御を参照してください	`claude --strict-mcp-config --mcp-config ./mcp.json`
`--system-prompt`	デフォルトシステムプロンプト全体をカスタムテキストに置き換えます	`claude --system-prompt "You are a Python expert"`
`--system-prompt-file`	ファイルからシステムプロンプトを読み込み、デフォルトプロンプトを置き換えます	`claude --system-prompt-file ./custom-prompt.txt`
`--system-prompt-snapshot`	`off` を渡して、会話の最初のリクエストで**記録されたプロンプト** (ページ 73) を再利用する代わりに、すべてのリクエストでシステムプロンプトを再構築します。例えば、`--continue` 実行全体で `--append-system-prompt` テキストを反復処理する場合。Claude Code v2.1.257 以降が必要です	`claude --system-prompt-snapshot off`
`--teleport`	**クラウドセッション** (ページ 223) をローカルターミナルで再開します	`claude --teleport`
`--teammate-mode`	エージェントチームチームメイトの表示方法を設定します。`in-process`（デフォルト）、`auto`、`tmux`、または `iterm2`。このセッションの `teammateMode` 設定をオーバーライドします。表示モードを選択を参照してください	`claude --teammate-mode auto`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--tmux`	ワークツリーの tmux セッションを作成します。`--worktree` が必要です。利用可能な場合は iTerm2 ネイティブペインを使用します。従来の tmux の場合は `--tmux=classic` を渡します	`claude -w feature-auth --tmux`
`--tools`	Claude が使用できるビルトインツールを制限します。`""` を使用してすべてを無効にするか、`"default"` をデフォルトセットに使用するか、`"Bash,Edit,Read"` などのツール名を使用します。macOS、Linux、WSL では、デフォルトセットは Glob ツール動作で説明されているように `Glob` と `Grep` を除外します。タスク追跡ツールの 1 つをここで名前を付けた場合、Claude Code もセッションをオプトインします。フラグは MCP ツールに影響しません。それらも拒否するには、`--disallowedTools "mcp__*"` を使用します。`EndConversation` を省略するリストはそれを削除しません。`""` は MCP ツールが残っていない場合にのみそれを削除します	`claude --tools "Bash,Edit,Read"`
`--verbose`	詳細ログを有効にします。ターンバイターン出力全体を表示します。このセッションの `viewMode` 設定をオーバーライドします	`claude --verbose`
`--version, -v`	バージョン番号を出力します	`claude -v`

次のページに続く

表 2 – 前のページからの続き

フラグ	説明	例
`--worktree`, `-w`	Claude を `<repo>/.claude/worktrees/<name>` の分離された git ワークツリーで開始します。名前を指定しない場合、Claude Code は 1 つを生成します。`#<number>`、GitHub プルリクエスト URL、または GitLab マージリクエスト URL を渡して、origin からそのプルリクエストまたはマージリクエストをフェッチし、ワークツリーをそれから分岐させます。GitLab マージリクエストから分岐するには Claude Code v2.1.233 以降が必要です	`claude -w feature-auth`

5.2.1 システムプロンプトフラグ

Claude Code はシステムプロンプトをカスタマイズするための 5 つのフラグを提供します。4 つはそのテキストを設定し、`--system-prompt-snapshot` を使用して、会話がそれで開始したテキストを保持するかどうかを制御します。5 つすべて対話モードと非対話モードの両方で機能します。

フラグ	動作	例
`--system-prompt`	デフォルトプロンプト全体を置き換えます	`claude --system-prompt "You are a Python expert"`
`--system-prompt-file`	ファイルの内容で置き換えます	`claude --system-prompt-file ./prompts/review.txt`
`--append-system-prompt`	デフォルトプロンプトに追加します	`claude --append-system-prompt "Always use TypeScript"`
`--append-system-prompt-file`	ファイルの内容をデフォルトプロンプトに追加します	`claude --append-system-prompt-file ./style-rules.txt`
`--system-prompt-snapshot`	`off` を使用して、すべてのリクエストでプロンプトを再構築します。`on`（デフォルト）を使用して、**記録が適用される** (ページ 73) 場所で記録されたプロンプトを再利用します	`claude --append-system-prompt "Draft rules" --system-prompt-snapshot off`

これらのフラグは組み合わせることができます。デフォルトプロンプトを置き換えつつ独自のテキストも追加するには、`--append-system-prompt` または `--append-system-prompt-file` を `--system-prompt` または `--system-prompt-file` と一緒に渡します。Claude Code v2.1.283 以降では、`--append-system-`

prompt と --append-system-prompt-file のように、フラグをそれ自身のファイル形式と一緒に渡すこともでき、その場合 Claude Code は両方を使用します。

たとえば、シェルで次のコマンドを実行すると、ファイルのスタイルガイドと追加の指示 1 つの両方を追加できます。

```
claude -p --append-system-prompt-file ./style.md --append-system-prompt "Always reply␣
↪in French" "Summarize README.md"
```

Claude は、デフォルトのシステムプロンプトに続いて、style.md の内容、空行、Always reply in French の順で受け取ります。--append-system-prompt を --append-system-prompt-file より前に渡した場合でも、ファイルの内容が先に来ます。

置換テキストが、すべての実行で同じ指示と実行ごとに変わるコンテキストを組み合わせる場合、__SYSTEM_PROMPT_DYNAMIC_BOUNDARY__ のみを含む行を指示とコンテキストの間に追加します。Claude Code はその最初の行でプロンプトを分割し、その行を削除するため、その上の部分はキャッシュされたままで、その下の部分は変わります。Claude Code v2.1.275 以降が必要です。カスタムプロンプトの静的部分をキャッシュは、分割が適用される設定をリストアップしています。

Claude Code のデフォルト ID がタスクに適合するかどうかに基づいて選択します。Claude がコーディングアシスタントのままで、追加のルールも従うべき場合は、追加フラグを使用します。呼び出しごとの指示、出力形式、または -p スクリプトのドメインコンテキスト。追加するとデフォルトのツールガイダンス、安全指示、コーディング規約が保持されるため、異なる部分のみを提供します。Claude Code のサーフェス、ID、または権限モデルが異なる場合は、置換フラグを使用します。人間が見ていないパイプラインの非コーディングエージェントなど。置換するとデフォルトプロンプト全体が削除されます。ツールガイダンスと安全指示を含むため、タスクがまだ必要とするものについて責任を負います。

プロジェクト間で切り替えて共有できる永続的なペルソナについては、**出力スタイル** (ページ 821) を使用します。Claude が常に従うべきプロジェクト規約については、*CLAUDE.md* (ページ 997) を使用します。エージェント SDK ガイド（システムプロンプト）は同じ決定をより詳しく説明しています。

再開された会話でのシステムプロンプトフラグ

デフォルトでは、Claude Code はシステムプロンプトを 1 回構築します。会話の最初のリクエストで、適用されたシステムプロンプトフラグからのテキストを使用して、セッションに記録します。会話がコンパクトされるまで、その後のすべてのリクエストは、--resume または --continue で会話に戻った後を含めて、その記録されたプロンプトを使用します。その後の起動で異なるシステムプロンプトフラグテキストを渡すか、何も渡さない場合、会話がコンパクトされるか、新しい会話を開始するときに有効になります。

クラウドセッションの外で、--bare を渡すか CLAUDE_CODE_SIMPLE=1 を設定して Claude Code を**ベアモード** (ページ 696) で開始する場合、--system-prompt-snapshot on を渡さない限り、記録はオフのままです。v2.1.268 より前は、フィーチャーフラグをフェッチしないセッション（Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry のセッションを含む）は、すべてのリクエストでプロンプトを再構築し、--system-prompt-snapshot は効果がありませんでした。

代わりに、例えば --continue 実行全体でその文言を反復処理する場合など、すべてのリクエストでプロンプトを再構築するには、--system-prompt-snapshot off を渡します。v2.1.265 より前は、システムプロンプトフラグのいずれかを渡すと、--system-prompt-snapshot on を渡さない限り、記録もオフになりました。

5.3 関連項目

- *Chrome* **拡張機能** (ページ 213) - ブラウザ自動化と Web テスト
- **インタラクティブモード** (ページ 75) - ショートカット、入力モード、インタラクティブ機能
- **クイックスタートガイド** (ページ 11) - Claude Code の開始方法
- **一般的なワークフロー** (ページ 105) - 高度なワークフローとパターン
- **設定** (ページ 711) - 設定オプション
- Agent SDK ドキュメント - プログラムによる使用と統合

第6章　インタラクティブモード

Claude Code セッションのキーボードショートカット、入力モード、インタラクティブ機能の完全なリファレンス。

6.1 キーボードショートカット

注釈:

キーボードショートカットはプラットフォームとターミナルによって異なる場合があります。フルスクリーンレンダリングでは、トランスクリプトビューアで？を押すと、そこで利用可能なショートカットが表示されます。
macOS ユーザー: Option/Alt キーショートカット（`Alt+B`、`Alt+F`、`Alt+D`、`Alt+Y`、`Alt+P`）を使用するには、ターミナルで Option を Meta として設定する必要があります。各ターミナルの設定については、***macOS で Option キーショートカットを有効にする*** (ページ 830) を参照してください。

6.1.1 一般的なコントロール

ショートカット	説明	コンテキスト
`Ctrl+C`	割り込み、または入力をクリア	実行中の操作を割り込みます。何も実行されていない場合、最初のプレスはプロンプト入力をクリアし、2 番目のプレスで Claude Code を終了します
`Ctrl+X Ctrl+K`	このセッション内のすべての実行中の**バックグラウンドサブエージェント** (ページ 344) を停止し、アーティファクト自動返信をセッションの残りの部分で無効にします。3 秒以内に 2 回押して確認	サブエージェント制御
`Ctrl+D`	Claude Code セッションを終了	最初のプレスで確認ヒントが表示され、800ms 以内に 2 番目のプレスで終了します。プロンプトにテキストがある場合、`Ctrl+D` はカーソルの後の文字を削除します

次のページに続く

表 1 – 前のページからの続き

ショートカット	説明	コンテキスト
Ctrl+G または Ctrl+X Ctrl+E	デフォルトテキストエディタで開く	プロンプトまたはカスタム応答をデフォルトテキストエディタで編集します。Ctrl+X Ctrl+E は readline ネイティブバインディングです。/config で**外部エディタで最後の応答を表示**をオンにすると、Claude の前の返信を # コメント付きコンテキストとしてプロンプトの上に追加します。Claude Code は保存時にコメントブロックを削除します
Ctrl+L	画面を再描画	完全なターミナル再描画を強制し、入力と会話履歴を保持します。ディスプレイが乱れたり部分的に空白になった場合に使用して復旧します。会話をクリアを参照してください (フルスクリーンレンダリング)
Ctrl+O	トランスクリプトビューアを切り替え	詳細なツール使用法と実行を表示し、各アシスタントメッセージにタイムスタンプと使用されたモデルを表示します。また、デフォルトで折りたたまれる行（MCP 呼び出しなど、Called slack 3 times という 1 行として表示される行）や他のセッションからのメッセージ（Message from @<sender> プレビューとして 1 行で表示される）を展開します
Ctrl+R	コマンド履歴を逆検索	前のコマンドを対話的に検索
Ctrl+V または Cmd+V（iTerm2）または Alt+V（Windows および WSL）	クリップボードから画像を貼り付け	カーソルに [Image #N] チップを挿入して、プロンプト内で位置的に参照できます。WSL では、Ctrl+V と Alt+V の両方がバインドされています。ターミナルが Ctrl+V をインターセプトする場合は Alt+V を使用してください
Ctrl+B	バックグラウンドで実行中のタスク	Bash コマンドとエージェントをバックグラウンドで実行します。Tmux ユーザーは 2 回押してください

次のページに続く

表 1 – 前のページからの続き

ショートカット	説明	コンテキスト
Ctrl+T	Claude のタスクチェックリストを切り替え	ステータス領域に *Claude* **のやることリスト** (ページ 100) を表示または非表示にします。これはバックグラウンドタスクビューではありません。実行中のシェルとサブエージェントを確認するには /tasks を使用してください
Ctrl+S	プロンプトを保存または復元	入力にテキストがある場合、それを保存してプロンプトをクリアします。空のプロンプトで再度押すと、保存されたテキスト、カーソル位置、貼り付けられたコンテンツ、および入力モードが復元されるため、保存された！**シェルコマンド** (ページ 89) はシェルモードで戻ります
Ctrl+Z	Claude Code を一時停止	Unix のみ。プロセスをシェルに一時停止します。fg を実行して再開します
左/右矢印	ダイアログタブを循環	権限ダイアログとメニューのタブ間を移動します。タブ付きダイアログでは、タブ行がフォーカスされている間、キーはタブを切り替えます。**タブアクション** (ページ 771) を参照して、フォーカスがどのように移動するかを確認してください
Tab	オートコンプリート候補を受け入れるか、権限回答にコメントを追加	プロンプト入力にオートコンプリート候補が表示されている場合、選択された候補を受け入れます。ほとんどの権限プロンプトで、**はい**または**いいえ**がフォーカスされている場合、そのオプションのコメントフィールドを開き、再度押すとフィールドを閉じます。**権限プロンプトに回答するときにコメントを追加** (ページ 1040) を参照してください

次のページに続く

表 1 – 前のページからの続き

ショートカット	説明	コンテキスト
上/下矢印 または Ctrl+P/Ctrl+N	カーソルを移動またはコマンド履歴を移動	入力が複数の視覚行にまたがる場合 (ラップされているか複数行かどうか)、最初にプロンプト内でカーソルを移動します。カーソルが最初または最後の視覚行にある場合、再度押すとコマンド履歴を移動します。メッセージがキューに入っている場合、最初の行から Up を押すと代わりに**キューに入れたものを取り戻します** (ページ 91)
Esc	Claude を割り込むか、ダイアログを閉じる	現在の応答またはツール呼び出しを途中で停止して、リダイレクトできます。Claude はこれまでの作業を保持します。**メッセージがキューに入っている** (ページ 90) 場合、Claude Code は次にそれらを送信します。ダイアログが開いている場合、Esc はダイアログを閉じます。フッター項目 (**サブエージェントパネル** (ページ 344) 内の行など) が選択されている場合、Esc は割り込みの代わりに**それを選択解除** (ページ 772) します。権限プロンプトでは、Esc はアクション (**コメントなしのいいえ** (ページ 1040) と同じ) を拒否します
Esc + Esc	入力ドラフトをクリアするか、巻き戻す	プロンプト入力にテキストが含まれている場合、ダブル Esc はそれをクリアし、ドラフトを履歴に保存して Up で呼び出せるようにします。入力が空の場合、ダブル Esc は**巻き戻しメニュー** (ページ 1019) を開いて、前の時点からコードと会話を復元または要約できます

次のページに続く

表 1 – 前のページからの続き

ショートカット	説明	コンテキスト
Ctrl+Enter または Ctrl+X Ctrl+S	キューに入れたメッセージを今すぐ送信	**キューに入れたメッセージ** (ページ 90) とドラフトを直ちに送信します。*Claude Code* **がキューに入れたものを送信するとき** (ページ 90) は、Claude が作業中のターンに何が起こるかについて説明しています。**シェルモード** (ページ 89) では、キーはコマンドをキューに入れるだけです。拡張キーを報告しないターミナルでは、Ctrl+Enter は通常の Enter として到着します。Ctrl+X Ctrl+S はすべてのターミナルで機能します。Claude Code v2.1.275 以降が必要です
Shift+Tab、または Node または Bun ランタイムが VT 入力モードを有効にしない場合は Windows で Alt+M	権限モードを循環	default（モード指標で Manual とラベル付け）、acceptEdits、plan、および利用可能な場合は bypassPermissions と auto を循環します。auto から、最初のプレスは default に切り替わります。権限モードを参照してください。ファイル権限プロンプトでは、同じキーが開いている**コメントフィールド** (ページ 1040) を閉じます。フィールドが開いていない場合、プロンプトがそのオプションを提供するときに、セッションの残りの部分でアクションを許可するオプションを選択します
Option+P（macOS）または Alt+P（Windows/Linux）	モデルを切り替え	プロンプトをクリアせずにモデルを切り替え
Option+T（macOS）または Alt+T（Windows/Linux）	拡張思考を切り替え	拡張思考モードを有効または無効にします。Opus 5.5、Sonnet 5.5、または Fable モデルには効果がありません。これらは常に拡張思考を使用します。macOS で Option を Meta として設定せずに動作します
Option+O（macOS）または Alt+O（Windows/Linux）	高速モードを切り替え	高速モードを有効または無効にします

6.1.2 テキスト編集

ショートカット	説明	コンテキスト
Ctrl+A	カーソルを現在の行の開始に移動	複数行入力では、現在の論理行の開始に移動します
Ctrl+E	カーソルを現在の行の終了に移動	複数行入力では、現在の論理行の終了に移動します
Ctrl+K	行の終了まで削除	削除されたテキストは貼り付け用に保存されます
Ctrl+U	カーソルから行の開始まで削除	削除されたテキストは貼り付け用に保存されます。複数行入力で複数行にわたってクリアするには繰り返してください。macOS では、iTerm2 および Terminal.app を含むターミナルエミュレータは Cmd+Backspace をこのショートカットにマップします
Ctrl+W	前の空白文字まで戻って削除	削除されたテキストは貼り付け用に保存されます。1 回のプレスでパス全体または --flag=value を削除します。前の単語だけを削除するには、macOS で Option+Delete を押すか、Windows で Ctrl+Backspace を押してください
Ctrl+Y	削除されたテキストを貼り付け	Ctrl+K、Ctrl+U、Ctrl+W などの単語または行削除ショートカットで最後に削除したテキストを貼り付けます
Alt+Y（Ctrl+Y の後）	貼り付け履歴を循環	貼り付けた後、以前に削除されたテキストを循環します。macOS では *Option を Meta として* (ページ 75) 設定する必要があります
Alt+B	カーソルを 1 単語戻す	単語ナビゲーション。macOS では *Option を Meta として* (ページ 75) 設定する必要があります
Alt+F	カーソルを 1 単語進める	現在の単語の終了に移動するか、カーソルが単語間にある場合は次の単語の終了に移動します。macOS では *Option を Meta として* (ページ 75) 設定する必要があります
Alt+D	単語の終了まで削除	現在の単語の終了まで削除するか、カーソルが単語間にある場合は次の単語の終了まで削除します。削除されたテキストは貼り付け用に保存されます。macOS では *Option を Meta として* (ページ 75) 設定する必要があります

次のページに続く

表 2 – 前のページからの続き

ショートカット	説明	コンテキスト
Ctrl+_ または Ctrl+Shift+-	最後の入力編集を元に戻す	前の入力テキストとカーソル位置を復元します

6.1.3 編集ショートカットの単語境界

単語ショートカット Alt+B、Alt+F、Alt+D、Option+Delete、Ctrl+Backspace は、単語を文字と数字の実行として扱うため、_、.、/ などの句読点は単語を分離します。プロンプトに src/utils/foo.ts がある場合、Alt+B を繰り返し押すと ts、foo、utils、src の開始で停止します。

Ctrl+W は異なります。句読点を無視して前の空白文字まで戻って削除するため、1 回のプレスで src/utils/foo.ts 全体を削除します。

中国語や日本語など、スペースなしで書かれたテキストでは、単語ショートカットは依然として 1 回に 1 単語移動または削除します。

これらの readline 規約は Claude Code v2.1.261 以降に適用されます。以前のバージョンでそれらをオンにした keybindingFlavor 設定は非推奨であり、効果がありません。

これらのショートカットは**キーバインディング設定ファイル** (ページ 765) で再マップできません。このファイルにはそれらのアクションがありません。

6.1.4 テーマと表示

ショートカット	説明	コンテキスト
Ctrl+T	コードブロックの構文強調表示を切り替え	/theme ピッカーメニュー内でのみ機能します。Claude の応答のコードが構文色を使用するかどうかを制御します

6.1.5 複数行入力

方法	ショートカット	コンテキスト
クイックエスケープ	\ + Enter	すべてのターミナルで機能
Option キー	Option+Enter	macOS で *Option* を *Meta* として (ページ 830) 有効にした後
Shift+Enter	Shift+Enter	iTerm2、WezTerm、Ghostty、Kitty、Warp、Apple Terminal、Windows Terminal でネイティブ。他のターミナルについては、**複数行プロンプトを入力** (ページ 829) を参照してください
制御シーケンス	Ctrl+J	設定なしで任意のターミナルで機能
ペーストモード	直接貼り付け	コードブロック、ログの場合

6.1.6 クイックコマンド

ショートカット	説明	注記
/ で開始	コマンドまたはスキル	**コマンド** (ページ 83) と **スキル** (ページ 275) を参照してください
! で開始	シェルモード	コマンドを直接実行し、その出力をセッションに追加し、Claude がそれに応答するようにします
@	ファイルパスメンション	ファイルパスオートコンプリートをトリガーします。クロスセッションメッセージングを使用するセッションでは、@ の後に少なくとも 1 文字を入力すると、Claude Code はこのマシン上の他のライブセッションも提案するため、Claude にメッセージを送信するセッションを選択するよう指示できます。Claude Code v2.1.232 以降が必要です
:	絵文字ショートコード	完全な :name: を入力して絵文字を挿入するか、2 文字以上で候補を表示します。**絵文字ショートコード** (ページ 93) を参照してください。Claude Code v2.1.217 以降が必要です
? 空の入力で	ショートカットヘルプパネルを切り替え	入力にテキストが既に含まれている場合、? を入力すると文字が挿入されます

6.1.7 トランスクリプトビューア

トランスクリプトビューアが開いている場合 (Ctrl+O で切り替え)、これらのショートカットが利用可能です。/tui を引数なしで実行して、どのレンダラーがアクティブかを確認します。Ctrl+E は *transcript:toggleShowAll* (ページ 765) で再バインドできます。

ショートカット	説明
?	キーボードショートカットヘルプパネルを切り替えます。フルスクリーンレンダリングが必要です
{ / }	vim 段落モーションのように、前または次のユーザープロンプトにジャンプします。フルスクリーンレンダリングが必要です
Ctrl+E	すべてのコンテンツを表示を切り替えます。クラシックレンダラーでのみ利用可能で、フルスクリーンレンダリングでは利用できません
[	完全な会話をターミナルのネイティブスクロールバックに書き込んで、Cmd+F、tmux コピーモード、その他のネイティブツールで検索できるようにします。フルスクリーンレンダリングが必要です
v	会話を一時ファイルに書き込んで、$VISUAL または $EDITOR で開きます。フルスクリーンレンダリングが必要です
q、Ctrl+C、Esc	トランスクリプトビューを終了します。3 つすべて *transcript:exit* (ページ 765) で再バインドできます

6.1.8 音声入力

ショートカット	説明	注記
Space を保持またはタップ	音声ディクテーション	音声ディクテーションが有効になっている必要があります。保持して記録するか、/voice tap を実行してタップトグルを実行します。再バインド可能

6.2 コマンド

Claude Code で / と入力すると、利用可能なコマンドが表示されます。または / の後に任意の文字を入力してフィルタリングできます。/ メニューには、組み込みコマンド、バンドルされたユーザー作成の *skills* (ページ 275)、および plugins と *MCP servers* (ページ 409) によって提供されるコマンドが表示されます。すべての組み込みコマンドがすべてのユーザーに表示されるわけではありません。プラットフォームまたはプランに依存するコマンドもあり、設計上、メニューから非表示になっているが、完全な名前を入力すると実行される利用可能なコマンドもいくつかあります。

フルスクリーンレンダリング では、/ コマンドと @ ファイル提案リストもマウスに応答します。行にマウスを置くとハイライトされ、クリックするとそれが選択されます。

Claude Code に含まれるコマンドの完全なリストについては、コマンドリファレンス を参照してください。

6.2.1 プロンプトの途中でコマンドを完成させる

コマンド補完はプロンプトの途中でも機能します。スペースの後に / を入力し、その後に名前の最初の文字を入力します。例えば run the tests, then /com のようにします。名前がそれらの文字で始まるコマンドのみが一致するため、/tmp/notes.md のようなファイルパスではリストが開いたままになりません。Claude Code がコマンド自体を実行するのは、コマンドが メッセージを開始する 場合のみです。

- フルスクリーンレンダリング **の場合**：入力中に一致するコマンドがリストとして開き、行がハイライトされていないため、Enter キーを押すとプロンプトがそのまま送信されます。Tab キーを押すと最上位の一致が挿入されます。または矢印キーと Enter キーで行を選択します。
- **フルスクリーン外の場合**：最上位の一致の残りがカーソルでゴーストテキストとして表示され、複数のコマンドが一致する場合は +2 などのカウントが表示されます。Tab キーを押すと唯一の一致が挿入されるか、複数が一致する場合はリストが開き、矢印キーと Enter キーで行を選択します。

両方のレンダラーで、プロンプトの途中の裸の / で Tab キーを押すと、すべてのコマンドがリストされます。

プラグインスキルはその裸の名前でも一致するため、/deploy は myplugin:deploy-app という名前のスキルを見つけます。一致を挿入すると、Claude Code は完全な /myplugin:deploy-app を書き込みます。

6.3 Vim エディタモード

`/config` → Editor mode から vim スタイルの編集を有効にします。

Claude Code は、`Ctrl+O` で**トランスクリプトビューア** (ページ 82) を切り替えたり、`/config` などのパネルを開いたり閉じたりするときに、vim モードとカーソル位置を保持します。プロンプトを NORMAL モードのままにしておくと、戻ってきたときも NORMAL モードのままで、カーソルは置いた場所にあります。

6.3.1 モード切り替え

コマンド	アクション	元のモード
`Esc` または `Ctrl+[`	NORMAL モードに入ります。Kitty キーボードプロトコルを使用するターミナルでは、`Ctrl+[` は v2.1.242 以降が必要です	INSERT、VISUAL
`i`	カーソルの前に挿入	NORMAL
`I`	行の開始位置に挿入	NORMAL
`a`	カーソルの後に挿入	NORMAL
`A`	行の終了位置に挿入	NORMAL
`o`	下に行を開く	NORMAL
`O`	上に行を開く	NORMAL
`v`	文字単位のビジュアル選択を開始	NORMAL
`V`	行単位のビジュアル選択を開始	NORMAL

6.3.2 INSERT モードのキーシーケンスを再マップ

`vimInsertModeRemaps` 設定は、2 キーの INSERT モードシーケンスを Escape にマップするため、`jj` のようなマッピングで NORMAL モードに戻ります。Claude Code v2.1.208 以降が必要です。

以下の `~/.claude/settings.json` の例は、vim モードをオンにして `jj` を Escape にマップします：

```
{
  "editorMode": "vim",
  "vimInsertModeRemaps": { "jj": "<Esc>" }
}
```

各キーは連続して入力される正確に 2 つの印字可能文字で、`"<Esc>"` は唯一サポートされているターゲットです。異なる長さまたはターゲットを持つエントリは無視されます。

シーケンスの最初の文字を入力すると、通常通り挿入されます。1 秒以内に 2 番目の文字を押すと、保留中の文字が削除され、NORMAL モードに切り替わり、入力に文字は残りません。1 秒のウィンドウの後、または異なるキーが続く場合、両方の文字はリテラルテキストとして残るため、2 つのキーの間で一時停止することで、シーケンスを含む単語を入力できます。

Claude Code は、ユーザー設定ファイル、`--settings` フラグ、および**マネージド設定**からのみこの設定を読み込みます。プロジェクトの `.claude/settings.json` または `.claude/settings.local.json` のエントリは無視されるため、チェックアウトされたリポジトリはキーストロークを再マップできません。

6.3.3 ナビゲーション（NORMAL モード）

コマンド	アクション
h/j/k/l	左/下/上/右に移動
Space	右に移動
w	次の単語
e	単語の終わり
b	前の単語
0	行の開始
$	行の終わり
^	最初の空白以外の文字
gg	入力の開始
G	入力の終わり
f{char}	次の文字の出現位置にジャンプ
F{char}	前の文字の出現位置にジャンプ
t{char}	次の文字の出現位置の直前にジャンプ
T{char}	前の文字の出現位置の直後にジャンプ
;	最後の f/F/t/T モーションを繰り返す
,	最後の f/F/t/T モーションを逆方向で繰り返す
/	逆履歴検索を開く、Ctrl+R と同じです。空の検索プロンプトはヒントを表示します：Esc を押してから i を押してから / を押してコマンドメニューを開く代わりに

注釈:

vim NORMAL モードでは、カーソルが入力の開始または終了にあり、それ以上移動できない場合、j/k と ↑/↓ はコマンド履歴をナビゲートします。← は空のプロンプトで INSERT だけでなく NORMAL モードからもエージェントビューを開きます。v2.1.219 より前は、← は空のプロンプトで NORMAL モードでは何もしませんでした。

6.3.4 編集（NORMAL モード）

コマンド	アクション
x	文字を削除
r{char}	カーソルの下の文字を {char} に置換
dd	行を削除
D	行の終わりまで削除
dw/de/db	単語を削除/終わりまで/戻す
df{char}/dt{char}	次の文字の出現位置まで削除（含む）、または直前まで
dj/dk	現在の行と下または上の行を削除
dgg/dG	現在の行から最初または最後の行まで削除
d0/c0/y0	カーソルから行の開始まで削除、変更、またはヤンク。Claude Code v2.1.281 以降が必要です

次のページに続く

表 10 – 前のページからの続き

コマンド	アクション
cc	行を変更
C	行の終わりまで変更
cw/ce/cb	単語を変更/終わりまで/戻す
s	文字を置換：カーソルの下の文字を削除して INSERT モードに入ります。Claude Code v2.1.211 以降が必要です
S	行を置換：行をクリアして INSERT モードに入ります。Claude Code v2.1.211 以降が必要です
yy/Y	行をヤンク（コピー）
yw/ye/yb	単語をヤンク/終わりまで/戻す
p	カーソルの後に貼り付け
P	カーソルの前に貼り付け
>>	行をインデント
<<	行をデデント
J	行を結合
u	元に戻す
.	最後の変更を繰り返す

6.3.5 テキストオブジェクト（NORMAL モード）

テキストオブジェクトは d、c、y などのオペレータで機能します：

コマンド	アクション
iw/aw	内部/周囲の単語
iW/aW	内部/周囲の WORD（空白区切り）
i"/a"	内部/周囲のダブルクォート
i'/a'	内部/周囲のシングルクォート
i(/a(	内部/周囲の括弧
i[/a[	内部/周囲の角括弧
i{/a{	内部/周囲の波括弧

6.3.6 ビジュアルモード

文字単位の選択は v を、行単位の選択は V を押します。モーションは選択を拡張し、オペレータはそれに直接作用します。

コマンド	アクション
d/x	選択を削除
y	選択をヤンク
c/s	選択を変更
p	選択をレジスタの内容に置換

次のページに続く

表 12 – 前のページからの続き

コマンド	アクション
r{char}	選択されたすべての文字を {char} に置換
~/u/U	選択を切り替え、小文字、または大文字
>/<	選択された行をインデントまたはデデント
J	選択された行を結合
o	カーソルとアンカーを交換
iw/aw/i"/…	テキストオブジェクトを選択
v/V	文字単位と行単位を切り替え、またはビジュアルモードを終了

Ctrl+V を使用したブロック単位のビジュアルモードはサポートされていません。

6.4 コマンド履歴

Claude Code はあなたが入力したプロンプトの履歴を保持し、上矢印キーでの呼び出しは同じプロジェクトの過去のセッションからプロンプトを取得します。

- 入力履歴は作業ディレクトリごとに保存されます
- /clear を実行すると新しいセッションが開始されます。呼び出しの際には新しいセッションのプロンプトが最初に表示され、その後に以前のセッションのプロンプトが表示されます。前のセッションの会話は保存され、再開することができます。
- 同じプロンプトを連続して 2 回送信すると、1 つの履歴エントリが記録されるため、上矢印キーを押すと前の異なるプロンプトにステップします
- 貼り付けたテキストを含むプロンプトを呼び出すと、再度送信する際に Claude Code は完全な貼り付けコンテンツを送信します。コンテンツが 自動的にクリーンアップ されている場合、Claude Code は文字通りの [Pasted text #N] 文字列を送信しません。プロンプトに何が起こるかについては、**大きなコンテンツを貼り付ける** (ページ 838) を参照してください
- ! を使用した履歴展開はデフォルトで無効になっています

6.4.1 Ctrl+R での逆方向検索

Ctrl+R を押してコマンド履歴をインタラクティブに検索します。フルスクリーンレンダリング では、Ctrl+R は検索ダイアログを開きます。入力してフィルタリングし、Up と Down を押してマッチを移動し、Ctrl+S を押してスコープをこのセッション、このプロジェクト、すべてのプロジェクトの間で循環させます。Enter または Tab を押してマッチをプロンプト入力に配置するか、Esc を押してキャンセルします。以下のステップは、クラシックレンダラーのインライン検索について説明しています。

1. **検索を開始**: Ctrl+R を押して逆方向履歴検索を有効にします
2. **クエリを入力**: 前のコマンドで検索するテキストを入力します。検索用語は一致する結果で強調表示されます
3. **マッチを移動**: Ctrl+R をもう一度押して、より古いマッチを循環させます
4. **検索スコープ**: インライン検索は常にすべてのプロジェクトからプロンプトを検索します

5. **マッチを受け入れる**:
    - `Tab` または `Esc` を押して現在のマッチを受け入れ、編集を続けます
    - `Enter` を押して受け入れ、コマンドを直ちに実行します
6. **検索をキャンセル**:
    - `Ctrl+C` を押してキャンセルし、元の入力を復元します
    - 空の検索で `Backspace` を押してキャンセルします

インライン検索は完全なプロンプト履歴をスキャンし、最新のものから順に、重複は最新の出現に折りたたまれます。フルスクリーン ダイアログは選択されたスコープ内の全プロンプト履歴を検索し、最新のものから順に、重複は最新の出現に折りたたまれます。最も最近のプロンプトが直ちに表示され、古いプロンプトからのマッチは Claude Code が残りの履歴を読み込むにつれて入力されます。マッチするプロンプトは検索用語が強調表示されて表示されるため、前の入力を見つけて再利用できます。

マッチを受け入れるか検索をキャンセルすると、Claude Code がまだ履歴を読み込んでいる場合でも、直ちに有効になります。

6.5 バックグラウンド Bash コマンド

Claude Code はバックグラウンドで Bash コマンドを実行することをサポートしており、長時間実行されるプロセスが実行されている間も作業を続けることができます。

6.5.1 バックグラウンド実行の仕組み

Claude Code がコマンドをバックグラウンドで実行する場合、コマンドを非同期で実行し、すぐにバックグラウンドタスク ID を返します。Claude Code はコマンドがバックグラウンドで実行され続けている間、新しいプロンプトに応答できます。

コマンドをバックグラウンドで実行するには、以下のいずれかを行うことができます。

- Claude Code にコマンドをバックグラウンドで実行するよう指示する
- `Ctrl+B` を押して、通常の Bash ツール呼び出しをバックグラウンドに移動する。Tmux ユーザーは tmux のプリフィックスキーのため、`Ctrl+B` を 2 回押す必要があります。

コマンドが完了する前にタイムアウトに達した場合、Claude Code は自動的にそれをバックグラウンドに移動します。停止する代わりにバックグラウンドに移動します。ただし、コマンドが `sleep` で始まる場合は除きます。`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` でバックグラウンドタスクをオフにしているか、**ベアモード** (ページ 696) で開始している場合、コマンドはタイムアウト時に停止します。タイムアウトを変更するには、Bash タイムアウト環境変数を設定してください。

主な機能：

- 出力はファイルに書き込まれ、Claude は Read ツールを使用してそれを取得できます
- バックグラウンドタスクには、追跡と出力取得のための一意の ID があります
- バックグラウンドタスクは、Claude Code が終了するときに自動的にクリーンアップされます。macOS と Linux では、`/tasks` からバックグラウンドタスクを停止するか、Claude Code が終了時に停止す

る場合、`setsid` または `timeout` の下で開始されたものなど、タスクのシェルから切り離されたプロセスも停止します

- セッションを終了する代わりにセッションをバックグラウンドにする場合、バックグラウンドセッションでバックグラウンドタスクが実行され続けます。実行中のセッションをバックグラウンドにするを参照してください
- 出力が 5GB を超える場合、バックグラウンドタスクは自動的に終了され、stderr に理由を示すメモが表示されます
- macOS と Linux では、オペレーティングシステムが重大なメモリプレッシャーを報告する場合、Claude Code はバックグラウンドタスクを停止します。ただし、セッションが少なくとも 30 分間アイドル状態にあり、ターンまたはサブエージェントが実行されていない場合に限ります。Claude Code v2.1.193 以降が必要です
  - デバッグログには、タスクが停止された理由、またはプレッシャーイベントがそれらを実行し続けた理由が記載されています
  - `CLAUDE_CODE_DISABLE_BG_SHELL_PRESSURE_REAP` を 1 に設定して、メモリプレッシャー停止をオフにします
- ターミナル、デスクトップアプリ、または VS Code 拡張機能から作業するローカルセッションでのバックグラウンドコマンドには時間制限がありません。`-p` 実行やクラウドセッションなど、無人で実行されるセッションでは、Claude Code はバックグラウンドコマンドを時間制限に達した時点で停止します
- フォアグラウンド**サブエージェント** (ページ 344) が開始したバックグラウンドコマンドは、そのサブエージェントの実行が終了するときに終了します。完了、失敗、または中断されたかどうかに関わらず。ツール参照のバックグラウンドコマンドが停止するときを参照してください

すべてのバックグラウンドタスク機能を無効にするには、`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` 環境変数を 1 に設定してください。**ベアモード** (ページ 696) で開始することでもオフになります。

一般的なバックグラウンド実行コマンド：

- ビルドツール（webpack、vite、make）
- パッケージマネージャー（npm、yarn、pnpm）
- テストランナー（jest、pytest）
- 開発サーバー
- 長時間実行プロセス（docker、terraform）

6.5.2 ！プリフィックスを使用したシェルモード

入力に！をプリフィックスして、Claude を経由せずにシェルコマンドを直接実行します。

```
! npm test
! git status
! ls -la
```

シェルモード：

- コマンドとその出力を会話コンテキストに追加します
- リアルタイムの進捗と出力を表示します
- 長時間実行コマンドの同じ Ctrl+B バックグラウンド実行をサポートします
- Claude がコマンドを解釈または承認する必要がありません
- 履歴ベースのオートコンプリートをサポートします。部分的なコマンドを入力して Tab を押すと、現在のプロジェクトの前の ! コマンドから完成します
- v2.1.193 以降、すべてのプラットフォームでライブファイルパスオートコンプリートをサポートします。./src/ または ~/ などのフォワードスラッシュを含むトークンを入力して、一致するファイルとディレクトリのドロップダウンを表示し、Tab を押して受け入れます。Windows でもフォワードスラッシュを使用してください。ドロップダウンは / でトリガーされ、\ ではありません
- 空のプロンプトで Escape、Backspace、または Ctrl+U で終了します
- ! で始まるテキストを空のプロンプトに貼り付けると、シェルモードが自動的に入力されます。入力された ! の動作と一致します

セッションが**厳密なサンドボックスモード** (ページ 1076) の下にリストされているものの 1 つでない限り、シェルモードで入力するコマンドは、サンドボックスを有効にしている場合でも**サンドボックス** (ページ 1069) の外で実行されます。これは、サンドボックスが Claude が実行するコマンドに適用されるためです。

Claude はコマンド出力がトランスクリプトに到達すると自動的に応答するため、! npm test を実行して、2 番目のプロンプトなしで失敗の説明を取得できます。応答は通常のプロンプトを送信するのと同じコストがかかります。出力がコンテキストに追加されて応答がない以前の動作を復元するには、settings.json で respondToBashCommands を false に設定します。v2.1.186 より前では、シェルモードは常に出力をコンテキストに追加して応答がありませんでした。

6.6 Claude が作業中にメッセージをキューに入れる

Claude が作業中に、メッセージを入力して Enter キーを押すと、Claude Code はターンを中断する代わりにメッセージをキューに入れ、送信するまでキューに入れられたエントリを会話に表示します。! **シェルコマンド** (ページ 89) と、ほとんどのコマンドも同じようにキューに入れることができます。ただし、Claude Code が送信直後に実行する /status などのコマンドは除きます。

送信されたメッセージとキューに入れられたメッセージは、Claude が応答を開始するまでグレーで表示されるため、Claude がまだ処理していないメッセージを判別できます。

接続された *IDE* (ページ 200) からの選択が付いたメッセージをキューに入れた場合、Enter キーを押したときに選択していた内容が保持されます。その後に選択した内容がどうであれ関係ありません。

6.6.1 Claude Code がキューに入れたものを送信するタイミング

キューに入れたエントリが Claude に到達するタイミングは、何をキューに入れたかによって異なります。

- メッセージ：Claude がツール呼び出しを実行している間にメッセージをキューに入れた場合、Claude Code はそれらのツール呼び出しが完了するとすぐに、同じターン内で Claude に渡します。ターンが

メッセージがまだキューに入った状態で終了した場合、別のキープレスなしに送信され、入力した順序で送信されます。

- コマンドとシェルコマンド：Claude Code はターンが終了するまでそれらを保持し、その後 1 つずつ実行し、キューに入れた順序を保持します。

キューに入れたものをターンの終了を待たずに送信するには、`Ctrl+Enter` キーを押します。キューに入れたメッセージがすぐに送信され、入力していたドラフトがある場合はそれがその後ろにキューに入れられます。Claude Code v2.1.275 以降が必要です。

キューに入れた ! シェルコマンドがメッセージの前にある場合、このキーはターンを中断します。それ以外の場合、ターンに何が起こるかは、キーを押したときに Claude が何をしているかによって異なります。

- シェルコマンド、サブエージェント、または**バックグラウンド** (ページ 88) に移動できるその他の作業を実行している：その作業はバックグラウンドに移動して実行を続け、Claude は同じターン内でメッセージを読みます。
- 応答を書き込むだけ、またはバックグラウンドに移動できないものを実行している：Claude Code はターンを中断し、メッセージを次に送信します。v2.1.281 より前では、このキーは両方の場合でターンを中断していました。

シェルモード (ページ 89) では、このキーはコマンドをキューに入れるだけです。拡張キーを報告しないターミナルでは、`Ctrl+Enter` は通常の `Enter` として到達し、ドラフトをキューに入れます。`Ctrl+X Ctrl+S` はどのターミナルでも機能します。どちらのキーも *chat:sendNow* **アクション** (ページ 767) のバインディングです。

`Esc` キーを押してターンを中断し、ドラフトを送信しません。Claude Code はキューに入れたものを保持し、すぐに送信します。

Claude Code は、キューに入れる代わりに、送信直後に実行するコマンドがあります。その中には `/model`、`/effort`、`/fast` があります。これら 3 つはそれぞれ設定を変更します。モデル、effort レベル、または fast mode です。Claude Code が新しい設定を Claude が既に作業しているターンに適用するか、次のターンからのみ適用するかは、コマンドによって異なります。

- */model* (ページ 728)：Claude Code がキャッシュ警告を表示する場合は、確認後、Claude Code はそのターン内で次に行うリクエストに変更を適用します。
- */effort* (ページ 747)：Claude Code がキャッシュ警告を表示する場合は、確認後、Claude Code はそのターン内で次に行うリクエストに変更を適用します。
- `/fast`：Claude Code はターンが開始されたときにアクティブだった fast mode 設定を保持するため、速度の変更は次のターンから適用されます。現在のモデルが fast mode をサポートしていない場合、それをオンにするとモデルも切り替わり、Claude Code はそのターン内で次に行うリクエストから新しいモデルを使用します。

6.6.2 キューに入れたものを取り戻す

入力ボックスの最初の行から `Up` キーを押すと、キューに入れたメッセージとコマンドを取り戻すことができます。Claude Code はそれらをキューから削除し、入力ボックスに 1 行に 1 つずつ、入力していたテキストの前に配置します。テキストを編集して `Enter` キーを押すと、1 つのエントリとして再度キューに入れられます。または、入力ボックスをクリアして削除します。

Claude Code は、入力ボックスが空で、キューに入れたものが他にない場合にのみ、キューに入れたシェルコマンドを取り戻し、その際に入力ボックスをシェルモードに切り替えます。それ以外の場合は、それらをキューに留めておき、! プレフィックス付きでリストし、ターンが終了した後に実行します。

6.7 プロンプト提案

セッションを最初に開くと、Claude Code はプロンプト入力に薄い灰色で表示されたコマンド例を表示して、開始を支援します。このコマンド例はプロジェクトの git 履歴から選択されるため、最近作業していたファイルが反映されます。

Claude が応答した後、Claude Code は会話履歴に基づいて次のプロンプトを提案できます。例えば、複数部分のリクエストからのフォローアップステップや、ワークフローの自然な続きなどです。

- `Tab` キーまたは `Right arrow` キーを押して提案をプロンプト入力に配置し、`Enter` キーを押して送信します
- 入力を開始して提案を閉じます

Claude Code は、これらの次のプロンプト提案を、セッションが使用しているのと同じモデルへのバックグラウンドリクエストで生成します。このリクエストはプランの使用制限または API コストにカウントされます。会話のプロンプトキャッシュを再利用するため、主にキャッシュ読み取りと少数の出力トークンで構成されるため、追加コストは最小限です。

6.7.1 Claude Code が提案をスキップする場合

インタラクティブモードでは、Claude Code はデフォルトでプロンプト提案をオフにし、`/config` の **Prompt suggestions** トグルを、フィーチャーフラグを取得しないセッション（サードパーティプロバイダー上のセッションや Claude アプリゲートウェイを通じたセッションなど）およびインストールまたはアップグレード後の最初のセッション（フラグがまだ到着していないセッション）で非表示にします。

Claude Code は、以下を含むいくつかの状況で個別の提案もスキップします。

- 不要なコストを避けるため、プロンプトキャッシュがコールドの場合
- 会話の最初のターンの後、一部のセッションで
- 前の応答がエラーで終了した場合
- Plan Mode にいる場合
- アカウントが使用制限に近い、またはその制限に達している場合。制限に達するまで提案をオンのままにするには、`CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION` を `true` に設定します。v2.1.238 より前では、変数が `true` に設定されていても、Claude Code は制限に近い場合にそれらをスキップしていました
- agent team では、デフォルトでチームメイトのセッションで。リーダーのセッションは提案を表示します

Print Mode では、Claude Code はデフォルトでは提案を生成しません。Claude Code が生成した各ターンの後に `prompt_suggestion` メッセージを出力するには、`-p "<prompt>" --output-format stream-json --verbose` で *`--prompt-suggestions`* (ページ 51) を渡します。ジェネレータはここでも非常に短い会話

とコールドプロンプトキャッシュをスキップするため、単一の短い -p クエリは何も出力しない可能性があります。

6.7.2 プロンプト提案をオフにする

プロンプト提案を完全に無効にするには、以下のいずれかを使用します。

- /config で **Prompt suggestions** をオフにします
- 設定ファイルで promptSuggestionEnabled を false に設定します
- CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION 環境変数を false に設定します。これは設定よりも優先されます。

```
export CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION=false
```

組織全体でプロンプト提案をオフにするには、managed settings で promptSuggestionEnabled を false に設定します。また、ユーザーが独自の環境変数で再度有効にできないように、managed env キーの下で CLAUDE_CODE_ENABLE_PROMPT_SUGGESTION を false に設定します。

6.8 絵文字ショートコード

プロンプト入力で : に続けて絵文字ショートコードを入力すると、絵文字が挿入されます。Claude Code v2.1.217 以降が必要です。

- :heart: などの完全なショートコードを入力すると、Claude Code は閉じる : を入力した直後に ☒ ☒ に置き換えます
- : に加えて名前の少なくとも 2 文字を入力します（例：:hea）。提案ポップアップが開き、Tab または Enter を押してハイライトされた絵文字を挿入します

ショートコードは入力の開始時またはスペースの後に続く必要があります。単語または URL 内の : は提案を開きません。

この機能をオフにするには、settings.json で emojiCompletionEnabled を false に設定します。これにより、提案ポップアップとインライン置換の両方が無効になります。

6.9 入力中にスペルチェックを実行する

Claude Code は、入力中にプロンプト入力のスペルミスのある単語に下線を引くことができます。入力ボックス内のテキストのみをチェックし、Claude の返信やファイルはチェックしません。また、入力ボックスが **シェルモード** (ページ 89)、Ctrl+R 履歴検索、または 音声入力 にある場合は何もチェックしません。

スペルチェックはデフォルトでオフになっており、Claude Code は スクリーンリーダーモード では何もチェックしません。Claude Code v2.1.235 以降が必要です。

6.9.1 前提条件

- aspell、hunspell、または ispell をインストールし、PATH に含まれていることを確認してください。Claude Code は、Windows の .cmd シムを含むすべてのプラットフォームで、見つかった 3 つのうち最初のものを実行します。
- プログラムが PATH にあることを確認するには、ターミナルで aspell --version、hunspell --version、または ispell -v を実行してください。「command not found」エラーが表示される場合は、まだ PATH に含まれていません。

6.9.2 スペルチェックをオンまたはオフにする

Claude Code は spellcheck 設定を 3 つの場所から読み込み、プロジェクトの .claude/settings.json と .claude/settings.local.json では無視します。使用している場所からオンにしてください。

ユーザー設定

~/.claude/settings.json に spellcheck を追加してください。これは、**ユーザー設定** (ページ 711) の残りの部分と同様に、開くすべてのプロジェクトに適用されます。

```
{
  "spellcheck": { "enabled": true }
}
```

コマンドライン

spellcheck.json などの JSON ファイルに spellcheck を保存してください。

```
{
  "spellcheck": { "enabled": true }
}
```

次に、ファイルを --settings に渡してください。これはそのセッションのみに適用されます。

```
claude --settings spellcheck.json
```

管理設定

組織の **管理設定ソース** (ページ 1063) の 1 つに spellcheck を追加してください。これは、それらの設定を受け取るすべてのユーザーに適用され、ユーザーはそれをオフにすることはできません。

```
{
  "spellcheck": { "enabled": true }
}
```

スペルチェックがオンになっていることを確認するには、スペルミスのある単語とスペースを入力してください。Claude Code は単語に下線を引きます。下線が引かれない場合は、*Claude Code* **が何も下線を引かない場合** (ページ 95) を参照してください。スペルチェックを再度オフにするには、同じ場所で enabled を false に設定するか、spellcheck を削除してください。

Claude Code が実行する 3 つのプログラムのうちどれを実行するか、どの辞書を使用するか、または下線の色を選択するには、同じ場所で enabled の隣に次のフィールドのいずれかを追加してください。

- `checker`: `aspell`、`hunspell`、または `ispell`。Claude Code は指定したチェッカーからフォールバックせず、他の値を `auto` として扱います。
- `language`: チェッカーの形式の辞書名（`en_GB` など）。Claude Code は、パスや空白を含む名前など、プレーンな辞書名ではない値を無視し、チェッカーはデフォルト辞書を使用します。
- `color`: `yellow` などの色名、または `#rrggbb`、`#rgb`、`rgb(r,g,b)`、`ansi256(n)`、または `ansi:<name>` 値。Claude Code はデフォルトではテーマのエラー色を使用し、認識しない値に対しても同様です。

例えば、この `spellcheck` 設定は hunspell を `en_GB` 辞書で実行し、単語を黄色で下線を引きます。`~/.claude/settings.json`、`--settings` に渡すファイル、および管理設定で同じように機能します。

```
{
  "spellcheck": {
    "enabled": true,
    "checker": "hunspell",
    "language": "en_GB",
    "color": "yellow"
  }
}
```

3 つの場所のうち複数に `spellcheck` 設定がある場合、Claude Code は 1 つだけを使用します。管理設定が最初に、次に `--settings`、次にユーザー設定です。2 つの場所からフィールドを組み合わせることはありません。例えば、`--settings` が `spellcheck` を設定する場合、ユーザー設定の `language` は効果がありません。

6.9.3 Claude Code が下線を引く内容

入力を一時停止した直後、Claude Code は辞書が認識しない単語に下線を引きます。入力中の単語はそれを超えるまで放置し、テキストを変更することはありません。また、コードのように見えるテキストもスキップします。

- `/help`、`@` メンション、URL、ファイルパス、`--verbose` などのフラグなどのコマンド
- 数字、アンダースコア、または最初の後に大文字を含む単語、およびバッククォート内のテキスト

Claude Code は、中国語、日本語、韓国語、タイ語、ラオ語、クメール語、およびミャンマー語のテキストもスキップします。

Claude Code には独自の単語リストがありません。単語がスペルミスであるかどうかは、チェッカーが判断します。Claude Code が単語に下線を引くのを停止するには、チェッカーのドキュメントに従って、チェッカーの個人辞書に単語を追加してください。Claude Code はそれを再起動した後、新しい単語を取得します。

6.9.4 Claude Code が何も下線を引かない場合

Claude Code がチェッカーを実行し続けることができない場合、何も下線を引きません。

- チェッカーがインストールされていない、または `checker` で指定したものが見つからない
- チェッカーが 2 回連続で失敗する（起動時またはセッション中）。Claude Code は最初の失敗後に再起動し、2 番目の失敗後はチェックを停止します（Claude Code を再起動するまで）

- チェッカーが 15 秒以上かかって回答する（3 回）。毎回、Claude Code は待機していた単語をマークしません。3 回目の後、Claude Code を再起動するまでチェックを停止します

これらのうちどれが起こったかを確認するには、スペルチェックをオンにして `claude --debug` を開始し、単語を入力してください。次に、`~/.claude/debug/<session-id>.txt` のデバッグログで `[spellcheck]` 行を探してください。1 行は Claude Code が開始したプログラムに名前を付けるか、探したが見つからなかったプログラムをリストします。後の行は停止した理由を説明します。そこの辞書欠落エラーは、チェッカーが `language` 値の辞書を持っていないか、`language` が設定されていない場合はデフォルトの辞書を持っていないことを意味します。1 つをインストールするか、`language` を持っている辞書に設定してください。

6.10 プロンプト内の見えない文字

貼り付けたテキストには、タグ文字、双方向制御、ゼロ幅スペースなど、ターミナルが何も描画しない Unicode 文字が含まれる可能性があるため、プロンプトには見えないテキストが含まれることがあります。コピーしたテキストがターミナルに描画されない命令を含まないようにするため、Claude Code は Enter キーを押すときにそれらの文字を削除してから、何かを送信します。プロンプトと、プロンプトに含まれる任意の折りたたまれた**貼り付けテキスト参照** (ページ 838) の内容の両方をクリーンアップします。Claude Code はペルシア語とインド系スクリプトが書く結合文字と、絵文字シーケンス内のセレクタを保持します。

Claude Code が何かを削除した場合、その Enter は何も送信しません。クリーンアップされたプロンプトは、`Removed 3 invisible characters · review and press Enter to send` などの通知とともに入力ボックスに戻り、もう一度 Enter キーを押すと、表示されたテキストが送信されます。

`claude "fix the login bug"` のようにコマンドラインでプロンプトを渡す場合、Claude Code は 2 番目の Enter を待ちません。文字を削除し、通知を表示し、クリーンアップされたプロンプトを送信します。クリーンアップされたプロンプトが `/` で始まる場合、Claude Code はそれを入力ボックスに入れて、確認して送信するようにします。

6.11 /diff で変更内容を確認する

`/diff` を実行して、Claude Code を離れることなく、ワーキングツリーの変更内容を確認できます。Claude がこれまでに行った編集と、コミットしていないその他の変更が表示されます。`/diff` で開くものは、有効になっているレンダラーによって異なります。

- フルスクリーンレンダリング：*diff* **パネル** (ページ 97) が会話の横に開きます。パネルは開いたままで、作業を続ける間も更新されます。
- **クラシックレンダラー**：*diff* **ダイアログ** (ページ 98) がプロンプトの上に開き、読み終わったら閉じます。

パネルとダイアログはどちらも、Claude Code に組み込まれた mod の 1 つである `cc-plugin-diff` によって提供されます。`/plugin` でこの mod を無効にすると、`/diff` は代わりに Claude Code の以前のパネルと *diff* **ビューアー** (ページ 773) を開きます。

`/diff` が git から読み取る変更では、サブモジュールは単一のエントリとして表示され、それもサブモジュールが指すコミットが変わった場合のみです。サブモジュール内のファイルへの編集はそこには表示されません。Claude がファイルを編集すると、パネルとダイアログには各ターンの編集を示すビューも表示されます。これらのターンビューは git ではなく Claude のファイル編集から作成されるため、Claude がシェルコマンドを通じて行った変更は、ワーキングツリーのビューである `Current` の下にのみ表示されます。

6.11.1 Diff パネル

diff パネルは、変更されたファイルを追加行数と削除行数とともにリストアップし、リストの下に各ファイルの差分を表示します。Claude Code は、Claude がファイルを編集するか、シェルコマンドを実行するたびに、パネルを更新します。パネルを閉じるには、`/diff` を再度実行するか、ヘッダーの × をクリックします。

パネルを使用するには、以下が必要です。

- フルスクリーンレンダリング
- Git リポジトリ
- 少なくとも 110 列幅のターミナル
- Claude Code v2.1.287 以降

ターミナルが少なくとも 144 列幅の場合、Claude がファイルの編集を開始すると、パネルは自動的に開きます。`/diff` で自分で開いた後、後続のセッションでは、Claude がそれに適した幅のターミナルでファイルを編集するとすぐにパネルが開きます。パネルを閉じると、このセッションと後続のセッションでも閉じたままになり、`/diff` を再度実行するまで開きません。

パネルが開いている間、以下のことができます。

- **ファイルにジャンプする**：リストの行をクリックします。マウスホイールでパネルをスクロールします。ファイルリスト自体が長すぎて収まらない場合は、`Alt+Up` と `Alt+Down`、または `Ctrl+Up` と `Ctrl+Down` でスクロールします。
- **ファイルの変更について Claude に質問する**：ファイルの差分の上にあるファイル名の右側の `ask` をクリックします。Claude Code はそのファイルの差分を次のプロンプトに添付し、そのプロンプトを送信するまでボタンには `asked ✓` と表示されます。別のファイルで質問すると、最初のファイルが置き換えられます。
- **1 つのターンの編集を表示する**：パネルのヘッダーにある `source` ピッカーをクリックし、`Up` と `Down` でターンを選択して `Enter` を押します。ターンには `T1`、`T2` などのラベルが付きます。ピッカーは Claude がファイルを編集すると表示され、`Current` を選ぶとワーキングツリーに戻ります。
- **パネルが除外するファイルを表示する**：リストはテストファイルと生成されたファイルをスキップし、このセッション前の変更を下部の 1 行に折りたたみます。いずれかのカウント行をクリックして展開します。
- **パネルが比較する対象を変更する**：`Ctrl+X B` を押して、このセッションの変更から、コミットされていない変更を 1 つのリストとして、デフォルトブランチからの分岐以降のすべてに循環します。Claude Code は、各リポジトリの選択を記憶します。

ファイルリストのスクロールや比較対象の変更に使うキーを再バインドするには、*Diff* **パネルアクション** (ページ 774) を参照してください。

6.11.2 Diff ダイアログ

diff ダイアログは、プロンプトの上に枠付きのブロックとして開き、変更されたファイルを追加行数と削除行数とともにリストアップします。ファイルは `HEAD` と比較されるか、このリポジトリで最後に *diff* **パネルで選択した** (ページ 97) 対象と比較されます。

Claude がファイルを編集すると、リストの上に `source` ピッカーが表示されます。ピッカーには、Claude にファイルを編集させた各プロンプトに対して 1 つずつ、`T1`、`T2` などのラベルが付いたターンビューが用意されます。ターンビューにはそのターンの編集のみが表示され、`Current` を選ぶとワーキングツリーに戻ります。

ダイアログでは次のキーを使用します。

- **上と下**：ファイルを選択します。
- **Enter**：選択したファイルの差分を開きます。上下、または PageUp と PageDown でスクロールします。
- **Esc**：ファイルの差分からリストに戻るか、リストからダイアログを閉じます。
- **Tab**：`source` ピッカーに移動し、上下で `Current` またはターンビューを選択して Enter を押します。ファイルの差分では、Tab で `ask` ボタンに移動します。Enter を押すと、そのファイルの差分が次のプロンプトに添付されます。

6.12 /btw で補足質問をする

`/btw` を使用して、現在の作業に関する質問をしながら、会話履歴に追加しないようにします。

```
/btw what was the name of that config file again?
```

Claude は補足質問に対して、会話に既に含まれている内容から回答します。つまり、あなたのメッセージ、Claude の返信、および収集したツール結果から回答します。Claude が既に読んだコード、以前に下した決定、またはセッションの他の内容について質問できます。後の補足質問は、以前の補足質問も確認します。Claude Code は最新の 20 個のやり取りを各質問で再生します。これは、それらをクリアするまで続きます。質問と回答は会話履歴に入りません。ターミナルでは、却下可能なオーバーレイに表示されます。ターミナルはスレッドをメモリに保持します。`x` を押して以前のやり取りをクリアすると、Claude Code を終了するときに消えます。

VS Code **拡張機能** (ページ 177) のチャットパネルでは、`/btw` はこのセクションで説明するオーバーレイではなくパネルを開き、パネル内で直接フォローアップ質問をします。パネルのスレッドはウィンドウの再読み込みを通じて保持され、そのページで説明されている保持スケジュールに従います。拡張機能は v2.1.227 以降が必要です。以前の拡張機能バージョンは `/btw` を提供していません。

- **Claude が作業中に利用可能**：Claude が応答を処理している間でも `/btw` を実行できます。補足質問は独立して実行され、メインのターンを中断しません。これまでの会話のすべてが表示されます。ただし、Claude がまだ作成している返信は除きます。
- **ツールアクセスなし**：補足質問は既にコンテキストにあるものからのみ回答します。Claude は補足質問に回答する際に、ファイルを読み取ったり、コマンドを実行したり、検索したりすることはできません。Claude がテキストとしてツール呼び出しを書き出す場合でも、回答は何も実行されなかったというメモで終わります。

- **単一の応答**：オーバーレイにはフォローアップターンがありません。スレッドを続行するには、別の /btw 質問をしてください。ローカルセッションで完全なツールアクセスを続行するには、f を押してこの質問と回答を**バックグラウンドサブエージェント** (ページ 354) にフォークします。
- **低コスト**：会話のプロンプトキャッシュがウォーム状態の間、補足質問は回答自体を超えるコストはほとんどかかりません。

5 つの最新の以前の補足質問は、現在の回答の上に薄い色のリストとして表示され、古いものの数がカウントされます。これらは会話履歴から除外されます。

オーバーレイを却下した後に戻るには、質問なしで /btw を実行します。オーバーレイは最新のやり取りで再度開きます。v2.1.212 より前では、質問なしの /btw は代わりに使用メッセージを出力していました。

回答が表示されたら、オーバーレイはこれらのキーを受け入れます。

キー	アクション
Space、Enter、Escape	回答を却下してプロンプトに戻る
Up / Down	回答をスクロール
Shift+Left / Shift+Right	この回答と以前の /btw 回答の間をステップします。Shift+Left は古い回答に移動し、Shift+Right は現在の回答に戻ります。[と] も同じことを行い、ターミナルが矢印キーで Shift を報告しない場合に使用します。Tab / Shift+Tab は同じ回答をサイクルします。Claude Code v2.1.257 以降が必要です。v2.1.187 から v2.1.256 の間、キーはプレーン Left / Right でした
c	回答をクリップボードに生の Markdown としてコピーします。マウス選択の代わりにこれを使用してください。マウス選択はソーステキストではなく、ハードラップされたターミナルレンダリングをキャプチャします
f	親会話にこの質問と回答を継承する**フォークされたサブエージェント** (ページ 354) を開始して、完全なツールアクセスで続行できるようにします。現在のセッションに留まり、プロンプトの下の**パネル** (ページ 354) でフォークを見つけます。ローカルセッションでのみ利用可能
x	現在の回答の上に表示される以前の /btw やり取りのリストをクリア

接続されたバックグラウンドセッションでは、Left はデタッチしてエージェントビューに戻ります。これは、回答がまだ到着している間でも行われます。補足質問は、あなたが離れている間も実行を続けます。次回セッションに接続するときに、オーバーレイは補足質問、またはその回答で再度開きます。v2.1.257 より前では、Left はそこでデタッチしませんでした。

/btw はあなたの完全な会話を見ていますが、ツールはありません。**サブエージェント** (ページ 317) はツールを持ち、受け取るプロンプトから開始するか、**フォーク** (ページ 354) の場合はこの会話のコピーから開始します。/btw を使用して、Claude がこのセッションから既に知っていることについて質問してください。サブエージェントを使用して、何か新しいことを見つけてください。

6.13 タスクリスト

タスクリストは Claude のやることリストです。複数ステップの作業を計画するために Claude が作成したアイテムで、保留中、進行中、完了を示すインジケーターが付いています。バックグラウンドタスクビューとは別です。実行中のシェルとサブエージェントを確認するには、代わりに `/tasks` を使用してください。

リストは、タスク追跡ツールを持つセッションにのみ入力されます。Claude Code は Claude 3.x モデル、Opus 4 から 4.7、Sonnet 4 から 4.6、Haiku 4.5 でデフォルトでこれらのツールを提供します。他のモデル（Claude Code が認識しないモデル ID を含む）では、`CLAUDE_CODE_ENABLE_TODO_TOOLS=1` でオプトインするか、タスクツールの利用可能性 の下にある他の方法のいずれかを使用しない限り、リストは空のままです。セッションにツールがある場合、タスクリストは次のように機能します。

- `Ctrl+T` を押してタスクリストビューを切り替えます。表示は一度に最大 5 つのタスクを表示します。Claude がまだチェックリストアイテムを作成していない場合、表示するものがないため、切り替えは目に見える効果がありません
- リストを展開したままにしておくと、Claude Code は `--resume` や `--continue` などでタスクがまだある セッションを起動する次回、展開されたビューを復元します。タスクリストが空の場合、Claude Code はそれを折りたたまれた状態で開始します
- すべてのタスクを表示したり、クリアしたりするには、Claude に直接尋ねてください。「すべてのタスクを表示して」または「すべてのタスクをクリアして」
- タスクはコンテキストコンパクション全体で保持され、Claude がより大規模なプロジェクトで整理された状態を保つのに役立ちます
- タスクリストをセッション間で共有するには、`CLAUDE_CODE_TASK_LIST_ID` を `~/.claude/tasks/` 内の名前付きディレクトリを使用するように設定します。`CLAUDE_CODE_TASK_LIST_ID=my-project claude`

6.14 セッションの要約

ターミナルから離れた後に戻ると、Claude Code はこれまでのセッションで何が起きたかの 1 行の要約を表示します。この要約は、最後に完了したターンから少なくとも 3 分が経過し、ターミナルがフォーカスされていない状態で、バックグラウンドで生成されるため、戻ってきたときに準備ができています。要約はセッションに少なくとも 3 ターンがある場合にのみ表示され、連続して 2 回表示されることはありません。

`/recap` を実行して、オンデマンドで要約を生成します。これはユーザー自身が要求した場合にのみ実行されます。Slack、Teams、またはプロジェクトのスレッドから中継されたメッセージや、ルーティンが送信したプロンプトに含まれていた場合は、要約の代わりに通知が表示されます。

セッションの要約はすべてのプランとプロバイダーでデフォルトでオンになっています。自動要約をオフにするには、`/config` を開いて **Session recap** をオフにします。自動要約は非対話モードでは表示されません。Claude Code は自動要約と `/recap` 出力の両方を 400 文字に制限しています。

6.15 使用制限がリセットされるまで待機する

claude.ai の使用制限が Claude Code の途中でタスクを停止した場合、Claude Code はオープンセッションで待機し、制限がリセットされた後に自動的にタスクを続行します。自動続行は、claude.ai サブスクリプ

ションでサインインしたインタラクティブセッションではデフォルトで有効です。Claude Code v2.1.234 以降が必要です。

Claude Code が待機している間、セッションの下部に、制限がリセットされる時刻と Claude が続行する時刻を示す行が表示されます。

```
Usage limit reached · limit resets 3:45pm
Continuing automatically at 3:45pm · esc to cancel
```

いずれの行にも、これらの文言の後に追加の内容が続く場合があります。たとえば、1 行目にはヘルプリンク、2 行目には `/usage-credits to continue now` が表示されることがあります。待機が自動的に開始された場合は、会話にも `Usage limit reached · continuing automatically at 3:45pm · esc to cancel` という行が記録されます。

セッションをオープンのままにしておいてください。その後の動作は、待機の終わり方によって異なります。

- **リセット時**：2 行目が `Continuing shortly · esc to cancel` に変わります。その後、会話に `Usage limit reset · continuing automatically` が表示され、Claude Code は Claude に停止した場所からタスクを再開するよう促します。最後のメッセージは再送信されません。
- **コンピューターがスリープした後**：約 30 分以上スリープし、スリープ中に制限がリセットされた場合、1 行目は `Your usage limit has reset`、2 行目は `Press enter to continue` と表示されます。`Enter` キーを押して続行してください。スリープが短かった場合や、リセット前にスリープが終了した場合は、Claude Code が自動的に続行します。
- **早期**：待機中に `/usage-credits` で**使用クレジットを追加** (ページ 914) したり、`/upgrade` の後に再度サインインしたり、`/model` でモデルを切り替えたりした場合、Claude Code は使用可能な使用量があるかどうかを確認し、ある場合はすぐに続行します。ブラウザで自分で行ったアップグレードまたは購入の後は確認されません。*`opusplan`* (ページ 742) および他のモデル設定で異なるモデルでプランモードを実行する場合、Claude Code はリセットを待機します。

続行されたタスクは他のターンと同様に実行されます。Claude Code は通常通り**権限** (ページ 1039) を要求するため、タスクはあなたが不在の間にプロンプトで停止する可能性があります。再度制限に達した場合、Claude Code は最大 2 回連続で待機を再度有効にしてから停止し、`Automatic continue stopped after repeated usage-limit hits · /rate-limit-options to try again` と表示します。

6.15.1 待機をキャンセルする

行が表示されている間に、空のプロンプトで `Esc` キーを押すか `Ctrl+C` を押すか、`/rate-limit-options` を実行して **Don't continue automatically** を選択してください。Claude Code は `Automatic continue cancelled` で始まる行で確認します。

キャンセル後、プロンプトを送信するか、`/rate-limit-options` から **Wait here, then continue automatically** で始まる行を再度選択するまで、何も続行されません。Claude Code はそのリセットウィンドウに対して自動的に待機を開始しません。次のリセットウィンドウは新たに開始されます。

待機は以下の場合にもタスクを続行せずに終了します。

- **プロンプトを送信する**：Claude Code は待機する代わりにプロンプトを送信します。そのプロンプトも制限に達した場合、プロンプトは会話に残り、Claude Code は再び待機を開始します。
- **Claude Code を終了する**：セッションを再開しても待機は再開されません。

- **会話の所有者が変わる**：`/login` でアカウントを切り替えたり、会話をクリアまたはリワインドしたり、`/resume` で別のセッションを再開したり、`/teleport` で取得したり、`/tui` で再起動したり、セッションを Claude Desktop、バックグラウンドセッション、またはクラウドに渡したりします。
- **設定がオフになるか、リセットが 24 時間を超える**：これは Claude Code が自動的に開始した待機のみを終了します。`/rate-limit-options` から選択した待機はカウントダウンを続けます。
- **続行がブロックされる**：続行プロンプトをブロックする *UserPromptSubmit* **フック** (ページ 487)、またはモデルに到達する前の失敗により、待機が終了します。Claude Code は続行が実行されなかったことを通知します。プロンプトを送信して続行してください。

6.15.2 自分で待機を開始する

Claude Code は以下の場合に自動的に待機を開始しません。

- **リモートコントロールとエージェントチームメイトセッション**：そのターミナルにいる人は依然として開始できます。
- **24 時間以上先のリセット**：週単位の制限は数日先にリセットされる可能性があります。
- **そのファミリー外のモデルを実行している間の Opus または Sonnet 制限**：次のターン時にその制限に達しない可能性があります。*opusplan* (ページ 742) および制限されたファミリーでプランモードを実行する他のモデル設定はこの例外を取得しません。

これらの場合、および自動続行がオフの場合、Claude Code は自分のターミナルで制限に達したときに、リセットウィンドウごとに 1 回使用制限オプションメニューを開きます。**Wait here, then continue automatically** で始まる行を選択して待機を開始してください。リモートコントロールまたはエージェントチームメイトセッションでは、自分で `/rate-limit-options` を実行してメニューを開いてください。

Claude Code は以下の場合に待機をまったく提供しません。

- **バックグラウンドセッションと `-p` 実行**：メニュー行は利用できません。
- **API キー、クラウドプロバイダー、および使用量ベースの課金**：そこでの使用はリクエストごとにメーター化されるため、待機するリセットはありません。
- **保存された claude.ai ログインのない** *LLM* **ゲートウェイ** (ページ 842)：Claude Code は保存された claude.ai ログインがアクティブな認証情報である場合にのみ待機を提供します。

6.15.3 自動続行をオフにする

`/config` で **Continue automatically at usage limit** をオフにするか、ユーザー設定で `autoContinueAtUsageLimit` を `false` に設定してください。`/config autoContinueAtUsageLimit=false` も機能します（`-p` を含む）が、`key=value` 形式は設定が無人実行を許可するため、オンに戻すことはできません。Claude Code がこのキーを読み込むどの設定ファイルかは、設定リファレンスに記載されています。

6.16 PR レビューステータス

オープンなプルリクエストを持つブランチで作業している場合、Claude Code はフッターに「PR #446」などのクリック可能な PR リンクを表示します。リンクには、レビュー状態を示す色付きの下線があります：

- 緑：承認済み

- 黄：レビュー待ち
- 赤：変更をリクエスト
- グレー：ドラフト

プルリクエストがマージまたはクローズされると、バッジは消えます。

Cmd+click（macOS）または Ctrl+click（Windows/Linux）でリンクをクリックして、プルリクエストをブラウザで開きます。

ステータスは、git push または gh pr create や gh pr merge などのプルリクエストを変更する gh pr コマンドがセッションで成功するとすぐに更新されます。

Claude Code は、SSH または tmux 経由でよく発生するターミナルでハイパーリンクサポートを検出できない場合でも、バッジをハイパーリンクとしてレンダリングします。FORCE_HYPERLINK=0 を設定して、バッジをプレーンテキストとしてレンダリングします。

CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC を設定すると、Claude Code はプルリクエストまたはマージリクエストのステータスをチェックしません。

注釈:

GitHub リポジトリの PR ステータスには GitHub トークンが必要です。Claude Code はリモートのホストに基づいてトークンを検出します：

- **github.com**：GH_TOKEN または GITHUB_TOKEN、または gh auth login で保存されたトークン。トークンがない場合、フッターは gh CLI がインストールされていないときは install gh for PR status を表示し、インストールされているときは gh auth login for PR status を表示します
- **GH_HOST として設定された GitHub Enterprise ホスト**：GH_ENTERPRISE_TOKEN または GITHUB_ENTERPRISE_TOKEN、または gh auth login --hostname <host> で保存されたトークン。トークンがない場合、フッターは同じヒントを表示します
- **その他の GitHub ホスト**：gh auth login --hostname <host> で保存されたトークン。トークンがない場合、Claude Code はバッジもヒントも表示しません

6.16.1 GitLab マージリクエスト

オープンな GitLab マージリクエストを持つブランチで作業している場合、Claude Code はフッタースロットに MR !N バッジを表示します。このスロットは通常 GitHub PR リンクを保持しています。!N は GitLab 独自のマージリクエスト番号 N の参照構文です。色付きの下線はマージリクエストの状態を示します：

- 緑：GitLab がマージリクエストをマージ可能として報告
- 黄：その他のオープン状態
- グレー：ドラフト

マージリクエストがマージまたはクローズされると、バッジは消えます。

git push または glab mr create や glab mr merge などのマージリクエストを変更する glab mr コマンドがセッションで成功するとすぐに更新されます。

バッジを取得するには、以下が必要です：

- Claude Code v2.1.234 以降
- gitlab.com またはセルフマネージドインスタンスのいずれかを指す、GitLab ホストへのリポジトリリモート
- `PATH` 上の glab CLI。`glab auth login` で認証済み

Claude Code はステータスをチェックするときに `glab` のトークン環境変数（`GITLAB_TOKEN` など）を無視するため、エクスポートされたトークンだけではバッジが取得されません。Claude Code はまた、セッションごとに 1 回 `glab` とそのログインを探すため、`glab` をインストールした後または `glab auth login` を実行した後に Claude Code を再起動してください。

6.17 Issue 参照リンク

Claude が `owner/repo#123` という形式で issue を言及する場合、ターミナルがハイパーリンクをサポートしていれば、その参照をクリックして開くことができます。Claude Code がターミナルのハイパーリンクサポートを検出できない場合は、`FORCE_HYPERLINK` を `1` に設定してリンクをオンにするか、`0` に設定して参照をプレーンテキストのままにしておくことができます。

リンクは 2 部構成の `owner/repo#123` 形式でのみ取得できます。以下は プレーンテキストのままです：

- 単独の `#123`
- `group/subgroup/project#123` などのネストされた GitLab パス
- コードスパンまたはコードブロック内の任意の参照

Claude Code は、参照が指すリポジトリではなく、git remote から識別したリポジトリのホストに対してリンクを構築します：

リポジトリのホスト	`owner/repo#123` のリンク先
github.com、GitHub Enterprise ホスト、または以下にリストされていないホスト	`https://<host>/owner/repo/issues/123`
gitlab.com	`https://gitlab.com/owner/repo/-/issues/123`
bitbucket.org、codeberg.org、または gitea.com	リンクなし；参照はプレーンテキストのままです

6.18 関連項目

- **スキル** (ページ 275) - カスタムプロンプトとワークフロー
- **チェックポイント** (ページ 1019) - Claude の編集を巻き戻し、前の状態を復元
- *CLI* **リファレンス** (ページ 45) - コマンドラインフラグとオプション
- **設定** (ページ 711) - 設定オプション
- **メモリ管理** (ページ 997) - CLAUDE.md ファイルの管理

第7章　一般的なワークフロー

Claude Code を使用してコードベースの探索、バグ修正、リファクタリング、テスト、その他の日常的なタスクを実行するためのステップバイステップガイド。

このページでは、日常的な開発のための短いレシピを集めています。プロンプティングとコンテキスト管理に関する高度なガイダンスについては、**ベストプラクティス** (ページ 115) を参照してください。

このページでは以下をカバーしています：

- **プロンプトレシピ** (ページ 105)：コード探索、バグ修正、リファクタリング、テスト、PR、ドキュメント用
- **以前の会話を再開する** (ページ 113)：タスクを複数回に分けて実行できるようにする
- ***worktree* を使用して並列セッションを実行する** (ページ 113)：同時編集が衝突しないようにする
- **編集前に計画する** (ページ 113)：ディスクに変更を加える前に確認する
- **研究を *subagent* に委譲する** (ページ 114)：メインコンテキストをクリーンに保つ
- ***Claude* をスクリプトにパイプする** (ページ 114)：CI とバッチ処理用

7.1 プロンプトレシピ

これらは、未知のコード探索、デバッグ、リファクタリング、テスト作成、PR 作成などの日常的なタスク用のプロンプトパターンです。各パターンは任意の Claude Code サーフェスで機能します。プロジェクトに合わせて表現を調整してください。

7.1.1 新しいコードベースを理解する

モノレポまたは大規模なコードベースで Claude Code を設定する場合は、モノレポと大規模リポジトリを参照してください。

コードベースの概要を素早く把握する

新しいプロジェクトに参加したばかりで、その構造を素早く理解する必要があるとします。

1. **プロジェクトルートディレクトリに移動する**

   ```
   cd /path/to/project
   ```

   `/path/to/project` をプロジェクトへのパスに置き換えてください。

2. **Claude Code を起動する**

```
claude
```

3. **高レベルの概要をリクエストする**

```
give me an overview of this codebase
```

4. **特定のコンポーネントについてさらに詳しく調べる**

```
explain the main architecture patterns used here
```

```
what are the key data models?
```

```
how is authentication handled?
```

Tip:

ヒント：

- 広い質問から始めて、特定の領域に絞り込んでいく
- プロジェクトで使用されているコーディング規約とパターンについて質問する
- プロジェクト固有の用語の用語集をリクエストする

関連するコードを見つける

特定の機能または機能に関連するコードを見つける必要があるとします。

1. **Claude に関連ファイルを見つけるよう依頼する**

```
find the files that handle user authentication
```

2. **コンポーネントがどのように相互作用するかについてのコンテキストを取得する**

```
how do these authentication files work together?
```

3. **実行フローを理解する**

```
trace the login process from front-end to database
```

Tip:

ヒント：

- 探しているものについて具体的に説明する
- プロジェクトのドメイン言語を使用する
- 言語のコード インテリジェンス プラグインをインストールして、Claude に正確な「定義に移動」と「参照を検索」のナビゲーションを提供する

7.1.2 バグを効率的に修正する

エラーメッセージが表示され、そのソースを見つけて修正する必要があるとします。

1. **Claude とエラーを共有する**

```
I'm seeing an error when I run npm test
```

2. **修正の推奨事項をリクエストする**

```
suggest a few ways to fix the @ts-ignore in user.ts
```

3. **修正を適用する**

```
update user.ts to add the null check you suggested
```

Tip:

ヒント：

- Claude に問題を再現するコマンドとスタックトレースを伝える
- エラーを再現するための手順を記載する
- エラーが断続的か一貫しているかを Claude に知らせる

7.1.3 コードをリファクタリングする

古いコードを最新のパターンとプラクティスを使用するように更新する必要があるとします。

コードベース全体を新しい言語に移植する場合は、ブログの Anthropic が Claude Code で大規模なコード移行を実行する方法を参照してください。

1. **リファクタリング対象のレガシーコードを特定する**

```
find deprecated API usage in our codebase
```

2. **リファクタリングの推奨事項を取得する**

```
suggest how to refactor utils.js to use modern JavaScript features
```

3. **変更を安全に適用する**

```
refactor utils.js to use ES2024 features while maintaining the same behavior
```

4. **リファクタリングを検証する**

```
run tests for the refactored code
```

Tip:

ヒント：

- Claude に最新のアプローチの利点を説明するよう依頼する
- 必要に応じて変更が後方互換性を維持することをリクエストする
- リファクタリングを小さくテスト可能な増分で実行する

7.1.4 テストを使用する

カバーされていないコードのテストを追加する必要があるとします。

1. **テストされていないコードを特定する**

```
find functions in NotificationsService.swift that are not covered by tests
```

2. **テストスキャフォルディングを生成する**

```
add tests for the notification service
```

3. **意味のあるテストケースを追加する**

```
add test cases for edge conditions in the notification service
```

4. **テストを実行して検証する**

```
run the new tests and fix any failures
```

Claude は、プロジェクトの既存のパターンと規約に従うテストを生成できます。テストをリクエストするときは、検証したい動作について具体的に説明してください。Claude は既存のテストファイルを調べて、既に使用されているスタイル、フレームワーク、アサーションパターンに一致させます。

包括的なカバレッジのために、Claude に見落とした可能性のあるエッジケースを特定するよう依頼してください。Claude はコードパスを分析し、エラー条件、境界値、見落としやすい予期しない入力のテストを提案できます。

7.1.5 プルリクエストを作成する

Claude に直接プルリクエストを作成するよう依頼するか（「create a pr for my changes」）、ステップバイステップで Claude をガイドできます：

1. **変更内容を要約する**

```
summarize the changes I've made to the authentication module
```

2. **プルリクエストを生成する**

```
create a pr
```

3. **レビューと改善**

```
enhance the PR description with more context about the security improvements
```

後でセッションを見つけるには、`claude --from-pr 1234` を実行してください。1234 を PR 番号に置き換えてください。これにより、その PR にリンクされたセッションにフィルタリングされたセッションピッカーが開きます。または、/resume ピッカーの検索に PR URL を貼り付けてください。Claude が `gh pr create` または `glab mr create` でプルリクエストを作成するとき、および Claude が既存の PR で作業するときに、Claude Code はセッションを PR にリンクします。

Tip:

Claude が生成した PR を送信する前にレビューし、Claude に潜在的なリスクや考慮事項を強調するよう依頼してください。

7.1.6 ドキュメントを処理する

コードのドキュメントを追加または更新する必要があるとします。

1. **ドキュメント化されていないコードを特定する**

```
find functions without proper JSDoc comments in the auth module
```

2. **ドキュメントを生成する**

```
add JSDoc comments to the undocumented functions in auth.js
```

3. **レビューと改善**

```
improve the generated documentation with more context and examples
```

4. **ドキュメントを検証する**

```
check if the documentation follows our project standards
```

Tip:

ヒント：

- 必要なドキュメントスタイル（JSDoc、docstring など）を指定する
- ドキュメント内の例をリクエストする
- パブリック API、インターフェース、複雑なロジックのドキュメントをリクエストする

7.1.7 ノートと非コードフォルダで作業する

Claude Code はどのディレクトリでも機能します。ノートボルト、ドキュメントフォルダ、またはマークダウンファイルの任意のコレクション内で実行して、コードと同じ方法でコンテンツを検索、編集、再編成します。

`.claude/` ディレクトリと `CLAUDE.md` は他のツールの設定ディレクトリと並んで競合なく存在します。Claude は各ツール呼び出しで新しくファイルを読み込むため、別のアプリケーションで行った編集は次回そのファイルを読み込むときに表示されます。

7.1.8 画像を使用する

コードベース内の画像を使用する必要があり、Claude の画像コンテンツ分析を支援したいとします。

1. **会話に画像を追加する**

   次のいずれかの方法を使用できます：

   1. Claude Code ウィンドウに画像をドラッグアンドドロップする
   2. 画像をコピーして、CLI に `Ctrl+V` で貼り付ける。または *Alt+V を Windows と WSL で* (ページ 75) 使用する
   3. Claude に画像パスを提供する。例：「Analyze this image: /path/to/your/image.png」

2. **Claude に画像を分析するよう依頼する**

```
What does this image show?
```

```
Describe the UI elements in this screenshot
```

```
Are there any problematic elements in this diagram?
```

3. **コンテキストに画像を使用する**

```
Here's a screenshot of the error. What's causing it?
```

```
This is our current database schema. How should we modify it for the new feature?
```

4. **ビジュアルコンテンツからコード提案を取得する**

```
Generate CSS to match this design mockup
```

```
What HTML structure would recreate this component?
```

Tip:

ヒント：

- テキスト説明が不明確または面倒な場合は画像を使用する
- より良いコンテキストのために、エラー、UI デザイン、図のスクリーンショットを含める
- 会話で複数の画像を使用できます
- 画像分析は図、スクリーンショット、モックアップなどで機能します
- Claude が画像を参照する場合（例：`[Image #1]`）、`Cmd+Click`（Mac）または `Ctrl+Click`（Windows/Linux）リンクをクリックして、デフォルトビューアで画像を開きます

7.1.9 ファイルとディレクトリを参照する

@ を使用して、Claude に読み込まれるのを待たずにファイルまたはディレクトリをすばやく含めます。

1. **単一ファイルを参照する**

```
Explain the logic in @src/utils/auth.js
```

これにより、ファイルの完全な内容が会話に含まれます。

2. **ディレクトリを参照する**

```
What's the structure of @src/components?
```

3. **MCP リソースを参照する**

```
Show me the data from @github:repos/owner/repo/issues
```

これにより、@server:resource 形式を使用して接続された MCP サーバーからデータを取得します。詳細については、*MCP* **リソース** (ページ 405) を参照してください。

Tip:

ヒント：

- ファイルパスは相対パスまたは絶対パスにできます
- @ を入力してパス提案メニューを開き、Enter または Tab キーを押してハイライトされたパスを受け入れ、もう一度 Enter キーを押してメッセージを送信します
- @ ファイル参照は、ファイルのディレクトリと親ディレクトリに `CLAUDE.md` を追加してコンテキストに含めます
- ディレクトリ参照はコンテンツではなくファイルリストを表示します
- 単一のメッセージで複数のファイルを参照できます（例：「@file1.js and @file2.js」）

7.1.10 スケジュールで Claude を実行する

Claude に長時間実行されるタスクを自動的に定期的に処理させたいとします。例えば、毎朝オープン PR をレビューしたり、毎週依存関係を監査したり、夜間に CI の失敗をチェックしたりします。

実行場所に基づいてスケジューリングオプションを選択します：

オプション	実行場所	最適な用途
ルーチン	クラウド、Anthropic 管理（デフォルト）	コンピュータがオフの場合でも実行する必要があるタスク。claude.ai/code/routines で設定します。スケジュールに加えて API 呼び出しまたは GitHub イベントでトリガーすることもできます。

次のページに続く

表 1 – 前のページからの続き

オプション	実行場所	最適な用途
デスクトップスケジュール済みタスク	デスクトップアプリ経由のマシン	ローカルファイル、ツール、またはコミットされていない変更への直接アクセスが必要なタスク。
GitHub Actions (ページ 247)	CI パイプライン	オープン PR などのリポジトリイベント、またはワークフロー設定と一緒に存在する必要がある cron スケジュールに関連するタスク。
`/loop`	現在の CLI セッション	セッションが開いている間のクイックポーリング。`--resume` と `--continue` は期限切れでないものを復元します。

Tip:

スケジュール済みタスク用のプロンプトを作成するときは、成功がどのように見えるか、および結果をどうするかについて明示的に説明してください。タスクは自律的に実行されるため、質問を明確にすることはできません。例えば：「`needs-review` ラベルが付いたオープン PR をレビューし、問題に関するインラインコメントを残し、`#eng-reviews` Slack チャネルに要約を投稿します。」

7.1.11 Claude にその機能について質問する

Claude は自分のドキュメントへの組み込みアクセスを持っており、自分の機能と制限について質問に答えることができます。

質問例

```
can Claude Code create pull requests?
```

```
how does Claude Code handle permissions?
```

```
what skills are available?
```

```
how do I use MCP with Claude Code?
```

```
how do I configure Claude Code for Amazon Bedrock?
```

```
what are the limitations of Claude Code?
```

注釈:

Claude はこれらの質問に対してドキュメントベースの回答を提供します。実行可能な例と実践的なデモンストレーションについては、`/powerup` を実行してアニメーション化されたデモを含むインタラクティブレッスンを受けるか、上記の特定のワークフローセクションを参照してください。

Tip:

ヒント：

- Claude は使用しているバージョンに関係なく、常に最新の Claude Code ドキュメントにアクセスできます
- 詳細な回答を得るために具体的な質問をする
- Claude は MCP 統合、エンタープライズ設定、高度なワークフローなどの複雑な機能を説明できます

7.2 以前の会話を再開する

タスクが複数回に分けて実行される場合は、コンテキストを再度説明するのではなく、中断したところから再開してください。Claude Code はすべての会話をローカルに保存します。

```
claude --continue
```

これにより、現在のディレクトリで最新のセッションが再開されます。まだセッションがない場合は、`No conversation found to continue` と出力して終了します。`claude --resume` を使用してリストから選択するか、実行中のセッション内から `/resume` を使用してください。セッションを管理するを参照して、名前付け、ブランチ、および完全なピッカーリファレンスを確認してください。

7.3 worktree を使用して並列セッションを実行する

1 つのターミナルで機能に取り組みながら、別のターミナルで Claude がバグを修正するようにします。編集が衝突しないようにしてください。各 git worktree は既存のコミットから作成された独自のブランチ上の個別のチェックアウトです。そのため、リポジトリには最初に少なくとも 1 つのコミットが必要です。

```
claude --worktree feature-auth
```

別の名前で 2 番目のターミナルで同じコマンドを実行して、分離された並列セッションを開始します。コミットがないリポジトリでは、コマンドは `Failed to resolve base branch "HEAD": git rev-parse failed` というエラーで失敗します。クリーンアップ、`.worktreeinclude`、および非 git VCS サポートについては、Worktrees を参照してください。1 つの画面から並列セッションを監視するには、別のターミナルではなく、バックグラウンドエージェント を参照してください。

7.4 編集前に計画する

ディスクに変更を加える前に確認したい変更については、計画モードに切り替えてください。Claude はファイルを読み込んで計画を提案しますが、承認されるまで編集は行いません。ステータスバーは計画モードがアクティブな間、☒ `plan mode on` と表示されます。

```
claude --permission-mode plan
```

セッション中に `Shift+Tab` を押してステータスバーが ☒ `plan mode on` と表示されるまで切り替えることもできます。承認フローとテキストエディタで計画を編集する方法については、計画モードを参照してください。

7.5 研究を subagent に委譲する

大規模なコードベースの探索はコンテキストをファイル読み込みで満たします。探索を委譲して、結果のみが戻るようにしてください。

```
use a subagent to investigate how our auth system handles token refresh
```

subagent は独自のコンテキストウィンドウでファイルを読み込み、要約を報告します。*Subagents* (ページ 317) を参照して、独自のツールとプロンプトを持つカスタムエージェントを定義することを確認してください。

7.6 Claude をスクリプトにパイプする

CI、プリコミットフック、またはバッチ処理用に Claude を非対話的に実行します。stdin と stdout は任意の Unix ツールのように機能します。

```
git log --oneline -20 | claude -p "summarize these recent commits"
```

非対話モード (ページ 695) を参照して、出力形式、許可フラグ、およびファンアウトパターンを確認してください。

7.7 次のステップ

- **ベストプラクティス** (ページ 115)

  Claude Code から最大限の価値を得るためのパターン

- セッションを管理する

  会話を再開、名前付け、ブランチする

- Worktrees

  分離された並列セッションを実行する

- Claude Code を拡張する

  skill、フック、MCP、subagent、プラグインを追加する

第8章　Claude Code のベストプラクティス

> 環境設定から並列セッションでのスケーリングまで、Claude Code を最大限に活用するためのヒントとパターン。

Claude Code は agentic coding 環境です。質問に答えて待つチャットボットとは異なり、Claude Code はファイルを読み取り、コマンドを実行し、変更を加え、あなたが見守ったり、方向を変えたり、完全に任せたりしながら、自律的に問題を解決できます。

これはあなたの作業方法を変えます。自分でコードを書いて Claude にレビューしてもらう代わりに、やりたいことを説明すると Claude がそれをどのように構築するかを考え出します。Claude は探索し、計画し、実装します。

しかし、この自律性にも学習曲線があります。Claude は理解する必要がある特定の制約の中で動作します。

このガイドでは、Anthropic の内部チームと、様々なコードベース、言語、環境で Claude Code を使用しているエンジニアの間で効果的であることが証明されたパターンについて説明します。agentic ループがどのように機能するかについては、Claude Code の仕組みを参照してください。

ほとんどのベストプラクティスは 1 つの制約に基づいています。Claude のコンテキストウィンドウはすぐにいっぱいになり、満杯になるにつれてパフォーマンスが低下します。

Claude のコンテキストウィンドウは、すべてのメッセージ、Claude が読み取ったすべてのファイル、およびすべてのコマンド出力を含む、会話全体を保持します。ただし、これはすぐにいっぱいになる可能性があります。単一のデバッグセッションまたはコードベース探索でも、数万のトークンを生成および消費する可能性があります。

LLM のパフォーマンスはコンテキストが満杯になるにつれて低下するため、これは重要です。コンテキストウィンドウがいっぱいになると、Claude は以前の指示を「忘れる」か、より多くの間違いを犯す可能性があります。コンテキストウィンドウは管理する最も重要なリソースです。セッションがどのように満杯になるかを実際に確認するには、スタートアップで何が読み込まれるか、各ファイル読み取りのコストについてのインタラクティブなウォークスルーを参照してください。**カスタムステータスライン** (ページ 785) でコンテキスト使用量を継続的に追跡し、トークン使用量を削減するための戦略については**トークン使用量を削減** (ページ 919) を参照してください。

8.1 Claude に自分の作業を検証する方法を与える

Tip:

Claude が実行できるチェックを与えてください。テスト、ビルド、比較するスクリーンショットです。これは、あなたが見守るセッションと、あなたが立ち去ることができるセッションの違いです。

Claude は、作業が完了したように見えるときに停止します。実行できるチェックがないと、「完了したように見える」が唯一の利用可能なシグナルであり、あなたが検証ループになります。すべての間違いはあなたがそれに気付くのを待ちます。Claude が実行できるパスまたはフェイルを生成するものを与えると、ループは自動的に閉じます。Claude は作業を行い、チェックを実行し、結果を読み、チェックが合格するまで反復します。

チェックは、会話で Claude が読むことができるシグナルを返すものです。テストスイート、ビルド終了コード、リンター、出力を固定値と比較するスクリプト、またはデザインと比較される**ブラウザスクリーンショット** (ページ 213) です。*/verify* (ページ 276) を実行してください。Claude のチェックが合格した後、実行中のアプリに対する変更を確認します。

戦略	前	後
検証基準を提供する	**「メールアドレスを検証する関数を実装する」**	**「*validateEmail* 関数を書く。テストケースの例：user@example.com は *true*、*invalid* は *false*、user@.com は *false*。実装後にテストを実行する」**
UI の変更を視覚的に検証する	**「ダッシュボードをより良く見えるようにする」**	**「*[スクリーンショットを貼り付け]* このデザインを実装する。結果のスクリーンショットを撮り、元のものと比較する。違いをリストアップして修正する」**
症状ではなく根本原因に対処する	**「ビルドが失敗している」**	**「ビルドがこのエラーで失敗している：*[エラーを貼り付け]*。修正して、ビルドが成功することを確認する。根本原因に対処し、エラーを抑制しない」**

チェックが存在したら、停止をどの程度厳しくゲートするかを決定します。

- **1 つのプロンプトで**：Claude にチェックを実行し、同じメッセージ内で反復するよう求めます。上記の表のように。
- **セッション全体で**：チェックを/goal 条件として設定します。別の評価者がすべてのターンの後に再度チェックし、Claude はそれが保持されるまで作業を続けます。Claude が停止した場合、Claude Code は最終的にゴールがまだ設定されたままでランを停止します。/goal 評価がどのように機能するかを参照してください。
- **決定論的ゲートとして**：*Stop hook* (ページ 526) がスクリプトとしてチェックを実行し、合格するまでターンが終了するのをブロックします。*Stop input* (ページ 526) は連続ブロックの上限をカバーします。
- **第二の意見によって**：**検証サブエージェント** (ページ 317) または動的ワークフローが独自の調査結果をチェックし、新しいモデルが結果を反論しようとするため、作業を行っているエージェントがそれを採点しているわけではありません。

各ステップは、セットアップと注意をトレードオフします。プロンプト版は今日、任意のタスクで機能します。`/goal` と Stop hook 版は、あなたなしで無人実行が正しく完了できるようにするものです。

Claude に成功を主張するのではなく、証拠を示すよう指示してください。テスト出力、実行したコマンドとその戻り値、または結果のスクリーンショットです。証拠を確認することは、検証を自分で再実行する

よりも高速であり、監視していなかったセッションでも機能します。

8.2 最初に探索し、次に計画し、その後コーディングする

Tip:

研究と計画を実装から分離して、間違った問題を解決することを避けます。

Claude が直接コーディングにジャンプさせると、間違った問題を解決するコードが生成される可能性があります。Plan Mode を使用して、探索を実行から分離します。

推奨されるワークフローには 4 つのフェーズがあります。

1. **探索**

   Shift+Tab を押してステータスバーに ☒ plan mode on と表示されるまで Plan Mode に入るか、claude --permission-mode plan でセッションを開始します。Claude はファイルを読み取り、変更を加えずに質問に答えます。

   リスト 1: claude (plan mode)

   ```
   read /src/auth and understand how we handle sessions and login.
   also look at how we manage environment variables for secrets.
   ```

2. **計画**

   Claude に詳細な実装計画を作成するよう依頼します。

   リスト 2: claude (plan mode)

   ```
   I want to add Google OAuth. What files need to change?
   What's the session flow? Create a plan.
   ```

   Ctrl+G を押して、Claude が進む前に、テキストエディタで計画を開いて直接編集します。

3. **実装**

   計画を承認するか Shift+Tab を押して Plan Mode を終了し、Claude にコーディングさせ、計画に対して検証します。

   リスト 3: claude

   ```
   implement the OAuth flow from your plan. write tests for the
   callback handler, run the test suite and fix any failures.
   ```

4. **コミット**

   Claude に説明的なメッセージでコミットし、PR を作成するよう依頼します。

リスト 4: claude

```
commit with a descriptive message and open a PR
```

注釈:

Plan Mode は便利ですが、オーバーヘッドも追加します。
スコープが明確で修正が小さいタスク（タイプミスの修正、ログ行の追加、変数の名前変更など）の場合は、Claude に直接実行するよう依頼します。
計画は、アプローチについて不確実な場合、変更が複数のファイルを変更する場合、または変更されるコードに不慣れな場合に最も役立ちます。差分を 1 文で説明できる場合は、計画をスキップします。

8.3 プロンプトで具体的なコンテキストを提供する

Tip:

指示がより正確であるほど、必要な修正が少なくなります。

Claude は意図を推測できますが、あなたの心を読むことはできません。特定のファイルを参照し、制約を述べ、例のパターンを指摘します。

戦略	前	後
タスクをスコープする。 どのファイル、どのシナリオ、テスト設定を指定します。	**「*foo.py* のテストを追加する」**	**「ユーザーがログアウトしているエッジケースをカバーする *foo.py* のテストを書く。モックを避ける。」**
ソースを指摘する。 Claude を質問に答えることができるソースに向けます。	**「*ExecutionFactory* がこんなに奇妙な *API* を持っているのはなぜですか？」**	**「*ExecutionFactory* の *git* 履歴を調べて、その *API* がどのようになったかを要約する」**
既存のパターンを参照する。 Claude をコードベースのパターンに向けます。	**「カレンダーウィジェットを追加する」**	**「ホームページで既存のウィジェットがどのように実装されているかを見て、パターンを理解する。*HotDogWidget.php* は良い例です。パターンに従って、ユーザーが月を選択し、前後にページネーションして年を選択できる新しいカレンダーウィジェットを実装する。コードベースで既に使用されているもの以外のライブラリを使用せずにゼロから構築する。」**

次のページに続く

表 2 – 前のページからの続き

戦略	前	後
症状を説明する。 症状、可能性のある場所、「修正」の外観を提供します。	**「ログインバグを修正する」**	**「ユーザーはセッションタイムアウト後にログインが失敗すると報告しています。*src/auth/* の認証フロー、特にトークン更新を確認します。問題を再現する失敗するテストを書き、修正する」**

曖昧なプロンプトは、探索していて方向転換を余裕を持ってできるときに役立つことがあります。「このファイルで何を改善しますか？」のようなプロンプトは、あなたが尋ねることを考えなかったことを表面化させることができます。

8.3.1 リッチコンテンツを提供する

Tip:

`@` を使用してファイルを参照したり、スクリーンショット/画像を貼り付けたり、データを直接パイプしたりします。

Claude にリッチデータを提供するにはいくつかの方法があります。

- **`@` でファイルを参照する** コードがどこにあるかを説明する代わりに。Claude は応答する前にファイルを読み取ります。
- **画像を直接貼り付ける**。画像をコピー/貼り付けまたはドラッグアンドドロップしてプロンプトに入れます。
- **ドキュメントと API リファレンスの URL を指定する**。`/permissions` を使用して、頻繁に使用されるドメインをホワイトリストに登録します。
- **データをパイプする** `cat error.log | claude -p "explain this error"` を実行してファイルの内容を直接送信します。
- **Claude に必要なものを取得させる**。Bash コマンド、MCP ツール、またはファイルを読み取ることを使用して、Claude 自身がコンテキストをプルするよう指示します。

8.4 環境を設定する

いくつかのセットアップステップにより、Claude Code はすべてのセッション全体で大幅に効果的になります。拡張機能の完全な概要と各機能をいつ使用するかについては、Claude Code を拡張を参照してください。

8.4.1 効果的な CLAUDE.md を書く

Tip:

`/init` を実行して、現在のプロジェクト構造に基づいてスターター CLAUDE.md ファイルを生成し、時間をかけて改善します。

CLAUDE.md は Claude がすべての会話の開始時に読む特別なファイルです。Bash コマンド、コードスタイル、ワークフロールールを含めます。これにより、Claude はコードだけからは推測できない永続的なコンテキストを取得します。

CLAUDE.md ファイルに必須の形式はありませんが、短く人間が読める状態に保ちます。例えば：

リスト 5: CLAUDE.md

```
# Code style
- Use ES modules (import/export) syntax, not CommonJS (require)
- Destructure imports when possible (eg. import { foo } from 'bar')

# Workflow
- Be sure to typecheck when you're done making a series of code changes
- Prefer running single tests, and not the whole test suite, for performance
```

`/context` を実行して、Claude がファイルを読み込んだことを確認します。CLAUDE.md はすべてのセッションで読み込まれるため、広く適用されるもののみを含めます。ドメイン知識またはときどきのみ関連するワークフローについては、代わりに *skills* (ページ 275) を使用します。Claude はそれらをオンデマンドで読み込み、すべての会話を膨らませることなく使用します。

簡潔に保ちます。各行について、次のように尋ねます。**「これを削除すると *Claude* が間違いを犯しますか？」** そうでない場合は、削除します。膨らんだ CLAUDE.md ファイルは Claude があなたの実際の指示を無視するようにします。

☒ 含める	☒ 除外する
Claude が推測できない Bash コマンド	Claude がコードを読むことで理解できるもの
デフォルトと異なるコードスタイルルール	Claude が既に知っている標準言語規約
テスト指示と推奨テストランナー	詳細な API ドキュメント（代わりにドキュメントにリンク）
リポジトリのエチケット（ブランチ命名、PR 規約）	頻繁に変わる情報
プロジェクト固有のアーキテクチャ決定	長い説明またはチュートリアル
開発者環境の癖（必須環境変数）	ファイルごとのコードベースの説明
一般的な落とし穴または明白でない動作	「きれいなコードを書く」のような自明なプラクティス

Claude が CLAUDE.md にルールがあるにもかかわらず、あなたが望まないことをし続ける場合、ファイルはおそらく長すぎて、ルールが失われています。Claude が CLAUDE.md で答えられている質問をあなたに尋ねる場合、フレーズが曖昧かもしれません。CLAUDE.md をコードのように扱う：物事がうまくいかないときにレビューし、定期的に削除し、Claude の動作が実際に変わるかどうかを観察することで変更をテ

ストします。チェックインされた CLAUDE.md については、`/doctor` を実行すると、Claude はコードベースから導き出せるコンテンツの削除を提案します。

Claude が 1 つの指示をスキップし続ける場合は、その行だけに「IMPORTANT」などの強調を追加します。多くの行を強調する場合、どれも目立ちません。CLAUDE.md を git にチェックインして、チームが貢献できるようにします。ファイルは時間とともに価値が複合します。

CLAUDE.md ファイルは `@path/to/import` 構文を使用して追加ファイルをインポートできます。インポートルールと CLAUDE.md ファイルが存在できる場所については、*CLAUDE.md* **ファイル** (ページ 998) を参照してください。

8.4.2 権限を設定する

Tip:

制御を失わずにプロンプトを減らすには、`/permissions` で信頼するツールを事前承認し、`/sandbox` でサンドボックス化されたコマンドを質問なしで実行させます。自分でエディットとコマンドを承認したい場合は、Manual モードに切り替えます。

Claude Code v2.1.283 以降では、auto mode は対話型ターミナルと VS Code セッションの組み込みの開始権限モードです。別の分類器モデルがほとんどのアクションをレビューし、スコープエスカレーション、未知のインフラストラクチャ、敵対的なコンテンツ駆動のアクションなど、リスクがあるように見えるものだけをブロックします。以前のバージョンでは、auto mode は Pro、Max、Team プランでのみ対話型ターミナルと VS Code セッションの組み込みの開始権限モードです。

Manual モードでは、Claude Code はシステムを変更する可能性のあるアクション（ファイル書き込み、Bash コマンド、MCP ツール）の前に尋ねます。これは安全ですが、面倒です。10 回目の承認後、あなたはクリックしているだけで、本当にレビューしていません。Manual モードでこれらの中断を減らす 2 つのツールがあり、auto mode でも同様に適用されます。

- **権限ホワイトリスト**：`npm run lint` や `git commit` など、安全であることがわかっているツールを許可します
- **サンドボックス**：OS レベルの分離を有効にして、ファイルシステムとネットワークアクセスを制限し、Claude が定義された境界内でより自由に動作できるようにします

権限モード、**権限ルール** (ページ 1039)、**サンドボックス** (ページ 1069) の詳細をお読みください。

8.4.3 CLI ツールを使用する

Tip:

Claude Code に `gh`、`aws`、`gcloud`、`sentry-cli` などの CLI ツールを使用して外部サービスと対話するよう指示します。

CLI ツールは外部サービスと対話する最もコンテキスト効率的な方法です。GitHub を使用する場合は、`gh` CLI をインストールします。Claude は問題の作成、プルリクエストのオープン、コメントの読み取りにそ

れを使用する方法を知っています。gh がなければ、Claude は GitHub API を使用できますが、認証されていないリクエストはしばしばレート制限に達します。

Claude は、それが既に知らない CLI ツールを学ぶのにも効果的です。`Use 'foo-cli-tool --help' to learn about foo tool, then use it to solve A, B, C.` のようなプロンプトを試してください。

8.4.4 MCP サーバーを接続する

Tip:

`claude mcp add` をサーバー名と URL またはコマンドで実行して、Notion、Figma、またはデータベースなどの外部ツールを接続します。例えば：`claude mcp add --transport http notion https://mcp.notion.com/mcp`。

MCP **サーバー** (ページ 363) を使用すると、Claude に問題トラッカーから機能を実装したり、データベースをクエリしたり、監視データを分析したり、Figma からデザインを統合したり、ワークフローを自動化したりするよう依頼できます。

8.4.5 フックを設定する

Tip:

例外なしで毎回発生する必要があるアクションにはフックを使用します。

フック (ページ 411) は Claude のワークフロー内の特定のポイントで自動的にスクリプトを実行します。CLAUDE.md の指示とは異なり、フックは決定論的であり、アクションが発生することを保証します。

Claude はあなたのためにフックを書くことができます。**「すべてのファイル編集後に *eslint* を実行するフックを書く」** または **「*migrations* フォルダへの書き込みをブロックするフックを書く」** のようなプロンプトを試してください。`.claude/settings.json` を直接編集してフックを設定し、`/hooks` を実行して設定されているものを参照します。

8.4.6 スキルを作成する

Tip:

`.claude/skills/` に `SKILL.md` ファイルを作成して、Claude にドメイン知識と再利用可能なワークフローを提供します。

スキル (ページ 275) は、プロジェクト、チーム、またはドメイン固有の情報で Claude の知識を拡張します。Claude は関連するときに自動的にそれらを適用するか、`/skill-name` で直接呼び出すことができます。

`.claude/skills/` にディレクトリと `SKILL.md` を追加してスキルを作成します。

リスト 6: .claude/skills/api-conventions/SKILL.md

```
---
name: api-conventions
description: REST API design conventions for our services
---
# API Conventions
- Use kebab-case for URL paths
- Use camelCase for JSON properties
- Always include pagination for list endpoints
- Version APIs in the URL path (/v1/, /v2/)
```

スキルは、直接呼び出す再利用可能なワークフローを定義することもできます。

リスト 7: .claude/skills/fix-issue/SKILL.md

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---
Analyze and fix the GitHub issue: $ARGUMENTS.

1. Use `gh issue view` to get the issue details
2. Understand the problem described in the issue
3. Search the codebase for relevant files
4. Implement the necessary changes to fix the issue
5. Write and run tests to verify the fix
6. Ensure code passes linting and type checking
7. Create a descriptive commit message
8. Push and create a PR
```

`/fix-issue 1234` を実行して呼び出します。副作用のあるワークフローで、手動でトリガーしたい場合は `disable-model-invocation: true` を使用します。

8.4.7 カスタムサブエージェントを作成する

Tip:

`.claude/agents/` に特化したアシスタントを定義して、Claude が分離されたタスクに委譲できるようにします。

サブエージェント (ページ 317) は独自のコンテキストと独自の許可されたツールセットで実行されます。メインの会話を乱さずに、多くのファイルを読み取ったり、特化した焦点が必要なタスクに役立ちます。

リスト 8: .claude/agents/security-reviewer.md

```
---
name: security-reviewer
description: Reviews code for security vulnerabilities
tools: Read, Grep, Glob, Bash
model: opus
---
You are a senior security engineer. Review code for:
- Injection vulnerabilities (SQL, XSS, command injection)
- Authentication and authorization flaws
- Secrets or credentials in code
- Insecure data handling

Provide specific line references and suggested fixes.
```

Claude に明示的にサブエージェントを使用するよう指示します。**「サブエージェントを使用してこのコードをセキュリティの問題についてレビューする。」**

8.4.8 プラグインをインストールする

Tip:

`/plugin` を実行してマーケットプレイスを参照します。プラグインは設定なしでスキル、ツール、統合を追加します。

プラグインは、コミュニティと Anthropic からの単一のインストール可能なユニットにスキル、フック、サブエージェント、MCP サーバーをバンドルします。型付き言語を使用する場合は、コード インテリジェンス プラグインをインストールして、Claude に正確なシンボルナビゲーションと編集後の自動エラー検出を提供します。

スキル、サブエージェント、フック、MCP の選択に関するガイダンスについては、Claude Code を拡張を参照してください。

8.5 効果的にコミュニケーションする

Claude に他のエンジニアに尋ねるような質問をしてください。より大きな機能については、Claude にあなたにインタビューさせて、実装を開始する前に仕様を書いてもらいます。

8.5.1 コードベースの質問をする

Tip:

シニアエンジニアに尋ねるような質問を Claude にしてください。

新しいコードベースにオンボーディングするときは、Claude Code を学習と探索に使用します。別のエンジニアに尋ねるのと同じ種類の質問を Claude に尋ねることができます。

- ロギングはどのように機能しますか？
- 新しい API エンドポイントを作成するにはどうすればよいですか？
- `foo.rs` の 134 行目の `async move { ... }` は何をしていますか？
- `CustomerOnboardingFlowImpl` はどのエッジケースを処理しますか？
- このコードが 333 行目で `bar()` の代わりに `foo()` を呼び出すのはなぜですか？

Claude Code をこのように使用することは、効果的なオンボーディングワークフローであり、ラップアップ時間を改善し、他のエンジニアの負荷を軽減します。特別なプロンプトは必要ありません。直接質問してください。

8.5.2 Claude にあなたにインタビューさせる

Tip:

より大きな機能については、Claude に最初にあなたにインタビューさせます。最小限のプロンプトで開始し、Claude に `AskUserQuestion` ツールを使用してあなたにインタビューするよう依頼します。

Claude は、技術的な実装、UI/UX、エッジケース、トレードオフなど、あなたがまだ考えていないことについて質問します。`[brief description]` をあなたの機能に置き換えてからプロンプトを送信してください。

```
I want to build [brief description]. Interview me in detail using the AskUserQuestion␣
↪tool.

Ask about technical implementation, UI/UX, edge cases, concerns, and tradeoffs. Don't␣
↪ask obvious questions, dig into the hard parts I might not have considered.

Keep interviewing until we've covered everything, then write a complete spec to SPEC.
↪md.
```

仕様が完成したら、新しいセッションを開始して実行します。新しいセッションはクリーンなコンテキストを持ち、実装に完全に焦点を当てており、参照する書かれた仕様があります。

最も有用な仕様は自己完結型です。関連するファイルとインターフェースに名前を付け、スコープ外のものを述べ、機能が機能することを証明するエンドツーエンド検証ステップで終わります。仕様を正確にするのに費やした時間は、実装を見守るのに費やした時間よりも多くの見返りがあります。

8.6 セッションを管理する

会話は永続的で可逆的です。これを有利に使用してください。

8.6.1 早期かつ頻繁に方向転換する

Tip:

Claude が軌道を外れていることに気付いたらすぐに修正してください。

最良の結果は、タイトなフィードバックループから生まれます。Claude は時々最初の試みで問題を完全に解決しますが、それを迅速に修正することは一般的により良い解決策をより速く生成します。

- `Esc`：`Esc` キーで Claude の途中のアクションを停止します。コンテキストは保持されるため、リダイレクトできます。
- `Esc + Esc` **または** `/rewind`：`Esc` を 2 回押すか `/rewind` を実行して、巻き戻しメニューを開き、以前の会話とコード状態を復元するか、選択したメッセージから要約します。
- `"Undo that"`：Claude に変更を元に戻すよう依頼します。
- `/clear`：関連のないタスク間でコンテキストをリセットします。関連のないコンテキストを持つ長いセッションはパフォーマンスを低下させる可能性があります。

1 つのセッションで同じ問題について Claude を 2 回以上修正した場合、コンテキストは失敗したアプローチで乱雑です。`/clear` を実行し、学んだことを組み込んだより具体的なプロンプトで新しく開始します。より良いプロンプトを持つクリーンなセッションは、ほぼ常に蓄積された修正を持つ長いセッションを上回ります。

8.6.2 コンテキストを積極的に管理する

Tip:

関連のないタスク間で `/clear` を実行してコンテキストをリセットします。

Claude Code はコンテキスト制限に近づくと会話履歴を自動的にコンパクトにします。これにより、重要なコードと決定を保持しながらスペースを解放します。

長いセッション中に、Claude のコンテキストウィンドウは関連のない会話、ファイルの内容、コマンドで満杯になる可能性があります。これはパフォーマンスを低下させ、時々 Claude を気を散らすことができます。

- タスク間で頻繁に `/clear` を使用してコンテキストウィンドウを完全にリセットします
- 自動コンパクションがトリガーされると、Claude は最も重要なもの（コードパターン、ファイル状態、主要な決定を含む）を要約します
- より多くの制御のために、`/compact <instructions>` を実行します。例えば `/compact Focus on the API changes`
- 会話の一部のみをコンパクトするには、`Esc + Esc` または `/rewind` を使用し、メッセージチェックポイントを選択し、**ここから要約** または **ここまで要約** を選択します。最初のものはそのポイント以降のメッセージを凝縮し、以前のコンテキストは保持されます。2 番目のものは以前のメッセージを凝縮し、最近のものは完全に保持されます。**巻き戻しメニューの要約オプション** (ページ 1019) を参照してください。
- CLAUDE.md でコンパクション動作をカスタマイズします。`"When compacting, always preserve the full list of modified files and any test commands"` のような指示を使用して、重要なコンテキストが要約を生き残ることを確認します
- 会話履歴に入らない簡単な質問については、*/btw* (ページ 98) を使用します。答えは会話履歴に入らないため、コンテキストを増やさずに詳細をチェックできます。

8.6.3 調査にサブエージェントを使用する

Tip:

`"use subagents to investigate X"` で研究を委譲します。彼らは別のコンテキストで探索し、実装のためにメインの会話をクリーンに保ちます。

コンテキストが基本的な制約であるため、サブエージェントを使用して研究をコンテキストの外に保ちます。Claude がコードベースを研究するとき、多くのファイルを読み取り、すべてがコンテキストを消費します。サブエージェントは別のコンテキストウィンドウで実行され、要約を報告します。

```
Use subagents to investigate how our authentication system handles token
refresh, and whether we have any existing OAuth utilities I should reuse.
```

Claude が何かを実装した後、検証にサブエージェントを使用することもできます。**敵対的なレビューステップを追加する** (ページ 131) を参照してください。

8.6.4 チェックポイントで巻き戻す

Tip:

送信するすべてのプロンプトはチェックポイントを作成します。以前のチェックポイントに会話、コード、またはその両方を復元できます。

Claude は変更前に自動的にファイルをスナップショットするため、チェックポイントはそれらを復元できます。`Escape` をダブルタップするか `/rewind` を実行して、巻き戻しメニューを開きます。会話のみを復

元したり、コードのみを復元したり、両方を復元したり、選択したメッセージから要約したりできます。詳細については、**チェックポイント** (ページ 1019) を参照してください。

すべての動きを慎重に計画する代わりに、Claude に何か危険なことを試すよう指示できます。うまくいかない場合は、巻き戻して別のアプローチを試してください。チェックポイントはセッション全体で保持されるため、ターミナルを閉じても後で巻き戻すことができます。

警告:

チェックポイントは Claude が行った変更のみを追跡します。Bash コマンドまたは外部プロセスを通じて行われた変更はキャプチャされません。これは git の代替ではありません。

8.6.5 会話を再開する

Tip:

`/rename` でセッションに名前を付け、ブランチのように扱います。各ワークストリームは独自の永続的なコンテキストを取得します。

Claude Code は会話をローカルに保存するため、タスクが複数のセッションにまたがる場合、コンテキストを再度説明する必要はありません。`claude --continue` を実行して最新のセッションを選択するか、`claude --resume` を実行してリストから選択します。`oauth-migration` などの説明的な名前でセッションに名前を付けて、後で見つけやすくします。セッションを管理を参照して、再開、ブランチ、命名制御の完全なセットを確認してください。

8.7 自動化とスケール

1 つの Claude で効果的になったら、並列セッション、非対話型モード、ファンアウトパターンで出力を乗算します。

8.7.1 非対話型モードを実行する

Tip:

CI、プリコミットフック、またはスクリプトで `claude -p "prompt"` を使用します。ストリーミング JSON 出力の場合は `--output-format stream-json --verbose` を追加します。

`claude -p "your prompt"` を使用すると、対話型プロンプトなしで Claude を非対話的に実行できます。実行は `--no-session-persistence` を渡さない限り、再開可能なセッションを作成します。**非対話型モード** (ページ 695) は、Claude を CI パイプライン、プリコミットフック、または自動化されたワークフローに統合する方法です。出力形式を使用すると、結果をプログラムで解析できます。プレーンテキスト、JSON、またはストリーミング JSON です。

```
# One-off queries
claude -p "Explain what this project does"

# Structured output for scripts
claude -p "List all API endpoints" --output-format json

# Streaming for real-time processing
claude -p "Analyze this log file" --output-format stream-json --verbose
```

最初のコマンドはプレーンテキストを出力します。`json` 形式は `result` フィールドを持つ単一の JSON オブジェクトを返します。`stream-json` 形式は 1 行に 1 つの JSON オブジェクトを出力し、初期化イベントで始まります。

8.7.2 複数の Claude セッションを実行する

Tip:

複数の Claude セッションを並列で実行して、開発を高速化し、分離された実験を実行するか、複雑なワークフローを開始します。

自分で行いたい調整の量に合わせて、並列アプローチを選択し、セッションが相互に検出結果を渡す必要があるときはメッセージングを追加します。

- Worktrees：分離された git チェックアウトで個別の CLI セッションを実行して、編集が衝突しないようにします
- クロスセッションメッセージング：自分で実行するセッションが相互に検出結果を渡すことができます
- **デスクトップアプリ** (ページ 148)：複数のローカルセッションを視覚的に管理します。各セッションは独自の worktree にあります
- *Web* **上の** *Claude Code* (ページ 223)：デフォルトで Anthropic が管理するインフラストラクチャ上のクラウドでセッションを実行します
- エージェントビュー：研究プレビュー。`claude agents` を実行して、バックグラウンドで実行し続けるセッションをディスパッチし、1 つの画面から監視します
- エージェントチーム：実験的で、デフォルトで無効です。共有タスク、メッセージング、チームリーダーを備えた複数のセッションの自動調整

作業を並列化することを超えて、複数のセッションは品質に焦点を当てたワークフローを有効にします。新しいコンテキストは、Claude がちょうど書いたコードに偏らないため、コードレビューを改善します。

例えば、Writer/Reviewer パターンを使用します。

セッション A（ライター）	セッション B（レビュアー）
`Implement a rate limiter for our API endpoints`	
	`Review the rate limiter implementation in @src/middleware/rateLimiter.ts. Look for edge cases, race conditions, and consistency with our existing middleware patterns.`
`Here's the review feedback: [Session B output]. Address these issues.`	

テストで同様のことを行うことができます。1 つの Claude にテストを書かせ、別の Claude にそれらを渡すコードを書かせます。

8.7.3 ファイル全体にファンアウトする

Tip:

各タスクに対して `claude -p` を呼び出すループを実行します。バッチ操作のスコープ権限に `--allowedTools` を使用します。

大規模な移行または分析の場合、多くの並列 Claude 呼び出し全体で作業を配布できます。`/batch <instruction>` を実行して、Claude が変更を 5〜30 個のサブエージェント全体に分割させます。各サブエージェントは独自の worktree で作業します。代わりに独自のスクリプトからファンアウトを駆動するには、`claude -p` をループします。

1. **タスクリストを生成する**

   Claude に移行が必要なファイルのリストをファイルに書き込ませます。次のステップのループがそれを読み取ることができるように、`list all 2,000 Python files that need migrating and save the list to files.txt` のようなプロンプトを使用します

2. **リストをループするスクリプトを書く**

```
for file in $(cat files.txt); do
  claude -p "Migrate $file from Python 2 to Python 3. Return OK or FAIL." \
    --allowedTools "Edit,Bash(git commit *)"
done
```

3. **いくつかのファイルでテストしてから、すべてのファイルで実行する**

   最初の 2〜3 ファイルで何が悪いかに基づいてプロンプトを改善し、完全なセットで実行します。`--allowedTools` フラグは Claude が何ができるかを制限します。これは無人で実行しているときに重要です。

Claude を既存のデータ/処理パイプラインに統合することもできます。

```
claude -p "<your prompt>" --output-format json | your_command
```

8.7.4 auto mode で自律的に実行する

無中断の実行とバックグラウンドのセーフティチェックについては、auto mode を使用します。分類器モデルはコマンドを実行前にレビューし、スコープエスカレーション、未知のインフラストラクチャ、敵対的なコンテンツ駆動のアクションをブロックしながら、ルーチンワークをプロンプトなしで進めさせます。

```
claude --permission-mode auto -p "fix all lint errors"
```

非対話型実行で -p フラグを使用する場合、分類器が繰り返しアクションをブロックするとき、Claude Code は実行を停止しません。auto mode がフォールバックするときを参照して、代わりに何が起こるか、およびしきい値を確認してください。

8.7.5 敵対的なレビューステップを追加する

Tip:

タスクが完了したと見なす前に、サブエージェントに新しいコンテキストで diff をレビューさせ、ギャップを報告させます。

Claude が無人で作業する期間が長いほど、作業が完了したと見なす前に独立したチェックが重要になります。新しい**サブエージェント** (ページ 317) コンテキストで実行されるレビュアーは、diff と提供されたクライテリアのみを見て、変更を生成した推論は見ないため、独自の条件で結果を評価します。

正確性チェックの場合、バンドルされた /code-review スキルを実行します。これは新しいサブエージェントで現在の diff をバグについてレビューし、検出結果をセッションに返します。代わりに diff をプランに対してチェックするには、レビュープロンプトを自分で作成します。チェックする作業、チェック対象のプラン、および検出結果として何がカウントされるかを名前付けします。

```
Use a subagent to review the rate limiter diff against PLAN.md. Check that
every requirement is implemented, the listed edge cases have tests, and
nothing outside the task's scope changed. Report gaps, not style preferences.
```

レビュアーはサブエージェントとして実行されるため、実装セッションはギャップを直接受け取り、ウィンドウ間で検出結果をコピーすることなく、それらを修正して再レビューできます。

注釈:

ギャップを見つけるようにプロンプトされたレビュアーは、作業が健全であっても、通常はいくつかを報告します。それはそれが求められたことだからです。すべての検出結果を追求することは、過度なエンジニアリングにつながります。追加の抽象化レイヤー、防御的なコード、および発生できないケースのテストです。レビュアーに、正確性または記載された要件に影響するギャップのみをフラグ付けするよう指示し、残りはオプションとして扱います。

8.8 一般的な失敗パターンを避ける

これらは一般的な間違いです。早期に認識することで時間を節約できます。

- **キッチンシンクセッション。** 1 つのタスクで開始し、関連のないことを Claude に尋ねてから、最初のタスクに戻ります。コンテキストは関連のない情報でいっぱいです。

  修正：関連のないタスク間で `/clear` を実行します。

- **何度も修正する。** Claude が何か間違ったことをし、修正し、まだ間違っています。修正します。コンテキストは失敗したアプローチで乱雑です。

  修正：2 回の失敗した修正の後、`/clear` を実行し、学んだことを組み込んだより良い初期プロンプトを書きます。

- **過度に指定された CLAUDE.md。** CLAUDE.md が長すぎる場合、Claude は重要なルールがノイズに失われるため、半分を無視します。

  修正：容赦なく削除します。Claude が指示なしで既に何かを正しく行う場合、削除するか、フックに変換します。

- **信頼してから検証するギャップ。** Claude はもっともらしく見える実装を生成しますが、エッジケースを処理しません。

  修正：常に検証を提供します（テスト、スクリプト、スクリーンショット）。検証できない場合は、出荷しないでください。

- **無限探索。** スコープなしで何かを「調査」するよう Claude に依頼します。Claude は数百のファイルを読み取り、コンテキストを満たします。

  修正：調査を狭くスコープするか、サブエージェントを使用して、探索がメインコンテキストを消費しないようにします。

8.9 直感を開発する

このガイドのパターンは固定されていません。それらはすべての状況で一般的にうまく機能する出発点ですが、すべての状況に最適ではない可能性があります。

時々、あなたは 1 つの複雑な問題に深く入り込んでいて、履歴が価値があるため、コンテキストを蓄積させるべきです。時々、タスクが探索的であるため、計画をスキップして Claude にそれを理解させるべきです。時々、曖昧なプロンプトは、Claude が問題をどのように解釈するかを制約する前に見たいため、正確です。

何が機能するかに注意を払います。Claude が素晴らしい出力を生成するとき、あなたが何をしたかに注意してください。プロンプト構造、提供したコンテキスト、あなたがいたモード。Claude が苦労するとき、なぜ尋ねてください。コンテキストがノイズが多すぎましたか？プロンプトが曖昧すぎましたか？タスクが 1 回のパスには大きすぎましたか？

時間をかけて、ガイドが捉えることができない直感を開発します。具体的にするべき時と開放的にするべき時、計画すべき時と探索すべき時、コンテキストをクリアすべき時と蓄積させるべき時を知ります。

8.10 関連リソース

- Claude Code の仕組み：agentic ループ、ツール、コンテキスト管理
- Claude Code を拡張：スキル、フック、MCP、サブエージェント、プラグイン
- **一般的なワークフロー** (ページ 105)：デバッグ、テスト、PR などのステップバイステップレシピ
- *CLAUDE.md* (ページ 997)：プロジェクト規約と永続的なコンテキストを保存する

第III部

統合環境

第9章　Desktop application

> Claude Code Desktop をさらに活用する：Git 分離による並列セッション、ドラッグアンドドロップペインレイアウト、統合ターミナルとファイルエディタ、サイドチャット、コンピュータ使用、電話から Dispatch セッションを送信、ビジュアル diff レビュー、アプリプレビュー、PR 監視、コネクタ、エンタープライズ設定。

Claude Desktop アプリには 3 つのタブがあります：**Chat** は会話用、**Cowork** は Dispatch とより長い agentic work 用、**Code** はソフトウェア開発用です。このページは Code タブのリファレンスです。

- macOS 用にダウンロード

  Intel と Apple Silicon 向けのユニバーサルビルド

- Windows 用にダウンロード

  x64 プロセッサ向け

- Linux 用 Claude を入手（ベータ版）

  Ubuntu と Debian 向けの apt または .deb

Windows ARM64 の場合は、ARM64 インストーラーをダウンロードしてください。Linux では apt でインストールします。Claude Desktop on Linux を参照してください。

インストール後、Claude を起動してサインインし、**Code** タブをクリックします。最初のセッションのウォークスルーについては、はじめにガイドを参照してください。

Code タブでは、各会話は **セッション** です：独自のチャット履歴とプロジェクトフォルダを持ち、他のセッションとは独立しています。サイドバーはセッションをリストアップし、複数を並列で実行できます。セッション内では以下のことができます：

- ***diff* ビューで変更をレビューしてコメント** (ページ 142) してから、***CI* を通じて結果の *PR* を監視** (ページ 143)
- **埋め込みブラウザで実行中のアプリをプレビュー** (ページ 141) し、Claude が独自の変更を検証し、**外部サイトを並べて開く** (ページ 141)
- iOS Simulator ペインで iOS アプリを実行してテストする Claude を監視
- **ペインを配置** (ページ 143) して、チャット、diff、ブラウザ、ターミナル、ファイルエディタを並べて表示
- セッションのコンテキストを使用する**サイド質問** (ページ 149) を尋ねて、セッションを脱線させない
- Claude に**他のセッションを確認、メッセージ送信、またはアーカイブ** (ページ 149) させる
- **外部ツールを接続** (ページ 152)（GitHub、Slack、Linear など）

- Claude に**アプリを開いてスクリーンを制御** (ページ 146) させる
- マシン上、**クラウド** (ページ 150) 上、または *SSH* (ページ 159) 上で実行

スケジュール済みの定期的な作業、**キーボードショートカット** (ページ 145)、または**電話からタスクを送信** (ページ 151) については、リンクされたページとセクションを参照してください。既にターミナルベースの CLI を使用している場合は、*CLI* **比較** (ページ 166) を参照して、何が引き継がれるかを確認してください。

9.1 セッションを開始する

最初のメッセージを送信する前に、プロンプト領域で 4 つのことを設定してください：

- **環境**：Claude が実行される場所を選択します。ローカルマシンの場合は **Local**、**クラウドセッション** (ページ 159) の場合は **Cloud**、管理するリモートマシンの場合は *SSH* **接続** (ページ 159) を選択するか、Windows の場合は WSL ディストリビューションを選択します。**環境設定** (ページ 158) を参照してください。
- **プロジェクトフォルダ**：Claude が作業するフォルダまたはリポジトリを選択します。クラウドセッションの場合、**複数のリポジトリ** (ページ 150) を追加できます。
- **モデル**：送信ボタンの横のドロップダウンから**モデル** (ページ 725) を選択します。セッション中にこれを変更できます。
- **権限モード**：**モードセレクタ** (ページ 139) から Claude がどの程度の自律性を持つかを選択します。セッション中にこれを変更できます。

タスクを入力して **Enter** キーを押してセッションを開始します。各セッションは独自のコンテキストと変更を追跡します。

9.2 コードの操作

Claude に適切なコンテキストを提供し、それが独立して実行する量を制御し、変更内容を確認します。

9.2.1 プロンプトボックスを使用する

Claude に実行させたいことを入力して **Enter** キーを押して送信します。Claude はプロジェクトファイルを読み取り、**権限モード** (ページ 139) に基づいて変更を加えてコマンドを実行します。いつでも Claude を中断できます：停止ボタンをクリックして即座に中断するか、修正を入力して **Enter** キーを押して実行中のアクションを停止せずに送信します。Claude は現在のアクションが完了した後に修正を読み取り、次のステップの前に調整します。

プロンプトボックスの横の**+**ボタンをクリックすると、ファイル添付、**スキル** (ページ 152)、**コネクタ** (ページ 152)、および**プラグイン** (ページ 152) にアクセスできます。

9.2.2 ファイルとコンテキストをプロンプトに追加する

プロンプトボックスは外部コンテキストを取り込む 2 つの方法をサポートしています：

- **@mention ファイル**：@の後にファイル名を入力して、ファイルを会話コンテキストに追加します。Claude はそのファイルを読み取り、参照できます。@mention はクラウドセッションおよび WSL セッションでは利用できません。
- **ファイルを添付**：添付ボタンを使用するか、ファイルをプロンプトに直接ドラッグアンドドロップして、画像、PDF、およびその他のファイルをプロンプトに添付します。これはバグのスクリーンショット、デザインモックアップ、または参照ドキュメントを共有するのに便利です。

9.2.3 権限モードを選択する

権限モードは、セッション中に Claude がどの程度の自律性を持つかを制御します：ファイルの編集、コマンドの実行、またはその両方の前に確認するかどうかです。送信ボタンの横のモードセレクタを使用して、いつでもモードを切り替えることができます。送信ボタンの横のモードセレクタを使用して、いつでも権限モードを切り替えることができます。各変更を自分で承認するには、Manual に切り替えます。

新しいローカルセッションのデフォルトモードを設定するには、**設定ファイル** (ページ 711) に `permissions.defaultMode` を追加します。デスクトップアプリは CLI と同じ設定ファイルを読み取ります。セレクタで選択したモードはフォルダごとに記憶され、そのフォルダの `defaultMode` より優先されます。ただし Plan は現在のセッションにのみ適用されます。

モード	設定キー	動作
Manual	`default`	Claude はファイルの編集またはコマンドの実行の前に確認を求めます。diff を確認し、各変更を受け入れるか拒否できます。
Accept edits	`acceptEdits`	Claude はファイル編集と `mkdir`、`touch`、`mv` などの一般的なファイルシステムコマンドを自動的に受け入れますが、他のターミナルコマンドの実行前には確認を求めます。ファイル変更を信頼し、より高速な反復を望む場合に使用します。
Plan	`plan`	Claude はファイルを読み取り、コマンドを実行して探索してから、ソースコードを編集せずにプランを提案します。アプローチを最初に確認したい複雑なタスクに適しています。
Auto	`auto`	Claude はルーチンプロンプトなしで実行されます。シェルコマンドやネットワークリクエストなどのアクションが実行される前に、バックグラウンド分類器がリクエストとの整合性を確認します。*auto mode* **が利用可能** (ページ 140) な場合に表示されます。設定用の個別トグルはありません。

次のページに続く

表 1 – 前のページからの続き

モード	設定キー	動作
Bypass permissions	`bypassPermissions`	Claude は権限プロンプトなしで実行されます。ただし、どのモードも自動承認しないアクション、Claude が**外部サイトで機能する** (ページ 141) 場合の安全分類器、またはデスクトップアクション（**セッションをアーカイブする** (ページ 149) など Claude が常に最初に確認するもの）は除きます。CLI の`--dangerously-skip-permissions`と同等です。Pro および Max プランでは、Settings → Claude Code の「Allow bypass permissions mode」で有効にします。Team および Enterprise プランでは設定トグルはなく、組織ポリシーで制御されます。サンドボックス化されたコンテナまたは VM でのみ使用してください。

Code タブの以前のバージョンでは、これらのモードを Ask permissions、Auto accept edits、および Plan mode というラベルが付けられていました。

`dontAsk` 権限モードは CLI でのみ利用可能です。

Tip:

ベストプラクティス
複雑なタスクを Plan で開始して、Claude が変更を加える前にアプローチをマップアウトするようにします。プランを承認したら、Accept edits または Manual に切り替えて実行します。このワークフローの詳細については、**最初に探索してからプランしてからコード化する** (ページ 117) を参照してください。

クラウドセッションは Accept edits、Plan、および Auto をサポートしています。Accept edits は `default` モードに対応しています：クラウドセッションはファイル編集を事前に承認するため、セレクタは Manual ではなく Accept edits を表示します。Bypass permissions はクラウドセッションでは利用できません。自己ホスト環境のセッションを含みます。

エンタープライズ管理者は利用可能な権限モードを制限できます。詳細については、**エンタープライズ設定** (ページ 160) を参照してください。

auto モードの利用可否

auto モードは Anthropic API のすべてのユーザーが利用でき、Claude Opus 4.6 以降、Sonnet 4.6 以降、または *Fable* **モデル** (ページ 727) が必要です。組織管理者は**管理設定** (ページ 161) の `disableAutoMode` キーで auto モードをオフにできます。

Desktop を Google Cloud の Agent Platform にルーティングする Enterprise デプロイでも、auto モードはデフォルトで利用可能です。サポートされているモデルについては、Bedrock、Agent Platform、または Foundry での auto モードの有効化を参照してください。

9.2.4 アプリをプレビューする

Claude は dev サーバーを起動し、Browser ペインで開いて変更を確認できます。これはフロントエンド Web アプリとバックエンドサーバーの両方で機能します：Claude は API エンドポイントをテストし、サーバーログを表示し、見つけた問題を反復処理できます。ほとんどの場合、Claude はプロジェクトファイルを編集した後、サーバーを自動的に起動します。いつでも Claude にプレビューを要求することもできます。デフォルトでは、Claude は編集後に**変更を自動検証** (ページ 154) します。

Browser ペインは、プロジェクトから静的 HTML ファイル、PDF、画像、およびビデオを開くこともできます。チャットで HTML、PDF、画像、またはビデオパスをクリックして、Browser ペインで開きます。

Browser ペインから、以下を実行できます：

- Browser ペインで実行中のアプリと直接対話する
- Claude が自動的に独自の変更を検証するのを監視する：スクリーンショットを撮影し、DOM を検査し、要素をクリックし、フォームに入力し、見つけた問題を修正します
- セッションツールバーのサーバードロップダウンからサーバーを開始または停止する
- ドロップダウンで **Persist sessions** を選択して、サーバーの再起動時にクッキーとローカルストレージを保持し、開発中に再度ログインする必要がないようにする
- サーバー設定を編集するか、すべてのサーバーを一度に停止する

Claude はプロジェクトに基づいて初期サーバー設定を作成します。アプリがカスタム dev コマンドを使用する場合、`.claude/launch.json` を編集してセットアップに合わせます。完全なリファレンスについては、**プレビューサーバーを設定する** (ページ 153) を参照してください。

保存されたセッションデータをクリアするには、Settings → Claude Code でトグルを使用するか、Browser を完全に無効にするには、Settings → Claude Code でトグルを使用します。

9.2.5 外部サイトを閲覧する

Browser ペインはタブ付きブラウザなので、ドキュメント、issue トラッカー、または実行中のアプリの横に他のサイトを開くことができます。Browser を開くには、macOS で **Cmd+Shift+B**、Windows で **Ctrl+Shift+B** を押すか、**Views** メニューから選択します。チャットで外部リンクをクリックすると、Browser ペインを使用する **Open in app** または自分のブラウザを使用する **Default browser** を提供するチューザーが表示されます。macOS で **Cmd** キーを押しながらクリックするか、Windows で **Ctrl** キーを押しながらクリックすると、システムブラウザでリンクが直接開きます。Google OAuth などのポップアップサインインフローを含む、ペインのサイトにサインインできます。

Claude は**アプリを検証** (ページ 141) するために使用するのと同じツールを使用して外部ページを読み取り、対話できます。2 つの追加の安全チェックがあります：

- 安全分類器は、すべての権限モードで、クリックやタイピングなど、外部ページでの Claude の書き込みアクションを確認します。これらは *auto mode* (ページ 139) が使用するのと同じ分類器であり、アクションにフラグが立てられた場合、モードに関係なく権限プロンプトが表示されます。
- Auto および Bypass permissions 以外の権限モードでは、Claude が新しいサイトに移動する前にドメイン許可リストチェックも適用されます。

サイトで Claude のアクションを承認する

Claude が外部サイトで初めてアクションを実行すると、権限カードが表示され、Claude は選択を待ちます：**Allow once**、**Always allow**、または **Deny**。**Allow once** は何も保存せずにアクションを承認します。**Always allow** はそのサイトの承認をデバイスに保存し、Settings で取り消すことができます。サブドメインを含む各サイトは独自の承認が必要です。ローカル dev サーバーとプロジェクトファイルは承認が不要なため、*auto-verify* (ページ 154) はプロンプトなしで機能し続けます。

承認されたサイトでも、Claude はあなたの入力なしに商品を購入したり、アカウントを作成したり、CAPTCHA をバイパスしたりしません。Browser ペインでの閲覧は、*Claude in Chrome* **拡張機能** (ページ 213) と同じ安全モデルを使用します。Claude が機密サイトとリスクのあるアクションをどのように処理するかについては、Claude in Chrome を安全に使用するを参照してください。

Browser と Chrome 拡張機能を選択する

Browser ペインは、個人用ブラウザとは別の、保存されたログインや履歴がないクリーンなブラウザプロファイルを使用します。アプリの構築とテスト、および ID が不要なサイトに使用します。Claude があなたのログイン済みセッションであなたとして機能するようにしたい場合は、代わりにブラウザのログイン状態を共有する *Claude in Chrome* **拡張機能** (ページ 213) を使用します。

組織の外部閲覧を制限する

Browser は、Claude in Chrome 拡張機能と同じサイト許可リストとブロックリストコントロールに従います。組織が既に拡張機能用にこれらのリストを設定している場合、Browser は自動的にそれらを尊重します。管理者は、*browserExternalPageTools* **マネージド設定** (ページ 161) で外部ページの Claude のツールをオフにすることもできます。ツールが無効になっている場合、ユーザーは外部サイトに移動できます。Claude のツールはそれらを読み取ったり、アクションを実行したりできません。

外部閲覧を完全に無効にするには、*disableBrowserExternalNavigation* **マネージド設定** (ページ 161) を `true` に設定します。これは Browser 内のすべての外部ナビゲーションをブロックします。組織の許可リストのサイトを含みます。localhost dev サーバーとファイルプレビューは機能し続けます。`browserExternalPageTools` を使用して、ユーザーが Claude のツールなしで外部サイトの閲覧を続けられるようにし、`disableBrowserExternalNavigation` を使用して、ユーザーと Claude の両方に対して外部サイトをブロックします。

9.2.6 diff ビューで変更を確認する

Claude がコードに変更を加えた後、diff ビューを使用して、プルリクエストを作成する前にファイルごとに変更を確認できます。

Claude がファイルを変更すると、`+12 -1` などの追加および削除された行数を示す diff 統計インジケータが表示されます。このインジケータをクリックして diff ビューアを開きます。左側にファイルリストが表示され、右側に各ファイルの変更が表示されます。

特定の行にコメントするには、diff 内の任意の行をクリックしてコメントボックスを開きます。フィードバックを入力して **Enter** キーを押してコメントを追加します。複数の行にコメントを追加した後、すべてのコメントを一度に送信します：

- **macOS**：**Cmd+Enter** を押す

- **Windows**：**Ctrl+Enter** を押す

Claude はコメントを読み取り、要求された変更を加えます。これは確認できる新しい diff として表示されます。

9.2.7 コードを確認する

diff ビューで、右上のツールバーの **Review code** をクリックして、Claude にコミット前に変更を評価するよう依頼します。Claude は現在の diff を検査し、diff ビューに直接コメントを残します。任意のコメントに応答するか、Claude に修正を依頼できます。

レビューは高シグナル問題に焦点を当てています：コンパイルエラー、明確なロジックエラー、セキュリティ脆弱性、および明らかなバグです。スタイル、フォーマット、既存の問題、またはリンターが検出するものにはフラグを立てません。

9.2.8 プルリクエストステータスを監視する

プルリクエストを開いた後、CI ステータスバーがセッションに表示されます。Claude Code は GitHub CLI を使用してチェック結果をポーリングし、失敗を表示します。

- **Auto-fix**：有効にすると、Claude は失敗出力を読み取り、反復することで、失敗した CI チェックを自動的に修正しようとします。
- **Auto-merge**：有効にすると、Claude はすべてのチェックが成功したら PR をマージします。マージ方法はスカッシュです。Auto-merge がこれを機能させるために GitHub リポジトリ設定で有効にされている必要があります。

CI ステータスバーの **Auto-fix** および **Auto-merge** トグルを使用して、いずれかのオプションを有効にします。Claude Code はまた、CI が完了したときにデスクトップ通知を送信します。PR がマージまたはクローズされた後にセッションを自動的にアーカイブするには、Settings → Claude Code で *auto-archive* (ページ 148) をオンにします。

注釈:

PR 監視には、GitHub CLI（gh）がマシンにインストールされ、認証されている必要があります。gh がインストールされていない場合、Desktop は PR を作成しようとする最初の時点でインストールを促します。

9.3 ワークスペースを配置する

Code タブはペインを任意のレイアウトで配置できるように構築されています：チャット、diff、ブラウザ、ターミナル、ファイル、プラン、タスク、およびサブエージェント。macOS では iOS Simulator も含まれます。ペインをヘッダーでドラッグして位置を変更するか、ペインエッジをドラッグしてサイズを変更します。macOS では **Cmd+\を、Windows では Ctrl+\を押してフォーカスされたペインを閉じます。セッションツールバーの Views** メニューから追加のペインを開きます。

複数の画面で作業するには、diff やターミナルなどのペインをポップアウトして独立したウィンドウにし、完了したらドックバックします。Claude はメインウィンドウで作業を続けます。

注釈: このセクションのペインレイアウト、ターミナル、ファイルエディタ、およびビューモードには Claude Desktop v1.2581.0 以降が必要です。macOS では **Claude → Check for Updates** を、Windows では **Help → Check for Updates** を開いて更新してください。

9.3.1 ターミナルでコマンドを実行する

統合ターミナルを使用すると、別のアプリに切り替えることなく、セッションと並行してコマンドを実行できます。**Views** メニューから開くか、macOS または Windows で **Ctrl+`を押します。ターミナルはセッションの作業ディレクトリで開き、Claude と同じ環境を共有するため、`npm test` や `git status` などのコマンドは Claude が編集しているのと同じファイルを見ます。2 番目のターミナルタブを開くには、ターミナルペインヘッダーの+をクリックするか、チャットのフォルダを右クリックして Open in terminal** を選択します。ターミナルはローカルセッションでのみ利用可能です。

9.3.2 ファイルを開いて編集する

チャットまたは diff ビューアのファイルパスをクリックして、ファイルペインで開きます。HTML、PDF、画像、およびビデオパスは代わりに**ブラウザペイン** (ページ 141) で開きます。スポット編集を行い、**Save** をクリックして書き戻します。ファイルを開いてからディスク上で変更された場合、ペインは警告を表示し、オーバーライドまたは破棄できます。**Discard** をクリックして編集を元に戻すか、ペインヘッダーのパスをクリックして絶対パスをコピーします。

ファイルペインはローカルおよび SSH セッションで利用可能です。クラウドセッションの場合、Claude に変更を加えるよう依頼します。

9.3.3 ファイルを他のアプリで開く

チャット、diff ビューア、またはファイルペイン内のファイルパスを右クリックしてコンテキストメニューを開きます：

- **Attach as context**：ファイルを次のプロンプトに追加
- **Open in**：VS Code、Cursor、Zed などのインストール済みエディタでファイルを開く
- **Show in Finder**（macOS）、**Show in Explorer**（Windows）：含まれるフォルダを開く
- **Copy path**：絶対パスをクリップボードにコピー

9.3.4 ビューモードを切り替える

ビューモードは、チャットトランスクリプトに表示される詳細の量を制御します。送信ボタンの横の **Transcript view** ドロップダウンからモードを切り替えるか、macOS または Windows で **Ctrl+O** を押してモードをサイクルします。Thinking モードは、Claude がセッションで思考を生成した後にのみドロップダウンに表示されます。

モード	表示内容
Normal	ツール呼び出しは要約に折りたたまれ、完全なテキスト応答
Thinking	ツール呼び出しは要約に折りたたまれ、Claude の思考を含む
Verbose	すべてのツール呼び出し、ファイル読み取り、Claude が実行した中間ステップ、および Claude の思考

ツール呼び出しがまだ折りたたまれた状態で Claude の推論を追跡するには Thinking を使用します。Claude が特定のアクションを実行した理由をデバッグするときは Verbose を使用します。Claude Desktop v1.46388.1 より前のバージョンは Summary モードもリストしており、Summary に設定されたセッションは更新後に Normal で開きます。

9.3.5 キーボードショートカット

macOS で **Cmd+/を、Windows で Ctrl+/を押して、Code タブで利用可能なすべてのショートカットを表示します。Windows では、以下のショートカットに対して Cmd** の代わりに **Ctrl** を使用します。セッションサイクリング、ターミナルトグル、およびビューモードトグルはすべてのプラットフォームで **Ctrl** を使用します。

ショートカット	アクション
`Cmd /`	キーボードショートカットを表示
`Cmd N`	新しいセッション
`Cmd W`	セッションを閉じる
`Ctrl Tab` / `Ctrl Shift Tab`	次または前のセッション
`Cmd Shift ]` / `Cmd Shift [`	次または前のセッション
`Esc`	Claude の応答を停止
`Cmd Shift D`	diff ペインを切り替え
`Cmd Shift B`	ブラウザペインを切り替え
`Cmd Shift S`	ブラウザで要素を選択
`Ctrl ` `	ターミナルペインを切り替え
`Cmd \`	フォーカスされたペインを閉じる
`Cmd ;`	サイドチャットを開く
`Ctrl O`	ビューモードをサイクル
`Cmd Shift M`	権限モードメニューを開く
`Cmd Shift I`	モデルメニューを開く
`Cmd Shift E`	努力メニューを開く
`1–9`	開いているメニューの項目を選択

これらのショートカットは Code タブにのみ適用されます。ターミナルベースの**インタラクティブモードショートカット** (ページ 75)（モードをサイクルするための `Shift+Tab` など）は Desktop では適用されません。

9.3.6 使用状況を確認する

モデルピッカーの横の使用状況リングをクリックして、現在のコンテキストウィンドウ使用状況とプラン使用状況を確認します。コンテキスト使用状況はセッションごと、プラン使用状況はすべての Claude Code サーフェス全体で共有されます。

9.4 Claude にコンピュータを使用させる

コンピュータ使用により、Claude はアプリを開き、スクリーンを制御し、あなたがするのと同じ方法でマシンで直接作業できます。CLI がないデスクトップツールと対話するよう Claude に依頼したり、GUI を通じてのみ機能する何かを自動化したりしてください。iOS アプリの実行とテストの場合、Desktop はスクリーン制御の代わりに専用の iOS Simulator ペインを開きます。このペインはコンピュータ使用を有効にしなくても機能します。

注釈:

コンピュータ使用は macOS と Windows の研究プレビューであり、Pro または Max プランが必要です。Team または Enterprise プランでは利用できません。Claude Desktop アプリが実行されている必要があります。

コンピュータ使用はデフォルトでオフです。**設定で有効にして** (ページ 147)、Claude がスクリーンを制御する前に必要な権限を付与してください。macOS では、Accessibility と Screen Recording の権限も付与する必要があります。

macOS では、コンピュータ使用はバックグラウンドで実行することもできます。Claude は承認したアプリで作業しながら、あなたは作業を続けることができます。

警告:

サンドボックス化された *Bash* ツール (ページ 1069) とは異なり、コンピュータ使用は実際のデスクトップで実行され、承認したものへのアクセス権があります。Claude は各アクションをチェックし、オンスクリーンコンテンツからの潜在的なプロンプトインジェクションにフラグを立てますが、信頼境界は異なります。ベストプラクティスについては、コンピュータ使用安全ガイドを参照してください。

9.4.1 コンピュータ使用が適用される場合

Claude はアプリまたはサービスと対話するための複数の方法を持ち、コンピュータ使用は最も広範で最も遅いです。最も正確なツールを最初に試します：

- サービスの**コネクタ** (ページ 152) がある場合、Claude はコネクタを使用します。
- タスクがシェルコマンドの場合、Claude は Bash を使用します。
- タスクがブラウザ作業であり、*Chrome* の *Claude* (ページ 213) がセットアップされている場合、Claude はそれを使用します。
- タスクが iOS アプリの実行またはテストの場合、Claude は iOS Simulator ペインを使用します。これはスクリーン制御を使用しません。
- これらのいずれも適用されない場合、Claude はコンピュータ使用を使用します。

アプリごとのアクセス層 (ページ 147) はこれを強化します：ブラウザはビューのみに制限され、ターミナルと IDE はクリックのみに制限され、Claude をコンピュータ使用がアクティブな場合でも専用ツールに向けます。スクリーン制御は、ネイティブアプリ、ハードウェア制御パネル、または API のない独自ツールなど、他に何も到達できないものに予約されています。

9.4.2 コンピュータ使用を有効にする

コンピュータ使用はデフォルトでオフです。それが必要な何かをするよう Claude に依頼し、それがオフの場合、Claude は Settings でコンピュータ使用を有効にすれば、タスクを実行できることを伝えます。

1. **デスクトップアプリを更新する**

   Claude Desktop の最新バージョンがあることを確認してください。macOS と Windows では、claude.com/download でダウンロードまたは更新してください。Linux では、パッケージマネージャーを通じて更新してください（手順）。その後、アプリを再起動します。

2. **トグルをオンにする**

   デスクトップアプリで、**Settings > This computer > System** に移動します。**Computer use** の下で **Enable computer use** をオンにします。Windows では、トグルはすぐに有効になり、セットアップは完了です。macOS では、次のステップに進みます。

   トグルが表示されない場合は、macOS または Windows で Pro または Max プランを使用していることを確認してから、アプリを更新して再起動します。

3. **macOS 権限を付与する**

   macOS では、トグルが有効になる前に 2 つのシステム権限を付与します：

   - **Accessibility**：Claude がクリック、入力、スクロールできるようにします
   - **Screen Recording**：Claude がスクリーンに表示されているものを見ることができるようにします

   Settings ページは各権限の現在のステータスを表示します。いずれかが拒否されている場合、バッジをクリックして関連するシステム設定ペインを開きます。

9.4.3 アプリ権限

Claude が初めてアプリを使用する必要がある場合、セッションにプロンプトが表示されます。**Allow for this session** または **Deny** をクリックします。承認は現在のセッション、または *Dispatch* **が生成したセッション** (ページ 151) では 30 分間有効です。

プロンプトは、Claude がそのアプリに対して取得するコントロールのレベルも表示します。これらの層はアプリカテゴリによって固定され、変更できません：

層	Claude ができること	適用対象
ビューのみ	スクリーンショットでアプリを見る	ブラウザ、取引プラットフォーム
クリックのみ	クリックとスクロール、ただし入力またはキーボードショートカットは不可	ターミナル、IDE

次のページに続く

表 4 – 前のページからの続き

層	Claude ができること	適用対象
フルコントロール	クリック、入力、ドラッグ、キーボードショートカットの使用	その他すべて

Terminal、Finder または File Explorer、System Settings または Settings などの広範なリーチを持つアプリは、承認が何を付与するかを知るようにプロンプトに追加の警告を表示します。

Settings > This computer > System の **Computer use** セクションには、次のオプションがあります：

- **Denied apps**：ここにアプリを追加して、プロンプトなしで拒否します。Claude は許可されたアプリのアクションを通じて拒否されたアプリに間接的に影響を与える可能性がありますが、拒否されたアプリと直接対話することはできません。
- **Unhide apps when Claude finishes**：Claude が作業している間、他のウィンドウは非表示になり、承認されたアプリのみと対話します。Claude が完了すると、この設定をオフにしない限り、非表示のウィンドウが復元されます。

9.5 セッションを管理する

各セッションは独立した会話であり、独自のコンテキストと変更があります。複数のセッションを並列で実行するか、サイドチャットを分岐させるか、Claude に他のセッションをチェックしてメッセージを送信させるか、作業をクラウドに送信するか、Dispatch にセッションを電話から開始させることができます。

9.5.1 セッションで並列に作業する

サイドバーの**+ New session **をクリックするか、macOS で** Cmd+N **を、Windows で** Ctrl+N **を押して、複数のタスクを並列で作業します。Ctrl+Tab と** Ctrl+Shift+Tab **を押してサイドバーのセッションをサイクルします。Git リポジトリの場合、ブランチ名の横の** worktree**オプションを選択して、Git worktrees を使用してセッションにプロジェクトの独立した分離コピーを与えるため、1 つのセッションの変更は、コミットするまで他のセッションに影響しません。

2 つのセッションを同時に表示するには、macOS で **Cmd** を、Windows で **Ctrl** を押しながらサイドバーのセッションをクリックします。セッションは既に開いているセッションの横の 2 番目のペインで開きます。分割がアクティブな間、別のサイドバーセッションをクリックすると、フォーカスがあるペインが置き換わります。macOS で**Cmd+**を、Windows で** Ctrl+**を押して、フォーカスされたペインを閉じて、単一のセッションに戻ります。

Worktrees はデフォルトで`<project-root>/.claude/worktrees/`に保存されます。Settings → Claude Code の「Worktree location」でカスタムディレクトリに変更できます。また、すべての worktree ブランチ名の前に付加されるブランチプレフィックスを設定することもできます。これは Claude が作成したブランチを整理するのに便利です。完了したら、サイドバーのセッションにマウスを合わせてアーカイブアイコンをクリックして worktree を削除します。PR がマージまたはクローズされた後にセッションを自動的にアーカイブするには、Settings → Claude Code で **Auto-archive after PR merge or close** をオンにします。Auto-archive はローカルセッションで実行が完了したものにのみ適用されます。

gitignored ファイル（`.env` など）を新しい worktrees に含めるには、プロジェクトルートに.worktreeinclude ファイルを作成します。

注釈:

セッション分離には Git が必要です。ほとんどの Mac には Git がデフォルトで含まれています。Terminal で `git --version` を実行して確認してください。バージョン番号が表示されれば、Git がインストールされています。Git エラーが発生した場合は、Cowork タブで Claude に助けを求めてセットアップのトラブルシューティングを行ってください。

サイドバーの上部のコントロールを使用して、ステータス、プロジェクト、または環境でセッションをフィルタリングし、プロジェクトでセッションをグループ化します。セッション名を変更するには、アクティブセッションの上部のツールバーのセッションタイトルをクリックします。

コンテキスト使用状況を確認するには、**使用状況を確認する** (ページ 146) を参照してください。コンテキストがいっぱいになると、Claude は自動的に会話を要約して作業を続けます。`/compact` を入力して要約をより早くトリガーし、コンテキストスペースを解放することもできます。コンテキストウィンドウを参照して、圧縮がどのように機能するかについての詳細を確認してください。

デスクトップアプリは、Code セッションがタスクを完了し、現在そのセッションを表示していない場合に OS 通知を送信します。プロジェクトに属するセッションの場合、代わりにプロジェクトの通知を受け取ります。

9.5.2 メインセッションを脱線させずにサイドクエスチョンを尋ねる

サイドチャットを使用すると、セッションのコンテキストを使用するが、メインの会話に何も追加しない質問を Claude に尋ねることができます。コードの一部を理解したい、仮定を確認したい、またはセッションを脱線させずにアイデアを探索したい場合に使用します。

macOS で**Cmd+; **を、Windows で** Ctrl+;**を押してサイドチャットを開くか、プロンプトボックスで`/btw`を入力します。サイドチャットはその時点までのメインスレッドのすべてを読み取ることができます。完了したら、サイドチャットを閉じてメインセッションを続行します。

サイドチャットはローカル、SSH、および WSL セッションで利用可能です。デスクトップアプリはサイドチャットをディスクに保存しないため、アプリを閉じた後に戻ることはできません。

9.5.3 バックグラウンドタスクを監視する

タスクペインは、現在のセッション内で実行されているバックグラウンド作業を表示します：サブエージェント、バックグラウンドシェルコマンド、および動的ワークフロー。**Views** メニューから開くか、レイアウトにドラッグします。

任意のエントリをクリックして、サブエージェントペインで出力を確認するか、停止します。他のセッションが何をしているかを確認するには、**サイドバー** (ページ 148) を使用するか、Claude に**それらをチェックしてもらう** (ページ 149) ことができます。

9.5.4 セッション間で作業する

Claude はあなたの他の Code タブセッションをリストアップし、各セッションが何をしてきたかを読み取り、セッション間でメッセージを送信できます。平易な言語で尋ねてください：「どのセッションが認証リファクタリングに触れましたか？」、「API セッションは何を結論づけましたか？」、または「支払いセッションにスキーマが変更されたことを伝えてください」。また、Claude にセッションの名前変更またはアー

カイブを依頼することもできます。Claude はサイドバーのアーカイブアイコンと同じ方法でセッションをアーカイブするため、PR がマージされたセッションをクリーンアップするよう依頼してください。

このサーフェスを通じて、Claude はデスクトップアプリが自身で実行するセッションのみを見ます：ローカル、*SSH* (ページ 159)、および Code タブの WSL セッション。Claude はクラウドセッション、またはターミナル CLI または VS Code 拡張機能から開始したセッションを見ません。同じプロジェクトの worktrees にあっても見ません。そのため、9 つのターミナル worktrees が開いていて 2 つのデスクトップセッションがある場合、それらの 1 つで回答する Claude は他の 1 つのデスクトップセッションを報告します。Claude は質問しているセッションをリストアップしません。デフォルトでは、最近アクティブな 20 個のセッションを見て、アーカイブされたセッションをスキップします。ただし、それらを要求する場合を除きます。クロスセッションメッセージングは、Claude があなたの他の Claude Code セッション（ターミナルセッションを含む）にメッセージを送信することを別途許可します。

Claude がこのサーフェスを通じて別のセッションにメッセージを送信する場合、Claude Code はそこにカードとして表示され、送信セッションのタイトルでラベル付けされ、戻るリンクが付いているため、メッセージがどこから来たかを常に知ることができます。受信セッションがタスクの途中にある場合、Claude Code はメッセージを保持し、Claude は現在の作業が完了したら読み取ります。受信する Claude は返信でき、Claude Code はこのサーフェスを通じて返信を配信します。Claude はアーカイブされたセッションに配信できず、メッセージが通らない場合はあなたに通知します。

Claude Code はセッション間で 4 つの安全動作を適用します：

- セッションをアーカイブする前に、Claude はまずあなたに尋ねます。Auto および Bypass 権限を含むすべての権限モードで承認カードが表示されます。
- このサーフェスを通じて、Claude はスケジュール済みタスク実行など、誰も見ていないセッションからクロスセッションメッセージを送信できず、1 つに配信することもできません。
- Claude Code は、受信セッションがクロスセッションメッセージング自体を持たない場合でも、このサーフェスからの各メッセージを受信セッションのインバウンドコントロールに対してチェックします。受信セッションで `crossSessionInbound` を `refuse` に設定した場合、Claude Code はこのサーフェスからのメッセージをドロップします。Claude Code は拒否を Claude デスクトップアプリに報告します。v2.1.234 より前では、Claude Code はこのサーフェスからクロスセッションメッセージングを持たない受信セッションへのすべてのメッセージをドロップしました。
- Claude Code は各受信メッセージを引用し、それを送信したセッションに属性を付け、Claude は受信セッション自体の権限設定に従って 1 つに対して行動します。

Claude は新しいセッションを提案することもできます。現在のタスクの範囲外で修正する価値があるものに気付くと、作業をチャットのタスクチップとして提供します。チップをクリックして、独自の worktree を持つ新しいセッションでその作業を開始します。Claude は現在のセッションを中断なく続行します。

9.5.5 長時間実行されるタスクをクラウドで実行する

大規模なリファクタリング、テストスイート、マイグレーション、またはその他の長時間実行されるタスクの場合、セッションを開始するときに **Local** の代わりに **Cloud** を選択します。Cloud セッションはデフォルトで Anthropic が管理するインフラストラクチャで実行され、アプリを閉じたりコンピュータをシャットダウンしたりしても続行します。いつでも戻ってきて進捗を確認するか、Claude を別の方向に導くことができます。claude.ai/code または Claude モバイルアプリから Cloud セッションを監視することもできます。

Cloud セッションは複数のリポジトリもサポートしています。クラウド環境を選択した後、選択されたリポジトリの横の**+**ボタンをクリックして、セッションに追加のリポジトリを追加します。各リポジトリは独自のブランチセレクタを取得します。これは共有ライブラリとそのコンシューマーの更新など、複数のコードベースにまたがるタスクに便利です。

Cloud セッションがどのように機能するかについての詳細については、*Web 上の Claude Code* (ページ 223) を参照してください。1 つの作業本体が多くの Cloud セッションを必要とする場合、サイドバーで **Projects** を選択してプロジェクトを作成します。Claude はそこから 1 つの会話でセッションを開始して追跡します。

9.5.6 別のサーフェスで続行する

セッションツールバーの右下の VS Code アイコンからアクセスできる **Continue in** メニューを使用すると、セッションを別のサーフェスに移動できます：

- **Claude Code on the Web**：ローカルセッションをクラウドで実行し続けるために送信します。Desktop はブランチをプッシュし、会話の要約を生成し、完全なコンテキストを持つ新しい Cloud セッションを作成します。その後、ローカルセッションをアーカイブするか保持するかを選択できます。これはクリーンなワーキングツリーが必要であり、SSH セッションでは利用できません。
- **Your IDE**：現在の作業ディレクトリでサポートされている IDE でプロジェクトを開きます。

9.5.7 Dispatch からのセッション

Dispatch は、Cowork タブに存在する Claude との永続的な会話です。Dispatch にタスクをメッセージで送信すると、それをどのように処理するかを決定します。

タスクは 2 つの方法で Code セッションになります：「Claude Code セッションを開いてログインバグを修正する」など直接要求するか、Dispatch がタスクが開発作業であると判断して自動的に生成するかです。通常 Code にルーティングされるタスクには、バグの修正、依存関係の更新、テストの実行、またはプルリクエストの開くが含まれます。研究、ドキュメント編集、スプレッドシート作業は Cowork に留まります。

どちらの方法でも、Code セッションは Code タブのサイドバーに **Dispatch** バッジ付きで表示されます。完了したときまたは承認が必要なときに、電話でプッシュ通知を受け取ります。

コンピュータ使用 (ページ 146) が有効な場合、Dispatch が生成した Code セッションもそれを使用できます。これらのセッションのアプリ承認は 30 分後に期限切れになり、通常の Code セッションのようにセッション全体を続けるのではなく、再度プロンプトが表示されます。

セットアップ、ペアリング、Dispatch 設定については、Dispatch ヘルプ記事を参照してください。Dispatch には Pro または Max プランが必要であり、Team または Enterprise プランでは利用できません。

Dispatch は、ターミナルから離れているときに Claude で作業する複数の方法の 1 つです。他のオプションとの比較については、プラットフォームと統合を参照してください。

9.6 Claude Code を拡張する

外部サービスを接続し、再利用可能なワークフローを追加し、Claude の動作をカスタマイズし、プレビューサーバーを設定します。コネクタ、スキル、プラグインを 1 か所で管理するには、サイドバーの **Customize** をクリックします。Desktop アプリの Cowork タブは、CLI の `~/.claude` ディレクトリではなく、この Customize 設定からスキル、プラグイン、コネクタをソースとし、claude.ai アカウント経由で同期します。

Claude Code はまた、同じアカウントでサインインしたターミナルセッションで、claude.ai アカウント用に有効化されたスキルとプラグインもロードします。*claude.ai* **から同期されたスキル** (ページ 283) および claude.ai から同期されたプラグインを参照してください。

9.6.1 外部ツールを接続する

ローカルおよび *SSH* (ページ 159) セッションの場合、プロンプトボックスの横の**+ボタンをクリックして Connectors を選択し、Google Calendar、Slack、GitHub、Linear、Notion などの統合を追加します。セッションの前または中にコネクタを追加できます。**+**ボタンはクラウドセッションおよび WSL セッションでは利用できませんが、ルーチンはルーチン作成時にコネクタを設定します。

コネクタを管理または切断するには、デスクトップアプリの Settings → Connectors に移動するか、プロンプトボックスの Connectors メニューから **Manage connectors** を選択します。

接続すると、Claude はカレンダーを読み取り、メッセージを送信し、問題を作成し、ツールと直接対話できます。セッションで設定されているコネクタについて Claude に尋ねることができます。

コネクタは *MCP* **サーバー** (ページ 363) であり、グラフィカルセットアップフローを備えています。サポートされているサービスとの迅速な統合に使用します。Connectors にリストされていない統合の場合、**設定ファイル** (ページ 364) を介して MCP サーバーを手動で追加します。また、カスタムコネクタを作成することもできます。

9.6.2 スキルを使用する

スキル (ページ 275) は Claude ができることを拡張します。Claude は関連する場合に自動的にロードするか、直接呼び出すことができます：プロンプトボックスで / を入力するか、**+ボタンをクリックして Slash commands** を選択して、利用可能なものを参照します。これには 組み込みコマンド、**カスタムスキル** (ページ 278)、コードベースからのプロジェクトスキル、および インストール済みプラグインからのスキルが含まれます。1 つを選択すると、入力フィールドで強調表示されます。その後にタスクを入力して、通常どおり送信します。

Claude が作業中でも、他のメッセージと同じようにコマンドを送信でき、ターンが終了するとセッションはアイドル状態に戻ります。v2.1.206 より前では、ターン中に送信されたコマンドはセッションを実行中として表示したままにし、その後に送信したメッセージは配信されませんでした。

ローカルセッションは `~/.claude/skills/` からパーソナルスキルをロードします。*SSH* (ページ 159) セッションは、マシンからではなく、リモートホストのホームディレクトリから `~/.claude/skills/` を読み取ります。

ローカルおよびクラウドセッションは、claude.ai アカウント用に有効化されたスキルもロードします。ただし、組織が `disableSideloadFlags` を設定している場合、ローカルセッションはそれらをロードしません。クラウドセッションは、*Cowork* **およびクラウドセッションのスキル** (ページ 282) で説明されているように、`~/.claude/skills/` の代わりにそれらをロードします。

9.6.3 プラグインをインストールする

プラグインは、スキル、エージェント、hooks、MCP サーバー、および LSP 設定を Claude Code に追加する再利用可能なパッケージです。ターミナルを使用せずにデスクトップアプリからプラグインをインストールできます。

ローカルおよび *SSH* (ページ 159) セッションの場合、プロンプトボックスの横の**+ボタンをクリックして Plugins を選択して、インストール済みプラグインとそのスキルを確認します。プラグインを追加するには、サブメニューから Add plugin**を選択してプラグインブラウザを開きます。これは、公式 Anthropic マーケットプレイスを含む、設定された マーケットプレイスから利用可能なプラグインを表示します。**Manage plugins** を選択して、プラグインを有効化、無効化、またはアンインストールします。

プラグインはユーザーアカウント、特定のプロジェクト、またはローカルのみにスコープできます。組織がプラグインを一元管理する場合、それらのプラグインは CLI と同じ方法で Desktop セッションで利用可能です。ただし、デスクトップアプリが `disableSideloadFlags` の下で保留しているものは除きます。

プラグインブラウザはクラウドセッションでは利用できず、デスクトップアプリからインストールしたプラグインはクラウドセッションでは利用できません。クラウドセッションでもプラグインをインストールしません。リポジトリの `.claude/settings.json` で宣言されているプラグインは、セットアップから引き継がれるもので説明されているとおりです。プラグインは WSL セッションでは利用できません。プラグインの作成を含む完全なプラグインリファレンスについては、プラグインを参照してください。

9.6.4 プレビューサーバーを設定する

Claude は dev サーバーセットアップを自動的に検出し、セッションを開始するときに選択したフォルダのルートの `.claude/launch.json` に設定を保存します。Preview はこのフォルダを作業ディレクトリとして使用するため、親フォルダを選択した場合、独自の dev サーバーを持つサブフォルダは自動的に検出されません。サブフォルダのサーバーで作業するには、そのフォルダで直接セッションを開始するか、設定を手動で追加します。

サーバーの起動方法をカスタマイズするには、たとえば `npm run dev` の代わりに `yarn dev` を使用するか、ポートを変更するには、ファイルを手動で編集するか、サーバードロップダウンの **Edit configuration** をクリックしてコードエディタで開きます。ファイルはコメント付き JSON をサポートしています。

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

同じプロジェクトから異なるサーバーを実行するために複数の設定を定義できます。たとえば、フロントエンドと API です。以下の **例** (ページ 156) を参照してください。

変更を自動検証する

autoVerify が有効な場合、Claude はファイルを編集した後、コード変更を自動的に検証します。スクリーンショットを撮影し、エラーをチェックし、応答を完了する前に変更が機能することを確認します。

Auto-verify はデフォルトで有効です。.claude/launch.json に "autoVerify": false を追加してプロジェクトごとに無効にするか、サーバードロップダウンメニューから切り替えます。

```
{
  "version": "0.0.1",
  "autoVerify": false,
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 3000
    }
  ]
}
```

無効にすると、プレビューツールは引き続き利用可能であり、いつでも Claude に検証を依頼できます。Auto-verify は編集後に自動的に実行します。

設定フィールド

configurations 配列の各エントリは、以下のフィールドを受け入れます：

フィールド	型	説明
name	string	このサーバーの一意の識別子
runtimeExecutable	string	実行するコマンド（npm、yarn、node など）
runtimeArgs	string[]	runtimeExecutable に渡される引数（["run", "dev"] など）
port	number	サーバーがリッスンするポート。デフォルトは 3000
cwd	string	プロジェクトルートに相対的な作業ディレクトリ。デフォルトはプロジェクトルート。プロジェクトルートを明示的に参照するには ${workspaceFolder} を使用します
env	object	{ "NODE_ENV": "development" } などのキーと値のペアとしての追加環境変数。このファイルはリポジトリにコミットされるため、ここにシークレットを入れないでください。dev サーバーにシークレットを渡すには、**ローカル環境エディタ** (ページ 158) で設定します。
autoPort	boolean	ポート競合の処理方法。**ポート競合** (ページ 156) を参照してください

次のページに続く

表 5 – 前のページからの続き

フィールド	型	説明
program	string	node で実行するスクリプト。*program と runtimeExecutable を使用する場合* (ページ 155) を参照してください
args	string[]	program に渡される引数。program が設定されている場合のみ使用されます
url	string	http://localhost:<port> の代わりにプレビューが開くアドレス。**特定の *URL* でプレビューを開く** (ページ 155) を参照してください

program と runtimeExecutable を使用する場合

runtimeExecutable を runtimeArgs と共に使用して、パッケージマネージャーを通じて dev サーバーを起動します。たとえば、"runtimeExecutable": "npm" と "runtimeArgs": ["run", "dev"] は npm run dev を実行します。

node で直接実行したいスタンドアロンスクリプトがある場合は program を使用します。たとえば、"program": "server.js" は node server.js を実行します。args で追加フラグを渡します。

特定の URL でプレビューを開く

デフォルトでは、プレビューは http://localhost:<port> を開きます。サーバーが別のアドレスを必要とする場合は url を設定します。一般的なケースは、ローカル HTTPS が必要なサーバー、*.localhost サブドメインを使用するアプリ、およびリダイレクトを通じてサインインするアプリです。

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "port": 8443,
      "url": "https://localhost:8443"
    }
  ]
}
```

Localhost アドレスは直接開き、デフォルトのポートアドレスとまったく同じです。これには localhost、任意の *.localhost サブドメイン、127.0.0.1、および ::1 が含まれます。セキュリティのため、localhost url はサーバーのオリジンのみである必要があります。パスやクエリはなく、ポートはエントリのポートと一致する必要があります。特定のページを表示するには、プレビューが開いた後、Claude にそこに移動するよう依頼します。パス、クエリ、または一致しないポートを持つ localhost url は、url を名前付けして修正を表示する設定エラーとして報告されます。

他のアドレスの場合、Desktop はプレビューが初めてそれを開くときに許可を求めます。これは、プレビューで新しいサイトを閲覧するときと同じです。外部アドレスはパスを含む場合があります。**Always allow** を

選択して、将来そのサイトのプロンプトをスキップします。プレビューで外部サイトを制限する組織ポリシーは引き続き適用されます。

既に自分で実行しているサーバーをプレビューするには、コマンドなしで `url` を設定します。Claude はサーバーを起動する代わりに、実行中のサーバーにプレビューを接続します：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "my-app",
      "url": "https://app.localhost:3000"
    }
  ]
}
```

`url` は `http` または `https` である必要があり、ユーザー名またはパスワードを含まない必要があります。

ポート競合

`autoPort` フィールドは、優先ポートが既に使用されている場合の処理を制御します：

- `true`：Claude は自動的に空きポートを見つけて使用します。ほとんどの dev サーバーに適しています。
- `false`：Claude はエラーで失敗します。OAuth コールバックまたは CORS 許可リストなど、サーバーが特定のポートを使用する必要がある場合に使用します。
- **設定されていない（デフォルト）**：Claude はサーバーがそのポートを必要とするかどうかを尋ねてから、答えを保存します。

Claude が別のポートを選択すると、割り当てられたポートを `PORT` 環境変数を通じてサーバーに渡します。

例

これらの設定は、異なるプロジェクトタイプの一般的なセットアップを示しています：

Next.js

この設定は、Yarn を使用してポート 3000 で Next.js アプリを実行します：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "web",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["dev"],
      "port": 3000
    }
  ]
}
```

複数のサーバー

フロントエンドと API サーバーを持つモノレポの場合、複数の設定を定義します。フロントエンドは `autoPort: true` を使用して、3000 が使用されている場合は空きポートを選択し、API サーバーはポート 8080 を正確に必要とします：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "frontend",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "dev"],
      "cwd": "apps/web",
      "port": 3000,
      "autoPort": true
    },
    {
      "name": "api",
      "runtimeExecutable": "npm",
      "runtimeArgs": ["run", "start"],
      "cwd": "server",
      "port": 8080,
      "env": { "NODE_ENV": "development" },
      "autoPort": false
    }
  ]
}
```

Node.js スクリプト

パッケージマネージャーコマンドを使用する代わりに Node.js スクリプトを直接実行するには、`program` フィールドを使用します：

```
{
  "version": "0.0.1",
  "configurations": [
    {
      "name": "server",
      "program": "server.js",
      "args": ["--verbose"],
      "port": 4000
    }
  ]
}
```

9.7 環境設定

セッションを開始する (ページ 138) ときに選択する環境は、Claude が実行される場所と接続方法を決定します：

- **Local**：マシンで実行され、ファイルに直接アクセスできます
- **Cloud**：デフォルトで Anthropic が管理するインフラストラクチャで実行されます。アプリを閉じても、セッションは続行されます。
- **SSH**：SSH 経由で接続するリモートマシンで実行されます。たとえば、独自のサーバー、クラウド VM、または dev コンテナなどです。
- **WSL**（Windows）：マシン上の WSL 2 ディストリビューション内で実行され、その Linux ツールチェーンとネイティブパスを使用します

9.7.1 ローカルセッション

デスクトップアプリは常にシェル環境全体を継承するわけではありません。macOS では、Dock または Finder からアプリを起動すると、`~/.zshrc` または `~/.bashrc` などのシェルプロファイルを読み取り、`PATH` と固定された Claude Code 変数セットを抽出しますが、そこでエクスポートする他の変数は取得されません。Windows では、アプリはユーザーおよびシステム環境変数を継承しますが、PowerShell プロファイルは読み取りません。

ローカルセッションと dev サーバーの環境変数を設定するには、プロンプトボックスの環境ドロップダウンを開き、**Local** にマウスを合わせて、ギアアイコンをクリックしてローカル環境エディタを開きます。ここで保存する変数は、マシンに暗号化されて保存され、開始するすべてのローカルセッションとプレビューサーバーに適用されます。また、`~/.claude/settings.json` ファイルの `env` キーに変数を追加することもできます。ただし、これらは Claude セッションにのみ到達し、dev サーバーには到達しません。サポートされている変数の完全なリストについては、環境変数を参照してください。

拡張思考 (ページ 751) はデフォルトで有効になっており、複雑な推論タスクのパフォーマンスを向上させますが、追加のトークンを使用します。Anthropic API では、ローカル環境エディタで `MAX_THINKING_TOKENS` を `0` に設定して思考をオフにします。これは Opus 5.5、Sonnet 5.5、または Fable モデルには効果がなく、常に拡張思考を使用します。Anthropic API で思考をオフにした場合、Claude Code は、Opus 5 などのその組み合わせを受け入れないことが分かっているモデルに、より高いレベルではなく effort `high` を送信します。

適応的推論 (ページ 747) を持つモデルでは、適応的推論が思考の深さを制御するため、`0` 以外の `MAX_THINKING_TOKENS` 値は無視されます。Opus 4.6 と Sonnet 4.6 では、固定思考予算を使用するために `CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING` を `1` に設定します。Fable モデル、Sonnet 5 以降、および Opus 4.7 以降は常に適応的推論を使用し、固定予算モードはありません。

管理デバイス上のローカルセッション

管理者は *disableDesktopLocalSessions* **管理設定** (ページ 161) でローカルセッションをオフにできます。オフにされた場合、**Local** は環境ドロップダウンに留まりますが、グレーアウトされて選択できず、組織がオフにしたことを示すツールチップが表示されます。Windows では、WSL エントリも、管理デバイスでの可用性が別途管理される場合、同じようにグレーアウトされます。新しいセッションは、SSH 接続が設定されている場合、最初の SSH 接続にデフォルト設定されます。Desktop は、既存のセッションを続行しよ

うとした場合、このデバイスではローカルセッションが利用できないというメッセージを表示します。代わりに *SSH* (ページ 159) または *Cloud* (ページ 159) 環境を選択するか、IT チームに連絡してください。

9.7.2 クラウドセッション

クラウドセッションはアプリを閉じても、バックグラウンドで続行されます。使用状況は**サブスクリプションプランの制限** (ページ 911) にカウントされ、別の計算料金はありません。

異なるネットワークアクセスレベルと環境変数を持つカスタムクラウド環境を作成できます。それらを管理するには、プロンプトボックスの環境ドロップダウンを開いて **Cloud** を選択します：

- **環境を追加する**：**Add cloud environment** を選択します
- **自分の環境の 1 つを編集またはアーカイブする**：それにマウスを合わせて、ギアアイコンをクリックします

ネットワークアクセスと環境変数の設定の詳細については、クラウド環境を設定するを参照してください。

9.7.3 SSH セッション

SSH セッションを使用すると、デスクトップアプリをインターフェイスとして使用しながら、リモートマシンで Claude Code を実行できます。これは、クラウド VM、dev コンテナ、または特定のハードウェアまたは依存関係を持つサーバーに存在するコードベースで作業するのに便利です。

SSH 接続を追加するには、セッションを開始する前に環境ドロップダウンをクリックして、**+ Add SSH connection** を選択します。ダイアログは以下を要求します：

- **Name**：この接続のフレンドリーラベル
- **SSH Host**：`user@hostname` または `~/.ssh/config` で定義されたホスト
- **SSH Port**：空のままの場合はデフォルトの 22、または SSH config からのポート
- **Identity File**：`~/.ssh/id_rsa` などの秘密鍵へのパス。デフォルトキーまたは SSH config を使用するには空のままにします。

追加されると、接続は環境ドロップダウンに表示されます。それを選択して、そのマシンでセッションを開始します。Claude はリモートマシンで実行され、そのファイルとツールにアクセスできます。

リモートマシンは Linux または macOS を実行する必要があります。Desktop は初回接続時にリモートマシンに Claude Code を自動的にインストールします。接続されると、SSH セッションは権限モード、コネクタ、プラグイン、および MCP サーバーをサポートします。

チームの SSH 接続を事前設定する

管理者は、管理設定ファイルに `sshConfigs` を追加することで、SSH 接続をチームメンバーに配布できます。この方法で定義された接続は、各ユーザーの環境ドロップダウンに自動的に表示され、管理対象として表示されるため、ユーザーはそれらを選択できますが、アプリで編集または削除することはできません。

次の例は、単一の接続を事前設定しています：

```
{
  "sshConfigs": [
    {
      "id": "shared-dev-vm",
      "name": "Shared Dev VM",
      "sshHost": "user@dev.example.com",
      "sshPort": 22,
      "sshIdentityFile": "~/.ssh/id_ed25519"
    }
  ]
}
```

各エントリには id、name、および sshHost が必要です。sshPort および sshIdentityFile フィールドはオプションです。ユーザーは、ダイアログを通じて追加された接続が保存される独自の ~/.claude/settings.json に sshConfigs を追加することもできます。

ユーザーが接続できる SSH ホストを制限する

管理者は、管理設定ファイルに sshHostAllowlist を追加することで、Desktop の SSH セッションを承認されたホストのセットに制限できます。設定されると、ユーザーは解決されたホスト名がパターンの 1 つと一致するホストにのみ接続できます。SSH セッションを完全に無効にするには、空の配列に設定します。

次の例は、devboxes.example.com の下のすべてのホストと、単一の名前付きバスティオンホストへの接続を許可しています：

```
{
  "sshHostAllowlist": ["*.devboxes.example.com", "bastion.example.com"]
}
```

パターンは大文字と小文字を区別しません。* はすべてのホストと一致し、*.example.com は example.com とすべてのサブドメインと一致します。その他はすべて完全一致です。チェックは ssh -G を経由した ~/.ssh/config 解決後のホスト名に対して実行されるため、Host エイリアスと ProxyCommand/ProxyJump エントリは、解決された HostName が一致する限り許可されます。

sshHostAllowlist は管理設定からのみ読み取られます。ユーザーまたはプロジェクト設定の値は無視されます。Claude Desktop アプリのみがこの設定を尊重します。Claude Code CLI と IDE 拡張機能はこれを読み取らず、Bash ツールを通じて実行される ssh コマンドを制限しません。これは Desktop アプリが接続するホストを管理し、ネットワーク出力ではないため、ハード境界が必要な場合は組織のネットワークまたはゼロトラストコントロールと組み合わせてください。

9.8 エンタープライズ設定

Team または Enterprise プランの組織は、管理コンソールコントロール、管理設定ファイル、およびデバイス管理ポリシーを通じてデスクトップアプリの動作を管理できます。

9.8.1 管理コンソールコントロール

これらの設定は管理設定コンソールを通じて設定されます：

- **Code in the desktop**：組織内のユーザーがデスクトップアプリで Claude Code にアクセスできるかどうかを制御します
- **Code in the web**：組織の**クラウドセッション** (ページ 223) を有効または無効にします
- **Remote Control**：組織の Remote Control を有効または無効にします
- **Disable Bypass permissions mode**：組織内のユーザーが bypass permissions モードを有効にするのを防ぎます

注釈:

管理コンソールのデータとプライバシー設定の **Monitoring** 下の Cowork 用 OpenTelemetry フォームは、Cowork セッションのみに適用されます。このマシン上の Cowork セッションでは、デスクトップアプリはそのコレクタを Claude Code に `OTEL_*`環境変数として渡すため、Claude Code がそのセッションで**管理コンソール設定をフェッチしない** (ページ 161) 場合でも、フォームは有効になります。Code タブセッションからテレメトリをエクスポートするには、Claude Code 管理設定の `env` ブロックで `CLAUDE_CODE_ENABLE_TELEMETRY` と `OTEL_*`変数を設定します。**監視用の管理者設定** (ページ 936) に示されているとおりです。ローカル、クラウド、SSH セッションは、それぞれ**異なるソースから管理設定を読み取ります** (ページ 161)。クラウドセッションが到達できるホストについては、ネットワークアクセスを参照してください。Code タブセッションが報告する `service.name` については、**サービス情報** (ページ 993) を参照してください。

9.8.2 管理設定

管理設定はプロジェクトおよびユーザー設定をオーバーライドし、Desktop の Claude Code セッションに適用されます。これらのキーを組織の管理設定ファイルで設定するか、管理コンソールを通じてリモートでプッシュできます。

キー	説明
`permissions.disableBypassPermissionsMode`	ユーザーが Bypass permissions モードを有効にするのを防ぐには`"disable"`に設定します。
`disableAutoMode`	Auto モードをモードセレクタから削除するには`"disable"`に設定します。`permissions` の下でも受け入れられます。
`autoMode`	組織全体で auto mode 分類器が信頼およびブロックするものをカスタマイズします。auto mode を設定するを参照してください。
`browserExternalPageTools`	Claude が *Browser* **ペイン** (ページ 141) の外部ページを読み取るまたは操作するためのツールを使用するのを防ぐには`"disabled"`に設定します。ユーザーは引き続き外部サイトに自分でナビゲートできます。ローカル開発サーバープレビューは影響を受けません。

次のページに続く

表 6 – 前のページからの続き

キー	説明
disableMobileSimulatorTools	Claude の iOS Simulator ペインでデバイスを制御およびキャプチャするためのツールをブロックするには true に設定します。ペインはユーザー自身のタップに対して使用可能なままです。Claude のアクセスのみが削除されます。値は JSON ブール値 true である必要があります。文字列"true"は無視されます。
disableBrowserExternalNavigation	*Browser* **ペイン** (ページ 141) の外部ブラウジングを完全にオフにするには true に設定します。ユーザーも Claude も外部サイトにナビゲートできません。localhost 開発サーバープレビューは影響を受けません。値は JSON ブール値 true である必要があります。文字列"true"は無視されます。
sshConfigs	環境ドロップダウンに表示される *SSH* **接続** (ページ 159) を事前設定します。ユーザーは管理接続を編集または削除できません。
sshHostAllowlist	*SSH* **セッション** (ページ 160) を、解決されたホスト名がこれらのパターンのいずれかと一致するホストに制限します。空の配列は SSH セッションを無効にします。管理設定からのみ読み取られます。
disableDesktopLocalSessions	**デバイスで実行されるコードセッション** (ページ 158) をオフにするには true に設定します。SSH セッションから他のホストへのセッションとクラウドセッションは利用可能なままです。値は JSON ブール値 true である必要があります。管理設定からのみ読み取られます。Claude Desktop v1.37937.0 以降が必要です。
managedMcpServers	MCP サーバー設定をすべてのユーザーにプッシュします。サードパーティ（3P）Desktop デプロイメントでのみ利用可能です。各エントリで、"http"、"sse"、または"stdio"のトランスポート、接続詳細、およびオプションで、そのサーバーのどのツールをユーザーが呼び出せるかを制限する toolPolicy マップを設定します。管理設定ファイル、MDM、または Claude apps gateway ポリシーの desktop ブロックを通じて配信します。3P デプロイメントは管理コンソール設定を受け取らないためです。ゲートウェイを通じて配信するには、ゲートウェイサーバーで Claude Code v2.1.232 以降が必要です。これはデスクトップアプリ自体のキーです。Claude Code は、異なるエントリ形状を持つ、同じ名前の管理設定を読み取ります。

Desktop セッションがどこで実行されるかに応じて、どの管理設定がそのセッションに到達するかが異なります。*availableModels* (ページ 732) などのモデル制限は、Desktop の Claude Code セッションでターミナル CLI と同じ方法で適用されます。*surface coverage* (ページ 735) を参照してください。

- **このマシン上のローカルセッション**：ディスクにデプロイされた管理設定ファイルが適用されます。管理コンソールを通じてリモートでプッシュされた管理設定は、セッションが適格なログインまたはキーで認証する場合、Anthropic の API でこれらのセッションに到達します。ターミナル CLI と同じ**設定の優先順位** (ページ 719) に従います。
- **クラウドセッション** (ページ 159)：サーバー管理設定を受け取ります。デバイスにデプロイされたファイルはこれらのセッションに到達しません。Anthropic が管理する VM で実行されるためです。自己ホスト環境にルーティングされたセッションは、ランナーイメージ内の管理設定ファイルも読み取ります。Claude Code が管理ソースを組み合わせる方法は、そのファイルが適用される場合を示しています。
- *SSH* **セッション** (ページ 159)：セッションはリモートホストから管理設定ファイルを読み取ります。Desktop 自体は、ローカルマシンの管理設定から `sshConfigs`、`sshHostAllowlist`、および `disableDesktopLocalSessions` を読み取ります。
- Cowork **セッション**：このマシン上の Cowork セッションでは、ユーザーが Team または Enterprise アカウントでサインインしている場合でも、Claude Code は管理コンソール設定をフェッチしません。マシンにデプロイされたポリシーを読み取ります。ただし、Claude Desktop 設定が `requireCoworkFullVmSandbox` を設定している場合を除きます。リモート Cowork セッションはどちらも受け取りません。ポリシーが適用される場所と時期を参照して、どのデバイスファイルが Cowork に到達するかを確認してください。*MCP* **権限ルール** (ページ 1058) を参照して、`Bash` および `WebFetch` ルールが Cowork のツールにどのように適用されるかを確認してください。

ローカルおよび SSH セッションでは、デスクトップアプリは各ユーザーの接続された claude.ai コネクタを Claude Code に直接配信します。MCP 設定または `managed-mcp.json` はこれらのコネクタに到達しません。使用する設定ソースまたはファイルの場所に関係なく。これらのセッションでコネクタのツールをブロックするには、組織の**コネクタツールコントロール** (ページ 399) を使用してください。**コネクタが** *Claude Code* **に到達する方法** (ページ 398) は、各種類のセッションでコネクタを管理する設定を示しています。

`permissions.disableBypassPermissionsMode` と `disableAutoMode` はユーザーおよびプロジェクト設定でも機能しますが、管理設定に配置するとユーザーがそれらをオーバーライドするのを防ぎます。

管理ソースのみが設定できる権限、プラグイン、および配信キーについては、管理設定のみが設定できるキーを参照してください。

9.8.3 デバイス管理ポリシー

IT チームは、macOS の MDM または Windows のグループポリシーを通じてデスクトップアプリを管理できます。利用可能なポリシーには、Claude Code 機能の有効化または無効化、自動更新の制御、およびカスタムデプロイメント URL の設定が含まれます。

- **macOS**：Jamf または Kandji などのツールを使用して `com.anthropic.claudefordesktop` プリファレンスドメインを通じて設定します
- **Windows**：`SOFTWARE\Policies\Claude` のレジストリを通じて設定します

9.8.4 ネットワークアクセス要件

Desktop はアプリケーションコードとユーザーコンテンツを Anthropic CDN ホストから読み込みます。

```
anthropic.com
*.anthropic.com
claude.ai
*.claude.ai
claude.com
*.claude.com
claude.app
*.claude.app
*.claudeusercontent.com
*.claudemcpcontent.com
```

トラフィックは HTTPS ポート 443 です。ただし、*OTLP* (ページ 935)、LLM ゲートウェイ、または MCP サーバーのカスタムポートを設定する場合を除きます。

プロキシサーバー、カスタム認証局、mTLS、およびスタンドアロン CLI が必要とするドメインについては、**ネットワーク設定** (ページ 897) を参照してください。

ファイアウォールワイルドカードの数を減らすために、代わりにこれらの Anthropic ホストを許可してください。特定のサブドメインは動的に生成されるため、ワイルドカードのままである必要があります。

```
anthropic.com
api.anthropic.com
a-api.anthropic.com
a-cdn.anthropic.com
s-cdn.anthropic.com
assets-proxy.anthropic.com
claude.ai
a.claude.ai
a-cdn.claude.ai
assets.claude.ai
downloads.claude.ai
*.livepreview.claude.ai
claude.com
platform.claude.com
*.livepreview.claude.app
*.claudeusercontent.com
*.claudemcpcontent.com
```

組織で Claude に対して IP 許可リストが有効になっている場合は、`bridge.claudeusercontent.com` を `claude.ai` および `api.anthropic.com` と同じプロキシ出口を通じてルーティングしてください。そのようにルーティングできない場合は、プロキシがそのホストに使用する出口アドレスを組織の IP 許可リストに追加してください。ただし、そのアドレスが組織に専用されている場合のみです。共有プロキシ出口範囲は、プロキシベンダーの他のカスタマーも許可します。

Anthropic は、到着元のアドレスを使用して、組織の IP 許可リストに対してそのホストへの接続をチェッ

クします。プロキシがそのホストのトラフィックを、その許可リストにないアドレスを通じて送信する場合、Chrome の Claude およびブリッジを通じて接続する他の機能は機能しなくなりますが、アプリの残りの部分は機能し続けます。

Artifact が Google Fonts からタイプフェイスを読み込む場合、`fonts.googleapis.com` および `fonts.gstatic.com` もリクエストします。両方のホストはオプションです。それらをブロックする場合、artifact はフォールバックタイプフェイスでレンダリングされます。フォントリクエストが即座に失敗するように、高速拒否でブロックしてください。ページの最初のレンダリングを遅延させるのではなく。

Artifact は、React またはチャートパッケージなどの JavaScript ライブラリを `cdnjs.cloudflare.com`、`cdn.jsdelivr.net`、`cdn.tailwindcss.com`、`code.jquery.com`、および `unpkg.com` から読み込むこともできます。他の外部ホストからは読み込みません。それらのホストをブロックする場合、ライブラリに依存する artifact の部分は機能しません。ブロックされたフォントとは異なり、ブロックされたライブラリにはフォールバックがありません。ここでも高速拒否でブロックしてください。ブロックされたライブラリリクエストが即座に失敗するように。タイムアウトするまでハングするのではなく。

9.8.5 認証と SSO

Team および Enterprise 組織はすべてのユーザーに SSO を要求できます。プランレベルの詳細については**認証** (ページ 851) を参照し、Setting up SSO で SAML 設定を参照してください。OIDC セットアップは Claude Enterprise Administrator Guide で説明されています。

9.8.6 データ処理

Claude Code はローカルセッションではコードをローカルで処理するか、クラウドセッションでは Anthropic が管理するインフラストラクチャで処理します。組織がそれらを自己ホスト環境にルーティングしない限り。クラウドセッション（自己ホスト環境を含む）は、処理のために会話とコードコンテキストを Anthropic の API に送信します。ローカルおよび SSH セッションは、デプロイメントが設定する**モデルプロバイダー** (ページ 168)（デフォルトでは Anthropic の API）に送信します。データ保持、プライバシー、およびコンプライアンスの詳細については、**データ処理** (ページ 1105) を参照してください。

9.8.7 デプロイメント

Desktop はエンタープライズデプロイメントツールを通じて配布できます：

- **macOS**：Jamf または Kandji などの MDM を使用して `.dmg` インストーラーを通じて配布します
- **Windows**：MSIX パッケージを通じてデプロイします。サイレントインストールを含むエンタープライズデプロイメントオプションについては、Deploy Claude Desktop for Windows を参照してください。

ファイアウォールで許可リストに登録するドメインについては、上記の**ネットワークアクセス要件** (ページ 164) を参照してください。プロキシ設定、カスタム認証局、および LLM ゲートウェイについては、**ネットワーク設定** (ページ 897) を参照してください。

完全なエンタープライズ設定リファレンスについては、エンタープライズ設定ガイドを参照してください。

9.9 CLI から移行する場合

Claude Code CLI を既に使用している場合、Desktop はグラフィカルインターフェイスで同じ基盤となるエンジンを実行します。同じマシン上で、同じプロジェクト上でも、両方を同時に実行できます。各々は独自のセッションリストを保持し、CLI セッションを Desktop に持ち込むことができます。CLAUDE.md ファイルを通じて設定とプロジェクトメモリを共有します。

CLI セッションを Desktop に移動するには、ターミナルで `/desktop` を実行します。Claude はセッションを保存し、デスクトップアプリで開いてから CLI を終了します。このコマンドは、Claude サブスクリプションでサインインしている場合、macOS と x64 Windows で利用可能です。API キー認証、Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry では利用できません。

シェルから、*`claude --desktop`* (ページ 51) は Desktop を直接開き、ターミナルセッションを開始しません。Claude Code v2.1.285 以降が必要で、`/desktop` と同じプラットフォームおよびサインイン要件があります。他の引数がない場合、現在のディレクトリで Desktop を開きます。既存の CLI セッションを Desktop で開くには、このディレクトリの最新の会話に対して `--continue` を追加するか、`/status` が表示するセッション ID で `--resume` を追加します：

```
claude --desktop --resume <session-id>
```

Claude Code は `Opening session <session-id> in Claude Desktop` を出力し、セッションはアプリで開き、コマンドは終了します。セッション名は ID の代わりに機能しません。Claude Code は別のターミナルで開いているセッションや、バックグラウンドで実行中のセッションを移動しません。Claude Desktop がインストールされていない場合、コマンドはダウンロードリンクを出力して終了します。

Desktop 内から CLI セッションを再開することもできます。`/resume` を使用します。このコマンドはローカルセッションで利用可能で、SSH、WSL、またはクラウドセッションでは利用できません。

Desktop でターミナルセッションを続行するには：

1. ターミナルでセッションを閉じます。
2. Desktop のプロンプトボックスで `/resume` と入力します。Desktop は、このコンピュータで CLI から開始したセッションをリストアップします。タイトル、フォルダ、またはブランチで検索し、各セッションがどこで終了したかをプレビューします。
3. セッションを選択します。アプリで完全な会話とコンテキストとともに続行されます。

Desktop は同じセッションを続行するため、コピーではなく、ターミナルで `claude --resume` を実行すると、その後も見つかります。

Tip:

Desktop と CLI をいつ使用するか：複数のセッションを 1 つのウィンドウで管理したい場合、ペインを並べて配置したい場合、または変更を視覚的に確認したい場合は Desktop を使用します。スクリプト、自動化が必要な場合、またはターミナルワークフローを好む場合は CLI を使用します。

9.9.1 CLI フラグの同等物

このテーブルは、一般的な CLI フラグの Desktop アプリの同等物を示しています。リストされていないフラグは、スクリプトまたは自動化用に設計されているため、Desktop の同等物がありません。

CLI	Desktop の同等物
`--model sonnet`	送信ボタンの横のモデルドロップダウン
`--resume`、`--continue`	サイドバーのセッションをクリックするか、プロンプトボックスで `/resume` と入力して CLI から開始したセッションを選択します
`--permission-mode`	送信ボタンの横のモードセレクタ
`--dangerously-skip-permissions`	権限モードをバイパスします。Pro および Max プランでは、設定 → Claude Code →「権限モードのバイパスを許可」で有効にします。Team および Enterprise プランでは、組織ポリシーで制御されます
`--add-dir`	クラウドセッションで + ボタンで複数のリポジトリを追加します
`--allowedTools`、`--disallowedTools`	セッションごとの同等物はありません。**設定ファイル** (ページ 711) の権限ルールは引き続き適用されます。
`--verbose`	トランスクリプトビューのドロップダウンの**詳細ビューモード** (ページ 144)
`--print`、`--output-format`	利用できません。Desktop は対話的のみです。
`ANTHROPIC_MODEL` 環境変数	送信ボタンの横のモデルドロップダウン
`MAX_THINKING_TOKENS` 環境変数	ローカル環境エディタで設定します。**環境設定** (ページ 158) を参照してください。

9.9.2 共有設定

Desktop と CLI は同じ設定ファイルを読み込むため、セットアップが引き継がれます：

- プロジェクト内の *CLAUDE.md* (ページ 997) および `CLAUDE.local.md` ファイルは両方で使用されます
- `~/.claude.json` または `.mcp.json` で設定された *MCP* **サーバー** (ページ 363) は両方で機能します
- 設定で定義された *Hooks* (ページ 443) および *skills* (ページ 275) は両方に適用されます
- `~/.claude.json` および `~/.claude/settings.json` の **設定** (ページ 711) は共有されます。`settings.json` の権限ルール、許可されたツール、およびその他の設定は Desktop セッションに適用されます。
- **モデル**：同じ**モデル** (ページ 725) が両方で利用可能です。Desktop では、送信ボタンの横のドロップダウンからモデルを選択します。セッション中にも同じドロップダウンからモデルを変更できます。

Claude Desktop チャットアプリからの MCP サーバー

Desktop アプリは、`claude_desktop_config.json` から MCP サーバーをローカルコードタブセッションに読み込み、`~/.claude.json` および `.mcp.json` のサーバーと並行して実行します。`claude_desktop_config.json` で定義したサーバーは、Desktop チャットサーフェイスとローカルコードタブセッションの両方で利用可能です。

`claude_desktop_config.json` と `~/.claude.json` または `.mcp.json` で同じサーバー名を定義した場合、ローカルセッションのコードタブは 1 回接続し、`claude_desktop_config.json` の定義を使用します。

アプリは、`~/.claude.json` の stdio サーバーをローカルセッションの組み込み CLI に再配信します。`~/.claude.json`（ユーザースコープ）と `.mcp.json` のトップレベルが同じ stdio サーバー名を定義する場合、コードタブは `~/.claude.json` の定義を使用し、CLI の**スコープ階層** (ページ 383) から外れます。

注釈:

スタンドアロン CLI は `claude_desktop_config.json` を読み込みません。macOS と WSL では、`claude mcp add-from-claude-desktop` を実行して、それらのサーバーを `~/.claude.json` にコピーします。*Claude Desktop* **から** *MCP* **サーバーをインポート** (ページ 396) を参照して、インポートフローとスコープオプションを確認してください。

9.9.3 機能比較

このテーブルは、CLI と Desktop の主要機能を比較しています。CLI フラグの完全なリストについては、*CLI* **リファレンス** (ページ 45) を参照してください。

機能	CLI	Desktop
権限モード	`dontAsk` を含むすべてのモード	Manual、Accept edits、Plan、および Auto。権限モードのバイパスは、モードセレクタで有効にされると表示されます：Pro および Max プランでは設定トグルを通じて、Team および Enterprise プランでは組織ポリシーを通じて
サードパーティプロバイダー (ページ 847)	Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry	デフォルトで Anthropic の API。ゲートウェイルーティングについては、Desktop アプリをゲートウェイに接続を参照してください。Code タブを Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry、または自己ホスト型 LLM ゲートウェイで実行するには、Claude Desktop on 3P を参照してください。
MCP **サーバー** (ページ 363)	設定ファイルで設定	ローカルおよび SSH セッションのコネクタ UI、または設定ファイル
プラグイン	`/plugin` コマンド	プラグインマネージャー UI

次のページに続く

表 8 – 前のページからの続き

機能	CLI	Desktop
@mention ファイル	テキストベース	オートコンプリート付き。ローカルおよび SSH セッションのみ
ファイル添付	利用できません	画像、PDF
セッション分離	`--worktree` (ページ 45) フラグ	セッション開始時の **worktree** オプション
複数セッション	別々のターミナル	サイドバータブ
定期的なタスク	Cron ジョブ、CI パイプライン	スケジュール済みタスク
コンピュータ使用	macOS で `/mcp` で有効化	macOS および Windows で**アプリとスクリーン制御** (ページ 146)
iOS シミュレータ	コンピュータ使用経由でシミュレータを駆動	iOS シミュレータペインが自動的に開きます
Dispatch 統合	利用できません	サイドバーの *Dispatch* **セッション** (ページ 151)
スクリプトと自動化	`--print` (ページ 45)、*Agent SDK* (ページ 695)	利用できません

9.9.4 Desktop で利用できない機能

以下の機能は Desktop では利用できません（記載されている場合を除く）：

- **サードパーティプロバイダー**：Desktop はデフォルトで Anthropic の API に接続します。Desktop をゲートウェイ経由でルーティングするか、Code タブを Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry、または自己ホスト型 LLM ゲートウェイで実行するには、**サードパーティプロバイダー行** (ページ 168) のリンクに従ってください。
- **Linux（ベータ）**：Linux デスクトップアプリではコンピュータ使用はまだ利用できません。Claude Desktop on Linux を参照してください。
- **インラインコード提案**：Desktop はオートコンプリートスタイルの提案を提供しません。会話型プロンプトと明示的なコード変更を通じて機能します。
- **エージェントチーム**：Claude がチームリーダーとして共有タスクリストからチームメイトにタスクを割り当てる調整されたチームは、CLI で利用可能で、Desktop では利用できません。1 つのセッション内でマルチエージェント作業を行う場合は、Desktop で実行される動的ワークフローを使用します。Claude は**他のセッションに直接メッセージを送信して管理** (ページ 149) することもできます。
- **ターミナルダイアログコマンド**：ターミナルで対話的パネルを開く組み込みコマンドは、コードタブで異なる動作をします。**設定ファイル** (ページ 711) を直接編集して権限ルールと設定を管理するか、スタンドアロン CLI からコマンドを実行します。
  - `/permissions` などの引数形式がないコマンドは、`isn't available in this environment` で応答します。
  - `/config` は設定 → Claude Code を開きます。コマンド後のテキストは無視されるため、`/config theme=dark` はテーマを設定しません。

9.10 トラブルシューティング

以下のセクションでは、デスクトップアプリに固有の問題について説明します。チャットに表示される `API Error: 500`、`529 Overloaded`、`429`、または `Prompt is too long` などのランタイム API エラーについては、エラーリファレンスを参照してください。これらのエラーとその修正は、CLI、Desktop、Web 全体で同じです。

9.10.1 バージョンを確認する

実行しているデスクトップアプリのバージョンを確認するには：

- **macOS**：メニューバーの **Claude** をクリックしてから、**About Claude** をクリック
- **Windows**：**Help** をクリックしてから、**About** をクリック

バージョン番号をクリックしてクリップボードにコピーします。

9.10.2 Code タブの 403 またはエラー認証エラー

Code タブを使用するときに `Error 403: Forbidden` またはその他の認証エラーが表示される場合：

1. アプリメニューからサインアウトして再度サインインします。これが最も一般的な修正です。
2. アクティブな有料サブスクリプション（Pro、Max、Team、または Enterprise）があることを確認します。
3. CLI は機能するが Desktop は機能しない場合、デスクトップアプリを完全に終了し（ウィンドウを閉じるだけではなく）、再度開いてサインインします。
4. インターネット接続とプロキシ設定を確認します。

9.10.3 起動時に空白または停止画面

アプリが開いても空白または応答しない画面が表示される場合：

1. アプリを再起動します。
2. 保留中の更新を確認します。macOS と Windows ではアプリは起動時に自動更新されます。Linux では、Claude Desktop on Linux で説明されているように apt を使用して更新します。
3. マネージドネットワーク上では、ファイアウォールが**ネットワークアクセス要件** (ページ 164) の CDN ホストを許可していることを確認します。
4. Windows では、**Windows Logs** → **Application** の Event Viewer でクラッシュログを確認します。

9.10.4 「Failed to load session」

`Failed to load session` が表示される場合、選択したフォルダが存在しなくなった可能性があります。Git リポジトリがインストールされていない Git LFS を必要とする可能性があります。またはファイル権限がアクセスを防ぐ可能性があります。別のフォルダを選択するか、アプリを再起動してみてください。

9.10.5 セッションがインストール済みツールを見つけられない

Claude が `npm`、`node`、またはその他の CLI コマンドなどのツールを見つけられない場合、ツールが通常のターミナルで機能することを確認し、シェルプロファイルが PATH を正しく設定していることを確認し、デスクトップアプリを再起動して環境変数を再度読み込みます。

9.10.6 Git および Git LFS エラー

独自の worktree で実行されるセッションには Git が必要です。「Git is required」が表示される場合、Git をインストールするか、Windows では Git for Windows をインストールして、もう一度試してください。Windows では、1.49585.0 より前の Claude Desktop バージョンはローカルセッションを開始する前に Git を要求していました。そのプロンプトが表示され、worktrees を使用していない場合は、アプリを更新してください。

「Git LFS is required by this repository but is not installed」が表示される場合、git-lfs.com から Git LFS をインストールし、`git lfs install` を実行してアプリを再起動します。

9.10.7 Windows で MCP サーバーが機能しない

MCP サーバートグルが応答しない場合、または Windows でサーバーが接続に失敗する場合、サーバーが設定で正しく設定されていることを確認し、アプリを再起動し、Task Manager でサーバープロセスが実行されていることを確認し、接続エラーについてサーバーログを確認します。

9.10.8 アプリが終了しない

- **macOS**：Cmd+Q を押します。アプリが応答しない場合、Cmd+Option+Esc で Force Quit を使用し、Claude を選択して Force Quit をクリックします。
- **Windows**：Ctrl+Shift+Esc で Task Manager を使用して Claude プロセスを終了します。

9.10.9 Windows 固有の問題

- **インストール後に PATH が更新されない**：新しいターミナルウィンドウを開きます。PATH の更新は新しいターミナルセッションにのみ適用されます。
- **同時インストールエラー**：別のインストールが進行中であるというエラーが表示されるが、実際には進行中でない場合、インストーラーを管理者として実行してみてください。

9.10.10 CLI で開くときに「Branch doesn't exist yet」

クラウドセッションはローカルマシンに存在しないブランチを作成できます。セッションツールバーのブランチ名をクリックしてコピーしてから、ローカルでフェッチします：

```
git fetch origin <branch-name>
git checkout <branch-name>
```

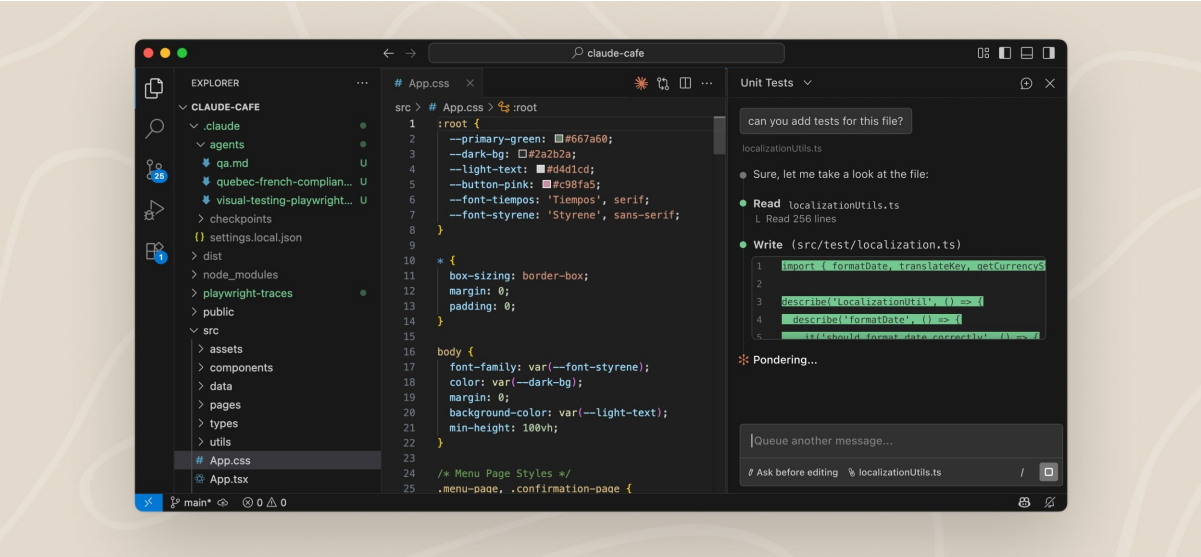
9.10.11 まだ立ち往生していますか？

- デスクトップアプリで Help → Get Support を開くか、Claude サポートセンターに直接アクセスします
- スタンドアロン `claude` CLI でも再現される問題については、GitHub Issues でバグを検索またはファイルします

問題を報告するときは、デスクトップアプリのバージョン、オペレーティングシステム、正確なエラーメッセージ、および関連ログを含めます。macOS では Console.app を確認します。Windows では Event Viewer → Windows Logs → Application を確認します。ログの抜粋を公開イシューに投稿する前に確認してください。環境からのファイルパスおよび他の詳細が含まれる可能性があります。

第10章　VS Code で Claude Code を使用する

Claude Code 拡張機能を VS Code にインストールして設定します。インラインの差分表示、@-メンション、プラン確認、キーボードショートカットを使用した AI コーディング支援を取得します。



VS Code 拡張機能は、Claude Code 用のネイティブグラフィカルインターフェースを提供し、IDE に直接統合されています。これは VS Code で Claude Code を使用する推奨方法です。

この拡張機能を使用すると、Claude のプランを受け入れる前に確認および編集でき、編集が行われるときに自動的に受け入れることができ、選択範囲から特定の行範囲を持つファイルを @-メンションでき、会話履歴にアクセスでき、複数の会話を別々のタブまたはウィンドウで開くことができます。

10.1 前提条件

インストール前に、以下があることを確認してください。

- VS Code 1.94.0 以上
- Anthropic アカウント：任意の有料 Claude サブスクリプション（Pro、Max、Team、または Enterprise）または Claude Console アカウントが機能し、API キーは不要です。拡張機能を初めて開くときに、このアカウントで**サインイン** (ページ 851) します。Amazon Bedrock や Google Cloud の Agent Platform などのサードパーティプロバイダーを通じて Claude にアクセスする場合は、セットアップ手順について**サードパーティプロバイダーを使用する** (ページ 199) を参照してください。

Tip:

拡張機能には、チャットパネル用の CLI（コマンドラインインターフェース）の独自コピーが含まれています。VS Code の統合ターミナルで `claude` を実行するには、**スタンドアロン *CLI* インストール** (ページ 19) も必要です。詳細については、*VS Code* **拡張機能と** *Claude Code CLI* (ページ 196) を参照してください。

10.2 拡張機能をインストールする

IDE のリンクをクリックして直接インストールします。

- VS Code 用にインストール
- Cursor 用にインストール

または、VS Code で `Cmd+Shift+X`（Mac）または `Ctrl+Shift+X`（Windows/Linux）を押して拡張機能ビューを開き、「Claude Code」を検索して、**インストール**をクリックします。

拡張機能のバージョン番号は、同梱されている Claude Code のバージョンです。たとえば、Claude Code v2.1.286 以降を必要とする機能には、拡張機能のバージョン 2.1.286 以降が必要です。拡張機能のバージョンは拡張機能ビューに表示されます。

拡張機能は Devin Desktop や Kiro などの他の VS Code フォークにもインストールされます。エディタの拡張機能ビューで「Claude Code」を検索するか、Open VSX レジストリからインストールしてください。エディタが拡張機能をインストールできない場合は、*CLI* (ページ 11) をインストールして、統合ターミナルで `claude` を実行してください。CLI はどのターミナルでも動作します。

注釈:

インストール後に拡張機能が表示されない場合は、VS Code を再起動するか、コマンドパレットから「Developer: Reload Window」を実行してください。

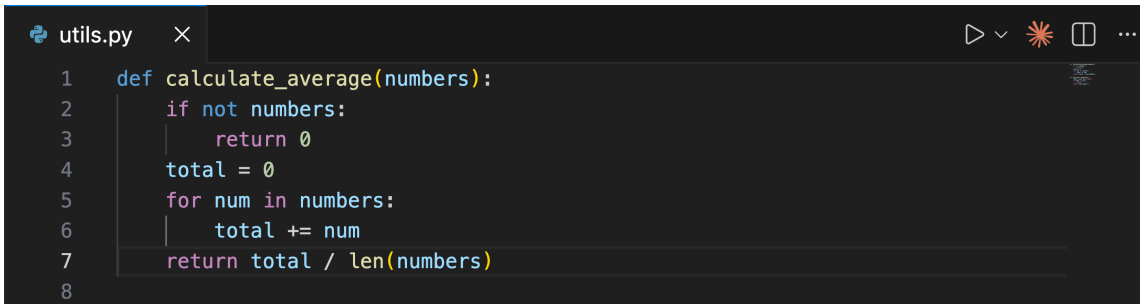
10.3 はじめに

インストール後、VS Code インターフェースを通じて Claude Code の使用を開始できます。

1. **Claude Code パネルを開く**

   VS Code 全体で、Spark アイコンは Claude Code を示します。✳

   Claude を開く最も速い方法は、**エディタツールバー**（エディタの右上隅）の Spark アイコンをクリックすることです。このアイコンはファイルを開いている場合にのみ表示されます。



Claude Code を開くその他の方法：

- **アクティビティバー**：左サイドバーの Spark アイコンをクリックしてセッションリストを開きます。任意のセッションをクリックして**優先位置** (ページ 193) で開くか、新しいセッションを開始します。このアイコンはアクティビティバーに常に表示されます。
- **コマンドパレット**：`Cmd+Shift+P`（Mac）または `Ctrl+Shift+P`（Windows/Linux）を押し、「Claude Code」と入力して、「Open in New Tab」などのオプションを選択します。
- **ステータスバー**：ウィンドウの右下隅の ☒ **Claude Code** をクリックします。ファイルが開いていない場合でも機能します。

Claude パネルをドラッグして VS Code 内の任意の場所に移動できます。詳細は**ワークフローをカスタマイズする** (ページ 184) を参照してください。

2. **サインイン**

   パネルを初めて開くと、サインイン画面が表示されます。**Sign in** をクリックしてブラウザで認可を完了します。

   後で **Not logged in · Please run /login** が表示される場合、拡張機能はサインイン画面を自動的に再度開きます。表示されない場合は、コマンドパレットから **Developer: Reload Window** でウィンドウをリロードします。

   シェルに `ANTHROPIC_API_KEY` が設定されているのにサインインプロンプトが表示される場合、VS Code がシェル環境を継承していない可能性があります。ターミナルから `code .` で VS Code を起動して環境変数を継承するか、代わりに Claude アカウントでサインインしてください。

   サインイン後、**Learn Claude Code** チェックリストが表示されます。**Show me** をクリックして各項目を実行するか、X で閉じます。後で再度開くには、VS Code 設定の Extensions → Claude Code で **Hide Onboarding** をオフにします。

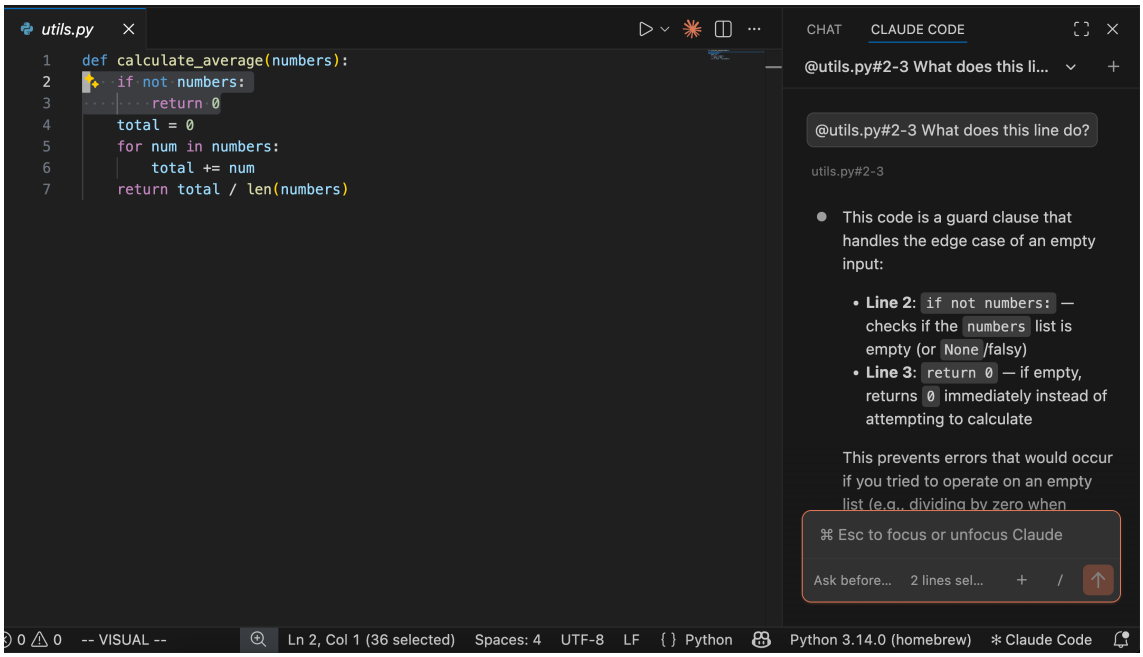
3. **プロンプトを送信する**

   コードやファイルについて Claude に支援を求めます。これは、何かの仕組みを説明すること、問題をデバッグすること、または変更を加えることなど、様々なことが含まれます。

Tip:

Claude は自動的に選択したテキストを認識します。`Option+K`（Mac）/ `Alt+K`（Windows/Linux）を押して、@-mention 参照（`@file.ts#5-10` など）をプロンプトに挿入することもできます。

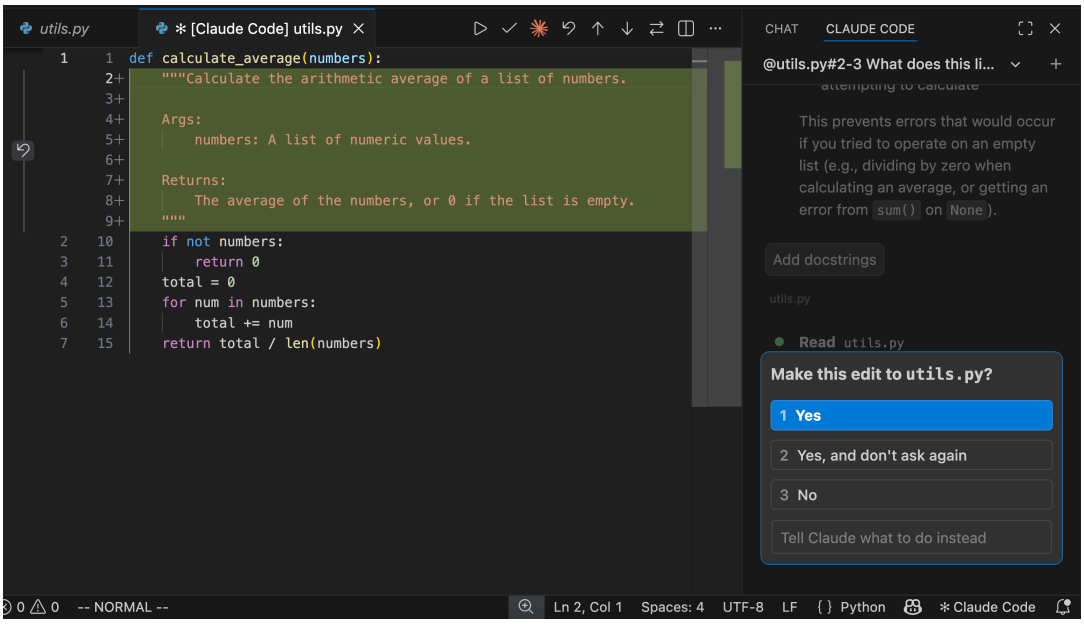
ファイル内の特定の行について質問する例を以下に示します。



4. **変更を確認する**

表示内容は、プロンプトボックスの下部に表示される権限モードによって異なります。

- Auto または Edit automatically モードでは、Claude はワークスペース内のほとんどのファイルを確認なしで編集します。
- Manual モードでは、Claude がファイルを編集したい場合、元のコンテンツと提案された変更の並べて比較を表示し、権限を求めます。受け入れたり、拒否したり、代わりに Claude に何をするかを指示したりできます。受け入れる前に diff ビューで提案されたコンテンツを直接編集した場合、Claude はそれを修正したことが通知されるため、ファイルが元の提案と一致していると想定されません。



提案された編集を 1 つずつ確認するには、diff 内の各変更の下にある **Accept this change** および **Reject this change** ボタンを使用します。変更を拒否すると、提案されたコンテンツ内でそれが元に戻され

ます。受け入れると、確認済みとしてマークされます。ファイル全体を受け入れるか拒否すると、確認が完了します。100 を超える変更を含む diff は、変更ごとのボタンなしで開かれるため、ファイル全体として確認してください。変更ごとの確認には Claude Code v2.1.275 以降が必要です。

同じアクションは、エディタのコンテキストメニューおよびコマンドパレットから **Claude Code: Accept Change at Cursor** および **Claude Code: Reject Change at Cursor** として利用できます。

Claude Code でできることについてのアイデアについては、**一般的なワークフロー** (ページ 105) を参照してください。

Tip:

コマンドパレットから「Claude Code: Open Walkthrough」を実行して、基本的な機能のガイド付きツアーを表示します。

10.4 プロンプトボックスを使用する

プロンプトボックスは複数の機能をサポートしています。

- **権限モード**: プロンプトボックスの下部にあるモード表示をクリックして、権限モードを切り替えます。Claude Code v2.1.283 以降では、Auto が組み込みの開始権限モードであり、それより前のバージョンでは Pro、Max、Team プランのみです。拡張機能が開始権限モードを選択する方法と、表示されるすべての権限モードについては、こちらをご覧ください。
  - **Auto**: ほとんどのアクションについて、分類器があなたに尋ねる代わりに確認します。自動モードで、確認およびブロックされる内容をご覧ください。
  - **Manual**: Claude はファイル編集とほとんどのシェルコマンドの前に権限を求めます。
  - **Plan**: Claude は変更を加える前に、実行内容を説明し、承認を待ちます。VS Code は計画を完全な Markdown ドキュメントとして自動的に開き、Claude が開始する前にフィードバックを提供するためのインラインコメントを追加できます。

    プロンプトボックスで `/plan` を入力することもできます。Claude Code v2.1.280 以降が必要です。

    * `/plan`: 計画モードに切り替えます。既に計画モードの場合は、現在の計画を表示します。
    * `/plan` とタスク（例：`/plan fix the auth bug`）: 計画モードに切り替えて、そのタスクの計画を開始します。
    * `/plan open`: 既に計画モードの場合、エディターで計画ファイルを開きます。
  - **Edit automatically**: Claude は確認を求めずに編集を行います。
- **Model**: コマンドメニューから **Switch model…** を選択して、セッション中にモデルを変更します。プロンプトボックスの下部にあるモデル名をクリックして、同じピッカーを開くこともできます。Claude Code v2.1.284 以降では、プロンプトボックスで `/model` のみを入力してもピッカーが開きます。

  現在のモデルが**努力レベル** (ページ 747) をサポートしている場合、ピッカーは **Effort** 行も表示し、モデル名ボタンは選択されたレベルを表示します。`max` 以外のレベルを選択すると、Claude Code はそれを現在のモデルのデフォルトとしてユーザー設定の `modelSettings` に保存します。`max` は現在の

セッションにのみ適用されます。モデル名ボタンと **Effort** 行には Claude Code v2.1.257 以降が必要です。

動的ワークフローが有効になっており、現在のモデルがそれをサポートしている場合、**Effort** 行の下に **Ultracode** スイッチが表示されます。オンにして、Claude がこのセッションの各実質的なタスクに対して、選択された努力レベルでワークフローを計画するようにします。オンの間、モデル名ボタンはレベルの後に・ `Ultracode` を表示します。スイッチには Claude Code v2.1.284 以降が必要です。

- **コマンドメニュー**: / をクリックするか / を入力してコマンドメニューを開きます。オプションには、ファイルの添付、モデルの切り替え、拡張思考の切り替えが含まれます。

  Customize セクションは、MCP サーバー、コマンド、出力スタイル、hooks、メモリ、instructions、権限、プラグインへのアクセスを提供します。ターミナルアイコンが付いているアイテムは統合ターミナルで開きます。

  - Customize セクションで **Slash commands** を選択して、`/usage` や `/remote-control` などのコマンドを参照します。ダイアログにはフィルターボックス付きでそれらが一覧表示されます。実行するコマンドを選択します。プロンプトボックスで / を入力すると、引き続きコマンドがインラインで提案されます。Claude Code v2.1.257 以降が必要です。

    また、`/skills` を入力してもこのダイアログが開きます。各**スキル** (ページ 275) 行は、**On** や **Name only** などの**表示設定** (ページ 306) を表示します。表示設定をクリックして変更できます。ただし、プラグインスキルなど **locked** とマークされた行は除きます。`/skills` ショートカットと表示設定コントロールには Claude Code v2.1.280 以降が必要です。

  - Customize セクションで **Output styles** を選択して、カスタムスタイルを含む**出力スタイル** (ページ 821) を選択します。Claude Code v2.1.257 以降が必要です。

    代わりにカスタムスタイルを作成するには、**Output styles** メニューから **Build a custom style** を選択します。Claude Code はプロジェクトまたはユーザーレベルで**スタイルファイル** (ページ 824) を作成します。Claude Code v2.1.261 以降が必要です。

  - Customize セクションで **Hooks** を選択して、セッションに読み込まれた *hooks* (ページ 443) をイベント別にグループ化して表示します。ユーザー、プロジェクト、ローカル設定ファイルに保存された hooks を追加、編集、または削除できます。マネージド設定やプラグインなどの他のソースからの Hooks は読み取り専用です。Claude Code v2.1.269 以降が必要です。

  - Customize セクションで **Permissions** を選択して、セッションの**権限ルール** (ページ 1039) を Allow、Ask、Deny にグループ化して表示します。ユーザー、プロジェクト、またはローカル設定にルールを追加し、そこに保存されたルールを削除できます。マネージド設定やこのセッションのみの承認など、他のソースからのルールは読み取り専用です。Claude Code v2.1.269 以降が必要です。

  - Customize セクションで **Memory** を選択して、**自動メモリ** (ページ 1013) をオンまたはオフにします。オンの場合、Claude が保存したメモリを参照し、それらを保存するフォルダをファイルマネージャーで表示することもできます。Claude Code v2.1.274 以降が必要です。

    保存されたメモリをクリックしてダイアログで読み取り、テキストを編集したり、メモリを削除したり、そのファイルをエディターで開いたりできます。ダイアログでメモリを表示、編集、削除するには Claude Code v2.1.275 以降が必要です。

  - Customize セクションで **Instructions** を選択して、Claude が読む *CLAUDE.md* **ファイル** (ページ 998) を編集します。ファイルを選択してエディターで開きます。ファイルがまだ存在しない

場合、Claude Code は最初にそれを作成します。Claude Code v2.1.274 以降が必要です。

- Customize セクションで **Status** を選択するか、`/status` を入力して、セッションの Claude Code バージョン、アカウント、モデル、MCP サーバーの詳細を確認します。Claude Code v2.1.280 以降が必要です。
- Customize セクションで **Sandbox** を選択するか、`/sandbox` を入力して、Claude の Bash コマンドが**サンドボックス化** (ページ 1069) されているかどうかを確認します。サンドボックスモードを切り替えたり、除外コマンドを追加したりできます。Claude Code v2.1.280 以降が必要です。
- Customize セクションで **Claude in Chrome** を選択するか、`/chrome` を入力して、*Claude in Chrome* (ページ 213) 接続を確認および管理します。どちらも claude.ai アカウントでサインインする必要があります。Claude Code v2.1.280 以降が必要です。
- Context セクションで **Export conversation** を選択するか、`/export` を入力して、会話をプレーンテキストとしてコピーするか、ファイルに保存します。`/export notes.txt` などのファイル名を追加して、ダイアログをスキップし、ファイルの保存場所を選択します。Claude Code v2.1.280 以降が必要です。
- Settings セクションには **Enable Remote Control for all sessions** が含まれており、これは `remoteControlAtStartup` を設定して、新しいインタラクティブセッションが Remote Control に自動的に接続するかどうかを制御します。Claude Code v2.1.203 以降が必要です。

  VS Code ウィンドウでトグルをオンまたはオフにすると、その変更は、その後に開始するセッションだけでなく、その VS Code ウィンドウで既に開いているセッションに適用されます。オフにすると、開いているセッションは切断されます。Claude Code v2.1.261 以降では、その変更は他の VS Code ウィンドウで開いているセッションにも適用されます。
- Settings セクションには **Focus view** も含まれており、これはツール呼び出し、ツール結果、思考を展開可能な行の背後に隠し、プロンプトと Claude の応答を残します。そこで切り替えるか、`Ctrl+Option+F`（Mac）/ `Ctrl+Alt+F`（Windows/Linux）で切り替えるか、コマンドパレットから **Claude Code: Toggle Focus view** で切り替えます。変更はすべての開いているセッションに適用され、セッション全体で保持されます。Claude Code v2.1.221 以降が必要です。

  Claude の最新のやることリストは表示されたままになり、Claude からの保留中の質問が尋ねているテキストも表示されたままになります。これには Claude Code v2.1.225 以降が必要です。Claude が**サブエージェント** (ページ 317) を実行している間、最新のアクティビティを含むライブ進捗行が、それらを開始したツール呼び出しグループの下に表示されます。これには Claude Code v2.1.269 以降が必要です。
- Anthropic アカウントからサインアウトするには、Settings セクションで **Sign out** を選択するか、`/logout` を入力します。**サードパーティプロバイダー** (ページ 199) では、メニューはどちらも提供しません。Claude Code v2.1.277 以降が必要です。
- バグを報告するには、メニューの下部にある **Report a problem** をクリックするか、`/bug` または `/feedback` をオプションの説明と共に入力して、レポートに事前入力します。レポートを送信し、ファーストパーティ接続で Anthropic にサインインしている場合、Claude Code はそれを Anthropic に送信します。Claude Code v2.1.229 以降が必要です。

  サードパーティプロバイダーまたは Anthropic 認証情報がない場合、何も送信されません。ダイアログは送信前にそのことを示します。レポートを送信すると、既知の API キーとトークンパターンが削除された状態で、*~/.claude/feedback-bundles/* **の下のローカルアーカイブ** (ページ 1109)

として保存されます。そのファイルを Anthropic アカウント担当者に送信するか、サポートリクエストに添付してください。確認ではファイル名が表示され、**Show folder** ボタンが含まれます。コンピューターにレポートを保存するには Claude Code v2.1.284 以降が必要です。

組織のポリシーが製品フィードバックをオフにしている場合、**Report a problem** はメニューに表示されず、`/bug` と `/feedback` はレポートを開く代わりに `Feedback is turned off by your organization's policy or this environment's settings.` という通知を表示します。Claude Code v2.1.284 以降では、`DISABLE_FEEDBACK_COMMAND` または `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` 環境変数を設定した場合、フィードバックもオフになり、レポートを開くとその通知が表示されます。

- **サイドクエスチョン**: `/btw` の後に質問を入力して、**会話に追加せずに** (ページ 98) セッションについて質問します。答えはチャットの横のパネルで開き、そこでフォローアップの質問をすることができます。スレッドはウィンドウの再読み込みを通じて保持されます。Claude Code は最新の 20 回の交換を保持し、Claude Code が保持期間を安全に判断できる限り、`cleanupPeriodDays` スケジュールに従って保存されたスレッドを期限切れにします。スレッドをクリアするには、パネルのゴミ箱アイコンをクリックします。Claude Code v2.1.227 以降が必要です。

- **応答をコピー**: 応答にマウスを置いて **Copy response** をクリックしてクリップボードにコピーするか、`/copy` を入力して最新の応答をコピーします。`/copy 2` は 2 番目に新しい応答をコピーします。Claude Code v2.1.277 以降が必要です。

- **ブックマーク**: 応答にマウスを置いて **Bookmark response** をクリックして保存するか、保存された応答で **Remove bookmark** をクリックして削除します。

  保存された応答を確認するには、Bookmarks パネルを開きます。Claude Code パネルの上部にあるブックマークアイコンをクリックするか、コマンドメニューの Context セクションで **Bookmarks** を選択するか、`/bookmarks` を入力します。Claude Code v2.1.286 以降が必要です。

- **コンテキスト表示**: プロンプトボックスは Claude のコンテキストウィンドウをどの程度使用しているかを表示します。Claude は必要に応じて自動的にコンパクトにするか、`/compact` を手動で実行できます。

- **プロンプトキャッシュクロック**: コンテキスト表示の横にある時計アイコンは、会話のプロンプトキャッシュが期限切れになるまでの時間を推定します。キャッシュの 5 分または 1 時間の有効期限からカウントダウンし、キャッシュを使用する各応答がカウントダウンを再開します。コンパクション以外に、キャッシュを無効にするアクションはクロックをリセットしないため、モデルを切り替えた後も残り時間を表示できます。

  - カウントダウンが終了するまで、アイコンは **12m** などの残り時間を表示します。
  - カウントダウンが終了すると、分が消え、アイコンは赤くなるか、次の応答までテーマのエラーカラーになります。キャッシュは期限切れの可能性があるため、次のメッセージへの応答が遅くなり、キャッシュが再構築される間、より高くなることを予想してください。5 分の有効期限がメッセージ間で実行され続ける場合は、TTL を自分で選択を参照してください。
  - 会話がコンパクト化された直後、アイコンは次の応答までの分なしで赤くなります。キャッシュはまだコンパクト化された会話をカバーしていないためです。

- **エージェントマップ**: 会話に**サブエージェント** (ページ 317) が含まれている場合、プロンプトボックスの下部に **2 agents** などのエージェント数が表示されます。そのドットは、サブエージェントが動作しているか、権限を待っているかを示します。

エージェント数をクリックしてエージェントマップを開きます。これは会話のサブエージェントをメインエージェントの下のツリーとして描画し、それぞれのステータス、経過時間、トークン数を表示します。サブエージェントをクリックして、そのプロンプトとツール呼び出しを表示したり、読み取り専用トランスクリプトを開いたり、実行中に停止したりできます。Claude Code v2.1.269 以降が必要です。

マップは、バックグラウンドシェルコマンドやモニターなどのセッションの他のバックグラウンドタスクもエージェントの下に一覧表示します。行をクリックしてタスクのカードを開き、そこで停止します。

エージェント数が表示されていない場合（Claude がバックグラウンドシェルを開始したがサブエージェントがない場合など）にマップを開くには、プロンプトボックスで `/tasks` を入力します。マップ内のバックグラウンドタスクと入力された `/tasks` には Claude Code v2.1.277 以降が必要です。

Claude Code v2.1.286 以降では、**Stop** をクリックするか `Esc` を押すと、現在のターンが終了します。バックグラウンドエージェントは、完了するかマップから停止するまで実行を続けます。

- **拡張思考**: Claude が複雑な問題を推論するためにより多くの時間を費やすことができます。コマンドメニュー（/）経由でオンに切り替えます。Claude の推論は会話に折りたたまれたブロックとして表示されます。ブロックをクリックして読むか、`Ctrl+O` を押してセッション内のすべての思考ブロックを展開または折りたたみます。詳細については、**拡張思考** (ページ 751) を参照してください。
- **複数行入力**: `Shift+Enter` を押して、送信せずに新しい行を追加します。これは質問ダイアログの「その他」フリーテキスト入力でも機能します。

10.4.1 ファイルとフォルダを参照する

@-メンションを使用して、特定のファイルまたはフォルダに関するコンテキストを Claude に提供します。@ の後にファイルまたはフォルダ名を入力すると、Claude はそのコンテンツを読み取り、それについて質問に答えたり、変更を加えたりできます。Claude Code はあいまい一致をサポートしているため、部分的な名前を入力して必要なものを見つけることができます。

```
Explain the logic in @auth (fuzzy matches auth.js, AuthService.ts, etc.)
What's in @src/components/ (include a trailing slash for folders)
```

大きな PDF の場合、ファイル全体ではなく特定のページを読むよう Claude に依頼できます。単一ページ、1〜10 ページなどの範囲、またはページ 3 以降などのオープンエンド範囲。特定のページを読むには、Claude Code が実行されるマシンに poppler-utils が必要です。

エディターでテキストを選択すると、Claude は強調表示されたコードを自動的に表示できます。プロンプトボックスのフッターは、選択されている行数を表示します。`Option+K`（Mac）/ `Alt+K`（Windows/Linux）を押して、ファイルパスと行番号を含む @-メンション（例：`@app.ts#5-10`）を挿入します。選択表示の **X** をクリックして削除し、Claude が選択を受け取らないようにします。他のテキストを選択すると、表示が戻ります。

拡張機能は一部のファイルから選択されたテキストを保留します。ファイルがワークスペース内にあり、`files.exclude` または `search.exclude` 設定と一致する場合、Claude は最大でもファイルのパスを受け取り、選択したテキストは受け取りません。同じことが git が無視するファイルにも適用されます。ただし、VS Code の `search.useIgnoreFiles` 設定と拡張機能の *respectGitIgnore* **設定** (ページ 193) の両方がオンになっている場合に限ります。これはデフォルトです。このフィルターはチャットパネルのみをカバー

します。Claude Code が統合ターミナルで実行される場合、CLI はファイルに関係なく選択されたテキストを送信するため、*Read* **拒否ルール** (ページ 200) を追加して、ファイルのコンテンツが Claude にそこに到達しないようにします。

Claude はエディターで開いているファイルも表示します。何も選択されていない場合でも、プロンプトボックスはその名前を表示します。選択されたテキストのみを追加するには、Attach Open File 設定をオフにします。この設定には Claude Code v2.1.271 以降が必要です。

メッセージに画像とファイルを添付することもできます。

- 画像を添付するには、クリップボードからプロンプトボックスに貼り付けます。
- ファイルを添付するには、`Shift` を押しながらプロンプトボックスにドラッグします。
- コンテキストから添付ファイルを削除するには、その X をクリックします。

10.4.2 テキストを貼り付ける

貼り付けたテキストはプロンプトボックスに表示されたままになり、**ターミナル** (ページ 838) のようにプレースホルダーに折りたたまれません。Claude Code が**貼り付けたテキストをマーク** (ページ 838) するセッションでは、Claude は大きな貼り付けを入力したテキストではなく貼り付けたテキストとして認識します。

Claude Code は、プロンプトボックスに貼り付けたテキストと送信するその他すべてのテキストから**目に見えない** *Unicode* **文字** (ページ 96) も削除します。

- `Removed 3 invisible characters from the pasted text` などの通知が貼り付け時に表示される場合、テキストはそれらの文字なしで入力されました。
- 送信時に削除された文字に関する通知が表示される場合、何も送信されていません。クリーンなテキストはプロンプトボックスに戻ります。表示されたテキストを送信するために再度送信します。

10.4.3 過去の会話を再開する

Claude Code パネルの上部にある **Session history** ボタンをクリックして、会話履歴にアクセスします。キーワードで検索するか、時間で参照できます。

任意の会話をクリックして、完全なメッセージ履歴で再開します。会話が現在のウィンドウの別のタブで既に開いている場合、クリックするとそのタブに切り替わります。セッションの再開の詳細については、セッションを管理を参照してください。

- **セッションタイトル**: 新しいセッションは最初のメッセージに基づいて AI が生成したタイトルを受け取ります。
- **名前変更とアーカイブ**: セッションにマウスを置いて、これらのアクションを表示します。説明的なタイトルを付けるために名前を変更するか、リストの下部にある **Archived sessions** グループに移動するためにアーカイブします。

会話が別の Claude Code プロセス（ターミナルの `claude` や別の VS Code ウィンドウなど）で開いている場合、プロンプトボックスの代わりに次の通知が表示されます：`This conversation is still open somewhere else. Using it in two places at once can mix up its messages.` ここで続行するには、別の場所で会話を閉じてから **Open here anyway** をクリックします。閉じずにクリックすると、会話

は両方の場所で開いた状態になります。*claudeProcessWrapper* (ページ 193) が設定されている場合、拡張機能はこのチェックをスキップし、会話を直接開きます。

デフォルトでは、14 日間アクティビティがないセッションは、開いている、未読、または**グループ** (ページ 185) 内にない限り、自動的に **Archived sessions** に移動します。自動アーカイブには Claude Code v2.1.265 以降が必要です。期間を変更するか、オフにするには、Archive Inactive Sessions 設定を開き、日数または **Never** を選択します。

アーカイブされたセッションを復元するには、**Archived sessions** を展開して **Unarchive session** をクリックします。アーカイブされたすべてのセッションを一度に復元するには、セッションリストの Activity Bar で **Archived sessions** ヘッダーにマウスを置き、その復元アイコンをクリックします。これには Claude Code v2.1.277 以降が必要です。v2.1.257 より前では、アクションは **Delete session** でした。これはセッションを非表示にし、復元する方法がありませんでした。その後削除したセッションは、アップグレード後に **Archived sessions** の下に表示されます。

再開した会話が計画モードで終了した場合、Claude Code は計画モードを復元します。Claude Code v2.1.246 以降が必要です。Claude Code は 2 つのケースでは復元しません。

- 拡張機能が `claudeCode.initialPermissionMode` から開始権限モードを選択するか、以前の会話から引き継がれたピックがある
- `claudeCode.claudeProcessWrapper` が設定されている

10.4.4 Claude.ai からクラウドセッションを再開する

クラウドセッション (ページ 223) を実行している場合、VS Code で直接再開できます。これには **Claude.ai Subscription** でのサインインが必要です。Anthropic Console ではありません。

1. **セッション履歴を開く**

   Claude Code パネルの上部にある **Session history** ボタンをクリックします。

2. **Web タブを選択**

   ダイアログには 2 つのタブが表示されます。Local と Web です。**Web** をクリックして claude.ai からのセッションを表示します。

3. **再開するセッションを選択**

   セッションを参照または検索します。任意のセッションをクリックして会話をローカルで続行します。

注釈:

フォルダが GitHub リポジトリの場合、Web タブはそのリポジトリからのセッションのみを表示します。
クラウドセッションを再開すると、拡張機能は会話履歴のコピーをダウンロードします。変更は claude.ai に同期されません。

Web タブは、Remote Control セッションもリストします。開いているフォルダで実行されたセッションをクリックすると、拡張機能はコピーをダウンロードする代わりにそのローカル会話を開き、それを表示しているタブが既にある場合はそのタブにフォーカスします。拡張機能が別の Claude プロセスが会話を開いている可能性を除外できない場合、代わりにダウンロードされたコピーを取得します。

会話のいずれかの部分がダウンロードに失敗した場合、エラーが表示され、コピーは保存されません。セッションを再度選択して再試行します。ダウンロードする会話がまだないセッションを選択した場合、エラーはそれを続行する場所を示します。

10.4.5 アカウントと使用状況を確認する

/usage を実行して Account & usage ダイアログを開きます。サインインしているアカウントを表示し、報告される使用状況はサインインによって異なります。

- **claude.ai プラン**: 現在のセッションと週など、プランの制限の使用状況バー。各バーは、制限がリセットされるまでの時間を表示します。

  ダイアログは、プランの制限に寄与しているものも詳しく説明します。キャッシュミス、長いコンテキスト、サブエージェント集約型または高度に並列化されたセッションなど、最近の使用状況の 10% 以上を占める動作にフラグを立て、それぞれを削減するためのヒントを提供します。属性テーブルは、各スキル、サブエージェント、プラグイン、MCP サーバーからどの程度の使用状況が来たかを示します。

  Day と Week トグルを使用して、過去 24 時間と過去 7 日間を切り替えます。数値は概算であり、このマシン上のローカルセッションから計算されるため、他のデバイスまたは claude.ai からの使用状況は含まれません。

- **その他のサインイン**: **サードパーティプロバイダー** (ページ 199) または API キーなど、プランの制限がサインインに適用されない場合、Usage セクションはセッション自体のコストとトークン使用状況を代わりに表示します。CLI の /usage は**セッションブロック** (ページ 911) で同じ合計を表示します。Activity Bar のセッションリストは、**Account & usage** ヘッダーの下にアクティブセッションの合計も表示します。Claude Code v2.1.277 以降が必要です。

使用状況の追跡と削減の詳細については、**コストを追跡** (ページ 911) を参照してください。

10.5 ワークフローをカスタマイズする

Claude パネルの位置を変更したり、複数の会話を実行したり、セッションリストをグループに整理したり、ターミナルモードに切り替えたりできます。

10.5.1 Claude の配置場所を選択する

Claude パネルを VS Code 内の任意の場所に移動できます。パネルのタブまたはタイトルバーをつかんで、以下の場所にドラッグします。

- **セカンダリサイドバー**: ウィンドウの右側。コード作成中に Claude を表示したままにします。
- **プライマリサイドバー**: Explorer、Search などのアイコンがある左側のサイドバー。
- **エディタ領域**: Claude をファイルの横のタブとして開きます。サイドタスクに便利です。

Claude がエディタグループの新しいタブを開くと、拡張機能はそのグループをロックするため、Claude タブがフォーカスされている間に開いたファイルは別のグループに移動します。

拡張機能がグループをロックするのを停止するには、Lock Editor Groups 設定をオフにします。既にロックされているグループは、ロック解除するまでロックされたままです。この設定には Claude Code v2.1.274 以降が必要です。

Tip:

メイン Claude セッションにはサイドバーを使用し、サイドタスク用に追加タブを開きます。Claude は優先される場所を記憶します。アクティビティバーのセッションリストアイコンは Claude パネルとは別です。セッションリストは常にアクティビティバーに表示されますが、Claude パネルアイコンは左側のサイドバーにドッキングされている場合にのみそこに表示されます。

10.5.2 リロード後に会話を続行する

Developer: Reload Window を実行するか VS Code を再起動した後、チャットがその会話とともに戻るかどうかは、それがどこで開かれていたかによって異なります。

- **エディタタブ**: 会話はそのタブとともに戻ります。
- **サイドバー**: 過去 10 分以内にメッセージを送信したか Claude が応答した場合、会話は戻ります。戻らない場合は、**セッション履歴** (ページ 182) から会話を再開してください。

別の Claude Code プロセスがまだその会話を開いている場合は、ここで開く前に確認を求められます。その際には、**セッション履歴から再開する** (ページ 182) ときと同じ **Open here anyway** の通知が表示されます。

リロードが Claude の途中のステップを中断した場合、会話が戻ると Claude はそのステップを続行し、チャット内の通知が継続をマークします。Claude Code v2.1.274 以降が必要です。ステップが 1 時間以上前に中断されたか、セッションが別の場所で開かれている場合、会話はアイドル状態で戻ります。

継続をオフにするには、Continue After Reload 設定を開いてチェックを外します。VS Code の環境または *environmentVariables* **設定** (ページ 193) で `CLAUDE_CODE_RESUME_INTERRUPTED_TURN` やその他の `CLAUDE_CODE_RESUME_` 変数を設定しても、パネルでは効果がありません。拡張機能がパネルのセッションを開始する前にこれらの変数を削除するためです。

10.5.3 複数の会話を実行する

コマンドパレットから **Open in New Tab** または **Open in New Window** を使用して、追加の会話を開始します。各会話は独自の履歴とコンテキストを保持し、異なるタスクを並行して処理できます。

タブを使用する場合、スパークアイコンの小さなカラーの点がステータスを示します。青は権限リクエストが保留中であることを意味し、オレンジはタブが非表示の間に Claude が完了したことを意味します。

10.5.4 セッションをグループに整理する

アクティビティバーのセッションリストで、関連するセッションを名前付きの折りたたみ可能なグループに集めることができます。Claude Code v2.1.229 以降が必要です。

- **セッションをグループ化または非グループ化する**: セッションを右クリックしてそれからグループを作成したり、既存のグループに移動したり、グループから削除したりします。各セッションは一度に 1 つのグループに属するため、別のグループに移動すると最初のグループから削除されます。

- **複数のセッションを一度に移動する**: `Cmd` キー（Mac）/ `Ctrl` キー（Windows/Linux）を押しながら各セッションをクリックするか、`Shift` キーを押しながらクリックして範囲を選択してから、選択内容を右クリックします。
- **タブからセッションをグループ化する**: コマンドパレットから **Claude Code: Add Session Tab to Group** を実行してからグループを選択または作成します。Claude Code v2.1.257 以降が必要です。
- **グループの名前を変更または削除する**: グループヘッダーを右クリックします。グループを削除するとグループのみが削除され、そのセッションは非グループ化されたリストに戻ります。

拡張機能はワークスペースフォルダごとにグループを保存するため、ウィンドウの再読み込み後も保持され、同じフォルダを開くすべてのウィンドウに表示されます。リストを検索すると、拡張機能はすべてのグループ全体で 1 つのフラットリストに一致するものを表示します。

10.5.5 セッションリストをフィルタリングする

アクティビティバーの長いセッションリストを絞り込むには、リストの上部にある 2 つのフィルタコントロールを使用します。Claude Code v2.1.271 以降が必要です。アーカイブされたセッションは、どちらかのフィルタがオンの間は表示されません。

- **アクティブ**: このトグルをオンにして、入力が必要なセッション、処理中のセッション、未読のセッション、および最後にフォーカスした Claude タブのセッションのみを表示します。
- **ステータスでフィルタリング**: ファネルアイコンをクリックしてから、**入力が必要**、**処理中**、または **完了** をチェックして、これらの状態のセッションを表示します。**開く** または **閉じる** をチェックして、セッションが開いているかどうかで絞り込みます。セッションは、このウィンドウにタブがある場合、またはターミナルなどのこのマシン上の別の Claude Code プロセスで実行されている場合、開いていると見なされます。

アクティブ がオンで、ステータス、**開く**、または **閉じる** をチェックすると、リストはチェックに一致するすべてのセッションも表示します。設定したフィルタはウィンドウの再読み込み後も保持されます。

10.5.6 ターミナルモードに切り替える

デフォルトでは、拡張機能はグラフィカルチャットパネルを開きます。CLI スタイルのインターフェースを使用する場合は、Use Terminal 設定を開いてチェックボックスをオンにします。

VS Code 設定（Mac では `Cmd+,`、Windows/Linux では `Ctrl+,`）を開き、Extensions → Claude Code に移動して、**Use Terminal** をチェックすることもできます。

10.6 プラグインを管理する

VS Code 拡張機能には、プラグインをインストールおよび管理するためのグラフィカルインターフェイスが含まれています。プロンプトボックスに `/plugins` と入力して、**プラグインを管理**インターフェイスを開きます。

10.6.1 プラグインをインストールする

プラグインダイアログには、**プラグイン**と**マーケットプレイス**の 2 つのタブが表示されます。

プラグインタブでは、以下のことができます。

- **インストール済みプラグイン**がトップに表示され、トグルスイッチで有効または無効にできます。
  - プロジェクトの共有 `.claude/settings.json` で有効になっているプラグインをオフにすると、拡張機能は最初に確認を求めます。**自分のみ無効にする**は自分に対してのみオフにし、**全員のために無効にする**は共有ファイルを変更します。
  - 読み込みに失敗したプラグインは、その行に短い理由が表示されます。理由をクリックすると対処方法が表示され、プラグインのトラブルシューティングで調べるために完全なエラーメッセージをコピーすることもできます。
- 設定されたマーケットプレイスからの**利用可能なプラグイン**が下に表示されます
- 名前または説明でプラグインをフィルタリングするために検索します
- 利用可能なプラグインの**インストール**をクリックします

プラグインをインストールするときは、インストールスコープを選択します。

- **あなたのためにインストール**：すべてのプロジェクトで利用可能（ユーザースコープ）
- **このプロジェクトのためにインストール**：プロジェクト協力者と共有（プロジェクトスコープ）
- **ローカルにインストール**：このリポジトリのみ、あなただけ（ローカルスコープ）

インストールが完了すると、フォームはまだ設定されていないプラグインの設定オプションを求めます。後でオプションを確認または変更するには、プラグインの行の歯車アイコンをクリックします。

機密テキストフィールドはマスクされ、以前保存したシークレットは**（変更なし）**と表示されます。保存された値を保持するには、フィールドを空白のままにします。

変更を保存した後、開いているセッションはプラグインをリロードし、ダイアログは**プラグイン変更を適用するために Claude を再起動してください**と表示します。

10.6.2 プラグインをアンインストールする

各インストール済み行は、それがインストールされているスコープを示します。そのインストールをアンインストールするには、行のゴミ箱アイコンをクリックします。薄いゴミ箱アイコンは、このワークスペースからアンインストールできない行（組織が管理するプラグインや別のプロジェクト用にインストールされたプラグインなど）を示します。

拡張機能は 2 つのケースで最初に確認を求めます：

- **プロジェクトの共有 `.claude/settings.json` で有効になっているプラグイン**：**自分のみ無効にする**を選択します。これはプラグインを協力者のためにインストール状態に保ちます。または**全員のためにアンインストール**を選択します。これはプロジェクトのインストールを `--keep-data` で削除するため、プラグインの保存されたデータディレクトリは残ります。既にプラグインを自分のためにオフにしている場合、ゴミ箱アイコンは質問なしにあなた自身のインストールを削除します。

- **それ以外の場合、保存されたデータを持つプラグインの最後のインストール**：データを保持するか削除するかを選択します。**保持**がデフォルトです

10.6.3 プラグインインストールリンクを共有する

特定のプラグインのインストールに直接誘導するリンクを送信するには、拡張機能の `install-plugin` URL を使用します。これを開くと VS Code が起動またはフォーカスされ、Claude Code パネルが開き、**プラグインを管理**ダイアログがそのプラグインのスコープ選択で開きます。ユーザーがスコープを選択するまで、何もインストールされません。プラグインのマーケットプレイスが Claude Code でまだ設定されていない場合、ダイアログはまずそれを追加するよう求めます。

```
vscode://anthropic.claude-code/install-plugin?plugin=code-review&
↪marketplace=anthropics/claude-plugins-official
```

URL は 2 つのクエリパラメータを受け取ります。

パラメータ	説明
`plugin`	マーケットプレイスにリストされているプラグインの名前。必須です。
`marketplace`	マーケットプレイスのソース：GitHub の `owner/repo`、`https://` URL、または `git@github.com:owner/repo.git` などの git SSH アドレス。省略した場合は `anthropics/claude-plugins-official` がデフォルトになります。

拡張機能は、何かを開く前に両方の値をチェックします。

- **プラグイン名**：最大 100 文字で、ASCII 英字または数字で始まり、それ以外は ASCII 英字、数字、`.`、`_`、`-` のみを使用します。
- **マーケットプレイスのソース**：`marketplace` パラメータに記載されている形式のみです。そのため、ローカルパス、`http://` アドレス、または `claude-plugins-official` のようなマーケットプレイスの名前は使用できません。`https://` URL には、ユーザー名、パスワード、クエリ文字列を含めることはできません。
- **Git ref**：マーケットプレイスをブランチまたはタグに固定するには、`#` をエンコードした形式である `%23` の後に ref を続けてソースに追加します（例：`marketplace=owner/repo%23v1.0`）。エンコードされていない `#` を含むリンクは失敗します。`anthropics` GitHub 組織内のマーケットプレイスは、リンクで固定できません。

これらのルールに違反するリンクを開くと、`Invalid plugin installation URL` で始まるエラーが表示されます。Claude Code パネルとダイアログは開かず、何もインストールされません。プラグインの名前またはマーケットプレイスをリンクに含められない場合は、**マーケットプレイス**タブでマーケットプレイスを追加してから、**プラグイン**タブでプラグインをインストールするよう案内してください。

次のケースでは、スコープ選択ではなくダイアログのメッセージで終了します。

- **マーケットプレイスにその名前のプラグインがリストされていない**：ダイアログはプラグインが見つからなかったことを報告します。`plugin` の値をマーケットプレイスのリストと照合してください。
- **プラグインが既にインストールされている**：ダイアログはそのことを示し、何も変わりません。

- **同じ名前の別のマーケットプレイスが既に追加されている**：ダイアログはリンクのマーケットプレイスが追加されなかったことを示し、何もインストールされません。

GitHub README、issue、およびその他の Markdown ホストの一部は、スキームが `http` または `https` ではないリンクを削除するため、`vscode://` リンクはプレーンテキストとしてレンダリングされます。これらのホストではコードブロック内に URL を配置してください。リンクがクリック可能ではなくプレーンテキストとしてレンダリングされるは `claude-cli://` リンクについて説明しています。

10.6.4 マーケットプレイスを管理する

マーケットプレイスタブに切り替えて、プラグインソースを追加または削除します。

- GitHub リポジトリ、URL、またはローカルパスを入力して、新しいマーケットプレイスを追加します
- 更新アイコンをクリックして、マーケットプレイスのプラグインリストを更新します
- ゴミ箱アイコンをクリックして、マーケットプレイスを削除します。削除するとそれからインストールしたすべてのプラグインがアンインストールされます。そのため、確認ではそれらのプラグインが最初に名前で示されます

ダイアログで加えたプラグイン変更は、その VS Code ウィンドウで開いている Claude Code セッションにすぐに適用されます。

ダイアログを開いたセッションがプラグインをリロードできない場合、ダイアログは再度試すか、そのセッションで Claude を再起動するオプションを提供します。

注釈:

VS Code のプラグイン管理は、内部的に同じ CLI コマンドを使用しています。拡張機能で設定したプラグインとマーケットプレイスは CLI でも利用でき、その逆も同様です。

プラグインシステムの詳細については、プラグインおよびプラグインマーケットプレイスを参照してください。

10.7 Chrome でブラウザタスクを自動化する

Claude を Chrome ブラウザに接続して、Web アプリをテストし、コンソールログでデバッグし、VS Code を離れることなくブラウザワークフローを自動化できます。これには Claude in Chrome 拡張機能 バージョン 1.0.36 以上が必要です。

プロンプトボックスに `@browser` と入力し、その後に Claude に実行させたい内容を入力します。

```
@browser go to localhost:3000 and check the console for errors
```

添付メニューを開いて、新しいタブを開く、ページコンテンツを読むなど、特定のブラウザツールを選択することもできます。

Claude はブラウザタスク用に新しいタブを開き、ブラウザのログイン状態を共有するため、既にサインインしているサイトにアクセスできます。

`@browser` と入力しなくても各セッションの開始時にブラウザへ接続されるようにするには、*Chrome* **をデフォルトで有効にする** (ページ 216) を参照してください。そのように接続されたセッションで Claude Code がブラウザ操作の前に確認を求める場合については、*VS Code* **セッションでの権限プロンプト** (ページ 216) を参照してください。

セットアップ手順、機能の完全なリスト、トラブルシューティングについては、*Claude Code* **を** *Chrome* **で使用する** (ページ 213) を参照してください。

10.8 VS Code コマンドとショートカット

コマンドパレット（Mac では `Cmd+Shift+P`、Windows/Linux では `Ctrl+Shift+P`）を開いて「Claude Code」と入力すると、Claude Code 拡張機能で利用可能なすべての VS Code コマンドが表示されます。

一部のショートカットは、どのパネルが「フォーカス」されているか（キーボード入力を受け取っているか）によって異なります。カーソルがコードファイル内にある場合、エディターがフォーカスされています。カーソルが Claude のプロンプトボックス内にある場合、Claude がフォーカスされています。`Cmd+Esc` / `Ctrl+Esc` を使用して、それらを切り替えます。

注釈:

これらは拡張機能を制御するための VS Code コマンドです。組み込みの Claude Code コマンドのすべてが拡張機能で利用可能なわけではありません。詳細については、*VS Code* **拡張機能と** *Claude Code CLI* (ページ 196) を参照してください。

コマンド	ショートカット	説明
Focus Input	`Cmd+Esc`（Mac）/ `Ctrl+Esc`（Windows/Linux）	エディターと Claude 間のフォーカスを切り替えます
Focus last message	-	会話内の最新メッセージ、または待機中の権限プロンプトにキーボードフォーカスを移動します。キーボードまたはスクリーンリーダーでそこから読み取ることができます。**ターミナルモード** (ページ 186) では利用できません。Claude Code v2.1.268 以降が必要です
Open in Side Bar	-	Claude をサイドバーで開きます
Open in Terminal	-	Claude をターミナルモードで開きます
Open in New Tab	`Cmd+Shift+Esc`（Mac）/ `Ctrl+Shift+Esc`（Windows/Linux）	新しい会話をエディタータブとして開きます
Open in New Window	-	新しい会話を別のウィンドウで開きます
New Conversation	`Cmd+N`（Mac）/ `Ctrl+N`（Windows/Linux）	新しい会話を開始します。Claude がフォーカスされており、`enableNewConversationShortcut` が `true` に設定されている必要があります

次のページに続く

表 2 – 前のページからの続き

コマンド	ショートカット	説明
Reopen Closed Session	`Cmd+Shift+T`（Mac）/ `Ctrl+Shift+T`（Windows/Linux）	最近閉じた Claude セッションタブを再度開きます。最後に閉じたタブが Claude セッションではない場合、VS Code の通常の再度開く機能にフォールスルーします。`enableReopenClosedSessionShortcut` で無効にできます
Insert @-Mention Reference	`Option+K`（Mac）/ `Alt+K`（Windows/Linux）	現在のファイルと選択範囲への参照を挿入します（エディターがフォーカスされている必要があります）
Accept Change at Cursor	-	**提案された編集をレビューする** (ページ 174) 際に、カーソル位置の変更を 1 つずつ受け入れます。Claude Code v2.1.275 以降が必要です
Reject Change at Cursor	-	提案された編集をレビューする際に、カーソル位置の変更を 1 つずつ元に戻します。Claude Code v2.1.275 以降が必要です
Toggle Focus view	`Ctrl+Option+F`（Mac）/ `Ctrl+Alt+F`（Windows/Linux）	会話内のツールアクティビティを非表示にするか表示します。Claude パネルまたはサイドバーが表示されている間に機能します。Claude Code v2.1.221 以降が必要です
Rename Session Tab	-	アクティブな Claude タブのセッションの名前を変更します。Claude Code v2.1.257 以降が必要です
Add Session Tab to Group	-	アクティブな Claude タブのセッションを、選択または作成する**セッショングループ** (ページ 185) に追加します。Claude Code v2.1.257 以降が必要です
Mark Session as Unread	-	アクティブな Claude タブのセッションをセッションリストで未読としてマークします。Claude Code v2.1.257 以降が必要です
Show Logs	-	拡張機能のデバッグログを表示します
Logout	-	Anthropic アカウントからサインアウトします

10.8.1 他のツールから VS Code タブを起動する

拡張機能は `vscode://anthropic.claude-code/open` で URI ハンドラーを登録します。これを使用して、独自のツール（シェルエイリアス、ブラウザーブックマークレット、または URL を開くことができるスクリプト）から新しい Claude Code タブを開きます。VS Code がまだ実行されていない場合、URL を開くと最初に起動します。VS Code が既に実行されている場合、URL は現在フォーカスされているウィンドウで

開きます。

オペレーティングシステムの URL オープナーでハンドラーを呼び出します。

macOS

```
open "vscode://anthropic.claude-code/open"
```

Linux

```
xdg-open "vscode://anthropic.claude-code/open"
```

`xdg-open` コマンドは `xdg-utils` パッケージから提供されます。シェルが見つからないと報告する場合は、xdg-open is not found on Linux を参照してください。

Windows

PowerShell では：

```
Start-Process "vscode://anthropic.claude-code/open"
```

`cmd.exe` では、`start` は最初の引用符付き引数をウィンドウタイトルとして扱うため、URL の前に空のタイトルを渡します：

```
start "" "vscode://anthropic.claude-code/open"
```

ハンドラーは 2 つのオプションのクエリパラメーターを受け入れます：

パラメーター	説明
`prompt`	プロンプトボックスに事前入力するテキスト。URL エンコードされている必要があります。プロンプトは事前入力されますが、自動的には送信されません。
`session`	新しい会話を開始する代わりに再開するセッション ID。セッションは、VS Code で現在開いているワークスペースに属している必要があります。セッションが見つからない場合は、代わりに新しい会話が開始されます。セッションが既にタブで開いている場合、そのタブがフォーカスされます。セッション ID をプログラムで取得するには、**会話を続ける** (ページ 707) を参照してください。

たとえば、「review my changes」で事前入力されたタブを開くには：

```
vscode://anthropic.claude-code/open?prompt=review%20my%20changes
```

拡張機能はまた `vscode://anthropic.claude-code/install-plugin` を処理します。これは ***1* つのプラグインでプラグインダイアログを開きます** (ページ 188)。VS Code タブの代わりにターミナルセッションを起動するには、CLI の `claude-cli://` ハンドラーを使用します。リンクからセッションを起動する を参照してください。

10.9 設定を構成する

拡張機能には 2 つのタイプの設定があります。

- **VS Code の拡張機能設定**：VS Code 内での拡張機能の動作を制御します。`Cmd+,`（Mac）または `Ctrl+,`（Windows/Linux）で開き、Extensions → Claude Code に移動します。`/` を入力して **General config…** を選択して設定を開くこともできます。
- **`~/.claude/settings.json` の Claude Code 設定**：拡張機能と CLI 間で共有されます。許可されたコマンド、環境変数、hooks、MCP サーバーに使用します。Claude Code v2.1.283 以降では、権限モード会話が開始される入力の 1 つでもあり、以前のバージョンでは Pro、Max、Team プランのみです。Switch permission modes に順序が記載されています。詳細は *Settings* (ページ 711) を参照してください。

Tip:

`settings.json` に `"$schema": "https://json.schemastore.org/claude-code-settings.json"` を追加して、VS Code 内で利用可能なすべての設定のオートコンプリートとインライン検証を取得します。

10.9.1 拡張機能設定

VS Code は `initialPermissionMode` をユーザー設定から読み込み、ワークスペース値を無視します。v2.1.225 より前では、VS Code は設定をデフォルトの `default` に設定し、ワークスペース値を適用していました。

設定	デフォルト	説明
`useTerminal`	`false`	Claude をグラフィカルパネルではなくターミナルモードで起動します
`initialPermissionMode`	-	新しい会話の承認プロンプトを制御します：`default`、`plan`、`acceptEdits`、または `bypassPermissions`。`manual` は `default` のエイリアスで、モード指示器で **Manual** というラベルが付いたモードを選択します。設定を未設定のままにすると、拡張機能は Switch permission modes で説明されているように開始権限モードを選択します。
`preferredLocation`	`panel`	Claude が開く場所：`sidebar`（右）または `panel`（新しいタブ）
`lockEditorGroups`	`true`	*Claude* **がそのタブを開始するエディターグループをロック** (ページ 184) して、Claude タブがフォーカスされている間に開いたファイルが別のグループに移動するようにします。オフの場合、拡張機能はエディターグループをロックしません。Claude Code v2.1.274 以降が必要です

次のページに続く

表 4 – 前のページからの続き

設定	デフォルト	説明
autosave	true	Claude がファイルを読み取りまたは書き込みする前に自動保存します
attachOpenFile	true	エディターで開いているファイルをメッセージに追加し、プロンプトボックスに表示します。オフの場合、選択したテキストのみが追加されます。Claude Code v2.1.271 以降が必要です
useCtrlEnterToSend	false	Enter の代わりに Ctrl/Cmd+Enter を使用してプロンプトを送信します
scrollToBottomOnSend	true	メッセージを送信するときに会話を下部にスクロールします。オフの場合、会話は元の位置に留まります。Claude Code v2.1.275 以降が必要です
showMessageTimestamps	false	各メッセージが送信された日時を表示します。日付が変わる位置には日付行が表示されます。Claude Code v2.1.284 以降が必要です
enableNewConversationShortcut	false	Cmd/Ctrl+N を有効にして新しい会話を開始します
enableReopenClosedSession Shortcut	true	Cmd/Ctrl+Shift+T を使用して、最近閉じた Claude セッションタブを再度開きます。最後に閉じたタブが Claude セッションではなかった場合、ショートカットは VS Code の通常の再度開く閉じたエディターコマンドを実行します。
archiveInactiveSessions	14	この日数アクティビティがない場合、**セッションを自動的にアーカイブします** (ページ 182)：1、2、7、または 14。0 に設定してオフにします。Claude Code v2.1.265 以降が必要です
continueAfterReload	true	ウィンドウをリロードした後、Claude は復元されたセッションで **中断されたステップを続行します** (ページ 185)。Claude Code v2.1.274 以降が必要です
hideOnboarding	false	オンボーディングチェックリスト（卒業帽アイコン）を非表示にします
focusView	false	ツール呼び出し、ツール結果、思考を展開可能な行の背後に非表示にして、プロンプトと Claude の応答を残します。Claude の最新のやることリストは表示されたままです。これには Claude Code v2.1.225 以降が必要です。コマンドメニューから Focus ビューを切り替えることもできます。Claude Code v2.1.221 以降が必要です
respectGitIgnore	true	ファイル検索と **選択コンテキスト** (ページ 181) から .gitignore パターンを除外します

次のページに続く

表 4 – 前のページからの続き

設定	デフォルト	説明
usePythonEnvironment	true	Claude を実行するときにワークスペースの Python 環境をアクティブにします。Python 拡張機能が必要です。
environmentVariables	[]	Claude プロセスの環境変数を設定します。共有構成には Claude Code 設定を使用してください。CLAUDE_CONFIG_DIR エントリは、その値が絶対パスである場合にのみ適用されます。拡張機能は ~ を展開せず、相対パスの値は無視します。
disableLoginPrompt	false	認証プロンプトをスキップします（サードパーティプロバイダーのセットアップ用）
allowDangerouslySkipPermissions	false	モードセレクターに権限をバイパスを追加します。インターネットアクセスのないサンドボックスでのみ使用してください。
claudeProcessWrapper	-	Claude プロセスを起動するために使用される実行可能ファイル。バンドルされたバイナリパスが存在する場合、引数として渡されます。プラットフォーム用のバイナリが拡張機能ビルドに含まれていない場合は、別途インストールされた claude バイナリに設定します。ラップされたセットアップでは、initialPermissionMode を設定するか、以前の会話で Manual、Edit automatically、または Auto を選択していない限り、会話は Manual モードで開始されます。これは、拡張機能が設定とビルトインデフォルトステップをスキップするためです。Switch permission modes を参照してください。アクティベーション時の「Unsupported platform」エラーは、プラットフォーム用にバイナリがバンドルされていないことを意味します。npm install 後にネイティブバイナリが見つからない を参照してください。

10.10 スクリーンリーダーを使用する

拡張機能のチャットパネルはスクリーンリーダーに対応しています。何も設定する必要はありません。拡張機能はすべてのユーザーの会話アクティビティを通知し、視覚的な変化はありません。これは CLI のオプトイン スクリーンリーダーモード とは別のもので、ターミナルインターフェースを適応させます。

チャットパネルのスクリーンリーダーサポートには Claude Code v2.1.236 以降が必要です。

会話中、拡張機能は以下を通知します。

- **Claude の返信**: 拡張機能は各返信が完了したときに 1 回通知し、テキストがストリーミングされている間は沈黙を保ちます。スクリーンリーダーはコードブロックを行数の概要として読み、リンクをラベルで読み、テーブルをセルごとに読みます。完全な返信はトランスクリプトで読み取り可能なままです。

- **権限リクエストと質問**: 拡張機能は権限プロンプトが表示されたときにリクエストを通知し、Claude が使用したいツールの名前を指定します。Claude があなたに質問するときと Claude がプランを完了してあなたのレビューを待つときも同じ方法で通知します。
- **ステータス変更**: 拡張機能は Claude が作業を開始したとき、Claude があなたの入力の準備ができたとき、および Claude Code が会話をコンパクト化し始めたときを通知します。
- **エラーとモデルプロンプト**: 拡張機能は会話内のエラーを通知し、**使用クレジット同意プロンプト** (ページ 728) または **フラグ付きリクエストプロンプト** (ページ 746) が表示されたときを通知します。

Claude が作業している間、スクリーンリーダーはプログレススピナーのアニメーションの代わりにテキストラベルを読みます。

セッションを再度開くか別のセッションに切り替えると、拡張機能は何も通知しません。復元された履歴、保留中の権限プロンプト、および進行中のステータスは、新しいことが発生するまで沈黙を保ちます。

10.10.1 キーボードからチャットパネルを使用する

トランスクリプト内の各ターンは、ターンを開始したプロンプトでラベル付けされた視覚的に隠されたヘッディングで始まるため、スクリーンリーダーのヘッディングナビゲーションを使用してターン間をジャンプできます。

ターン内で、スクリーンリーダーはメッセージを移動するときに、誰のメッセージであるかを通知します。

- **あなたのメッセージ**: 「You」
- **Claude のメッセージ**: 「Claude」
- **ツールステップ**: 「Claude」とツール名（「Claude, Bash」など）
- **思考ブロック**: 「Claude, thinking」

拡張機能はトランスクリプトをラベル付きリージョンとして公開しているため、`Tab` でトランスクリプト自体にフォーカスを移動して、自分のペースで読むこともできます。最新のメッセージまたは待機中の権限プロンプトにフォーカスを移動するには、代わりに **コマンドパレット** (ページ 190) から **Claude Code: Focus last message** を実行します。

権限プロンプトのオプションが権限ルールまたはディレクトリアクセスを保存する場合、そのラベルは「すべてのプロジェクト」や「このセッション」など、承認が保存される場所を名前で指定して終了します。そのオプションにフォーカスがある場合、`Left` または `Right` 矢印キーを押して保存先を変更でき、拡張機能は移動するたびに各保存先を通知します。ラベル内の保存先をクリックすることもできます。矢印キーには Claude Code v2.1.268 以降が必要です。

10.11 VS Code 拡張機能と Claude Code CLI

Claude Code は VS Code 拡張機能（グラフィカルパネル）と CLI（ターミナルのコマンドラインインターフェース）の両方で利用できます。一部の機能は CLI でのみ利用可能です。CLI のみの機能が必要な場合は、VS Code の統合ターミナルで `claude` を実行してください。これには**スタンドアロン *CLI* インストール** (ページ 19) が必要です。拡張機能は `claude` を PATH に追加しません。*VS Code* **で** *CLI* **を実行する** (ページ 197) を参照してください。

機能	CLI	VS Code 拡張機能
コマンドとスキル	すべて	サブセット（/ を入力して利用可能なものを表示）
MCP サーバー設定	はい	はい（チャットパネルで /mcp を使用して**サーバーを追加・管理** (ページ 198)）
チェックポイント	はい	はい
! Bash ショートカット	はい	いいえ
タブ補完	はい	いいえ

10.11.1 チェックポイントでの巻き戻し

VS Code 拡張機能はチェックポイントをサポートしており、Claude のファイル編集を追跡し、以前の状態に巻き戻すことができます。任意のメッセージにマウスを置くと巻き戻しボタンが表示されます。その後、3 つのオプションから選択します。

- **ここから会話をフォークする**：このメッセージからの新しい会話ブランチを開始し、すべてのコード変更はそのまま保持します
- **ここまでコードを巻き戻す**：会話履歴全体を保持しながら、ファイル変更をこのポイントまで戻します
- **会話をフォークしてコードを巻き戻す**：新しい会話ブランチを開始し、ファイル変更をこのポイントまで戻します

チェックポイントの仕組みと制限事項の詳細については、**チェックポイント** (ページ 1019) を参照してください。

10.11.2 VS Code で CLI を実行する

VS Code に留まりながら CLI を使用するには、統合ターミナルを開き（Windows/Linux では `Ctrl+`、Mac では `Cmd+`）、`claude` を実行します。CLI は diff 表示と診断共有などの機能のために IDE と自動的に統合されます。

拡張機能をインストールしても、`claude` がシェル PATH に追加されません。拡張機能はチャットパネル用に CLI のプライベートコピーをバンドルしていますが、ターミナルで `claude` を入力するには**スタンドアロン *CLI* インストール** (ページ 19) が必要です。インストールを 1 回実行すると、このページのコマンド（`claude mcp add` や `claude --resume` を含む）は任意のターミナルで機能します。インストール後も `claude` が見つからない場合は、PATH を確認してください。

外部ターミナルを使用している場合は、Claude Code 内で `/ide` を実行して VS Code に接続します。

10.11.3 拡張機能と CLI の間で切り替える

拡張機能と CLI は同じ会話履歴を共有します。拡張機能の会話を CLI で続行するには、ターミナルで `claude --resume` を実行します。これにより、会話を検索して選択できるインタラクティブピッカーが開きます。

10.11.4 プロンプトにターミナル出力を含める

プロンプトで `@terminal:name` を使用してターミナル出力を参照します。ここで `name` はターミナルのタイトルです。これにより、Claude はコマンド出力、エラーメッセージ、またはログをコピー&ペーストなしで確認できます。

10.11.5 実行中のコマンドやサブエージェントをバックグラウンドに移動する

Claude が待機しているコマンドや**サブエージェント** (ページ 317) に想定以上の時間がかかっている場合は、会話内のそのツール呼び出しの下にある **Run in background** をクリックします。このアクションは、コマンドが約 2 秒間実行された時点、またはサブエージェントが開始された直後に表示されます。Claude は待機をやめてターンを続行し、コマンドやサブエージェントはバックグラウンドタスクとして実行を続け、完了すると Claude に通知します。Claude Code v2.1.287 以降が必要です。

その間にタスクを確認したり停止したりするには、プロンプトボックスで `/tasks` を入力して**エージェントマップ** (ページ 177) を開きます。サブエージェントはそこでエージェントのツリー内の位置を維持し、コマンドはエージェントの下に一覧表示され、その**カードに最新の出力** (ページ 198) が表示されます。

10.11.6 バックグラウンドプロセスを監視する

プロンプトボックスで `/tasks` を入力して**エージェントマップ** (ページ 177) を開きます。これにより、セッションのバックグラウンドタスク（Claude がバックグラウンドシェルコマンドとして実行中の開発サーバーなど）が一覧表示されます。タスクをクリックしてそのカードを開き、そこで停止できます。Claude Code v2.1.277 以降が必要です。

バックグラウンドシェルコマンド、またはコマンドを実行するモニターの場合、カードにはコマンドの最新の出力も表示され、コマンドの実行中は更新されます。

10.11.7 MCP を使用して外部ツールに接続する

MCP（Model Context Protocol）サーバーは Claude に外部ツール、データベース、API へのアクセスを提供します。

VS Code を離れずに MCP サーバーを管理するには、チャットパネルで `/mcp` を入力します。開いたダイアログから、サーバーを追加したり、ローカル、ユーザー、またはプロジェクト**スコープ** (ページ 380) に保存されたサーバーを削除したり、サーバーを有効または無効にしたり、サーバーに再接続したり、OAuth 認証を管理したりできます。ダイアログでサーバーを追加・削除するには Claude Code v2.1.261 以降が必要です。

VS Code の統合ターミナル（`Ctrl+`` または `Cmd+``）で `claude mcp add` を実行することもできます。ダイアログとターミナルコマンドは同じ MCP 設定に保存され、どちらからの変更も、その後に開始する会話に反映されます。以下の例は GitHub のリモート MCP サーバーを追加します。このサーバーは、ヘッダーとして渡される個人用アクセストークンで認証します。

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/ \
  --header "Authorization: Bearer YOUR_GITHUB_PAT"
```

`YOUR_GITHUB_PAT` を個人用アクセストークンに置き換えます。`claude mcp add` コマンドは認証情報を検証せずに設定を保存するため、ここではプレースホルダー値が受け入れられますが、サーバーは後で接続

に失敗します。接続を確認するには、新しい会話を開始し、/mcp を入力して、サーバーが**接続済み**と表示されていることを確認します。認証情報が不正なサーバーは**失敗**と表示されます。

設定後、Claude にツールを使用するよう依頼します（例：「PR #456 をレビューしてください」）。

接続するサーバーを見つけるには、*MCP* **サーバーを検索・構築する** (ページ 364) を参照してください。

10.12 git で作業する

Claude Code は git と統合されており、VS Code 内で直接バージョン管理ワークフローをサポートします。Claude にコミット、プルリクエストの作成、またはブランチ間での作業を依頼できます。独立した worktree とそれ自身のファイルおよびブランチで Claude を開始するには、worktrees を使用した並列セッションの実行を参照してください。

10.12.1 コミットとプルリクエストを作成する

Claude はコミット、コミットメッセージの作成、および実装に基づいてプルリクエストを作成できます。

```
commit my changes with a descriptive message
create a pr for this feature
summarize the changes I've made to the auth module
```

プルリクエストを作成する場合、Claude は実際のコード変更に基づいて説明を生成し、テストまたは実装の決定に関するコンテキストを追加できます。

10.13 サードパーティプロバイダーを使用する

デフォルトでは、Claude Code は Anthropic の API に直接接続します。組織が Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry を使用して Claude にアクセスしている場合は、代わりにプロバイダーを使用するように拡張機能を設定してください。

1. **ログインプロンプトを無効化する**

   ログインプロンプトを無効化する設定を開き、チェックボックスをオンにします。

   VS Code の設定（Mac では `Cmd+,`、Windows/Linux では `Ctrl+,`）を開き、「Claude Code login」を検索して、**ログインプロンプトを無効化する**をオンにすることもできます。

2. **プロバイダーを設定する**

   プロバイダーのセットアップガイドに従ってください。

   - *Amazon Bedrock* **上の** *Claude Code* (ページ 863)
   - *Google Cloud* **の** *Agent Platform* **上の** *Claude Code* (ページ 881)
   - *Microsoft Foundry* **上の** *Claude Code* (ページ 891)

   これらのガイドでは、`~/.claude/settings.json` でプロバイダーを設定する方法について説明しており、VS Code 拡張機能と CLI の間で設定が共有されることを保証します。

サードパーティプロバイダーでは、拡張機能は claude.ai アカウントが必要な機能（プラン使用状況バー、音声入力、**クラウドセッション** (ページ 183) 用の Web タブなど）を提供しません。これらのサインインで Account & usage ダイアログが表示する内容については、**アカウントと使用状況を確認する** (ページ 184) を参照してください。以前の `/login` から残された claude.ai サインインは未使用のままです。拡張機能はリクエストと共にそれを送信しません。

10.14 セキュリティとプライバシー

あなたのコードはプライベートです。Claude Code はあなたのコードを処理して支援を提供しますが、モデルの訓練に使用することはありません。データ処理の詳細とログ記録をオプトアウトする方法については、**データとプライバシー** (ページ 1105) を参照してください。

自動編集権限が有効になっている場合、Claude Code は VS Code が自動的に実行する可能性のある VS Code 設定ファイル（`settings.json` や `tasks.json` など）を変更できます。信頼できないコードを操作する場合のリスクを軽減するには、以下の対策を講じてください。

- 信頼できないワークスペースに対して VS Code Restricted Mode を有効にする
- 自動編集または自動の代わりに Manual モードを使用する
- 変更を受け入れる前に慎重に確認する

10.14.1 組み込み IDE MCP サーバー

拡張機能がアクティブな場合、CLI が自動的に接続するローカル MCP サーバーが実行されます。これにより、CLI は VS Code のネイティブ diff ビューアーで diff を開き、`@` メンションの現在の選択を読み取り、Jupyter ノートブックで作業している場合はセルを実行するよう VS Code に要求できます。

サーバーは `ide` という名前で、設定するものがないため `/mcp` から非表示になっています。ただし、組織が MCP ツールをホワイトリストに登録するために `PreToolUse` フックを使用している場合は、それが存在することを知っておく必要があります。

選択とオープンファイルコンテキスト。 接続中、CLI は現在のエディター選択とアクティブファイルのパスを、送信する各プロンプトのコンテキストとして含めます。トランスクリプトには、これが発生したときに ☒ `Selected N lines from <file>` という行が表示されます。

Claude **が作業中にメッセージをキューに入れる** (ページ 90) 場合、`Enter` を押したときに持っていた選択を保持し、その後に選択したものは何でも保持します。

`.env` などの機密ファイルを除外するには、そのパスに対して *Read* **拒否ルール** (ページ 1051) を追加してください。一致する拒否ルールは、そのファイルの選択されたテキストとオープンファイル通知の両方が Claude に到達するのを防ぎます。

Attach Open File **設定** (ページ 193) をオフにすると、CLI はテキストを選択している間のみアクティブファイルのパスを受け取ります。

トランスポートと認証。 サーバーは `127.0.0.1` の 10000〜65535 の範囲内のランダムポートにバインドされ、ポートは設定できません。トランスポートは暗号化されていない `ws://` です。ソケットはループバックのみであるため、トラフィックをキャプチャできるプロセスはロックファイルからトークンを読み取ることもできるため、TLS は保護を追加しません。各拡張機能のアクティベーションは新しいランダム認証トークンを生成し、`~/.claude/ide/<port>.lock` のロックファイルに書き込み、CLI は `X-Claude-Code-`

`Ide-Authorization` ヘッダーとして提示して接続する必要があります。ロックファイルは `0700` ディレクトリ内で `0600` 権限を持つため、VS Code を実行しているユーザーのみがそれを読み取ることができます。`CLAUDE_CONFIG_DIR` が設定されている場合、ロックファイルは代わりに `$CLAUDE_CONFIG_DIR/ide/` に書き込まれます。

モデルに公開されるツール。 サーバーは 12 個のツールをホストしていますが、モデルに表示されるのは 2 つだけです。残りは CLI が独自の UI（diff を開く、選択を読み取る、ファイルを保存する）に使用する内部 RPC であり、ツールリストが Claude に到達する前にフィルタリングされます。

ツール名（フックで表示される）	機能	読み取り専用
`mcp__ide__getDiagnostics`	言語サーバー診断（VS Code の Problems パネルのエラーと警告）を返します。オプションで 1 つのファイルにスコープできます。	はい
`mcp__ide__executeCode`	アクティブな Jupyter ノートブックのカーネルで Python コードを実行します。以下の確認フローを参照してください。	いいえ

チャットパネルでの診断。 チャットパネルでは、Claude Code v2.1.285 以降の場合、Claude は `claude-vscode` という名前の別の組み込みサーバーを通じて VS Code の Problems パネルを読み取ります。Claude はこのサーバーに対して、1 つのファイル、または VS Code が診断情報を持つすべてのファイルについて、現在のエラーと警告を要求できます。

フックと権限ルールでは、チャットパネルの診断ツールは `mcp__claude-vscode__getDiagnostics` として認識されます。CLI とチャットパネルの両方で診断を対象にするには、フックまたはルールで `mcp__ide__getDiagnostics` と `mcp__claude-vscode__getDiagnostics` の両方を指定してください。

次の `settings.json` の例では、両方のツールを拒否しています。

```
{
  "permissions": {
    "deny": [
      "mcp__ide__getDiagnostics",
      "mcp__claude-vscode__getDiagnostics"
    ]
  }
}
```

`Read` 拒否ルールはどちらのツールも対象にしないため、この例のように**拒否ルール** (ページ 1058)で名前を指定してブロックしてください。

Jupyter 実行は常に最初に確認します。 `mcp__ide__executeCode` は何もサイレントに実行することはできません。各呼び出しで、コードはアクティブなノートブックの最後に新しいセルとして挿入され、VS Code がそれをビューにスクロールし、ネイティブ Quick Pick が **Execute** または **Cancel** を求めます。キャンセルするか、`Esc` でピッカーを閉じると、Claude にエラーが返され、何も実行されません。また、アクティブなノートブックがない場合、Jupyter 拡張機能（`ms-toolsai.jupyter`）がインストールされていない場合、またはカーネルが Python でない場合、ツールは完全に拒否します。

注釈:

Quick Pick 確認は `PreToolUse` フックとは別です。`mcp__ide__executeCode` のホワイトリストエントリにより、Claude はセルの実行を **提案** できます。VS Code 内の Quick Pick は、実際に実行できるようにするものです。

10.15 一般的な問題を解決する

10.15.1 拡張機能がインストールできない

- VS Code の互換性のあるバージョン（1.94.0 以降）があることを確認してください
- VS Code に拡張機能をインストールする権限があることを確認してください
- VS Code Marketplace から直接インストールしてみてください

10.15.2 Spark アイコンが表示されない

Spark アイコンは、ファイルを開いているときに **Editor Toolbar**（エディタの右上）に表示されます。表示されない場合は、以下の手順を実行してください。

1. **ファイルを開く**：アイコンを表示するにはファイルを開く必要があります。フォルダを開いているだけでは不十分です。
2. **VS Code のバージョンを確認する**：1.94.0 以降が必要です（Help → About）
3. **VS Code を再起動する**：コマンドパレットから「Developer: Reload Window」を実行してください
4. **競合する拡張機能を無効にする**：他の AI 拡張機能（Cline、Continue など）を一時的に無効にしてください
5. **ワークスペースの信頼を確認する**：拡張機能は制限モードでは動作しません

または、ウィンドウの右下隅の **Status Bar** にある ☒ **Claude Code** をクリックしてください。これはファイルを開いていなくても動作します。**Command Palette**（`Cmd+Shift+P` / `Ctrl+Shift+P`）を使用して「Claude Code」と入力することもできます。

10.15.3 macOS で Cmd+Esc が機能しない

macOS Tahoe 以降では、システムの Game Overlay ショートカットがデフォルトで `Cmd+Esc` にバインドされており、VS Code に到達する前にキープレスをインターセプトします。ショートカットを解放するには、以下の手順を実行してください。

1. System Settings を開く
2. Keyboard、Keyboard Shortcuts、Game Controllers の順に移動する
3. Game Overlay チェックボックスをクリアする

または、拡張機能を別のキーにリバインドしてください。VS Code の Keyboard Shortcuts editor（`Cmd+K Cmd+S`）を開き、`Claude Code: Focus input` を検索して、新しいバインディングを割り当ててください。

10.15.4 Claude Code が応答しない

Claude Code がプロンプトに応答しない場合は、以下の手順を実行してください。

1. **インターネット接続を確認する**：安定したインターネット接続があることを確認してください
2. **新しい会話を開始する**：新しい会話を開始して、問題が解決するかどうかを確認してください
3. **CLI を試す**：ターミナルから `claude` を実行して、より詳細なエラーメッセージが表示されるかどうかを確認してください

問題が解決しない場合は、GitHub で issue を報告してください。エラーの詳細を含めてください。

10.16 拡張機能をアンインストールする

Claude Code 拡張機能をアンインストールするには：

1. 拡張機能ビューを開きます（Mac では `Cmd+Shift+X`、Windows/Linux では `Ctrl+Shift+X`）
2. 「Claude Code」を検索します
3. **アンインストール** をクリックします

VS Code の統合ターミナルで `claude` を実行すると、Claude Code は拡張機能を自動的に再インストールします。拡張機能をインストールされたままにしないようにするには、`/config` で **Auto-install IDE extension** をオフにするか、`autoInstallIdeExtension` を `false` に設定します。また、`CLAUDE_CODE_IDE_SKIP_AUTO_INSTALL` 環境変数を 1 に設定することもできます。

拡張機能のデータを削除してすべての設定をリセットするには、プラットフォーム用の拡張機能のストレージディレクトリを削除します。

macOS の場合：

```
rm -rf ~/Library/"Application Support"/Code/User/globalStorage/anthropic.claude-code
```

Linux の場合：

```
rm -rf ~/.config/Code/User/globalStorage/anthropic.claude-code
```

Windows の場合（PowerShell）：

```
Remove-Item -Recurse -Force "$env:APPDATA\Code\User\globalStorage\anthropic.claude-
↪code"
```

詳細なヘルプについては、**トラブルシューティングガイド** (ページ 37) を参照してください。

10.17 次のステップ

Claude Code を VS Code で設定したら、以下を実行してください。

- **一般的なワークフローを確認する** (ページ 105) ことで、Claude Code を最大限に活用できます

- *MCP* **サーバーを設定する** (ページ 363) ことで、外部ツールを使用して Claude の機能を拡張できます。チャットパネルで `/mcp` を使用して、これらを追加および管理できます。
- *Claude Code* **の設定を構成する** (ページ 711) ことで、許可されたコマンド、hooks などをカスタマイズできます。これらの設定は、拡張機能と CLI 間で共有されます。

第11章　JetBrains IDEs

Claude Code を IntelliJ、PyCharm、WebStorm など JetBrains IDEs で使用する

Claude Code は専用プラグインを通じて JetBrains IDEs と統合され、インタラクティブな diff ビューイング、選択コンテキスト共有など、様々な機能を提供します。

11.1 サポートされている IDE

Claude Code プラグインは、以下を含むほとんどの JetBrains IDEs で動作します。

- IntelliJ IDEA
- PyCharm
- Android Studio
- WebStorm
- PhpStorm
- GoLand

11.2 機能

- **クイック起動**: `Cmd+Esc`（Mac）または `Ctrl+Esc`（Windows/Linux）を使用してエディタから Claude Code を直接開くか、UI の Claude Code ボタンをクリックします
- **Diff ビューイング**: Claude Code はコードの変更をターミナルではなく IDE の diff ビューアで開きます。`/config` の **Diff tool** 設定でこれを変更できます
- **選択コンテキスト**: IDE の現在の選択またはタブが Claude Code と自動的に共有されます。*Read* **拒否ルール** (ページ 1051) は、一致するファイルのこの共有をブロックします
- **ファイル参照ショートカット**: `Cmd+Option+K`（Mac）または `Alt+Ctrl+K`（Linux/Windows）を使用して `@src/auth.ts#L1-99` などのファイル参照を挿入します
- **診断共有**: Claude は *getDiagnostics* **ツール** (ページ 210) を呼び出して IDE の検査診断（lint や構文エラーなど）を読み取ります。Claude Code は編集後に独自にプラグインから診断をリクエストしません

11.3 インストール

プラグインは IDE の統合ターミナルで `claude` コマンドを実行し、それに接続します。独自の CLI コピーをバンドルしていないため、両方をインストールする必要があります。

1. **Claude Code CLI をインストールする**

   まだインストールしていない場合は、**クイックスタート** (ページ 11) に従って CLI をインストールしてください。`claude` が PATH にない場合、プラグインは'Claude Code を起動できません' という通知を表示します。

2. **JetBrains プラグインをインストールする**

   JetBrains マーケットプレイスから Claude Code プラグイン をインストールし、IDE を再起動します。

`claude` が IDE が見つけられない場所にインストールされている場合は、プラグインの *Claude* **コマンド設定** (ページ 207) でフルパスを設定してください。

Claude Code は、任意の有料 Claude サブスクリプション（Pro、Max、Team、または Enterprise）または Claude Console アカウントで動作し、API キーは不要です。`claude` を初めて実行するときに **ログイン** (ページ 851) するよう求められます。

11.4 使用方法

11.4.1 IDE から

IDE の統合ターミナルから `claude` を実行すると、すべての統合機能がアクティブになります。

11.4.2 外部ターミナルから

任意の外部ターミナルで `/ide` コマンドを使用して Claude Code を JetBrains IDE に接続し、すべての機能をアクティブにします。

```
claude
```

```
/ide
```

接続が成功すると、Claude Code は `Connected to IntelliJ IDEA.` のようなメッセージで確認します。Claude Code がプラグインを持たない実行中の IDE を検出した場合、`/ide` はプラグインをインストールし、IDE を再起動するよう求めます。

Claude が IDE と同じファイルにアクセスできるようにしたい場合は、IDE プロジェクトルートと同じディレクトリから Claude Code を起動してください。

11.5 設定

11.5.1 Claude Code 設定

Claude Code の設定を通じて IDE 統合を設定します。

1. `claude` を実行します
2. `/config` コマンドを入力します
3. **Diff tool** を `auto` に設定して IDE に diff を表示するか、`terminal` に設定してターミナルに表示したままにします

Diff tool エントリは Claude Code が IDE に接続されている場合にのみ `/config` に表示されるため、JetBrains ターミナルから `claude` を実行するか、外部ターミナルから最初に `/ide` を実行してください。基になる設定については `diffTool` を参照してください。

11.5.2 プラグイン設定

Settings → **Tools** → **Claude Code [Beta]** に移動して Claude Code プラグインを設定します。

一般設定

- **Claude command**: Claude を実行するカスタムコマンドを指定します（例：`claude`、`/usr/local/bin/claude`、または `npx @anthropic-ai/claude-code`）
- **Suppress notification for when Claude Command is not found**: Claude コマンドが見つからないことに関する通知をスキップします
- **Enable using Option+Enter for multi-line prompts**: macOS のみ。有効にすると、Option+Enter は Claude Code プロンプトに新しい行を挿入します。Option キーが予期せずキャプチャされる場合は無効にしてください。ターミナルの再起動が必要です。
- **Enable automatic updates**: プラグインの更新を自動的にチェックしてインストールします。再起動時に適用されます

Tip:

WSL ユーザー向け: Claude コマンドとして `wsl -d Ubuntu -- bash -lic "claude"` を設定します（`Ubuntu` を WSL ディストリビューション名に置き換えてください）

ESC キー設定

ESC キーが JetBrains ターミナルで Claude Code 操作を中断しない場合：

1. **Settings** → **Tools** → **Terminal** に移動します
2. 以下のいずれかを実行します。
   - 「Move focus to the editor with Escape」をオフにするか、
   - 「Configure terminal keybindings」をクリックして「Switch focus to Editor」ショートカットを削除します
3. 変更を適用します

これにより、ESC キーが Claude Code 操作を適切に中断できるようになります。

11.6 特別な設定

11.6.1 リモート開発

警告:

JetBrains リモート開発を使用する場合、ローカルクライアントマシンではなく、**Settings** → **Plugin (Host)** を通じてリモートホストにプラグインをインストールする必要があります。

11.6.2 WSL 設定

Claude Code を WSL2 の JetBrains IDE で使用していて「No available IDEs detected」が表示される場合、原因は通常 WSL2 の NAT ネットワークまたは Windows ファイアウォールが WSL2 と Windows ホストで実行されている IDE 間の接続をブロックしていることです。WSL1 はホストのネットワークを直接使用するため、影響を受けません。

Windows ファイアウォール経由で WSL2 トラフィックを許可する

これは推奨される修正方法です。既存の WSL2 ネットワークモードを保持するためです。

1. **WSL2 IP アドレスを見つける**

   WSL シェル内から以下を実行します。

   ```
   hostname -I
   ```

   サブネットをメモします。アドレスの最初の 2 つのセグメントを取得し、その後に `.0.0/16` を付けます。例えば、アドレスが `172.21.123.45` の場合、サブネットは `172.21.0.0/16` です。

2. **ファイアウォールルールを作成する**

   PowerShell を管理者として開き、以下を実行します。IP 範囲をサブネットに合わせて調整してください。

   ```
   New-NetFirewallRule -DisplayName "Allow WSL2 Internal Traffic" -Direction␣
   ↪Inbound -Protocol TCP -Action Allow -RemoteAddress 172.21.0.0/16 -LocalAddress␣
   ↪172.21.0.0/16
   ```

3. **IDE と Claude Code を再起動する**

   両方を閉じて再度開き、新しいルールが有効になるようにします。

WSL2 をミラーリングネットワークに切り替える

ミラーリングネットワークには Windows 11 22H2 以降が必要です。Windows 10 を使用している場合は、代わりに上記のファイアウォールルールを使用してください。

Windows ユーザーディレクトリの `.wslconfig` に以下を追加します。

```
[wsl2]
networkingMode=mirrored
```

その後、PowerShell から `wsl --shutdown` で WSL を再起動します。

11.7 トラブルシューティング

11.7.1 プラグインが動作しない

プラグインがインストールされているが Claude Code 機能が IDE に表示されない場合：

- Claude Code をプロジェクトルートディレクトリから実行していることを確認してください
- JetBrains プラグインが IDE 設定で有効になっていることを確認してください
- IDE を完全に再起動してください（複数回実行する必要がある場合があります）
- リモート開発の場合、プラグインがリモートホストにインストールされていることを確認してください

11.7.2 IDE が検出されない

`/ide` コマンドが「No available IDEs detected」を表示する場合：

- プラグインがインストールされて有効になっていることを確認してください
- IDE を完全に再起動してください
- `/ide` を実行せずに自動接続が期待される場合は、IDE の統合ターミナルから `claude` を起動したことを確認してください
- WSL ユーザーの場合、上記の *WSL* **設定** (ページ 208) を参照してください

11.7.3 コマンドが見つからない

Claude アイコンをクリックして「command not found」が表示される場合：

1. `claude --version` をターミナルで実行して Claude Code がインストールされていることを確認してください
2. プラグイン設定で Claude コマンドパスを設定してください
3. WSL ユーザーの場合、設定セクションで説明されている WSL コマンド形式を使用してください

11.8 セキュリティに関する考慮事項

Claude Code が acceptEdits 権限モードで JetBrains IDE で実行される場合、IDE によって自動的に実行される可能性のある IDE 設定ファイルを変更できる場合があります。これにより、`acceptEdits` モードで Claude Code を実行するリスクが増加し、Bash 実行に対する Claude Code の権限プロンプトをバイパスできる可能性があります。

JetBrains IDEs で実行する場合は、以下を検討してください。

- 編集に対して手動モードを使用する。`acceptEdits` と自動モードの両方は、保護されたパスを除いて、作業ディレクトリ内の編集を確認なしで承認するため
- Claude が信頼できるプロンプトでのみ使用されることを確認するために特に注意する
- Claude Code がアクセスして変更できるファイルを認識する

Claude Code のインストールまたはログインの問題については、インストールとログインのトラブルシューティング を参照してください。

11.8.1 組み込み IDE MCP サーバー

プラグインがアクティブな場合、CLI が自動的に接続するローカル MCP サーバーが実行されます。これは、CLI が IDE のネイティブ diff ビューアーで diff を開き、`@` メンションの現在の選択を読み取り、Claude が検査診断を読み取る方法です。

サーバーは `ide` という名前で、設定するものがないため `/mcp` から非表示になっています。ただし、組織が *PreToolUse* **フック** (ページ 495) を使用して MCP ツールをホワイトリストに登録している場合は、それが存在することを知っておく必要があります。

選択とオープンファイルコンテキスト。 接続中、CLI は現在のエディター選択とアクティブファイルのパスを、送信する各プロンプトのコンテキストとして含めます。トランスクリプトには、これが発生したときに ☒ `Selected N lines from <file>` という行が表示されます。

Claude **が作業中にメッセージをキューに入れる** (ページ 90) 場合、`Enter` キーを押したときに選択していた選択内容が保持され、その後に選択したものは何でも保持されます。

`.env` などの機密ファイルを除外するには、そのパスに対して *Read* **拒否ルール** (ページ 1051) を追加してください。一致する拒否ルールは、選択されたテキストとそのファイルのオープンファイル通知の両方が Claude に到達するのを防ぎます。

トランスポートと認証。 サーバーは OS が割り当てた一時的なポートでリッスンし、ポートは設定できません。トランスポートは暗号化されていない `ws://` です。ループバック上では、トラフィックをキャプチャできるプロセスはロックファイルからトークンも読み取ることができるため、TLS はローカル攻撃者に対する保護を追加しません。IDE の起動ごとに新しいランダム認証トークンが生成され、`~/.claude/ide/<port>.lock` のロックファイルに書き込まれ、CLI は `X-Claude-Code-Ide-Authorization` ヘッダーとして提示して接続する必要があります。`CLAUDE_CONFIG_DIR` が設定されている場合、ロックファイルは代わりに `$CLAUDE_CONFIG_DIR/ide/` に書き込まれます。

モデルに公開されるツール。 サーバーはいくつかのツールをホストしていますが、モデルに表示されるのは 1 つだけです。残りは、diff を開いたり選択を読み取ったりするなど、CLI が独自の UI に使用する内部 RPC であり、ツールリストが Claude に到達する前にフィルタリングされます。

ツール名（フックで表示）	機能	読み取り専用
`mcp__ide__getDiagnostics`	IDE の検査診断（エディターに表示されるエラーと警告）を返します。各呼び出しは 1 つのファイルをカバーします。Claude が指定するファイル、または Claude が指定しない場合はアクティブエディター内のファイルです。	はい

JetBrains プラグインは、モデルにコード実行ツールを公開しません。

リッスンインターフェース。 サーバーがバインドするネットワークインターフェースは、**Settings** → **Tools** → **Claude Code [Beta]** → **Networking (Advanced)** の下の **Accept connections from all network interfaces** によって制御されます。設定が無効な場合、サーバーは `127.0.0.1` のみでリッスンし、他のホストからはアクセスできません。有効な場合、ポートはローカルネットワークからアクセス可能です。この設定は、WSL2 のデフォルト NAT ネットワークやリモート IDE セットアップなど、CLI がループバック経由で IDE に到達できない場合のために存在します。*WSL* **設定** (ページ 208) を参照してください。

警告:

Accept connections from all network interfaces を有効にすると、IDE MCP ポートがローカルネットワークからアクセス可能になります。接続にはロックファイルからの認証トークンが必要ですが、トランスポートが暗号化されていない `ws://` であるため、設定がオンの場合、セッショントラフィックとそのトークンの両方がネットワーク上をクリアテキストで通過します。ループバックが本当に機能しない場合にのみオンにしてください。WSL2 の場合は、**ミラーリングネットワークに切り替える** (ページ 208) ことをお勧めします。これにより、Windows ループバックインターフェースが Linux VM と共有され、ソケットはループバック上に留まります。

第12章　Chrome で Claude Code を使用する

Claude Code を Chrome ブラウザに接続して、Web アプリをテストし、コンソールログでデバッグし、フォーム入力を自動化し、Web ページからデータを抽出します。

Claude Code は Claude in Chrome ブラウザ拡張機能 と統合され、CLI または *VS Code* **拡張機能** (ページ 189) からブラウザ自動化機能を提供します。コードをビルドしてから、コンテキストを切り替えることなくブラウザでテストおよびデバッグできます。

Claude はブラウザタスク用に新しいタブを開き、ブラウザのログイン状態を共有するため、既にサインインしているサイトにアクセスできます。ブラウザアクションはリアルタイムで表示される Chrome ウィンドウで実行されます。Claude がログインページまたは CAPTCHA に遭遇した場合、一時停止して手動で処理するよう求めます。

拡張機能は、Claude が開いたタブをセッションに紐付けられた Chrome タブグループにまとめます。ローカルセッションでは、セッション終了時に Claude Code がそのグループを閉じるかどうかは、セッションの終了方法によって異なります。

- `/clear` と入力すると、クリア後も継続する作業がまだ実行中でない限り、Claude Code は開いているページを含めてグループを閉じます
- `/resume` などのコマンドでセッションを切り替えた場合、Claude Code を終了した場合、またはクリア後も継続する作業がまだ実行中の状態で `/clear` を実行した場合、Claude Code はグループに空の新しいタブしか含まれていないときにのみグループを閉じます。そのため、まだ読んでいる可能性のあるページは開いたままになります

注釈:

Chrome 統合は Google Chrome と Microsoft Edge で動作します。Claude Code は、Brave、Arc、Vivaldi、Opera など、その他の Chromium ベースのブラウザでも拡張機能を検出して接続を設定します。Chrome 統合は Windows Subsystem for Linux（WSL）ではサポートされていません。

12.1 機能

Chrome が接続されている場合、単一のワークフロー内でブラウザアクションとコーディングタスクをチェーンできます。

- **ライブデバッグ**：コンソールエラーと DOM 状態を直接読み取り、それらを引き起こしたコードを修正します
- **デザイン検証**：Figma モックから UI をビルドしてから、ブラウザで開いて一致することを確認します
- **Web アプリテスト**：フォーム検証をテストし、ビジュアルリグレッションをチェックするか、ユーザーフローを検証します

- **認証済み Web アプリ**：API コネクタなしで、ログインしている Google Docs、Gmail、Notion、またはその他のアプリと対話します
- **データ抽出**：Web ページから構造化情報を取得してローカルに保存します
- **タスク自動化**：データ入力、フォーム入力、またはマルチサイトワークフローなどの反復的なブラウザタスクを自動化します
- **ファイルのアップロード**：マシン上のファイルを Web ページのアップロードフィールドに添付します
- **セッション記録**：ブラウザインタラクションを GIF として記録して、何が起こったかを文書化または共有します

12.2 前提条件

Claude Code を Chrome で使用する前に、以下が必要です。

- Google Chrome、Microsoft Edge、または Brave、Arc、Vivaldi、Opera などのその他の Chromium ベースのブラウザ
- Claude in Chrome 拡張機能 バージョン 1.0.36 以上（Chrome Web Store で入手可能）
- *Claude Code* (ページ 11)
- 直接 Anthropic プラン（Pro、Max、Team、または Enterprise）

Chrome 統合を使用するには、`/login` でサインインする必要もあります。API キーまたは *`claude setup-token`* (ページ 860) で発行した長期間有効なトークンで認証している場合、ブラウザ拡張機能はこれらの認証情報では認証できないため、`--chrome` を渡しても Claude Code は Chrome 統合を無効のままにします。v2.1.216 より前のバージョンでは、これらのセッションでも Chrome 統合を有効にできましたが、ブラウザ拡張機能への接続はすべて 403 エラーで失敗していました。

注釈:

Chrome 統合は Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry などのサードパーティプロバイダーを通じては利用できません。Claude にサードパーティプロバイダーを通じてのみアクセスする場合、この機能を使用するには別の claude.ai アカウントが必要です。

12.3 CLI で開始する

1. **Chrome で Claude Code を起動する**

   `--chrome` フラグで Claude Code を起動します。

   ```
   claude --chrome
   ```

   Chrome で初めて起動すると、Claude Code は統合の紹介とサイト権限の仕組みを説明する 1 回限りのダイアログを表示します。Enter キーを押して続行します。

   フラグなしで今後のセッションでも Chrome を有効にするには、*Chrome* **をデフォルトで有効にする** (ページ 216) を参照してください。

2. **Claude にブラウザを使用するよう依頼する**

   この例では、ページに移動してそのページを操作し、見つけた内容を報告します。これらはすべてターミナルまたはエディターから行えます。

   ```
   Go to code.claude.com/docs, click on the search box,
   type "hooks", and tell me what results appear
   ```

   ブラウザアクションの前に Claude Code が権限を求めた場合は、承認してください。ダイアログは `Claude in Chrome wants to` で始まり、そのサイトでのすべてのアクションをセッション中許可するオプションが表示されます。Claude は新しいタブを開いてタスクを開始します。

いつでも `/chrome` を実行して、接続ステータスの確認、権限の管理、拡張機能の再接続、または使用する接続済みブラウザの選択を行えます。ステータスパネルに「ステータス: 有効」と「拡張機能: インストール済み」が表示されていれば、統合は正常に動作しています。

複数のブラウザが接続されている場合は、Claude が使用するブラウザを選択します。選択する前にブラウザアクションが開始されると、Claude はいずれかを選択するよう促します。後でブラウザを切り替えるには、`/chrome` を実行して **ブラウザを選択…** を選びます。別のブラウザが接続された場合でも、Claude は選択したブラウザを使い続けます。

VS Code については、*VS Code* **でのブラウザ自動化** (ページ 189) を参照してください。

12.3.1 Claude から求められたときに拡張機能をインストールする

対話型セッションで Claude がブラウザを必要とし、Claude Code が拡張機能を検出できない場合、Claude Code は「Claude がブラウザの使用を求めています」というタイトルのインストールプロンプトを表示します。Claude Code が確認するのは 1 セッションにつき最大 1 回です。

プロンプトには 3 つの選択肢があります。

- **拡張機能をインストール**: ブラウザで拡張機能のインストールページを開き、ガイド付きセットアップを開始します。Claude Code はインストールを待機し、拡張機能を接続して、同じセッション内でブラウザツールを有効にします。接続の準備ができたら「ブラウザツールを使用して続行」を選択すると、Claude はブラウザでタスクを再開します。「ブラウザツールなしで続行」を選択してセットアップを中断し、後で `/chrome` で完了することもできます。
- **今はしない**: ブラウザツールなしでタスクを続行します。Claude Code は後のセッションで再度確認することがあります。
- **今後確認しない**: 今後のセッションでプロンプトを表示しないようにします。`/chrome` を使えば、いつでも統合をセットアップできます。

次の 2 つの管理 MCP ポリシーによってプロンプトはオフになります。

- 組織が deniedMcpServers 管理設定で `claude-in-chrome` MCP サーバーをブロックしている場合、Claude Code はインストールプロンプトを表示しません。
- 組織が `managed-mcp.json` ファイルをデプロイしており、管理対象セットと併せて Claude in Chrome を許可していない場合、Claude Code はインストールプロンプトを表示しません。

12.3.2 Chrome をデフォルトで有効にする

毎回 `--chrome` を渡さずに Chrome に接続するには、CLI セッションで `/chrome` を実行して **デフォルトで有効** を選択します。*VS Code* **拡張機能** (ページ 189) では、プロンプトボックスに `/chrome` と入力して **デフォルトで有効** スイッチをオンにします。両者は 1 つの設定を共有しているため、どちらかでオンにすると CLI と VS Code の両方でオンになります。

この設定がオンで Claude Code v2.1.287 以降を使用している場合、各 VS Code セッションは開始時にブラウザに接続するため、`@browser` と入力する前から Claude がブラウザを使用できます。それより前のバージョン、または設定がオフの場合、VS Code セッションは `@browser` と入力したときに接続します。

Chrome が実行されていない場合でも、Claude Code は通常どおり起動します。v2.1.211 より前では、Chrome 統合が有効で Chrome が実行されていない場合に、起動が停止することがありました。

注釈:

Chrome をデフォルトで有効にすると、ブラウザツールとその指示が常にロードされるため、コンテキスト使用量が増加します。コンテキスト消費の増加に気付いた場合は、この設定をオフにして、必要なときにのみ接続してください。CLI では `--chrome` を使用し、VS Code ではプロンプトボックスに *`@browser`* (ページ 189) と入力します。

12.3.3 サイト権限を管理する

サイトレベルの権限は Chrome 拡張機能から継承されます。Chrome 拡張機能の設定で権限を管理して、Claude がブラウズ、クリック、入力できるサイトを制御します。auto モードでは、auto モードの分類器自体があるサイトへのブラウザ呼び出しを承認した場合、権限ルールで Claude in Chrome に対していずれかのサイトを拒否していない限り、拡張機能はその呼び出しについて独自のサイトごとのチェックを省略します。

12.3.4 VS Code セッションでの権限プロンプト

VS Code セッションでは、ブラウザアクションの前に Claude Code が確認するかどうかは、セッションがブラウザに接続した方法によって異なります。

- **`@browser` と入力した場合**: Claude Code が通常であれば確認するブラウザアクションを、拡張機能がそれぞれ承認します。
- **デフォルトで有効** (ページ 216) **設定によって開始時に接続された場合**: そのセッションで `@browser` と入力するまで、Claude Code は Manual、Edit automatically、Auto、Bypass permissions の各モードで、許可していないサイトでのブラウザアクションの前に確認します。

12.3.5 plan モードでのブラウザツール

plan モードでは、Claude が GIF を記録する、新しいタブを開く、またはショートカットを実行する前に権限プロンプトが表示されます。ただし、*`@browser`* (ページ 216) と入力した VS Code セッションは除きます。対話型 CLI セッションでは、bypassPermissions モードが利用可能で、かつ機能フラグの取得がオフの場合、これらの呼び出しはプロンプトなしで実行されます。

`createIfEmpty` を設定する `tabs_context_mcp` 呼び出しや、これらのアクションのいずれかを含む `browser_batch` 呼び出しでもプロンプトが表示されます。

12.4 ワークフロー例

これらの例は、ブラウザアクションとコーディングタスクを組み合わせる一般的な方法を示しています。/mcp を実行して claude-in-chrome を選択し、**View tools** を選択すると、利用可能なブラウザツールの完全なリストが表示されます。

12.4.1 ローカル Web アプリケーションをテストする

Web アプリを開発する場合、変更が正しく機能することを確認するよう Claude に依頼します。

```
I just updated the login form validation. Can you open localhost:3000,
try submitting the form with invalid data, and check if the error
messages appear correctly?
```

Claude はローカルサーバーに移動し、フォームと対話し、観察したことを報告します。

12.4.2 コンソールログでデバッグする

Claude はコンソール出力を読み取って問題の診断を支援できます。ログが詳細になる可能性があるため、すべてのコンソール出力を要求するのではなく、探すパターンを Claude に伝えます。

```
Open the dashboard page and check the console for any errors when
the page loads.
```

Claude はコンソールメッセージを読み取り、特定のパターンまたはエラータイプでフィルタリングできます。

12.4.3 フォーム入力を自動化する

反復的なデータ入力タスクを高速化します。

```
I have a spreadsheet of customer contacts in contacts.csv. For each row,
go to the CRM at crm.example.com, click "Add Contact", and fill in the
name, email, and phone fields.
```

Claude はローカルファイルを読み取り、Web インターフェースをナビゲートし、各レコードのデータを入力します。

12.4.4 Web ページにファイルをアップロードする

Claude は、マシン上のファイルをページ上のアップロードフィールドに添付できます。Claude Code がファイルを読み取ってその内容をブラウザに送信するため、アップロードはローカルセッションとリモートセッションの両方で機能します。Claude Code v2.1.211 以降が必要です。

この例では、ログファイルをフォームに添付します。

```
Open the bug tracker at bugs.example.com, create a new issue,
and attach logs/session.log to it
```

アップロードには次の 3 つの制限が適用されます。

- **権限**: Claude がファイルをアップロードできるのは、セッションがそのファイルの読み取りを許可されている場合のみです。そのため、ファイルへの Read アクセスを拒否する権限ルールは、そのファイルのアップロードもブロックします。
- **サイズ**: 1 回のアップロードに含められるファイルは合計 10 MB までです。
- **ハードリンク**: Claude は複数のハードリンクを持つファイルを拒否します。これは node_modules などのパッケージマネージャーのストア内でよく見られます。ファイルをコピーし、そのコピーをアップロードしてください。

12.4.5 Google Docs でコンテンツをドラフトする

API セットアップなしで Claude を使用してドキュメントに直接書き込みます。

```
Draft a project update based on the recent commits and add it to my
Google Doc at docs.google.com/document/d/abc123
```

Claude はドキュメントを開き、エディターをクリックしてコンテンツを入力します。これは、ログインしているあらゆる Web アプリで機能します。Gmail、Notion、Sheets など。

12.4.6 Web ページからデータを抽出する

Web サイトから構造化情報を取得します。

```
Go to the product listings page and extract the name, price, and
availability for each item. Save the results as a CSV file.
```

Claude はページに移動し、コンテンツを読み取り、データを構造化形式にコンパイルします。

12.4.7 マルチサイトワークフローを実行する

複数の Web サイト間でタスクを調整します。

```
Check my calendar for meetings tomorrow, then for each meeting with
an external attendee, look up their company website and add a note
about what they do.
```

Claude はタブ間で動作して情報を収集し、ワークフローを完了します。

12.4.8 デモ GIF を記録する

ブラウザインタラクションの共有可能な記録を作成します。

```
Record a GIF showing how to complete the checkout flow, from adding
an item to the cart through to the confirmation page.
```

Claude はインタラクションシーケンスを記録し、GIF ファイルとして保存します。記録にはブラウザに表示されるすべてのもの（ログイン済みページのアカウント詳細を含む）が含まれるため、チーム外と共有する前に内容を確認してください。

12.4.9 スクリーンショットをディスクに保存する

スクリーンショットをファイルとして保存するよう Claude に依頼します。

```
Take a screenshot of the checkout page and save it to disk
```

Claude は画像をディスクに保存し、ファイルパスを報告します。v2.1.211 より前では、スクリーンショットツールの `save_to_disk` オプションはファイルを書き込みませんでした。

12.5 トラブルシューティング

12.5.1 拡張機能が検出されない

Claude Code が Chrome 拡張機能を検出できない場合：

1. Chrome 拡張機能が `chrome://extensions` にインストールされ、有効になっていることを確認します
2. `claude --version` を実行して Claude Code が最新であることを確認します
3. Chrome が実行されていることを確認します
4. `/chrome` を実行して「Reconnect extension」を選択し、接続を再確立します
5. 問題が解決しない場合、Claude Code と Chrome の両方を再起動します

Chrome 統合を初めて有効にすると、Claude Code はネイティブメッセージングホスト設定ファイルをインストールします。Chrome はスタートアップ時にこのファイルを読み取るため、最初の試行で拡張機能が検出されない場合、Chrome を再起動して新しい設定を取得します。

Claude Code は、初回インストール時にのみ拡張機能の接続を促すブラウザタブを開きます。後続のセッションで設定ファイルが書き直された場合（例えばビルドや設定ディレクトリを切り替えた後）、Claude Code はタブを再度開きません。

接続がまだ失敗する場合、ホスト設定ファイルが以下の場所に存在することを確認します。

Chrome の場合：

- **macOS**：`~/Library/Application Support/Google/Chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Linux**：`~/.config/google-chrome/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Windows**：Windows レジストリで `HKCU\Software\Google\Chrome\NativeMessagingHosts\` を確認します

Edge の場合：

- **macOS**：`~/Library/Application Support/Microsoft Edge/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`

- **Linux**：`~/.config/microsoft-edge/NativeMessagingHosts/com.anthropic.claude_code_browser_extension.json`
- **Windows**：Windows レジストリで `HKCU\Software\Microsoft\Edge\NativeMessagingHosts\` を確認します

その他の Chromium ベースのブラウザは、ブラウザ名にちなんだ独自の設定ディレクトリから同じファイルを読み取ります。例えば、macOS 上の Brave は `~/Library/Application Support/BraveSoftware/Brave-Browser/NativeMessagingHosts/` を使用し、Windows では各ブラウザが `HKCU\Software\BraveSoftware\Brave-Browser\NativeMessagingHosts\` のような独自のレジストリキーを持ちます。

12.5.2 ブラウザが応答しない

Claude のブラウザコマンドが機能しなくなった場合：

1. モーダルダイアログ（alert、confirm、prompt）がページをブロックしているかどうかを確認します。JavaScript ダイアログはブラウザイベントをブロックし、Claude がコマンドを受け取るのを防ぎます。ダイアログを手動で閉じてから、Claude に続行するよう伝えます。
2. Claude に新しいタブを作成して再度試すよう依頼します
3. `chrome://extensions` で拡張機能を無効にしてから再度有効にして Chrome 拡張機能を再起動します

12.5.3 長いセッション中に接続が切れる

Chrome 拡張機能のサービスワーカーは長時間のセッション中にアイドル状態になる可能性があり、接続が切れます。非アクティブ期間後にブラウザツールが機能しなくなった場合、`/chrome` を実行して「Reconnect extension」を選択します。

12.5.4 Windows 固有の問題

Windows では、以下の問題が発生する可能性があります。

- **名前付きパイプの競合（EADDRINUSE）**：別のプロセスが同じ名前付きパイプを使用している場合、Claude Code を再起動します。Chrome を使用している他の Claude Code セッションを閉じます。
- **ネイティブメッセージングホストエラー**：ネイティブメッセージングホストがスタートアップ時にクラッシュする場合、Claude Code を再インストールしてホスト設定を再生成してみてください。
- **セットアップページが開かない**：Claude Code を更新します。v2.1.211 より前では、Windows で拡張機能の接続を促すブラウザタブが開かないことがありました。

12.5.5 一般的なエラーメッセージ

これらは最も頻繁に遭遇するエラーと、それらを解決する方法です。

エラー	原因	修正
"Browser extension is not connected"	ネイティブメッセージングホストが拡張機能に到達できない、または組織の IP 許可リストが `bridge.claudeusercontent.com` への接続を拒否している	Chrome と Claude Code を再起動してから、`/chrome` を実行して再接続します。組織が IP 許可リストを使用しており、エラーが解決しない場合は、**組織の *IP* 許可リストとプロキシのエグレス** (ページ 907) を参照してください
`/chrome` で拡張機能に「Not detected」と表示される	Chrome 拡張機能がインストールされていないか、無効になっている	`chrome://extensions` で拡張機能をインストールまたは有効にします
"No tab available"	Claude がタブの準備ができる前に動作しようとした	Claude に新しいタブを作成して再度試すよう依頼します
"Receiving end does not exist"	拡張機能サービスワーカーがアイドル状態になった	`/chrome` を実行して「Reconnect extension」を選択します

12.6 関連項目

- コンピュータ使用：ブラウザでタスクを実行できない場合にネイティブ macOS アプリを制御します
- ***VS Code* で *Claude Code* を使用する** (ページ 189)：VS Code 拡張機能でのブラウザ自動化
- ***CLI* リファレンス** (ページ 45)：`--chrome` を含むコマンドラインフラグ
- **一般的なワークフロー** (ページ 105)：Claude Code を使用するその他の方法
- **データとプライバシー** (ページ 1105)：Claude Code がデータを処理する方法
- Chrome で Claude を使い始める：ショートカット、スケジューリング、権限を含む Chrome 拡張機能の完全なドキュメント

第13章　クラウドで Claude Code を使用する

ブラウザ、携帯電話、デスクトップアプリ、またはターミナルからクラウドで Claude Code セッションを実行し、--cloud と --teleport で移動し、プルリクエストを自動修正します。

注釈:

クラウドセッションは Pro、Max、Team プランで利用でき、Premium シートまたは Chat + Claude Code シートを持つ Enterprise ユーザーも対象です。

クラウドセッションは、マシン上ではなくクラウドインフラストラクチャで実行される Claude Code セッションです。デフォルトでは Anthropic が管理するインフラストラクチャで実行されるか、ルーティングされた場合は組織のセルフホスト環境で実行されます。セッションはラップトップを閉じた後も実行を続け、任意のデバイスから確認または操作できます。

以下のいずれかのサーフェスからクラウドセッションを開始できます：

- **ブラウザ**：claude.ai/code（ウェブ上の Claude Code とも呼ばれます）
- **モバイル**：Claude アプリの **Code** タブ
- **デスクトップアプリ**：**セッションを開始** (ページ 150) するときに **Local** の代わりに **Cloud** を選択
- **ターミナル**：`claude --cloud` (ページ 225)
- **ルーティン**：スケジュール実行とトリガー実行は各々クラウドセッションとして実行されます

1 つの作業本体に対して Claude が多くのクラウドセッションを開始して追跡するには、プロジェクトを使用します。ターミナル、IDE、または **Local** が選択されたデスクトップアプリのセッションは、代わりにマシン上で実行されます。これらのローカルセッションの 1 つを携帯電話またはブラウザから操作するには、リモートコントロールを使用します。

Tip:

クラウドセッションは初めてですか？はじめにから始めて、GitHub アカウントを接続し、最初のタスクを送信してください。

このページでは以下をカバーしています：

- **クラウド環境** (ページ 224)：セッションが実行される場所、およびそれを設定する場所
- *GitHub* **認証オプション** (ページ 224)：GitHub を接続する 2 つの方法
- **ターミナルとクラウド間でタスクを移動** (ページ 225)（`--cloud` と `--teleport` を使用）

- **セッションの操作** (ページ 231)：権限モード、確認、共有、アーカイブ、削除
- **プルリクエストの自動修正** (ページ 234)：CI 失敗とレビューコメントに自動的に応答
- **セキュリティと分離** (ページ 235)：セッションの分離方法
- **制限事項** (ページ 236)：レート制限とプラットフォーム制限

13.1 クラウド環境

すべてのクラウドセッションはクラウド環境で実行されます。これはネットワークアクセス、環境変数、セットアップスクリプトを制御する保存された設定です。環境がまだない場合、オンボーディングは Trusted ネットワークアクセスを持つ **Default** 環境をセットアップします。これは、あなたのために作成するか、作成するよう依頼するかのいずれかです。Default 環境を参照して、どちらがあなたのプランで発生するか、および複数の環境がある場合にセッションが環境を選択する方法を確認してください。

同じ環境はクラウドセッションを開始する場所に関係なく適用されます：ウェブ、ターミナル、Claude Tag、ルーチン、およびモバイルおよび Desktop アプリ。Claude Tag チャネルセッションは組織レベルの環境のみを使用します。共有環境またはセルフホスト環境のいずれかです。

クラウド環境を設定して、環境が許可するものを変更し、変数を設定するか、セットアップスクリプトを追加してください。インストール済みツールを参照して、設定なしでセッションに含まれるものを確認してください。

13.2 GitHub 認証オプション

クラウドセッションはコードをクローンしてブランチをプッシュするために GitHub リポジトリへのアクセスが必要です。2 つの方法でアクセスを許可できます：

方法	接続方法	セッションが到達できるリポジトリ	最適な用途
GitHub App	ウェブオンボーディング中に Claude GitHub App を認可します	任意のパブリックリポジトリ、および Claude GitHub App がインストールされているプライベートリポジトリ	ブラウザオンボーディング；*Auto-fix* (ページ 234) を希望するチーム
`/web-setup`	ターミナルで `/web-setup` を実行して、ローカル `gh` CLI トークンを Claude アカウントに送信します	`gh` トークンがアクセスできる任意のリポジトリ（App がインストールされているかどうかに関わらず）	すでに `gh` を使用している個別開発者

Claude GitHub App をリポジトリにインストールすると、そのリポジトリのプルリクエストに対して *Auto-fix* (ページ 234) も有効になります。

プロジェクト内のスレッドは、接続方法に関わらず、クローンする各リポジトリに Claude GitHub App がインストールされている必要があります。GitHub アクセスの設定を参照してください。

Anthropic ホスト環境では、GitHub 認証情報は Anthropic のサーバー上で暗号化されたままであり、セッションの VM に入ることはありません。VM からの GitHub 操作は GitHub プロキシを通じて行われ、サー

バー側で認証情報を添付します。

`/schedule` がルーチンを作成する前にリポジトリアクセスをチェックする方法については、リポジトリとブランチの権限を参照してください。`/web-setup` のウォークスルー（`/web-setup` が保存する内容と削除方法を含む）については、ターミナルから接続を参照してください。

Quick web setup は、メンバーが `/web-setup` で GitHub を接続できるようにする組織設定で、ブラウザオンボーディング中に Claude GitHub App インストールプロンプトをスキップし、環境フォームを表示する代わりに、ブラウザオンボーディングが Default 環境を作成するようにします。Team および Enterprise プランではデフォルトでオフになっており、`/web-setup` を非表示にします。Owner は Admin settings > Claude Code の **Quick web setup** トグルでオンにします。

注釈:

Zero Data Retention が有効な組織は `/web-setup` またはその他のクラウドセッション機能を使用できません。

13.3 ターミナルとクラウド間でタスクを移動

これらのワークフローには *Claude Code CLI* (ページ 11) が同じ claude.ai アカウントにサインインしている必要があります。ターミナルから新しいクラウドセッションを開始するか、クラウドセッションをターミナルにプルしてローカルで続行できます。クラウドセッションはラップトップを閉じても保持され、Claude モバイルアプリを含む任意の場所から監視できます。

注釈:

CLI からのセッションハンドオフは一方向です：`--teleport` でクラウドセッションをターミナルにプルできますが、既存のターミナルセッションをクラウドにプッシュすることはできません。`--cloud` フラグはタスク説明を使用して現在のリポジトリの新しいクラウドセッションを作成します；`-p` とセッション ID または claude.ai/code URL を使用すると、代わりに**その既存セッションにメッセージをキューイング** (ページ 228) します。*Desktop* **アプリ** (ページ 151) は別のサーフェスにローカルセッションを送信できる Continue in メニューを提供します。

13.3.1 ターミナルからクラウドへ

`--cloud` フラグを使用してコマンドラインからクラウドセッションを開始します：

```
claude --cloud "Fix the authentication bug in src/auth/login.ts"
```

これにより claude.ai 上に新しいクラウドセッションが作成されます。クラウド VM はローカルチェックアウトではなく、現在のブランチで現在のディレクトリの GitHub リモートをクローンするため、ローカルコミットがある場合は最初にプッシュしてください。**ローカルリポジトリを *GitHub* なしで送信** (ページ 226) を参照して、Claude Code がリモートをクローンする代わりにローカルリポジトリをアップロードする場合を確認してください。

`--cloud` は一度に 1 つのリポジトリで機能します。タスクはクラウドで実行され、ローカルで作業を続行できます。古い `--remote` スペルは `--cloud` の非推奨エイリアスとしてまだ機能します。

クラウドコンテナが起動している間、CLI はリポジトリのクローンやセットアップスクリプトの実行などのセットアップステップのライブチェックリストを表示します。プロビジョニング中に入力したメッセージはキューに入れられ、セッションの準備ができたら送信されます。

注釈:

`--cloud` はクラウドセッションを作成します。`--remote-control` は無関係です：ローカル CLI セッションを claude.ai または Claude アプリから監視および操舵できます。Remote Control を参照してください。

claude.ai または Claude モバイルアプリでセッションを開いて進捗をチェックするか、直接対話します。そこから Claude を操舵し、フィードバックを提供するか、他のすべての会話と同じように質問に答えることができます。

Claude が質問をして、セッションがアイドル状態のままの場合、**環境の有効期限切れ** (ページ 236) まで戻ってきたときに答えることができ、セッションはあなたの答えから続行されます。

クラウドタスクのヒント

ローカルで計画し、クラウドで実行する：複雑なタスクの場合、Claude をプランモードで開始してアプローチについて協力し、その後クラウドに作業を送信します：

```
claude --permission-mode plan
```

プランモードでは、Claude はファイルを読み取り、コマンドを実行して探索し、ソースコードを編集せずにプランを提案します。計画に満足したら、リポジトリにプランを保存し、コミットしてプッシュし、クラウド VM がそれをクローンできるようにします。その後、自律実行のためにクラウドセッションを開始します：

```
claude --cloud "Execute the migration plan in docs/migration-plan.md"
```

タスクを並列で実行：各 `--cloud` コマンドは独立して実行される独自のクラウドセッションを作成します。複数のタスクを開始でき、すべて別々のセッションで同時に実行されます：

```
claude --cloud "Fix the flaky test in auth.spec.ts"
claude --cloud "Update the API documentation"
claude --cloud "Refactor the logger to use structured output"
```

セッションが完了したら、claude.ai/code から PR を作成するか、**セッションをテレポート** (ページ 229) してターミナルで作業を続行できます。

GitHub なしでローカルリポジトリを送信

git リモートがないリポジトリから `claude --cloud` を実行する場合、または Claude GitHub App がインストールされていない github.com リポジトリから実行する場合、Claude Code はローカルリポジトリをバンドルしてクラウドセッションに直接アップロードします。これは `/web-setup` で GitHub を接続した場合でも適用されます。

完全なクローンの場合、バンドルにはすべてのブランチにわたるリポジトリ履歴と、追跡されたファイルへのコミットされていない変更が含まれます。

機密ファイルへのコミットされていない変更の扱いは、プラットフォームによって異なります：

- **macOS、Linux、WSL**：Claude Code は、認証情報やキーのような名前のファイルへのコミットされていない変更をアップロードから除外します。これには `.env` ファイル、Terraform `*.tfvars` ファイル、および `id_rsa` や `*.pem` などのキーファイルが含まれます。また、Git LFS などの git フィルターが管理するファイルへのコミットされていない変更も除外します。`Left on this machine:` という通知に除外されたファイルの名前が表示され、セッションは各ファイルのコミット済みバージョンで開始されます。コミット済みのものがない場合は、そのファイルなしで開始されます。
- **ネイティブ Windows**：追跡されたファイルへのコミットされていない変更は、ファイル名にかかわらずそのままアップロードされます。クラウドセッションに含めたくない編集は、セッションを開始する前にスタッシュするか元に戻してください。

Claude Code がリモートからクローンする場合でも強制するには、`CCR_FORCE_BUNDLE=1` を設定します：

```
CCR_FORCE_BUNDLE=1 claude --cloud "Run the test suite and fix any failures"
```

バンドルされたリポジトリはこれらの制限を満たす必要があります：

- ディレクトリは少なくとも 1 つのコミットを持つ git リポジトリである必要があります
- バンドルされたリポジトリは 100 MB 未満である必要があります。より大きなリポジトリは現在のブランチのみをバンドルすることにフォールバックし、その後ワーキングツリーの単一の圧縮スナップショットにフォールバックし、スナップショットがまだ大きすぎる場合のみ失敗します
- 追跡されていないファイルは含まれません。クラウドセッションが見るべきファイルで `git add` を実行します
- macOS、Linux、WSL では、Claude Code は属性ルールがファイルに適用される方法に影響する git 設定（インクルードされた設定ファイルで設定された `core.attributesFile` など）に従うことができない場合、アップロードを拒否します。拒否メッセージは設定と修正に名前を付けます
- バンドルから作成されたセッションは、*GitHub* **接続** (ページ 224) がそのリポジトリへのプッシュアクセスを持つ場合にのみ、GitHub リモートにプッシュバックできます

macOS、Linux、WSL では、アップロードには git 2.31 以降と、サポートされているチェックアウト構成も必要です。一方、ネイティブ Windows では、Claude Code はどちらのチェックも行わずにアップロードします。チェックアウトがこれらの要件を満たさない場合、Claude Code はセッションを開始しません。`Not uploading this working tree:` を含むエラーを出力し、原因と変更すべき点を示します。主な原因は次のとおりです：

- **古い git**：インストールされている git が 2.31 より古い。git を更新してから再試行してください。
- **アップロードがサポートしていないチェックアウト構成**：サブモジュール内、`git clone --separate-git-dir`、`--shared`、または `--reference` で作成したクローン内、`core.worktree` が設定されたチェックアウト内、または参照を reftable 形式で保持するリポジトリ内から開始した。代わりに、通常の `git clone` で作成したクローンのメインチェックアウトから開始してください。
- **スパースチェックアウトを使用したリンクされた worktree**：`git sparse-checkout` は worktree 独自の `config.worktree` ファイルに設定を書き込みますが、アップロードはこれを受け付けません。

そのため、これらの設定を持つ worktree はアップロードされず、Claude Code が `worktree.sparsePaths` で作成した worktree もアップロードされません。代わりにリポジトリのメインチェックアウトから開始してください。

- **作業ツリー内に置かれた git 設定**：git 設定が、チェックアウト内にあるファイルをインクルードしている（たとえば、リポジトリ内を指す `include.path` エントリ）。そのファイルを作業ツリーの外に移動するか、インクルードを削除してから再試行してください。

macOS、Linux、WSL では、`git clone --filter` で作成した部分クローンは、クローンがすべての追跡ファイルをローカルに保持している限り、履歴なしの作業ツリーのスナップショットとしてアップロードされます。

`claude --cloud` の場合、リポジトリが GitHub 上にあれば、アップロードとその要件を回避できます。ブランチをプッシュし、リポジトリに Claude GitHub App をインストールしてから、GitHub からクローンされるようにセッションを再度開始してください。

13.3.2 CLI からフォローアップを送信

クラウドセッションが実行されている場合、どこで実行されていても、`claude auth login` でサインインしている任意のマシンの `claude` CLI からそれにフォローアップメッセージを送信します。CLI は Anthropic アカウント認証情報で認証し、ローカルセッション状態を送信しないため、コマンドはセッションを開始したマシンから実行する必要がなく、PowerShell を含むすべてのシェルで同じです。

コマンドは 1 つのメッセージを投稿して終了します：

```
claude -p "your message" --cloud <session-id>
```

CLI はメッセージをセッションにキューイングして、返信を待たずに終了します。長時間実行されるセッションを操舵し、現在のセッションがまだ終了している間に次のステップをキューイングするか、CI スクリプトからフォローアップを送信するために使用します。代わりに stdin でメッセージをパイプすることもできます：`echo "your message" | claude -p --cloud <session-id>`。

`<session-id>` については、`session_...` または `cse_...` などの裸の ID、またはセッションの `claude.ai/code/<id>` URL をスキーム、クエリ文字列の有無で渡します。ID は claude.ai/code のセッションリストで見つけます。

注釈:

`--cloud` には Anthropic アカウントが必要です。Claude Code が Amazon Bedrock、Google Cloud の Agent Platform、または別のサードパーティプロバイダー用に設定されている場合は利用できません。`ANTHROPIC_BASE_URL` を通じてのみ設定された *LLM* **ゲートウェイ** (ページ 841) はこのチェックではサードパーティプロバイダーとしてカウントされませんが、`claude auth login` でサインインする必要があります。組織の `allow_remote_sessions` ポリシーも有効にする必要があります。Owner は claude.ai/admin-settings/claude-code の Claude Code 管理設定でオンにできます。

出力とエラー

成功時に、コマンドはセッション ID とセッションを表示するリンクを出力します：

```
Sent to cloud session.
Session ID: session_01DiUkqY2kzbUbDmW1w96rfi
View: https://claude.ai/code/session_01DiUkqY2kzbUbDmW1w96rfi?from=cli&m=0
```

マシン可読結果の場合は `--output-format json` を渡します：成功時は `{ok, session_id, url}`、送信が失敗した場合は `{ok: false, session_id, error}`（例えば、セッションが見つからないか、アーカイブされている場合）。設定エラー（サポートされていないプロバイダーや無効な組織ポリシーなど）は JSON なしで stderr に出力されます。`--output-format stream-json` は `--cloud <session-id>` ではサポートされていません。

CLI はエラーの前に `Error:` を付けます。失敗した配信は `failed to send message to cloud session <id>: <reason>` としてラップされます。

メッセージ	意味
`Cloud sessions aren't available with <provider>. They run on Anthropic's infrastructure and require an Anthropic account.`	Claude Code はサードパーティプロバイダー用に設定されています。メッセージは設定が使用するラベル（`Amazon Bedrock` や `Google Vertex AI` など）でプロバイダーに名前を付けます。そのプロバイダーの設定を削除します（例えば、`CLAUDE_CODE_USE_BEDROCK` を設定解除）、Anthropic アカウント（`claude auth login`）でサインインします。
`Cloud sessions are disabled by your organization's policy. Contact your organization admin to enable them.`	`allow_remote_sessions` 組織ポリシーがオフです。
`Couldn't verify your organization's policy for cloud sessions. Check your network connection and try again.`	Claude Code は組織のポリシーをフェッチできなかったため、クラウドセッションが許可されていると仮定するのではなく、送信を拒否します。ネットワーク接続を確認して再試行してください。
`Attaching to an existing cloud session is not enabled for your account.`	`-p` なしで `--cloud <session-id>` を実行しました。`claude -p "your message" --cloud <session-id>` でメッセージを送信します。
`Session not found: <id>`	ID または URL はアクセスできるセッションと一致しません。セッションの claude.ai/code URL に対して確認してください。
`cloud session <id> is archived and cannot accept new messages`	セッションはアーカイブされています。代わりに新しいセッションを開始します。

13.3.3 クラウドからターミナルへ

以下のいずれかを使用してクラウドセッションをターミナルにプルします：

- **`--teleport` を使用**：コマンドラインから `claude --teleport` を実行してインタラクティブセッションピッカーを表示するか、`claude --teleport <session-id>` を実行して特定のセッションを直接

再開します。コミットされていない変更がある場合は、最初にそれらをスタッシュするよう求められます。

- `/teleport` **を使用**：既存の CLI セッション内で `/teleport` または `/tp` を実行して、Claude Code を再起動せずに同じセッションピッカーを開きます。
- `/tasks` **から**：`/tasks` を実行してバックグラウンドセッションを表示し、`t` を押してセッションにテレポートします。
- **claude.ai/code から**：セッションメニューから **Open in > Terminal** を選択して、ターミナルに貼り付けられるコマンドをコピーします。
- **クラウドセッション内から**：`/teleport` を入力すると、Claude Code はそのセッションの正確な `claude --teleport <session-id>` コマンドで返信し、リポジトリのチェックアウトから実行する準備ができています。セッションの環境で Claude Code v2.1.223 以降が必要です。

セッションをテレポートすると、Claude はあなたが正しいリポジトリにいることを確認し、クラウドセッションからブランチをフェッチしてチェックアウトし、完全な会話履歴をターミナルに読み込みます。ターミナルはセッションの独自のコピーを取得します：そこでの新しい作業はローカルのままで、claude.ai または Claude モバイルアプリのクラウドセッションに表示されません。テレポート後に電話から操舵を続けるには、ローカルセッションで `/remote-control` を開始します。

`--teleport` は `--resume` とは異なります。`--resume` はこのマシンのローカル履歴から会話を再開し、クラウドセッションをリストしません。`--teleport` はクラウドセッションとそのブランチをプルします。

テレポート要件

テレポートはセッションを再開する前にこれらの要件をチェックします。要件が満たされていない場合は、エラーが表示されるか、問題を解決するよう求められます。

要件	詳細
クリーンな git 状態	作業ディレクトリにコミットされていない変更がないことが必要です。テレポートは必要に応じて変更をスタッシュするよう求めます。
正しいリポジトリ	フォークではなく、同じリポジトリのチェックアウトから `--teleport` を実行する必要があります。別のリポジトリのチェックアウトから実行する場合、Claude Code はセッションのリポジトリとチェックアウトのリポジトリの両方を示すエラーを表示します。v2.1.219 より前では、エラーはチェックアウトのリポジトリを示していませんでした。Claude Code がリモートをホスト名に解析できない場合（`git@work:owner/repo.git` のような SSH ホストエイリアスなど）、確認を求め、リモートの所有者とリポジトリ名がセッションのリポジトリと一致する場合、チェックアウトを受け入れます。
ブランチが利用可能	クラウドセッションからのブランチがリモートにプッシュされている必要があります。テレポートは自動的にフェッチしてチェックアウトします。
同じアカウント	クラウドセッションで使用された同じ claude.ai アカウントに認証される必要があります。

Teleport がセッションのブランチをフェッチする際、フェッチがターミナルでの入力を待つことはありません。git や ssh がパスワード、鍵のパスフレーズ、または新しい SSH ホストの確認を求める場合、フェッチは失敗し、その後のチェックアウトはローカルクローンにすでにそのブランチがある場合にのみ成功します。SSH の 2 つのケースについては、鍵を `ssh-agent` に読み込み、事前に一度手動で `git fetch` を実行してホストを記録してください。

`--teleport` が利用できない

テレポートには claude.ai サブスクリプション認証が必要です。API キーで認証されている場合は、代わりに claude.ai アカウントでサインインするために `/login` を実行してください。エラーがプロバイダーに名前を付ける場合、クラウドセッションはサードパーティプロバイダーを通じて利用できません；**エラーテーブル** (ページ 229) を参照してください。claude.ai 経由で既にサインインしており、`--teleport` がまだ利用できない場合は、組織がクラウドセッションを無効にしている可能性があります。

13.4 セッションの操作

セッションは claude.ai/code のサイドバーに表示されます。そこから変更を確認し、チームメイトと共有し、完了した作業をアーカイブするか、セッションを永続的に削除できます。

13.4.1 キューに入ったメッセージを取り戻す

Claude が作業中にメッセージを送信すると、Claude がそれを読むまでメッセージはキューに入ります。キューに入ったメッセージを取り戻すには、その上の × をクリックします。テキストはメッセージボックスに戻るため、編集するか別のものを送信できます。

Claude がすでにメッセージを読んでいる場合、それは会話に留まります。

13.4.2 コンテキストを管理

クラウドセッションは組み込みコマンドをサポートしており、テキスト出力を生成します。ターミナルインターフェイスでのみ実行されるコマンド（`/plugin` や `/resume` など）は利用できません。ターミナルでピッカーまたはパネルを開くコマンドはクラウドセッションで異なる動作をします：

- `/model`、`/effort`、`/color`、`/rename`：ターミナルピッカーまたはスライダーを開く代わりに、引数として値を渡します。例えば `/model sonnet` のように使用します。引数形式はセッションの環境で Claude Code v2.1.205 以降が必要であり、各コマンドの利用可能性に関する注記に従います。
- `/fast`：アカウントで利用可能な場合、セッションのファストモードを切り替えます。セッションの環境で Claude Code v2.1.271 以降が必要です。
- `/config`：ブラウザの claude.ai/code では、値を設定する代わりに Claude Code セクションの設定を開き、`key=value` を含むコマンド後のテキストは無視されます。クラウドセッションの設定を変更するには、環境変数を環境に設定するか、1 つのリポジトリを持つセッションでキーをそのリポジトリの `.claude/settings.json` にコミットします。**クラウドセッションの設定** (ページ 724) には各セッションが読み込むものが記載されています。

コンテキスト管理の場合：

コマンド	クラウドセッションで機能	注記
/compact	はい	会話を要約してコンテキストを解放します。/compact keep the test output のようなオプションのフォーカス指示を受け入れます
/context	はい	現在コンテキストウィンドウにあるものを表示します
/clear	いいえ	サイドバーから新しいセッションを開始します

自動圧縮はコンテキストウィンドウが容量に近づくと自動的に実行されます。クラウドセッションは CLAUDE_AUTOCOMPACT_PCT_OVERRIDE をセッション自身で設定するため、圧縮はウィンドウが満杯になるのではなく、**自動圧縮ウィンドウ** (ページ 754) の途中でトリガーされます。その値は環境変数に追加するものをオーバーライドするため、そこに変数を追加しても圧縮がトリガーされるタイミングは変わりません。

自動圧縮ウィンドウを変更するには、環境変数で CLAUDE_CODE_AUTO_COMPACT_WINDOW を設定するか、変数が設定されていないセッションで /autocompact をトークン数で実行します。

Subagents (ページ 317) はローカルと同じように機能します。Claude は Agent ツールでそれらをスポーンして、研究または並列作業を別のコンテキストウィンドウにオフロードし、メイン会話を軽くすることができます。リポジトリの .claude/agents/ で定義された Subagents は自動的にピックアップされます。

Agent teams はデフォルトでオフですが、環境変数に CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1 を追加することで有効にできます。

13.4.3 クラウドセッションの権限モード

クラウドセッションの権限モードはモードドロップダウンから選択します。タスクを作成するときとセッションが実行されている間の両方です。Anthropic ホスト型**環境が有効期限切れ** (ページ 236) になったセッションを再度開くか、セルフホスト型ランナーがアイドル中にリリースしたセッションにメッセージを送信する場合、Claude Code はセッションが存在していた権限モードで再開されます。

13.4.4 変更を確認

各セッションは追加および削除された行数を示す diff インジケーター（例：+42 -18）を表示します。それを選択して diff ビューを開き、特定の行にインラインコメントを残し、次のメッセージで Claude に送信します。

diff ビューは、デフォルトでセッションの変更をそのベースブランチと比較します。リポジトリ内の他の任意のブランチと比較するには、**Compare against** を選択してブランチを 1 つ選びます。

Claude Code はこれらの diff を計算します。これには Claude が編集するときに表示される per-file diff が含まれます。これは raw git blob コンテンツから計算されるため、リポジトリで設定された diff ドライバーと textconv フィルターは適用されません。セッション自体のチェックアウトではないリポジトリ内のファイル（セッション中にワークスペース内にクローンされたファイルなど）の場合、per-file diff は git 比較ではなく Claude の編集そのものを表示します。

完全なウォークスルー（PR 作成を含む）については Review and iterate を参照してください。Claude が PR の CI 失敗とレビューコメントを自動的に監視するようにするには、**プルリクエストの自動修正** (ページ 234) を参照してください。

13.4.5 セッションを共有

セッションを共有するには、以下のアカウントタイプに従ってその可視性を切り替えます。その後、セッションリンクをそのまま共有します。受信者はリンクを開くと最新の状態を表示しますが、ビューはリアルタイムで更新されません。

Enterprise または Team アカウントから共有

Enterprise および Team アカウントの場合、2 つの可視性オプションは **Private** と **Team** です。Team 可視性により、セッションは claude.ai 組織の他のメンバーに表示されます。*Claude in Slack* (ページ 239) セッションは自動的に Team 可視性で共有されます。

リポジトリアクセス検証はデフォルトで有効になっており、受信者のアカウントに接続された GitHub アカウントに基づいています。アカウントの表示名はアクセス権を持つすべての受信者に表示されます。

Max または Pro アカウントから共有

Max および Pro アカウントの場合、2 つの可視性オプションは **Private** と **Public** です。Public 可視性により、セッションは claude.ai にログインしているすべてのユーザーに表示されます。

共有する前にセッションで機密コンテンツを確認してください。セッションにはプライベート GitHub リポジトリのコードと認証情報が含まれる可能性があります。リポジトリアクセス検証はデフォルトで有効になっていません。

受信者がリポジトリアクセスを持つことを要求するか、共有セッションから名前を非表示にするには、Settings > Claude Code > Sharing settings に移動します。

13.4.6 セッションをアーカイブ

セッションをアーカイブしてセッションリストを整理できます。アーカイブされたセッションはデフォルトのセッションリストから非表示になりますが、アーカイブされたセッションをフィルタリングして表示できます。

セッションをアーカイブするには、サイドバーのセッションにマウスを合わせてアーカイブアイコンを選択します。

13.4.7 セッションを削除

セッションを削除すると、セッションとそのデータが永続的に削除されます。このアクションは取り消せません。セッションは 2 つの方法で削除できます：

- **サイドバーから**：アーカイブされたセッションをフィルタリングし、削除するセッションにマウスを合わせて削除アイコンを選択します
- **セッションメニューから**：セッションを開き、セッションタイトルの横のドロップダウンを選択し、**Delete** を選択します

セッションが削除される前に確認するよう求められます。

13.5 プルリクエストの自動修正

Claude はプルリクエストを監視し、CI 失敗とレビューコメントに自動的に応答できます。Claude は PR の GitHub アクティビティをサブスクライブし、チェックが失敗するかレビュアーがコメントを残すと、Claude は調査し、明確な場合は修正をプッシュします。

注釈:

Auto-fix には Claude GitHub App がリポジトリにインストールされている必要があります。まだインストールしていない場合は、GitHub App ページからインストールしてください。

PR がどこから来たか、どのデバイスを使用しているかに応じて、auto-fix をオンにするにはいくつかの方法があります：

- **クラウドセッションで作成された PR**：claude.ai/code でセッションを開き、CI ステータスバーを開き、**Auto-fix** を選択します
- **ターミナルから**：PR のブランチにいる間に `/autofix-pr` を実行します。Claude Code は `gh` で開いている PR を検出し、クラウドセッションをスポーンし、1 ステップで auto-fix をオンにします
- **モバイルアプリから**：Claude に PR を auto-fix するよう指示します。例えば「watch this PR and fix any CI failures or review comments」
- **既存の PR**：PR URL をセッションに貼り付けて、Claude に auto-fix するよう指示します

Auto-fix は PR ごとのトグルです。監視を停止するには、claude.ai/code のセッションで CI ステータスバーを開き、**Auto-fix** トグルをクリアするか、Claude に PR の監視を停止するよう指示します。

13.5.1 Claude が PR アクティビティにどのように応答するか

auto-fix がアクティブな場合、Claude は新しいレビューコメントと CI チェック失敗を含む PR の GitHub イベントを受け取ります。各イベントについて、Claude は調査して進め方を決定します：

- **明確な修正**：Claude が修正に確信があり、以前の指示と矛盾しない場合、Claude は変更を加え、プッシュし、セッションで何が行われたかを説明します
- **曖昧なリクエスト**：レビュアーのコメントが複数の方法で解釈される可能性がある場合、または建築的に重要なものが含まれている場合、Claude は行動する前にあなたに尋ねます
- **重複または無アクション イベント**：イベントが重複している場合、または変更が不要な場合、Claude はセッションでそれを記録して続行します

ベースブランチが進み、マージコンフリクトが作成されるときに GitHub は webhook を発行しないため、auto-fix は単独でコンフリクトに反応することはできません。コンフリクトを解決するには、セッションを開き、Claude にリベースするよう依頼してください。

Claude は PR を解決する際に GitHub のレビューコメントスレッドに返信する場合があります。これらの返信はあなたの GitHub アカウントを使用して投稿されるため、あなたのユーザー名の下に表示されますが、各返信は Claude Code から来たものとしてラベル付けされるため、レビュアーはそれがエージェントによって書かれたものであり、あなたが直接書いたものではないことを知っています。

警告:

リポジトリが Atlantis、Terraform Cloud、または `issue_comment` イベントで実行されるカスタム GitHub Actions などのコメントトリガー自動化を使用する場合、Claude の返信がそれらのワークフローをトリガーする可能性があることに注意してください。auto-fix を有効にする前にリポジトリの自動化を確認し、PR コメントがインフラストラクチャをデプロイするか特権操作を実行できるリポジトリでは auto-fix を無効にすることを検討してください。

13.6 セキュリティと分離

各クラウドセッションはいくつかのレイヤーを通じてマシンおよび他のセッションから分離されます：

- **分離された仮想マシン**：各セッションは分離された Anthropic 管理 VM で実行されます。セッションが組織によってルーティングされるセルフホスト環境は、代わりに独自のインフラストラクチャで実行され、分離はデプロイメントの責任です
- **ネットワークアクセス制御**：Anthropic ホスト型環境では、ネットワークアクセスはデフォルトで制限され、無効にできます。ネットワークアクセスでアクセスレベル、デフォルト許可ドメイン、および許可リストを通過しないトラフィックを参照してください。セルフホスト型環境では、独自のネットワーク境界でセッション出力を制限します。ネットワークアクセスを無効にして実行する場合、Claude Code は Anthropic API と通信できます。これにより VM からデータが出ることを許可する可能性があります。
- **認証情報保護**：Anthropic ホスト型環境では、git 認証情報と署名キーはサンドボックスの外に留まり、プロキシはスコープ付き認証情報で認証します。セルフホスト型環境では、デプロイメントが git 認証情報を提供します；git を設定を参照してください
- **API 認証情報**：Anthropic ホスト型環境の Pro および Max プランでは、クラウド環境に追加するキーはサンドボックスの外に留まり、セッションを離れた後、一致するリクエストに添付されます。セルフホスト型環境には API 認証情報がなく、Team および Enterprise プランはまだそれらを持っていません
- **セキュアな分析**：コードは PR を作成する前に分離されたセッション環境内で分析および変更されます

13.7 トラブルシューティング

`API Error: 500`、`529 Overloaded`、`429`、または `Prompt is too long` などの会話に表示される実行時 API エラーについては、エラーリファレンスを参照してください。これらのエラーとその修正は CLI および Desktop アプリと共有されます。以下のセクションはクラウドセッションに固有の問題をカバーしています。

13.7.1 セッション作成に失敗

新しいセッションが `Session creation failed` で開始に失敗するか、プロビジョニングで停止する場合、Claude Code は VM をセッションに割り当てることができませんでした。

- status.claude.com でクラウドセッションインシデントを確認してください

- 1 分後に再試行してください。容量はオンデマンドでプロビジョニングされます
- GitHub 接続後にリポジトリが表示されないに従って、GitHub 接続がリポジトリに到達できることを確認してください

13.7.2 組織 UUID を取得できない

`claude --cloud` と `claude --teleport` には claude.ai アカウントでのサインインが必要です。API キーで認証されている場合、または保存されたアカウント詳細が古い場合、これらのコマンドは `Unable to get organization UUID` で失敗するか、API キー認証が十分でないというメッセージが表示されます。API キー認証または古いアカウント詳細を使用して、セッション ID なしで `claude --teleport` を実行すると、どちらかのメッセージの代わりにセッションピッカーで `Error loading Claude Code sessions` が表示され、同じ修正が適用されます。

`/login` を実行して claude.ai アカウントでサインインし、コマンドを再試行してください。エラーがプロバイダーに名前を付ける場合は、**エラーテーブル** (ページ 229) を参照してください：クラウドセッションはサードパーティプロバイダーを通じて利用できません。

13.7.3 Remote Control セッションの有効期限切れまたはアクセス拒否

`--teleport` はクラウドセッションが使用する同じ Remote Control セッションインフラストラクチャを通じて接続するため、認証およびセッション有効期限エラーは Remote Control の表現で表示されます。`Remote Control session expired` または `Access denied` が表示される場合があります。接続トークンは短命で、アカウントにスコープされています。

- ローカルで `/login` を実行して認証情報をリフレッシュし、再接続してください
- セッションを所有する同じアカウントにサインインしていることを確認してください
- `Remote Control may not be available for this organization` が表示される場合、Owner がクラウドセッションを組織に対して有効にしていません

13.7.4 環境の有効期限切れ

クラウドセッションは非アクティブ期間後に停止し、セッションの VM は回収されます。セッションは MCP コネクタツール呼び出しを承認するか、MCP サーバーにサインインするのを待っている間、非アクティブとしてカウントされ、その待機中に有効期限が切れる可能性があります。

claude.ai/code からセッションを再度開いて、会話履歴が復元された新しい VM をプロビジョニングしてください。VM が回収されたときにまだ実行されていたバックグラウンド作業（subagents やシェルコマンドなど）は復元されません。

13.8 制限事項

クラウドセッションをワークフローに組み込む前に、以下の制約を考慮してください。

- **レート制限**: クラウドセッションは、アカウント内のすべての Claude および Claude Code の使用状況とレート制限を共有します。複数のタスクを並行実行すると、レート制限がそれに応じてより多く消費されます。クラウド VM に対する個別の計算料金はありません。

- **時間制限**: Claude が実行するコマンドと SessionStart フックにはデフォルトのタイムアウトがあり、変更できます。セットアップスクリプトはおよそ 5 分以内に完了した場合のみキャッシュされます。時間制限 を参照してください。
- **リポジトリ認証**: クラウドセッションをターミナルに取り込むことができるのは、同じアカウントで認証されている場合のみです。
- **プラットフォーム制限**: リポジトリのクローンとプルリクエストの作成には GitHub が必要です。自己ホスト型の GitHub Enterprise Server インスタンスは Team および Enterprise プランでサポートされています。GitLab、Bitbucket、またはその他の非 GitHub リポジトリをクラウドセッションに **ローカルバンドル** (ページ 226) として送信できます。これは `CCR_FORCE_BUNDLE=1` を設定することで実現できますが、セッションはその結果をリモートにプッシュバックできません。
- **組織 IP 許可リスト**: クラウドセッションは Anthropic 管理インフラストラクチャから Anthropic API を呼び出します。これはお客様のネットワークからではなく、自己ホスト環境 のセッションはお客様自身のネットワークから呼び出します。組織で IP 許可リスト が有効になっている場合、Anthropic ホスト型のすべてのクラウドセッションは認証エラーで失敗します。同じことが Code Review および Anthropic ホスト環境で実行される routines にも適用されます。自己ホスト環境にルーティングされたルーチンは、お客様自身のネットワークから API を呼び出します。Anthropic ホスト型サービスを組織の IP 許可リストから除外するには、Anthropic サポート にお問い合わせください。

13.9 関連リソース

- クラウド環境：クラウドセッションのネットワークアクセス、環境変数、セットアップスクリプトを設定
- Projects：Claude がリポジトリ上で並列クラウドセッションを調整し、結果を報告する 1 つの会話
- Ultrareview：クラウドサンドボックスで深いマルチエージェントコードレビューを実行
- Routines：スケジュール、API 呼び出し、または GitHub イベントに応答して作業を自動化
- **フック設定** (ページ 443)：セッションライフサイクルイベントでスクリプトを実行
- すべての設定：すべての設定オプション
- **セキュリティ** (ページ 1033)：分離保証とデータ処理
- **データ使用** (ページ 1105)：Anthropic がクラウドセッションから保持するもの
- Claude Tag：Slack で実行される組織管理の @Claude で、同じクラウドインフラストラクチャで動作

第14章　Slack での Claude Code

Slack ワークスペースから直接コーディングタスクを委任する。Anthropic は Team および Enterprise ワークスペース向けにこの以前のバージョンを Claude Tag に置き換えています。Pro および Max プランではセットアップパスのままです。

警告:

このページは、各セッションが個別ユーザーのアカウントで実行される以前の Claude Code in Slack について説明しています。

- **Team および Enterprise プラン：**Anthropic は、組織の共有 ID として @Claude を実行し、管理者が設定したアクセス権限を持つ Claude Tag に置き換えています。既存の Slack アプリと @Claude ハンドルは変わらず、Anthropic アカウントチームが切り替え日を通知できます。新しいワークスペースの場合は Claude Tag をセットアップしてください。このバージョンを既に使用しているワークスペースを移行する場合は、以前の Claude in Slack から移行するを参照してください。
- **Pro および Max プラン：** Claude Tag は個別プランでは利用できないため、このページがセットアップパスのままです。

Slack での Claude Code は、Claude Code の機能を Slack ワークスペースに直接もたらします。`@Claude` にコーディングタスクをメンションすると、Claude は自動的に意図を検出し、Claude Code クラウドセッションを作成します。これにより、チームの会話を離れることなく開発作業を委任できます。

この統合は既存の Claude for Slack アプリに基づいていますが、コーディング関連のリクエストに対して Claude Code クラウドセッションへのインテリジェントなルーティングを追加しています。各セッションは自分の Claude アカウントで実行され、接続されたリポジトリと自分のプラン制限を使用します。

14.1 ユースケース

- **バグ調査と修正**: Slack チャネルで報告されたバグを Claude に調査・修正させます。
- **迅速なコードレビューと修正**: Claude にチームのフィードバックに基づいて小さな機能を実装したりコードをリファクタリングさせます。
- **協調的なデバッグ**: チームの議論が重要なコンテキスト（エラーの再現やユーザーレポートなど）を提供する場合、Claude はその情報を使用してデバッグアプローチを知らせることができます。
- **並列タスク実行**: Slack でコーディングタスクを開始しながら他の作業を続け、完了時に通知を受け取ります。

14.2 前提条件

Claude Code in Slack を使用する前に、以下を確認してください：

要件	詳細
Claude プラン	Pro、Max、Team、または Claude Code アクセス付き Enterprise（プレミアムシート、または Chat + Claude Code シート）
クラウドセッション	**クラウドセッション** (ページ 223) がアカウントで有効になっている
GitHub アカウント	claude.ai/code で接続され、少なくとも 1 つのリポジトリが認証されている
Slack 認証	Slack アカウントが Claude アプリを通じて Claude アカウントにリンクされている

14.3 Slack での Claude Code のセットアップ

1. **Slack に Claude アプリをインストールする**

   ワークスペース管理者は Slack App Marketplace から Claude アプリをインストールする必要があります。Slack App Marketplace にアクセスして「Add to Slack」をクリックしてインストールプロセスを開始します。

2. **Claude アカウントを接続する**

   アプリがインストールされた後、個別の Claude アカウントを認証します：

   1. Apps セクションで「Claude」をクリックして Slack で Claude アプリを開きます
   2. App Home タブを開きます
   3. 「Connect」をクリックして Slack アカウントを Claude アカウントにリンクします
   4. ブラウザで認証フローを完了します

3. **クラウドセッションを設定する**

   クラウドセッションがアカウント用に適切に設定されていることを確認します：

   - claude.ai/code にアクセスして、Slack に接続したのと同じアカウントでサインインします
   - GitHub アカウントがまだ接続されていない場合は接続します
   - Claude が作業するリポジトリを少なくとも 1 つ認証します

4. **ルーティングモードを選択する**

   アカウントを接続した後、Claude が Slack のメッセージをどのように処理するかを設定します。Slack の Claude App Home を開いて、**ルーティングモード**設定を見つけます。

モード	動作
Code のみ	Claude はすべての @mentions を Claude Code セッションにルーティングします。Claude を Slack で開発タスク専用に使用するチームに最適です。
Code + Chat	Claude は各メッセージを分析し、Claude Code（コーディングタスク用）と Claude Chat（執筆、分析、一般的な質問用）の間でインテリジェントにルーティングします。すべてのタイプの作業に対して単一の @Claude エントリポイントが必要なチームに最適です。

注釈:

Code + Chat モードでは、Claude がメッセージを Chat にルーティングしたがコーディングセッションが必要な場合は、「Retry as Code」をクリックして Claude Code セッションを作成できます。同様に、Code にルーティングされたが Chat セッションが必要な場合は、そのスレッドでそのオプションを選択できます。

5. **Claude をチャネルに追加する**

   Claude はインストール後、自動的にどのチャネルにも追加されません。チャネルで Claude を使用するには、そのチャネルで `/invite @Claude` と入力して招待します。Claude は追加されたチャネルの @mentions にのみ応答できます。

14.4 仕組み

14.4.1 自動検出

Code + Chat ルーティングモードでは、Slack チャネルまたはスレッドで @Claude をメンションすると、Claude は自動的にメッセージがコーディングタスクかどうかを検出します。コーディングタスクは Claude Code クラウドセッションに送信されます。その他のものは通常のチャット返信を受け取ります。Code のみモードでは、すべての @mention が Claude Code に送信されます。

また、Claude が自動的に検出しない場合でも、リクエストをコーディングタスクとして処理するよう Claude に明示的に指示することもできます。

注釈:

Slack での Claude Code はチャネル（公開または非公開）でのみ機能します。ダイレクトメッセージ（DM）では機能しません。

14.4.2 コンテキスト収集

スレッドから: スレッドで @Claude をメンションすると、そのスレッド内のすべてのメッセージからコンテキストを収集して、完全な会話を理解します。

チャネルから: チャネルで直接メンションされた場合、Claude は関連するコンテキストについて最近のチャネルメッセージを確認します。

このコンテキストは Claude が問題を理解し、適切なリポジトリを選択し、タスクへのアプローチを知らせるのに役立ちます。

警告:

@Claude が Slack で呼び出されると、Claude はリクエストをより良く理解するために会話コンテキストへのアクセスが与えられます。Claude は他のメッセージからの指示に従う可能性があるため、ユーザーは信頼できる Slack 会話でのみ Claude を使用するようにしてください。

14.4.3 セッションフロー

1. **開始**: @Claude にコーディングリクエストをメンションします
2. **検出**: Claude がメッセージを分析してコーディング意図を検出します
3. **セッション作成**: claude.ai/code で新しい Claude Code セッションが作成されます
4. **進捗更新**: Claude は作業が進むにつれて Slack スレッドにステータス更新を投稿します
5. **完了**: 完了時に、Claude は概要とアクションボタンを含めてあなたをメンションします
6. **レビュー**: 「View Session」をクリックして完全なトランスクリプトを表示するか、「Create PR」をクリックしてプルリクエストを開きます

14.5 ユーザーインターフェース要素

14.5.1 メッセージアクション

- **View Session**: ブラウザで完全な Claude Code セッションを開き、実行されたすべての作業、セッションの継続、または追加のリクエストを確認できます。
- **Create PR**: セッションの変更から直接プルリクエストを作成します。
- **Retry as Code**: Claude が最初はチャットアシスタントとして応答したがコーディングセッションが必要な場合は、このボタンをクリックしてリクエストを Claude Code タスクとして再試行します。
- **Change Repo**: Claude が誤って選択した場合、別のリポジトリを選択できます。

14.5.2 リポジトリ選択

Claude は Slack 会話のコンテキストに基づいてリポジトリを自動的に選択します。複数のリポジトリが適用される可能性がある場合、Claude は正しいものを選択できるドロップダウンを表示する場合があります。

14.6 アクセスと権限

14.6.1 ユーザーレベルのアクセス

アクセスタイプ	要件
Claude Code セッション	各ユーザーは自分の Claude アカウントでセッションを実行します
使用状況とレート制限	セッションは個別ユーザーのプラン制限に対してカウントされます
リポジトリアクセス	ユーザーは個人的に接続したリポジトリにのみアクセスできます
セッション履歴	セッションは claude.ai/code の Claude Code 履歴に表示されます

14.6.2 ワークスペースレベルのアクセス

Slack ワークスペース管理者は、Claude アプリをワークスペースで利用可能にするかどうかを制御します：

コントロール	説明
アプリのインストール	ワークスペース管理者は Slack App Marketplace から Claude アプリをインストールするかどうかを決定します
Enterprise Grid の配布	Enterprise Grid 組織の場合、組織管理者は Claude アプリへのアクセスを持つワークスペースを制御できます
アプリの削除	ワークスペースからアプリを削除すると、そのワークスペース内のすべてのユーザーのアクセスが直ちに取り消されます

14.6.3 チャネルベースのアクセス制御

アプリをインストールしても、Claude は自動的にどのチャネルにも追加されません。Claude は @mentions に応答するのは、追加されたチャネルのみです。`/invite @Claude` で招待してください。公開チャネルと非公開チャネルの両方で機能します。管理者は Claude が招待されるチャネルと、それらのチャネルへのアクセス権を持つユーザーを管理することで、Claude Code の使用を制御できます。これにより、ワークスペースレベルの権限を超えた追加のアクセス制御層が提供されます。

14.7 どこでアクセスできるか

Slack で: ステータス更新、完了概要、アクションボタンが表示されます。完全なトランスクリプトは保存され、常にアクセス可能です。

claude.ai/code で: 完全な Claude Code セッション、完全な会話履歴、すべてのコード変更、ファイル操作があります。セッションは claude.ai/code の Claude Code 履歴に保存され、過去のセッションを続行したり、参照したり、プルリクエストを作成したりできます。

Enterprise および Team アカウントの場合、Slack の Claude から作成されたセッションは自動的に組織に表示されます。詳細については、**クラウドセッション共有** (ページ 233) を参照してください。

14.8 ベストプラクティス

14.8.1 効果的なリクエストの作成

- **具体的に**: ファイル名、関数名、またはエラーメッセージが関連する場合は含めます。
- **コンテキストを提供**: 会話から明確でない場合はリポジトリまたはプロジェクトをメンションします。

- **成功を定義**:「完了」とはどういう意味か説明します。Claude はテストを書くべきですか？ドキュメントを更新しますか？ PR を作成しますか？
- **スレッドを使用**: バグや機能について議論する場合はスレッドで返信して、Claude が完全なコンテキストを収集できるようにします。

14.8.2 Slack とウェブの使い分け

Slack を使用する場合: コンテキストが既に Slack の議論に存在する場合、タスクを非同期で開始したい場合、またはチームメイトが可視性を必要とする場合に協力しています。

ウェブを直接使用する場合: ファイルをアップロードする必要がある場合、開発中のリアルタイムインタラクションが必要な場合、またはより長く複雑なタスクに取り組んでいる場合。

14.9 トラブルシューティング

14.9.1 「Claude Code がアカウントで有効になっていません」

このエラーは、Claude アカウントにまだクラウド環境がないことを意味します。Slack に接続したのと同じアカウントで claude.ai/code に 1 回サインインして、ウェブオンボーディングを完了してください。これにより、デフォルトのクラウド環境が作成されるか、作成するよう求められます。エラーは次回のメンション時に解消されます。各ユーザーが個別に実行する必要があります。

14.9.2 セッションが開始しない

1. Claude アカウントが Claude App Home で接続されていることを確認します
2. クラウドセッションがアカウントで有効になっていることを確認します
3. Claude Code に接続された GitHub リポジトリが少なくとも 1 つあることを確認します

14.9.3 Claude Tag チャネルからのセッションが開始に失敗する

このエントリは Claude Tag を使用しているワークスペースに適用されます。Claude Tag では、Claude はメンバーのアカウントではなく、組織の共有 ID としてチャネルで機能します。claude.ai/code でチャネルのクラウド環境を作成した場合、それはあなたの個人アカウントに属しており、Claude は個人環境でチャネルセッションを開始できません。Claude Code はセッションを直ちに失敗させ、再試行しても役に立ちません。

Owner の場合で環境があなたのものである場合は、環境セレクターから 組織と共有してください。それ以外の場合は、Owner が admin settings の **Cloud environments** ページから組織共有環境として再作成します。

次の 2 つの方法で適用できます。

- claude.ai/admin-settings/claude-code で組織のデフォルトとして設定します。
- Claude Tag admin settings で チャネルに設定します。

Owner でない場合は、このエントリを Owner に送信してください。

14.9.4 リポジトリが表示されない

1. claude.ai/code でリポジトリを接続します
2. そのリポジトリの GitHub 権限を確認します
3. GitHub アカウントを切断して再接続してみます

14.9.5 誤ったリポジトリが選択された

1. 「Change Repo」ボタンをクリックして別のリポジトリを選択します
2. より正確な選択のためにリクエストにリポジトリ名を含めます

14.9.6 認証エラー

1. App Home で Claude アカウントを切断して再接続します
2. ブラウザで正しい Claude アカウントにサインインしていることを確認します
3. Claude プランに Claude Code アクセスが含まれていることを確認します

14.10 現在の制限事項

- **GitHub のみ**: リポジトリは GitHub 上にある必要があります。
- **一度に 1 つの PR**: 各セッションは 1 つのプルリクエストを作成できます。
- **クラウドセッションアクセスが必要**: ユーザーは **クラウドセッション** (ページ 223) へのアクセスが必要です。アクセスがない場合、Claude は標準的なチャット応答で返信します。

14.11 関連リソース

- **クラウド上の** *Claude Code* (ページ 223)

  クラウドセッションについて詳しく学ぶ

- Claude for Slack

  Claude for Slack の一般的なドキュメント

- Claude Tag

  Slack での組織管理の @Claude（管理者が設定したアクセス権限付き）

- Slack App Marketplace

  Slack Marketplace から Claude アプリをインストール

- Claude ヘルプセンター

  追加サポートを取得

第15章　Claude Code GitHub Actions

> @claude メンションに応答し、タスクを自動化し、イシューをプルリクエストに変換するために GitHub Actions ワークフロー内で Claude Code を実行します

Claude Code GitHub Actions は、リポジトリのワークフロー内で Claude Code を実行する GitHub Action です。プルリクエストまたはイシューコメントで `@claude` とメンションすると、Claude はコードを分析し、変更を実装し、コミットをプッシュします。また、Claude Code GitHub Action にプロンプトを与えて、任意の GitHub イベントで自動的に実行することもできます。イシューをプルリクエストに変換したり、コメントからバグを修正したり、繰り返されるタスクを自動化したりするために使用します。

複数の製品が Claude Code という名前を共有しています。このページは、リポジトリ内のワークフローファイルで設定する `claude-code-action` ワークフロー統合について説明しています。関連製品については、以下を参照してください。

- Code Review：ワークフローを記述せずに、すべてのプルリクエストで自動レビュー
- *Claude Code in the cloud* (ページ 223)：クラウドインフラストラクチャで実行される Claude Code セッション（マシンではなく）
- Claude Agent SDK：GitHub Actions 外のカスタム自動化。Claude Code GitHub Action は SDK の上に構築されています
- GitHub Enterprise Server：自己ホスト型 GitHub での Claude Code

15.1 セットアップ

Claude Code GitHub Action は 2 つの方法のいずれかでセットアップできます。

- **クイックセットアップ**：Claude Code から `/install-github-app` を実行します。Claude Code は GitHub App をインストールし、認証シークレットを追加し、ワークフロープルリクエストを準備します
- **手動セットアップ**：アプリをインストールし、シークレットを追加し、ワークフローファイルをリポジトリにコピーします。Claude Code をローカルで実行しない場合、コマンドが失敗した場合、またはワークフローファイルを完全に制御したい場合は、このパスを使用します

どちらのパスでも、リポジトリへの管理者アクセスが必要です。

15.1.1 クイックセットアップ

`/install-github-app` は github.com リポジトリでのみ機能します。リポジトリの git リモートが gitlab.com または bitbucket.org にある場合、コマンドは通知を出力して終了し、セットアップを開始しません。GitLab

パイプラインから Claude Code を実行するには、*Claude Code GitLab CI/CD* (ページ 261) を参照してください。

開始する前に、GitHub CLI をインストールし、`gh auth login` で認証します。Claude Code はそれをチェックし、不足している場合は警告します。

接続したいリポジトリで `claude` を開き、`/install-github-app` を実行して、プロンプトに従います。Claude Code は Claude GitHub App をインストールし、ワークフロー用の認証シークレットをセットアップします。

- Claude Code に既に API キーがある場合、そのキーを再利用し、リポジトリに既に設定されている既存の `ANTHROPIC_API_KEY` シークレットを保持することを提案します
- それ以外の場合は、Claude サブスクリプションで長期トークンを作成するか、API キーを貼り付けるかを選択します

Claude Code は認証情報をリポジトリシークレットとして保存します。API キーの場合は `ANTHROPIC_API_KEY`、サブスクリプショントークンの場合は `CLAUDE_CODE_OAUTH_TOKEN` という名前です。

Claude Code はその後、選択したワークフローファイルを含むブランチをプッシュし、既にそのシークレットを使用するように設定され、ブラウザで GitHub を開いてプルリクエストを作成する準備ができています。そのプルリクエストを作成してマージすると、リポジトリで `@claude` が機能します。

セットアップを途中で停止するには、Esc キーを押します。既に進行中のステップは完了し、後続のステップは開始しません。終了メッセージには、プッシュされたブランチやシークレットの保存など、リポジトリで既に発生したことがリストされます。

レビューワークフローを選択した場合、Claude は各レビューをプルリクエスト自体に投稿します。見つかった各イシューのインラインコメントとして、または見つからない場合は 1 つの概要コメントとして投稿します。Claude は下書きなどの一部のプルリクエストをスキップします。**レビューワークフロー例** (ページ 253) は同じスキルを使用し、それらをリストします。v2.1.229 より前では、Claude はレビューをワークフロー実行ログにのみ書き込みました。

以前のバージョンが生成したレビューワークフローを更新するには、以下のいずれかを実行します。

- `/install-github-app` を再度実行します。リポジトリに既に `claude.yml` がある場合は、**Update workflow file with latest version** を選択します。Claude Code は新しいブランチにワークフローファイルの新しいコピーをプッシュし、最初のインストールと同じようにプルリクエストを開きます。
- **レビューワークフロー例** (ページ 253) から `--comment` 引数と `claude_args` 行をチェックインファイルに自分で追加します。これにより、他の編集は保持されます。

GitHub App をインストールした後、Claude Code は GitHub Actions セットアップを続行するかどうかを尋ねます。**Skip for now** を選択して、GitHub App のインストールのみで停止します。後で `/install-github-app` を再度実行してワークフローとシークレットのステップを完了します。

注釈:

- GitHub App をインストールすると、複数の権限を付与します。完全なセットについては *GitHub App* **権限** (ページ 251) を参照してください
- クイックセットアップは Claude API と Claude サブスクリプションで機能します。Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry を使用する場合は、クラウドプロバイダーで Claude Code GitHub Actions を使用する を参照してください

15.1.2 手動セットアップ

`/install-github-app` を実行せずに Claude Code GitHub Action を設定するには、アプリをインストールし、シークレットを追加し、ワークフローファイルを自分でコピーします。

1. **Claude GitHub App をインストール**

   Claude GitHub App をリポジトリにインストールします。Claude Code GitHub Action はアプリの 3 つの権限に依存しています。

   - **Contents**：読み取りと書き込み。Claude がリポジトリファイルを変更できるようにするため
   - **Issues**：読み取りと書き込み。Claude がイシューに応答できるようにするため
   - **Pull requests**：読み取りと書き込み。Claude が PR を作成し、変更をプッシュできるようにするため

   インストール中に、他の Claude 機能が使用する権限も付与します。完全なセットについては *GitHub App* **権限** (ページ 251) を参照してください。

2. **認証シークレットを追加**

   認証方法に応じて、リポジトリに以下のシークレットのいずれかを追加します。GitHub の GitHub Actions でシークレットを使用する ガイドを参照してください。

   - `ANTHROPIC_API_KEY`：Claude Console からの Claude API キー
   - `CLAUDE_CODE_OAUTH_TOKEN`：Claude サブスクリプションで認証する OAuth トークン。Pro、Max、Team、Enterprise プランで利用可能。`claude setup-token` をローカルで実行して生成します。**長期トークンを生成する** (ページ 860) を参照してください

   ワークフローファイルで、シークレットを一致する入力に渡します。API キーの場合は `anthropic_api_key`、OAuth トークンの場合は `claude_code_oauth_token`。

3. **ワークフローファイルをコピー**

   examples/claude.yml をリポジトリの `.github/workflows/` ディレクトリにコピーします。ファイルは単なる例ではなく、動作するワークフローです。コミットされたとおり、Claude はイシューまたはプルリクエストで誰かが `@claude` とメンションするたびに応答し、`ANTHROPIC_API_KEY` シークレットで認証します。代わりに `CLAUDE_CODE_OAUTH_TOKEN` を追加した場合は、ワークフローの `anthropic_api_key` 行を `claude_code_oauth_token: ${{ secrets.CLAUDE_CODE_OAUTH_TOKEN }}` に変更します。

Tip:

セットアップ後、イシューまたは PR コメントで `@claude` をタグ付けして Claude Code GitHub Action をテストします。

15.1.3 組織向けセットアップ

クイックセットアップまたは手動セットアップでは、一度に 1 つのリポジトリを設定します。Claude Code GitHub Action を組織全体にロールアウトするには、以下を実行します。

- Claude GitHub App を組織レベルで 1 回インストールし、すべてのリポジトリまたは選択したリストを選択します
- 認証シークレットを組織レベルの Actions シークレットとして保存し、各リポジトリが独自のコピーを必要としないようにします
- Claude Code GitHub Action を実行する各リポジトリにワークフローファイルを追加するか、ジョブを 再利用可能なワークフロー として 1 回定義し、各リポジトリが呼び出すようにします

リポジトリ間で共有されるシークレットの場合、OAuth トークンは `claude setup-token` を実行した人のサブスクリプションに関連付けられているため、Claude Console から API キーで認証します。

長期シークレットを保存することを完全に回避するには、ワークロード ID フェデレーション経由で認証します。Claude Code GitHub Action はワークフローの GitHub OpenID Connect（OIDC）トークンを Claude Console サービスアカウント経由で Claude API アクセスと交換します。これらの入力を設定します。

- `anthropic_federation_rule_id`：フェデレーションルール ID、`fdrl_...`
- `anthropic_organization_id`：Anthropic 組織 ID
- `anthropic_service_account_id`：サービスアカウント ID、`svac_...`。オプション。Console で作成するフェデレーションルールが既にサービスアカウントをターゲットしているため
- `anthropic_workspace_id`：ワークスペース ID、`wrkspc_...`。フェデレーションルールが単一のワークスペースをターゲットしている場合はオプション

ワークフローに `id-token: write` 権限を付与します。これは Claude Code GitHub Action が独自の `github_token` を渡す場合でもフェデレーション交換に必要です。Console 側の設定については、Claude Code GitHub Action のセットアップガイド を参照してください。

セキュリティレビューでのデータ処理と保持に関する質問については、**データ使用** (ページ 1105) と **セキュリティ** (ページ 1033) を参照してください。

15.1.4 アンインストール

Claude Code GitHub Action を削除するには、インストールに適用される各セットアップを元に戻します。

- **ワークフローファイル**：`.github/workflows/` から `anthropics/claude-code-action` を使用するワークフローを削除します。クイックセットアップを使用した場合は、`claude.yml` を探し、レビューワークフローを選択した場合は `claude-code-review.yml` を探します。ワークフローが削除されると、Claude Code GitHub Action は実行されなくなります
- **シークレット**：リポジトリから `ANTHROPIC_API_KEY` または `CLAUDE_CODE_OAUTH_TOKEN` シークレットを削除し、**リポジトリ間で共有** (ページ 250) した場合は組織レベルの Actions シークレットから削除します。シークレットを削除しても、それが保持していた認証情報は有効なままです。API キーを完全に廃止するには、Claude Console でキーも削除します
- **GitHub App**：リポジトリまたは組織設定の GitHub Apps で Claude GitHub App をアンインストールします。ただし、Code Review や web auto-fix などの別の Claude 機能に使用しない場合のみです

クラウドプロバイダー を設定した場合は、AWS_ROLE_TO_ASSUME、GCP_* シークレット、または AZURE_* シークレットなどのプロバイダーシークレットも削除し、APP_ID と APP_PRIVATE_KEY シークレットとともにカスタム GitHub App をアンインストールします。

15.1.5 GitHub App 権限

Claude GitHub App は、Claude Code GitHub Action、Code Review、cloud sessions の Claude Code on the web の **プルリクエストの** *auto-fix* (ページ 234) など、GitHub と統合するすべての Claude 機能で共有されます。GitHub App は、すべての機能をカバーする単一の権限セットを持つため、セットには Claude Code GitHub Action が使用しない権限が含まれます。

アプリをインストールすると、以下の権限を付与します。

権限	アクセス
Actions	読み取りと書き込み
Checks	読み取りと書き込み
Contents	読み取りと書き込み
Discussions	読み取りと書き込み
Issues	読み取りと書き込み
Members	読み取り
Metadata	読み取り
Pull requests	読み取りと書き込み
Repository hooks	読み取りと書き込み
Statuses	読み取り
Workflows	読み取りと書き込み

権限セットは、それを使用する機能より前に変更される可能性があります。アプリが以前に持っていなかった権限をリクエストすると、GitHub はアカウント所有者に承認を促します。組織インストールの場合は組織所有者に促します。インストールは、承認されるまで古い権限を保持します。たとえば、Actions アクセスが読み取りから書き込みに変更されると、アプリはワークフローを再実行できるようになり、実行とログのみを表示できるようになるため、GitHub は所有者に変更を承認するよう求めます。

アプリをインストールすると、その完全な権限セットを受け入れます。GitHub では、サブセットを受け入れることはできません。組織が Claude Code GitHub Action が使用する権限のみを必要とする場合は、Claude Code GitHub Action のセットアップガイド に従って、Contents、Issues、Pull requests を持つカスタム GitHub App を作成します。カスタムアプリは Claude Code GitHub Action のみをカバーします。Code Review と web auto-fix には公式アプリが必要です。

Claude Code GitHub Action がこれらの権限で何ができるかを制限する方法の詳細については、セキュリティドキュメント を参照してください。

15.2 インタラクティブモードと自動化モード

Claude Code GitHub Action はワークフロー設定から実行方法を検出します。

- **インタラクティブモード**：ワークフローが prompt 入力を提供しない場合、Claude はトリガーフレーズ（デフォルトは @claude）をイシューまたはプルリクエストコメント、プルリクエストレビュー、

または新しく開かれたイシューの本文またはタイトルで待機し、そのリクエストに応答します。進捗と結果は、トリガーするイシューまたは PR のコメントとして表示されます。

- **自動化モード**：ワークフローが `prompt` 入力を提供する場合、Claude は **実行をトリガーできるユーザーのチェック** (ページ 252) のみに従って、待機せずに実行します。デフォルトでは、結果はコメントではなくワークフロー実行ログに表示されます。Claude は、プロンプトが指示し、**コード レビュー例** (ページ 253) のようにツールがある場合、イシューまたはプルリクエストに投稿できます。

15.2.1 実行をトリガーできるユーザー

両方のモードで、Claude Code GitHub Action はトリガーするアクターに対して 2 つのチェックを実行してから Claude を開始します。どちらかのチェックが拒否すると、実行は失敗します。

- **書き込みアクセス**：イシューおよびプルリクエストイベントで、トリガーするユーザーはリポジトリへの書き込みアクセスを持つ必要があります。書き込みアクセスのない特定のユーザーを許可するには、`allowed_non_write_users` を設定し、独自の `github_token` 入力を渡します。`schedule` トリガーなど、ユーザーが作成しないイベントはこのチェックをスキップします。
- **人間のアクター**：すべてのイベントで、Claude Code GitHub Action は `allowed_bots` にリストされていない限り、ボットアクターを拒否します。これにより、ボットが Claude をループでトリガーするのを防ぎます。このチェックはスケジュール実行にも適用されます。GitHub はこれらを、通常はワークフローの `cron` スケジュールを最後に変更したリポジトリユーザーに属性付けします。そのユーザーがボットの場合は、`allowed_bots` にリストします。

15.3 使用例

examples ディレクトリ には、さまざまなシナリオ用の使用可能なワークフローが含まれています。

このページの例は API キー認証を示しています。Claude サブスクリプションで認証する場合は、任意の例の `anthropic_api_key` 行を `claude_code_oauth_token: ${{ secrets.CLAUDE_CODE_OAUTH_TOKEN }}` に置き換えます。

15.3.1 @claude メンションに応答

このワークフローは Claude Code GitHub Action をインタラクティブモードで実行するため、イシューまたは PR コメントで誰かが `@claude` とメンションするたびに Claude が応答します。

```yaml
name: Claude Code
on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
jobs:
  claude:
    if: contains(github.event.comment.body, '@claude')
    runs-on: ubuntu-latest
    permissions:
```

(次のページに続く)

(前のページからの続き)

```
      contents: write
      pull-requests: write
      issues: write
      id-token: write
      actions: read
    steps:
      - uses: actions/checkout@v6
        with:
          fetch-depth: 1
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
```

このワークフローのボイラープレートではない部分：

- `id-token: write`：Claude Code GitHub Action のデフォルト GitHub App 認証に必要
- `actions: read`：Claude が PR の CI 結果を読み取ることができるようにします
- `actions/checkout`：Claude がリポジトリのローカルコピーで作業できるようにします
- `if`：`@claude` とメンションしないコメントでランナーが開始されるのを防ぎます。Claude Code GitHub Action はトリガーフレーズ自体をチェックしてから応答します

ワークフローが配置されたら、リクエストを含むイシューまたは PR コメントで `@claude` とメンションします。

```
@claude implement this feature based on the issue description
@claude how should I implement user authentication for this endpoint?
@claude fix the TypeError in the user dashboard component
```

Claude は同じイシューまたは PR のコメントで応答し、作業中に更新します。

15.3.2 スキルを実行

`prompt` 入力は、プレーンテキストだけでなく **スキル** (ページ 275) 呼び出しも受け入れます。

- リポジトリの `.claude/skills/` ディレクトリ内のスキルの場合、`anthropics/claude-code-action` ステップの前に `actions/checkout` を実行してスキルファイルをランナーで利用可能にし、`/skill-name` を `prompt` として渡します。
- plugin にパッケージされたスキルの場合、`plugin_marketplaces` と `plugins` 入力でプラグインをインストールし、名前空間付きの `/plugin-name:skill-name` を `prompt` として渡します。`plugins` 入力は `plugin-name@marketplace-name` を取ります。マーケットプレイス名はマーケットプレイスのリポジトリ URL ではなく、マーケットプレイス自体のマニフェストから取得されます。

次のワークフローは `code-review` プラグインをインストールし、プルリクエストが開かれた、更新された、レビュー準備完了、または再度開かれたときにそのスキルを実行します。クイックセットアップからのレビューワークフローと同じプラグインを実行します。プロンプト、モデル、トリガーを自分で制御したい場合は、このようなワークフローを使用します。ワークフローファイルを維持せずに自動レビューす

るには、Code Review を参照してください。パブリックリポジトリでは、GitHub はフォークプルリクエストでトリガーされた実行からシークレットを保留するため、レビューは同じリポジトリ内のブランチからのプルリクエストでのみ実行されます。

```yaml
name: Code Review
on:
  pull_request:
    types: [opened, synchronize, ready_for_review, reopened]
jobs:
  review:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      pull-requests: read
      issues: read
      id-token: write
    steps:
      - uses: actions/checkout@v6
        with:
          fetch-depth: 1
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
          plugin_marketplaces: "https://github.com/anthropics/claude-code.git"
          plugins: "code-review@claude-code-plugins"
          prompt: "/code-review:code-review --comment ${{ github.repository }}/pull/${
↪{ github.event.pull_request.number }}"
          claude_args: '--allowedTools "mcp__github_inline_comment__create_inline_
↪comment"'
```

このワークフローの 2 つの行がレビューの場所を制御します。

- `--comment`：Claude はプルリクエストにレビューを投稿します。見つかった各イシューのインラインコメントとして、または見つからない場合は 1 つの概要コメントとして投稿します。これがないと、Claude は何も投稿せず、ワークフロー実行ログで結果を読みます。
- `claude_args`：スキル自体の `allowed-tools` frontmatter が同じツールを名前付けしていても、この行を保持します。Claude Code GitHub Action はインラインコメントを投稿する MCP サーバーを開始するのは、`claude_args` の `--allowedTools` がそれを名前付けしている場合のみです。

Claude は下書きおよび閉じられたプルリクエスト、レビューが不要と判断するプルリクエスト（自動化されたものや些細なものなど）、および Claude からのコメントが既にあるプルリクエストをスキップします。

15.3.3 スケジュールで実行

prompt 入力を使用すると、Claude Code GitHub Action は cron スケジュールを含む任意の GitHub イベントで自動化モードで実行されます。プレーンテキストプロンプトの場合、Claude は claude_args の --allowedTools または settings 入力の *permissions.allow* **ルール** (ページ 1043) でプロンプトが必要とするツールを付与するまで、シェルまたは GitHub API アクセスを持ちません。代わりにスキルを呼び出す場合、Claude は *allowed-tools frontmatter* (ページ 298) が付与するツールを使用できます。GitHub はスケジュール済みワークフローをデフォルトブランチからのみ実行し、パブリックリポジトリではリポジトリアクティビティがない 60 日後にスケジュールを無効にします。

このワークフローは毎日 09:00 UTC にワークフロー実行ログでレポートを生成します。その claude_args 行は *CLI* **引数を渡す** (ページ 258) ことで、モデルを選択し、2 つの GitHub MCP ツールを許可します。Claude はこれらのツールで GitHub API を通じてコミットとイシューを読み取るため、チェックアウトステップを省略できます。

```yaml
name: Daily Report
on:
  schedule:
    - cron: "0 9 * * *"
jobs:
  report:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      issues: read
      id-token: write
    steps:
      - uses: anthropics/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
          prompt: "Generate a summary of yesterday's commits and open issues"
          claude_args: |
            --model claude-opus-5-5
            --allowedTools "mcp__github__list_commits,mcp__github__list_issues"
```

15.4 ベストプラクティス

15.4.1 CLAUDE.md でプロジェクト標準を定義

リポジトリルートに CLAUDE.md ファイルを作成して、コードスタイルガイドライン、レビュー基準、プロジェクト固有のルール、および推奨パターンを定義します。Claude は PR を作成し、リクエストに応答するときにこれらのガイドラインに従います。詳細については、**メモリドキュメント** (ページ 997) を参照してください。

15.4.2 認証情報を保護

警告:

API キーまたは OAuth トークンをリポジトリに直接コミットしないでください。常に GitHub Secrets として保存し、ワークフローで参照します。例えば `anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}`。

ワークフローに必要な権限のみを付与し、マージする前に Claude の変更を確認します。

権限と認証を含む包括的なセキュリティガイダンスについては、Claude Code Action セキュリティドキュメント を参照してください。

15.4.3 コストを管理

各実行は 2 種類のリソースを消費します。

- **GitHub Actions 分**：Claude Code GitHub Action は GitHub ホストランナーで実行され、GitHub Actions 分を消費します。価格と分の制限については、GitHub の請求ドキュメント を参照してください。
- **API トークン**：各インタラクションはプロンプトと応答の長さ、タスクの複雑さ、コードベースのサイズに基づいてトークンを消費します。現在のトークンレートについては、Claude の価格ページ を参照してください。OAuth トークンで認証する場合、実行は API 請求ではなく Claude サブスクリプションを使用します。

両方の種類のコストを低下させるには、Claude により明確なコンテキストを提供し、各実行が実行できる作業量を制限します。

- 特定の `@claude` リクエストを記述して、Claude が完了するのに必要なターン数を減らします
- イシューテンプレートを使用して事前にコンテキストを提供します
- `CLAUDE.md` を簡潔に保ちます。Claude は実行ごとにそれを読み取るため
- `claude_args` で `--max-turns` を設定して反復を制限します
- ワークフローレベルのタイムアウトを設定して暴走ジョブを回避します
- GitHub の並行制御を使用して並列実行を制限します

組織全体での使用追跡については、**分析ダッシュボード** (ページ 927) と **監視** (ページ 935) を参照してください。使用量がどのように測定され、請求されるかについては、**コスト** (ページ 911) を参照してください。

15.5 クラウドプロバイダーを使用

デフォルトでは、Claude Code GitHub Action は API キーまたは OAuth トークンで Claude API を直接呼び出します。代わりに独自のクラウドアカウント経由で推論をルーティングするには、プロバイダーの入力を設定し、クラウドプロバイダーで Claude Code GitHub Actions を使用する に従います。

- **Amazon Bedrock**：`use_bedrock: "true"`
- **Google Cloud の Agent Platform**：`use_vertex: "true"`

- **Microsoft Foundry**：`use_foundry: "true"`

3 つのプロバイダーすべてで、Claude API キーではなく OIDC ID フェデレーション経由で認証するため、リポジトリに静的クラウド認証情報を保存しません。

15.6 トラブルシューティング

15.6.1 Claude が @claude コマンドに応答しない

- GitHub App がリポジトリにインストールされていることを確認します
- リポジトリでワークフローが有効になっていることを確認します
- API キーまたは OAuth トークンがリポジトリシークレットに設定されていることを確認します
- コメントに `@claude` が完全な単語として含まれていることを確認します。`/claude` または `@claude-bot` ではなく
- コメントするユーザーがリポジトリへの書き込みアクセスを持つことを確認します。**実行をトリガーできるユーザー** (ページ 252) の例外を参照してください

15.6.2 CI が Claude のコミットで実行されない

- GitHub はデフォルトの `GITHUB_TOKEN` で作成されたコミットでワークフローをトリガーしません。Claude Code GitHub Action に `github_token: ${{ secrets.GITHUB_TOKEN }}` を渡す場合は、Claude GitHub App として認証するようにそれを削除するか、代わりにカスタムアプリトークンを渡します
- CI ワークフローのトリガーに Claude のプッシュが生成するイベント（`push` または `pull_request` など）が含まれていることを確認します

15.6.3 認証エラー

- API キーまたは OAuth トークンが有効であることを確認します。ワークフローをデバッグする前に `claude` でローカルでテストします
- Bedrock、Agent Platform、Foundry の場合は、クラウドプロバイダーページの トラブルシューティングセクション を参照してください

その他のソリューションについては、Claude Code GitHub Action の FAQ を参照してください。

15.7 高度な設定

15.7.1 アクション パラメータ

これらは最も一般的に使用される入力です。各パラメータは `anthropics/claude-code-action` ステップの `with:` キーにマップされます。

パラメータ	説明	必須
`prompt`	Claude への指示。プレーン テキストまたは**スキル** (ページ 275) 呼び出しとして指定します。省略した場合、Claude は**トリガー フレーズ** (ページ 251) に応答します	いいえ
`claude_args`	Claude Code に渡される CLI 引数	いいえ
`anthropic_api_key`	Claude API キー	Claude API の場合は必須です。ただし、`claude_code_oauth_token` または**ワークロード *ID* フェデレーション** (ページ 250) を使用する場合は不要です。Bedrock、Agent Platform、または Foundry では使用されません
`claude_code_oauth_token`	Claude サブスクリプションで認証するための OAuth トークン。`claude setup-token` で生成されます	いいえ
`github_token`	GitHub 操作用のトークン。省略した場合、Claude Code GitHub Action は Claude GitHub App として認証されます	いいえ
`plugin_marketplaces`	プラグイン マーケットプレイス Git URL の改行区切りリスト	いいえ
`plugins`	実行前にインストールするプラグイン名の改行区切りリスト	いいえ
`settings`	Claude Code 設定。JSON 文字列またはセッティング JSON ファイルへのパス	いいえ
`trigger_phrase`	Claude が応答するトリガー フレーズ。デフォルト: `@claude`	いいえ
`use_bedrock`	Claude API の代わりに Amazon Bedrock を使用します	いいえ
`use_vertex`	Claude API の代わりに Google Cloud の Agent Platform を使用します	いいえ
`use_foundry`	Claude API の代わりに Microsoft Foundry を使用します	いいえ

完全な入力リストについては、Claude Code GitHub Action の設定リファレンスを参照してください。

15.7.2 CLI 引数を渡す

`claude_args` パラメータは、任意の *Claude Code CLI* **引数** (ページ 45) を受け入れます。

```
claude_args: "--max-turns 5 --model claude-sonnet-5 --mcp-config /path/to/config.json"
```

一般的な引数:

- `--max-turns`: 会話ターン数を制限します
- `--model`: 使用するモデル。例えば `claude-sonnet-5`。この引数がない場合、Claude Code GitHub Action は Claude Code の**デフォルト モデル** (ページ 725) を使用します
- `--mcp-config`: *MCP* **設定** (ページ 363) へのパス
- `--allowedTools`: 許可されたツールのカンマ区切りリスト。`--allowed-tools` エイリアスも機能します
- `--debug`: デバッグ出力を有効にします

15.8 ベータ版からアップグレード

ワークフローがまだ `anthropics/claude-code-action@beta` を参照している場合は、v1 に更新します。

1. `uses` 行の `@beta` を `@v1` に変更します
2. `mode` 入力を削除します。Claude Code GitHub Action は **モードを自動的に検出** (ページ 251) するようになったため
3. `direct_prompt` を `prompt` に置き換えます
4. `max_turns` や `model` などの CLI オプションを `claude_args` に移動します。`custom_instructions` は同じ名前のフラグを持たず、`--append-system-prompt` になります

完全な入力マッピングと前後の例については、マイグレーションガイド を参照してください。

15.9 次のステップ

- クラウドプロバイダーで Claude Code GitHub Actions を使用する：Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry 経由で推論をルーティング
- 設定リファレンス：アクション入力の完全なリスト
- Examples ディレクトリ：より多くのシナリオ用の使用可能なワークフロー
- Code Review：ワークフローファイルを維持せずに自動プルリクエストレビュー

第16章　Claude Code GitLab CI/CD

Claude Code を GitLab CI/CD で開発ワークフローに統合する方法を学びます

注釈:

Claude Code for GitLab CI/CD は現在ベータ版です。機能と機能性は、エクスペリエンスを改善する際に進化する可能性があります。
この統合は GitLab によって保守されています。サポートについては、以下の GitLab issue を参照してください。

注釈:

この統合は Claude Code CLI and Agent SDK の上に構築されており、CI/CD ジョブとカスタム自動化ワークフローで Claude をプログラム的に使用できます。

16.1 GitLab で Claude Code を使用する理由

- **インスタント MR 作成**: 必要なことを説明すると、Claude は変更と説明を含む完全な MR を提案します
- **自動実装**: 単一のコマンドまたはメンションで issue を実行可能なコードに変換します
- **プロジェクト対応**: Claude は `CLAUDE.md` ガイドラインと既存のコードパターンに従います
- **シンプルなセットアップ**: `.gitlab-ci.yml` に 1 つのジョブとマスクされた CI/CD 変数を追加します
- **エンタープライズ対応**: Claude API、Amazon Bedrock、または Google Cloud の Agent Platform を選択して、データレジデンシーと調達のニーズを満たします
- **デフォルトでセキュア**: GitLab ランナーで実行され、ブランチ保護と承認が適用されます

16.2 仕組み

Claude Code は GitLab CI/CD を使用して AI タスクを分離されたジョブで実行し、MR 経由で結果をコミットバックします。

1. **イベント駆動型オーケストレーション**: GitLab は選択したトリガー（例えば、issue、MR、またはレビュースレッドで `@claude` をメンションするコメント）をリッスンします。ジョブはスレッドとリポジトリからコンテキストを収集し、その入力からプロンプトを構築し、Claude Code を実行します。
2. **プロバイダー抽象化**: 環境に適したプロバイダーを使用します。
   - Claude API（SaaS）

- Amazon Bedrock（IAM ベースのアクセス、クロスリージョンオプション）
- Google Cloud の Agent Platform（GCP ネイティブ、Workload Identity Federation）

3. **サンドボックス実行**: 各インタラクションは厳密なネットワークとファイルシステムルールを持つコンテナで実行されます。Claude Code はワークスペーススコープの権限を適用して書き込みを制限します。すべての変更は MR を通じてフローするため、レビュアーは diff を確認でき、承認が引き続き適用されます。

地域エンドポイントを選択して、既存のクラウド契約を使用しながらレイテンシーを削減し、データソブリンティ要件を満たします。

16.3 Claude にはどのようなことができますか？

GitLab パイプラインでは、Claude Code は以下のことができます：

- issue の説明またはコメントから MR を作成および更新する
- パフォーマンス低下を分析し、最適化を提案する
- ブランチに機能を直接実装してから MR を開く
- テストまたはコメントで特定されたバグおよび低下を修正する
- フォローアップコメントに応答して、リクエストされた変更を反復する

16.4 セットアップ

16.4.1 クイックセットアップ

最速で始める方法は、`.gitlab-ci.yml` に最小限のジョブを追加し、API キーをマスク変数として設定することです。

1. **マスク CI/CD 変数を追加する**
   - **Settings** → **CI/CD** → **Variables** に移動します
   - `ANTHROPIC_API_KEY` を追加します（マスク、必要に応じて保護）
2. **`.gitlab-ci.yml` に Claude ジョブを追加する**

```yaml
stages:
  - ai

claude:
  stage: ai
  image: node:24-alpine3.21
  # ジョブをトリガーする方法に合わせてルールを調整します：
  # - 手動実行
  # - マージリクエストイベント
  # - '@claude' を含むコメント時の web/API トリガー
  rules:
```

(次のページに続く)

(前のページからの続き)

```
    - if: '$CI_PIPELINE_SOURCE == "web"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
  variables:
    GIT_STRATEGY: fetch
  before_script:
    - apk update
    - apk add --no-cache git curl bash
    - curl -fsSL https://claude.ai/install.sh | bash
    # インストーラーは claude を ~/.local/bin に配置しますが、このイメージの PATH には含まれていません
    - export PATH="$HOME/.local/bin:$PATH"
  script:
    # オプション：セットアップが提供する場合は GitLab MCP サーバーを起動します
    - /bin/gitlab-mcp-server || true
    # コンテキストペイロード付きの web/API トリガー経由で呼び出す場合は AI_FLOW_* 変数を使用します
    - echo "$AI_FLOW_INPUT for $AI_FLOW_CONTEXT on $AI_FLOW_EVENT"
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Review this MR and implement the requested changes'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
```

ジョブと ANTHROPIC_API_KEY 変数を追加した後、**CI/CD** → **Pipelines** からジョブを手動で実行してテストするか、MR からトリガーして Claude が提案した更新をブランチで実装し、必要に応じて MR を開くことができます。

注釈:

Claude API の代わりに Amazon Bedrock または Google Cloud の Agent Platform で実行する場合は、認証と環境セットアップについて下記の *Amazon Bedrock と Google Cloud* **を使用する** (ページ 264) セクションを参照してください。

16.4.2 手動セットアップ（本番環境に推奨）

より制御されたセットアップが必要な場合またはエンタープライズプロバイダーが必要な場合：

1. **プロバイダーアクセスを設定する**：
   - **Claude API**：ANTHROPIC_API_KEY を作成してマスク CI/CD 変数として保存します
   - **Amazon Bedrock**：**Configure GitLab** → **AWS OIDC** を設定し、Amazon Bedrock 用の IAM ロールを作成します
   - **Google Cloud の Agent Platform**：**Configure Workload Identity Federation for GitLab** → **GCP** を設定します

2. **GitLab API 操作用のプロジェクト認証情報を追加する：**
   - デフォルトで `CI_JOB_TOKEN` を使用するか、`api` スコープを持つ Project Access Token を作成します
   - PAT を使用する場合は `GITLAB_ACCESS_TOKEN`（マスク）として保存します
3. **`.gitlab-ci.yml` に Claude ジョブを追加する：**Claude API の場合は **クイックセットアップ** (ページ 262) ジョブを使用するか、**設定例** (ページ 266) からプロバイダージョブを使用します
4. **（オプション）メンション駆動トリガーを有効にする：**
   - 「Comments (notes)」のプロジェクト webhook をイベントリスナー（使用している場合）に追加します
   - コメントに `@claude` が含まれている場合、リスナーが `AI_FLOW_INPUT` や `AI_FLOW_CONTEXT` などの変数を使用してパイプライントリガー API を呼び出すようにします

16.5 使用例

16.5.1 Issue を MR に変換する

Issue コメント内：

```
@claude implement this feature based on the issue description
```

Claude は Issue とコードベースを分析し、ブランチで変更を記述して、レビュー用の MR を開きます。

16.5.2 実装サポートを取得する

MR ディスカッション内：

```
@claude suggest a concrete approach to cache the results of this API call
```

Claude は変更を提案し、適切なキャッシング機能を備えたコードを追加し、MR を更新します。

16.5.3 バグを素早く修正する

Issue または MR コメント内：

```
@claude fix the TypeError in the user dashboard component
```

Claude はバグを特定し、修正を実装し、ブランチを更新するか新しい MR を開きます。

16.6 Amazon Bedrock と Google Cloud での使用

エンタープライズ環境では、Claude Code をクラウドインフラストラクチャ全体で実行でき、同じ開発者体験を得られます。

Amazon Bedrock

前提条件

Amazon Bedrock で Claude Code をセットアップする前に、以下が必要です。

1. 目的の Claude モデルへの Amazon Bedrock アクセス権を持つ AWS アカウント
2. AWS IAM で OIDC アイデンティティプロバイダーとして設定された GitLab
3. Amazon Bedrock 権限を持つ IAM ロールと、GitLab プロジェクト/refs に制限されたトラストポリシー
4. ロール引き受けのための GitLab CI/CD 変数：
   - `AWS_ROLE_TO_ASSUME`（ロール ARN）
   - `AWS_REGION`（Amazon Bedrock リージョン）

セットアップ手順

GitLab CI ジョブが OIDC 経由で IAM ロールを引き受けることを許可するよう AWS を設定します（静的キーは不要）。

必須セットアップ：

1. Amazon Bedrock を有効にし、目的の Claude モデルへのアクセスをリクエストします
2. まだ存在しない場合は、GitLab 用の IAM OIDC プロバイダーを作成します
3. GitLab OIDC プロバイダーによって信頼され、プロジェクトと保護された refs に制限された IAM ロールを作成します
4. Amazon Bedrock invoke API に対する最小権限パーミッションをアタッチします

Amazon Bedrock **ジョブの例** (ページ 266) を使用して、実行時にジョブの OIDC トークンを一時的な AWS 認証情報と交換します。

Google Cloud's Agent Platform

前提条件

Google Cloud's Agent Platform で Claude Code をセットアップする前に、以下が必要です。

1. 以下を備えた Google Cloud プロジェクト：
   - Google Cloud's Agent Platform API が有効化されている
   - GitLab OIDC を信頼するよう設定された Workload Identity Federation
2. 必要な Google Cloud's Agent Platform ロールのみを持つ専用サービスアカウント
3. GitLab CI/CD 変数：
   - `GCP_WORKLOAD_IDENTITY_PROVIDER`（`//iam.googleapis.com/` プレフィックスなしのプロバイダーリソース名。例：`projects/123456789/locations/global/workloadIdentityPools/my-pool/providers/my-provider`）
   - `GCP_SERVICE_ACCOUNT`（サービスアカウントメール）

- `GCP_PROJECT_ID`（Google Cloud プロジェクト ID）

セットアップ手順

Workload Identity Federation 経由で GitLab CI ジョブがサービスアカウントを偽装することを許可するよう Google Cloud を設定します。

必須セットアップ：

1. IAM Credentials API、STS API、および Google Cloud's Agent Platform API を有効にします
2. GitLab OIDC 用の Workload Identity Pool とプロバイダーを作成します
3. Google Cloud's Agent Platform ロールを持つ専用サービスアカウントを作成します
4. WIF プリンシパルにサービスアカウントを偽装するパーミッションを付与します

Agent Platform **ジョブの例** (ページ 266) を使用して、キーを保存することなく認証します。

16.7 設定例

以下は、パイプラインに適応させることができる、すぐに使用できるスニペットです。

16.7.1 Amazon Bedrock ジョブの例（OIDC）

前提条件：

- Amazon Bedrock が有効化され、選択した Claude モデルへのアクセスが可能
- GitLab OIDC が AWS で設定され、GitLab プロジェクトと refs を信頼するロールが存在
- Amazon Bedrock 権限を持つ IAM ロール（最小権限を推奨）

必須 CI/CD 変数：

- `AWS_ROLE_TO_ASSUME`：Amazon Bedrock アクセス用の IAM ロールの ARN
- `AWS_REGION`：Amazon Bedrock リージョン（例：`us-west-2`）

GitLab は `id_tokens:` ブロックからジョブの OIDC トークンを生成し、`GITLAB_OIDC_TOKEN` として公開します。aud を AWS の IAM OIDC アイデンティティプロバイダーで設定したオーディエンス値（例：GitLab インスタンス URL）に設定します。

```yaml
stages:
  - ai

claude-bedrock:
  stage: ai
  image: node:24-alpine3.21
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  id_tokens:
```

(次のページに続く)

(前のページからの続き)

```
    GITLAB_OIDC_TOKEN:
      aud: https://gitlab.example.com
  before_script:
    - apk add --no-cache bash curl jq git aws-cli
    - curl -fsSL https://claude.ai/install.sh | bash
    # The installer places claude in ~/.local/bin, which isn't on PATH in this image
    - export PATH="$HOME/.local/bin:$PATH"
    # Exchange the job's OIDC token for AWS credentials
    - export AWS_WEB_IDENTITY_TOKEN_FILE="/tmp/oidc_token"
    - printf "%s" "$GITLAB_OIDC_TOKEN" > "$AWS_WEB_IDENTITY_TOKEN_FILE"
    - >
      aws sts assume-role-with-web-identity
      --role-arn "$AWS_ROLE_TO_ASSUME"
      --role-session-name "gitlab-claude-$(date +%s)"
      --web-identity-token "file://$AWS_WEB_IDENTITY_TOKEN_FILE"
      --duration-seconds 3600 > /tmp/aws_creds.json
    - export AWS_ACCESS_KEY_ID="$(jq -r .Credentials.AccessKeyId /tmp/aws_creds.json)"
    - export AWS_SECRET_ACCESS_KEY="$(jq -r .Credentials.SecretAccessKey /tmp/aws_
↪creds.json)"
    - export AWS_SESSION_TOKEN="$(jq -r .Credentials.SessionToken /tmp/aws_creds.json)
↪"
  script:
    - /bin/gitlab-mcp-server || true
    - >
      claude
      -p "${AI_FLOW_INPUT:-'Implement the requested changes and open an MR'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    AWS_REGION: "us-west-2"
    CLAUDE_CODE_USE_BEDROCK: "1"
```

注釈:

Amazon Bedrock のモデル ID にはリージョン固有のプレフィックスが含まれます（例：`us.anthropic.claude-sonnet-4-6`）。ワークフローがサポートしている場合は、ジョブ設定またはプロンプトを通じて目的のモデルを渡します。

16.7.2 Agent Platform ジョブの例（Workload Identity Federation）

前提条件：

- Google Cloud の Agent Platform API が GCP プロジェクトで有効化
- Workload Identity Federation が GitLab OIDC を信頼するように設定

- Google Cloud の Agent Platform 権限を持つサービスアカウント

必須 CI/CD 変数：

- `GCP_WORKLOAD_IDENTITY_PROVIDER`：`//iam.googleapis.com/` プレフィックスなしのプロバイダーリソース名（例：`projects/123456789/locations/global/workloadIdentityPools/my-pool/providers/my-provider`）
- `GCP_SERVICE_ACCOUNT`：サービスアカウントメール
- `GCP_PROJECT_ID`：Google Cloud プロジェクト ID
- `CLOUD_ML_REGION`：Google Cloud の Agent Platform リージョン（例：`us-east5`）

GitLab は `id_tokens:` ブロックからジョブの OIDC トークンを生成し、`GITLAB_OIDC_TOKEN` として公開します。aud を Workload Identity Pool プロバイダーで設定したオーディエンス値（例：GitLab インスタンス URL）に設定します。ジョブはトークンをファイルに書き込み、認証情報設定の `credential_source` エントリは Google の認証ライブラリにそこから読み込むよう指示します。`GOOGLE_APPLICATION_CREDENTIALS` を認証情報設定ファイルに設定すると、*Application Default Credentials* (ページ 883) を通じて Claude Code で利用可能になります。

```yaml
stages:
  - ai

claude-vertex:
  stage: ai
  image: gcr.io/google.com/cloudsdktool/google-cloud-cli:slim
  rules:
    - if: '$CI_PIPELINE_SOURCE == "web"'
  id_tokens:
    GITLAB_OIDC_TOKEN:
      aud: https://gitlab.example.com
  before_script:
    - apt-get update && apt-get install -y git && apt-get clean
    - curl -fsSL https://claude.ai/install.sh | bash
    # The installer places claude in ~/.local/bin, which isn't on PATH in this image
    - export PATH="$HOME/.local/bin:$PATH"
    # Write the job's OIDC token where credential_source expects it
    - printf "%s" "$GITLAB_OIDC_TOKEN" > /tmp/oidc_token
    # Write the WIF credential configuration to a file (no downloaded keys)
    - |
      cat > /tmp/cred.json <<EOF
      {
        "type": "external_account",
        "audience": "//iam.googleapis.com/${GCP_WORKLOAD_IDENTITY_PROVIDER}",
        "subject_token_type": "urn:ietf:params:oauth:token-type:jwt",
        "token_url": "https://sts.googleapis.com/v1/token",
        "credential_source": {
          "file": "/tmp/oidc_token"
```

(次のページに続く)

(前のページからの続き)

```
        },
        "service_account_impersonation_url": "https://iamcredentials.googleapis.com/
↪v1/projects/-/serviceAccounts/${GCP_SERVICE_ACCOUNT}:generateAccessToken"
      }
      EOF
    # Expose the credentials to Claude Code via Application Default Credentials
    - export GOOGLE_APPLICATION_CREDENTIALS=/tmp/cred.json
    # Authenticate the gcloud CLI with the same credential configuration
    - gcloud auth login --cred-file=/tmp/cred.json
    - gcloud config set project "$GCP_PROJECT_ID"
  script:
    - /bin/gitlab-mcp-server || true
    - >
      CLOUD_ML_REGION="${CLOUD_ML_REGION:-us-east5}"
      claude
      -p "${AI_FLOW_INPUT:-'Review and update code as requested'}"
      --permission-mode acceptEdits
      --allowedTools "Bash Read Edit Write mcp__gitlab"
      --debug
  variables:
    CLOUD_ML_REGION: "us-east5"
    CLAUDE_CODE_USE_VERTEX: "1"
    ANTHROPIC_VERTEX_PROJECT_ID: "$GCP_PROJECT_ID"
```

注釈:

Workload Identity Federation では、サービスアカウントキーを保存する必要がありません。リポジトリ固有の信頼条件と最小権限のサービスアカウントを使用します。

16.8 ベストプラクティス

16.8.1 CLAUDE.md 設定

リポジトリのルートに `CLAUDE.md` ファイルを作成して、コーディング標準、レビュー基準、プロジェクト固有のルールを定義します。Claude はランの実行中にこのファイルを読み込み、変更を提案する際にあなたの規約に従います。

16.8.2 セキュリティに関する考慮事項

API キーやクラウド認証情報をリポジトリにコミットしないでください。常に GitLab CI/CD 変数を使用してください：

- `ANTHROPIC_API_KEY` をマスク変数として追加します（必要に応じて保護してください）
- 可能な限りプロバイダー固有の OIDC を使用します（長期的なキーは不要）

- ジョブの権限とネットワーク出力を制限します
- Claude の MR を他のコントリビューターと同じようにレビューします

16.8.3 パフォーマンスの最適化

- `CLAUDE.md` を焦点を絞った簡潔なものに保ちます
- 明確なイシュー/MR の説明を提供して、反復を減らします
- ランナーで npm とパッケージのインストールをキャッシュします（可能な場合）

16.8.4 CI コスト

Claude Code を GitLab CI/CD で使用する場合、関連するコストに注意してください：

- **GitLab ランナー時間**：
  - Claude はあなたの GitLab ランナーで実行され、コンピュート分を消費します
  - 詳細については、GitLab プランのランナー課金を参照してください
- **API コスト**：
  - Claude との各インタラクションは、プロンプトと応答のサイズに基づいてトークンを消費します
  - トークン使用量はタスクの複雑さとコードベースのサイズによって異なります
  - 詳細については、Anthropic 価格を参照してください
- **コスト最適化のヒント**：
  - 特定の `@claude` コマンドを使用して、不要なターンを減らします
  - 適切な `--max-turns` とジョブ `timeout` 値を設定します
  - 並列実行を制御するために同時実行を制限します

16.9 トラブルシューティング

16.9.1 Claude が @claude コマンドに応答しない

- パイプラインがトリガーされていることを確認してください（手動、MR イベント、またはノート イベント リスナー/webhook 経由）
- `ANTHROPIC_API_KEY` またはクラウドプロバイダー変数が存在することを確認してください
- コメントに `@claude` が含まれていること（`/claude` ではなく）、メンション トリガーが設定されていることを確認してください

16.9.2 ジョブがコメントを書き込めない、または MR を開けない

- `CI_JOB_TOKEN` がプロジェクトに対して十分な権限を持っていることを確認するか、`api` スコープを持つプロジェクト アクセス トークンを使用してください
- `mcp__gitlab` ツールが `--allowedTools` で有効になっていることを確認してください
- ジョブが MR のコンテキストで実行されているか、`AI_FLOW_*` 変数を経由して十分なコンテキストを持っていることを確認してください

16.9.3 認証エラー

- **Claude API の場合**: `ANTHROPIC_API_KEY` が有効で期限切れでないことを確認してください
- **Amazon Bedrock または Google Cloud の Agent Platform の場合**: OIDC/WIF 設定、ロール偽装、シークレット名を確認してください。リージョンとモデルの可用性を確認してください

16.10 高度な設定

16.10.1 一般的なパラメータと変数

これらの CLI フラグ、GitLab キーワード、および変数を使用して、ジョブ内の Claude Code 実行を制御します。

- `-p`: インラインで指示を提供します。例えば `claude -p "Review this MR"`
- `--max-turns`: バックアンドフォース反復の回数を制限します
- `timeout`: GitLab のジョブレベルの `timeout` キーワードで総ジョブ実行時間を制限します。例えば `timeout: 30m`
- `ANTHROPIC_API_KEY`: Claude API に必要です（Amazon Bedrock または Google Cloud の Agent Platform では使用されません）
- プロバイダー固有の環境: `AWS_REGION`、Google Cloud の Agent Platform のプロジェクト/リージョン変数

注釈:

正確なフラグとパラメータは `@anthropic-ai/claude-code` のバージョンによって異なる場合があります。ジョブ内で `claude --help` を実行して、サポートされているオプションを確認してください。

16.10.2 Claude の動作をカスタマイズする

Claude をガイドする方法は 2 つあります。

1. **CLAUDE.md**: コーディング標準、セキュリティ要件、およびプロジェクト規約を定義します。Claude は実行中にこれを読み、ルールに従います。
2. **カスタムプロンプト**: ジョブ内の `-p` を使用してタスク固有の指示を渡します。異なるジョブに異なるプロンプトを使用します（例えば、レビュー、実装、リファクタリング）。

第IV部

拡張機能

第17章　Claudeをスキルで拡張する

> Claude Code でスキルを作成、管理、共有して Claude の機能を拡張します。カスタムコマンドとバンドルされたスキルが含まれます。

スキルは Claude ができることを拡張します。`SKILL.md` ファイルに指示を記述すると、Claude はそれをツールキットに追加します。Claude は関連する場合にスキルを使用するか、`/skill-name` で直接呼び出すことができます。

同じ指示、チェックリスト、または複数ステップの手順をチャットに何度も貼り付けている場合、または CLAUDE.md のセクションが事実ではなく手順に成長している場合は、スキルを作成してください。CLAUDE.md のコンテンツとは異なり、スキルの本体は使用される場合にのみ読み込まれるため、長い参考資料は必要になるまでほぼコストがかかりません。

注釈:

`/help` や `/compact` などの組み込みコマンド、および `/debug` や `/code-review` などのバンドルされたスキルについては、コマンドリファレンスを参照してください。

カスタムコマンドはスキルにマージされました。 `.claude/commands/deploy.md` のファイルと `.claude/skills/deploy/SKILL.md` のスキルの両方が `/deploy` を作成し、同じように機能します。既存の `.claude/commands/` ファイルは引き続き機能します。スキルはオプション機能を追加します。サポートファイル用のディレクトリ、**スキルを呼び出すユーザーを制御する** (ページ 296) のためのフロントマター、および Claude が関連する場合に自動的にスキルを読み込む機能です。

Claude Code スキルは Agent Skills オープンスタンダードに従い、複数の AI ツール全体で機能します。Claude Code は **呼び出し制御** (ページ 296)、**サブエージェント実行** (ページ 303)、**動的コンテキスト注入** (ページ 300) などの追加機能でスタンダードを拡張します。*Claude Code* **外でスキルフロントマターを使用する** (ページ 292) を参照して、どのフロントマターフィールドがスタンダードの一部であり、どれが Claude Code 拡張機能であるかを確認してください。

17.1 バンドルされたスキル

Claude Code には、`/doctor`、`/code-review`、`/batch`、`/debug`、`/loop`、`/claude-api` などのバンドルされたスキルが含まれています。バンドルされたスキルはプロンプトベースです。Claude に詳細な指示を与え、ツールを使用して作業を調整させます。ほとんどの組み込みコマンドは、代わりに固定ロジックを直接実行します。

バンドルされたスキルは、他のスキルと同じ方法で呼び出します。`/` の後にスキル名を入力します。Claude は関連する場合、一部のバンドルされたスキルを自動的に呼び出します。`/verify` を含む他のスキルは、呼び出した場合にのみ実行されます。これにより、これらの長時間実行されるチェックが時間とトークンを費やすタイミングを制御できます。

ほとんどのバンドルされたスキルはすべてのセッションで利用可能です。いくつかは特定の機能に依存しています。たとえば、/workflow-authoring は 動的ワークフロー が有効な場合にのみ利用可能です。

バンドルされたスキルをオフにするには、disableBundledSkills 設定を使用します。

注釈:

/doctor セットアップチェックアップは、Claude Code v2.1.205 以降で disableBundledSkills がオンの場合でも入力可能なままです。これを非表示にするには、DISABLE_DOCTOR_COMMAND 環境変数を設定するか、*skillOverrides* (ページ 306) エントリ "doctor": "off" を設定します。v2.1.205 より前では、/doctor はバンドルされたスキルではなく組み込みコマンドでした。

バンドルされたスキルは、コマンドリファレンス に組み込みコマンドと一緒にリストされており、目的列に **Skill** とマークされています。

17.1.1 アプリを実行して検証する

3 つのバンドルされたスキルが連携して、アプリを起動し、テストだけでなく実行中のアプリに対して変更を確認します。

スキル	目的
/run	アプリを起動して駆動し、変更が機能していることを確認する
/verify	アプリをビルドして実行し、コード変更が意図したことを実行していることを確認する。テストまたは型チェックにフォールバックしない
/run-skill-generator	/run と /verify にプロジェクトをビルドして起動する方法を教える

/run と /verify はセットアップなしで動作します。プロジェクトタイプ（CLI、サーバー、TUI、ブラウザ駆動）と README、package.json、または Makefile の内容から起動を推測します。その推測は、標準的な起動を超えて何かが必要なプロジェクト（データベース、env ファイル、グラフィカルセッション、マルチステップビルド）では信頼性が低くなります。

/run-skill-generator は代わりにレシピを記録します。クリーン環境からアプリを実行し、機能したもの（インストールコマンド、環境変数、起動スクリプト）をキャプチャし、.claude/skills/run-<name>/ でプロジェクトごとのスキルとしてコミットします。その後、/run、/verify、およびリポジトリ内の他のエージェントは、記録されたレシピに従い、再度発見することはありません。プロジェクトごとに 1 回 /run-skill-generator を実行し、ビルドまたは起動プロセスが変更された場合は再度実行します。

/verify は独自のレシピを記録することもできます。記録されたレシピなしでアプリをビルドして駆動する必要がある場合、機能したもの（.claude/skills/verify/SKILL.md をリポジトリルートに、またはモノレポの場合は変更されたパッケージディレクトリに）を書き込むため、後の実行と他のエージェントは同じステップに従います。リポジトリルートでは、記録されたスキルはバンドルされた /verify に置き換わります。これには Claude Code v2.1.200 以降が必要です。

Claude は、失敗したコマンドや欠落したステップなど、実行を誤った場合にのみ記録されたファイルを編集するため、セッションごとの差分なしでファイルをコミットできます。v2.1.205 より前では、バンドルされたスキルは Claude に実行から学んだことをすべて折り込むよう指示し、頻繁なマージコンフリクトを引き起こしていました。

17.1.2 各コミットの前にチェックを実行する

`verify` または `simplify` という名前のスキルが存在する状態でセッションが開始されると、Claude Code のコミット指示により、Claude はドキュメントやテストへの変更を除き、各コミットの直前にそのスキルを実行します。これには Claude Code v2.1.286 以降が必要です。セッション開始時に次の条件を満たしている場合、Claude はその指示を受け取ります。

- **場所**: スキルがエンタープライズ、個人、プロジェクト、または追加ディレクトリの**場所** (ページ 279) から読み込まれるか、その名前の `.claude/commands/` ファイルから読み込まれること。`/verify` がリポジトリルートに記録するレシピはプロジェクトスキルであるため、対象になります。バンドルされた `/verify` と `/simplify`、プラグインスキル、claude.ai アカウントのスキルは対象外です。
- **呼び出し**: Claude がスキルを呼び出せること。たとえば `disable-model-invocation: true` などで *Claude* **による呼び出しを停止した** (ページ 296) 場合、Claude はこの指示を受け取りません。
- **Git の指示**: `includeGitInstructions` をオフにしていないこと。オフにすると、組み込みのコミットおよび PR に関するその他の指示とともに、この指示も削除されます。

17.1.3 Claude API プロジェクトで作業する

バンドルされた `/claude-api` スキルは、プロジェクトの言語に対して Claude API および Managed Agents リファレンス資料を読み込みます。コードが `anthropic` または `@anthropic-ai/sdk` をインポートする場合、Claude はこれを自動的にアクティベートします。

スキルのワークフローの 1 つを開始するには、Claude Code プロンプトでスキル名の後にサブコマンドを入力します。たとえば `/claude-api migrate` です。表は各サブコマンドが何をするか、およびそれを含む最も古い Claude Code バージョンをリストしています。`migrate` と `managed-agents-onboard` は v2.1.221 より前のバージョンです。これは表が追跡する最も古いバージョンです。

サブコマンド	実行内容	最小バージョン
`migrate`	既存の Claude API コードを新しいモデルに更新する	v2.1.221 より前
`upgrade`	プロジェクトの Anthropic SDK 依存関係をメジャーバージョン間で移動します。現在は Python `anthropic` パッケージを 0.x から 1.x に移動します	v2.1.236 以降
`managed-agents-onboard`	新しい Managed Agent の作成をウォークスルーする	v2.1.221 より前
`prompt-audit`	プロンプト、スキル、ツール説明に書かれた古いモデル向けの指示にフラグを立て、差分として修正を提案する	v2.1.221 以降
`cost-optimize`	プロジェクトの Claude API 支出がどこに行くかをプロファイルし、プロンプトキャッシング、不要な入出力トークンの削減、バッチ処理、努力、モデル選択などのオプションから節約を提案します。一度に 1 つの変更	v2.1.247 以降

次のページに続く

表 2 – 前のページからの続き

サブコマンド	実行内容	最小バージョン
build-eval	Claude を搭載したアプリ用の eval セットをビルドする	v2.1.259 以降
hillclimb	既存の eval に対してアプリを反復的に改善する	v2.1.259 以降
preserved-thinking-migration	統合が以前のターン、システムプロンプト、またはツールリストに対して行う編集を見つけます。これは preserved thinking ブロックを無効にし、各ブロックがどの程度の推論を削除するかを測定し、一度に 1 つずつ修正を提案します。各変更後に再測定します	v2.1.282 以降

17.2 はじめに

17.2.1 最初のスキルを作成する

この例では、Git リポジトリ内のコミットされていない変更を要約し、危険な点にフラグを付けるスキルを作成します。ライブ diff をプロンプトに取り込んでから Claude が読むため、Claude が開いているファイルから推測できるものではなく、実際の作業ツリーに基づいた応答が得られます。Claude は、変更について質問するときに自動的にスキルを読み込むか、/summarize-changes で直接呼び出すことができます。

1. **スキルディレクトリを作成する**

   個人用スキルフォルダにスキル用のディレクトリを作成します。個人用スキルはすべてのプロジェクト全体で利用できます。

```
mkdir -p ~/.claude/skills/summarize-changes
```

2. **SKILL.md を作成する**

   すべてのスキルには SKILL.md ファイルが必要です。このファイルには 2 つの部分があります。Claude がスキルをいつ使用するかを指定する --- マーカー間の YAML frontmatter と、スキルが実行されるときに Claude が従う指示を含む markdown コンテンツです。ディレクトリ名、またはフロントマターの name を設定した場合はそれが、入力するコマンドになり、description は Claude がスキルを自動的に読み込むかどうかを決定するのに役立ちます。

   これを ~/.claude/skills/summarize-changes/SKILL.md に保存します。

```
---
description: Summarizes uncommitted changes and flags anything risky. Use when␣
↪the user asks what changed, wants a commit message, or asks to review their␣
↪diff.
---

## Current changes
```

(次のページに続く)

(前のページからの続き)

```
!`git diff HEAD`

## Instructions

Summarize the changes above in two or three bullet points, then list any risks␣
↪you notice such as missing error handling, hardcoded values, or tests that␣
↪need updating. If the diff is empty, say there are no uncommitted changes.
```

!`git diff HEAD` 行は**動的コンテキスト注入** (ページ 300) を使用します。Claude Code はコマンドを実行し、Claude がスキルコンテンツを見る前にこの行をその出力に置き換えるため、指示は現在の diff がすでにインライン化された状態で到着します。

3. **スキルをテストする**

   Git プロジェクトを開き、任意のファイルに小さな編集を加えて、claude を実行して Claude Code を起動します。スキルは 2 つの方法でテストできます。

   説明に一致する内容を質問して Claude に自動的に呼び出させます。

   ```
   What did I change?
   ```

   またはスキル名で直接呼び出します。

   ```
   /summarize-changes
   ```

   どちらの方法でも、Claude は編集内容の短い要約とリスクのリストで応答する必要があります。

17.3 スキルの読み込み場所を選択する

スキルを保存する場所によって、どのセッションがそれを読み込むかが決まります。ホームディレクトリに保存すると、すべてのプロジェクトで読み込まれます。リポジトリにコミットすると、そこで作業するすべてのユーザーと共有できます。プラグインまたはマネージドセッティングを通じて配布すると、チーム全体に到達します。

場所	パス	読み込まれる場所
Enterprise	`.claude/skills/<skill-name>/SKILL.md` in the managed settings directory	組織がデプロイするマシン上のすべてのユーザー
Personal	`~/.claude/skills/<skill-name>/SKILL.md`	このマシン上のすべてのプロジェクト。ただし *Cowork* **またはクラウドセッション** (ページ 282) は除く
Project	`.claude/skills/<skill-name>/SKILL.md`	このリポジトリ内のセッション。コミットするとチームも取得できます

次のページに続く

表 3 – 前のページからの続き

場所	パス	読み込まれる場所
Nested	`<subdir>/.claude/skills/<skill-name>/SKILL.md`	`<subdir>` で開始されたセッション、またはその下で開始されたセッション。その上で開始されたセッションは、Claude がそこのファイルで作業を開始すると、スキルを 1 回読み込みます。*monorepos と subdirectories* (ページ 281) を参照してください
Additional directory	`.claude/skills/<skill-name>/SKILL.md` in a directory you pass with `--add-dir`	そのセッション。**プロジェクト外のディレクトリ** (ページ 281) を参照してください
Plugin	`<plugin>/skills/<skill-name>/SKILL.md`	プラグイン が有効な場所。`/plugin-name:skill-name` として
claude.ai account	claude.ai アカウント用に有効化されたスキル	Cowork セッション、クラウドセッション、およびそのアカウントでサインインするターミナルセッション。*claude.ai* **から同期されたスキル** (ページ 283) を参照してください

スキルフォルダは、これらのルールにも従います：

- **シンボリックリンク付きフォルダ**: enterprise、personal、または project の場所の `<skill-name>` エントリは、ディスク上の別の場所へのシンボリックリンクにすることができます。Claude Code は、複数の場所が同じターゲットを指している場合でも、ターゲットから `SKILL.md` を読み込み、スキルを 1 回だけ読み込みます。プラグインスキルは シンボリックリンクを異なる方法で処理します。
- **予約名** `synced`: スキルフォルダに `synced` という名前を付けないでください。大文字小文字は問いません。Claude Code は `~/.claude/skills/synced/` を *claude.ai* **からダウンロードされたスキル** (ページ 283) に使用し、enterprise、personal、および project の場所でこの名前で作成したスキルをスキップします。
- **予約名** `anthropic-skills`: プラグイン外では、`anthropic-skills` という名前のスキルフォルダまたはコマンドファイル、または `anthropic-skills:` で始まるものは読み込まれません。**同期されたスキル用に予約された名前** (ページ 285) を参照してください。
- **コマンドファイル**: `.claude/commands/` 内の Markdown ファイルは古い形式ですが、まだ機能します。`name` と `paths` を除く同じ *frontmatter* (ページ 288) をサポートしています。それを呼び出すために入力するコマンド名を見つけるには、**スキルがコマンド名を取得する方法** (ページ 292) を参照してください。新しい作業にはスキルを使用してください。スキルは **サポートファイル** (ページ 296) もサポートしているためです。
- **プラグインとしてのスキルフォルダ**: `.claude-plugin/plugin.json` をスキルフォルダに追加すると、`<name>@skills-dir` という名前の プラグイン として読み込まれます。これにより、エージェント、hooks、および MCP サーバーをバンドルできます。プロジェクトの `.claude/skills/` では、最初にワークスペーストラストダイアログを受け入れる必要があります。

17.3.1 monorepos と subdirectories でスキルを読み込む

Claude Code は、開始したディレクトリと、リポジトリルートまでのすべての親ディレクトリの `.claude/skills/` からプロジェクトスキルを読み込みます。そのため、`packages/frontend/` で開始しても、ルートで定義されたスキルが取得されます。v2.1.246 以降で *`/cd`* **でセッションを移動する** (ページ 1061) と、Claude Code は新しいディレクトリのプロジェクトスキルを追加します。

リンクされた git worktree で実行されているセッションでは、Claude Code は worktree ルートまでのみ親ディレクトリを検索します。Claude Code v2.1.277 以降では、worktree チェックアウトのルートに `.claude/skills` ディレクトリがない場合、Claude Code はメインチェックアウトのプロジェクトスキルを代わりに読み込みます。worktrees がメインチェックアウトと共有するもの を参照してください。

開始した場所の下の `.claude/skills/` ディレクトリ内のスキルは、起動時には読み込まれません。Claude がそのサブディレクトリ内のファイルを初めて読み込むか編集するときに読み込まれ、セッションの残りの間利用可能なままです。それまでは、`/` メニューに表示されず、名前で呼び出すことはできません。より早く読み込むには、サブディレクトリのパスで `/add-dir` を実行します。これには Claude Code v2.1.257 以降が必要です。

ネストされたスキルが別のスキルと同じ名前を共有する場合、両方が利用可能なままです。リポジトリルートに `deploy` スキルがあり、`apps/web/.claude/skills/` に別のスキルがある場合：

- `/deploy` はルートスキルを実行します。Claude Code は、Claude 用のディレクトリ修飾バリアントもリストし、作業しているファイルが存在するディレクトリのスキルを呼び出すように指示するため、ネストされたスキルは `apps/web/` での作業に適用されます。
- `/apps/web:deploy` はネストされたスキルを単独で実行します。その説明は、それが適用されるディレクトリに名前を付けます。

17.3.2 プロジェクト外のディレクトリからスキルを読み込む

`--add-dir` または `/add-dir` でディレクトリを追加すると、Claude Code はそのディレクトリの `.claude/skills/` 内のスキルを、その `.claude/commands/` および `.claude/agents/` とともに読み込みます。Agent SDK が TypeScript の `additionalDirectories` または Python の `add_dirs` を通じて追加するディレクトリは、SDK が `--add-dir` として渡すため、同じ方法で読み込まれます。`settings.json` の `permissions.additionalDirectories` セッティングはファイルアクセスのみを付与し、これらのいずれも読み込みません。

Claude Code は、起動時に `--add-dir` で渡したディレクトリの `.claude/skills/` を監視します。**セッション中にスキルを編集する** (ページ 286) で説明されています。追加されたディレクトリの `.claude/commands/` または `.claude/agents/` は監視しないため、そこでファイルを変更した後はセッションを再開してください。

これらの読み込みは、デフォルトで有効な `project` セッティングソース に依存します。`strictPluginOnlyCustomization` ポリシー、*bare mode* (ページ 696)、および *`--safe-mode`* (ページ 51) はそれぞれ、これらのページで説明されているようにさらに制限します。追加されたディレクトリが読み込むもの（`CLAUDE.md` およびプラグインセッティングを含む）の完全なテーブルについては、**追加ディレクトリはファイルアクセスを付与し、設定は付与しない** (ページ 1061) を参照してください。

17.3.3 同じ名前のスキルを解決する

2 つのスキルが同じディレクトリ名またはファイル名を共有する場合、各スキルがどこから来たかによって、/name がどちらを実行するかが決まります。フロントマターの name フィールドで設定された名前については、**スキルがコマンド名を取得する方法** (ページ 292) を参照してください。次の表は、enterprise、personal、project、nested、plugin、および claude.ai の場所、バンドルスキル、組み込みコマンド、およびコマンドファイルを対象としています：

同じ名前の場所	どのスキルが実行されるか
enterprise、personal、および project の 2 つ	Enterprise が personal より優先され、personal が project より優先されます。~/.claude/skills/ とプロジェクトの .claude/skills/ の両方に deploy がある場合、/deploy は personal スキルを実行します
これらの場所のいずれかと **バンドルされたスキル** (ページ 275)	あなたのスキルがバンドルされたコマンドを置き換えますが、そのエイリアスは置き換えません。プロジェクト code-review スキルは /code-review を置き換え、バンドルされたエイリアス /review はあなたのスキルを実行しません
これらの場所のいずれかと 組み込みコマンド	ローカルのターミナルセッションでは、自分のスキルが組み込みコマンドを置き換えますが、そのエイリアスは置き換えません。プロジェクトの usage スキルは /usage を置き換え、組み込みのエイリアス /cost は引き続き組み込みコマンドを実行します
スキルと .claude/commands/ 内のファイル	スキル
プロジェクトルートスキルとネストされたスキル	両方が読み込まれます。*monorepos と subdirectories* (ページ 281) を参照してください
プラグインスキルと上記の場所のいずれかのスキル	プラグインスキルは /plugin-name:skill-name として名前空間化されているため、両方が読み込まれます
上記のいずれかと *claude.ai* **アカウントから同期されたスキル** (ページ 283)	他のスキルまたはコマンド。同期されたスキルはその完全な名前でのみ実行されます。**同期されたスキル名が別のコマンドと一致する場合** (ページ 284) を参照してください

17.3.4 Cowork およびクラウドセッションでスキルを使用する

Cowork セッションおよび クラウドセッション（routines を含む）は、マシン上の ~/.claude/skills/ を読み込みません。インタラクティブおよびスケジュール済み Cowork セッションの両方は、claude.ai アカウント用に有効化されたスキルを読み込みます。これらはセッション開始時に同期されます。Desktop アプリサイドバーの **Customize** または claude.ai のスキルセッティングから管理します。クラウドセッションは、さらにクローンされたリポジトリの .claude/skills/ にコミットされたプロジェクトスキルを読み込みます。

スキルがマシン上の ~/.claude/skills/ にのみ存在する場合、routine がそれを呼び出すと、Claude Code はスキルが見つからないと報告します。各 routine 実行は新しいクラウドセッションとして開始されるためです。これらのセッションで personal スキルを利用可能にするには：

- Cowork およびクラウドセッションの場合、claude.ai アカウント用にスキルを有効化します。
- クラウドセッションの場合、代わりにスキルをリポジトリの `.claude/skills/` にコミットできます。リポジトリの `.claude/settings.json` で宣言されたプラグインおよびユーザーセッティングでのみ有効化されたプラグインは クラウドセッションで読み込まれません。

Desktop scheduled tasks はマシン上でローカルに実行されるため、`~/.claude/skills/` を読み込みます。

17.3.5 claude.ai から同期されたスキル

このセクションは、Cowork またはクラウドセッションを使用する場合、またはターミナルで claude.ai アカウントで Claude Code にサインインする場合に適用されます。これらのセッションでは、Claude Code は claude.ai アカウント用に有効化されたスキルを読み込みます。セットアップは不要です。**同期されたスキルが読み込まれる場所** (ページ 283) で説明されています。これらのスキルには、claude.ai セッティングで作成または有効化したスキル、組織がそこで提供するスキル、および `pdf` や `xlsx` などの Anthropic の組み込みスキルが含まれます。

Claude Code は、セッションが実行されるマシンで作成したファイルを読み込むのではなく、アカウントから同期されたスキルをダウンロードするため、**スキルの場所** (ページ 279) に保存するスキルには適用されないルールを同期されたスキルに適用します。

同期されたスキルが読み込まれる場所

Cowork またはクラウドセッションでは、Claude Code は claude.ai アカウント用に有効化されたスキルを読み込みます。*Cowork* **およびクラウドセッションでのスキル** (ページ 282) は、これらのセッションが取得するスキルを選択する方法を説明しています。

ターミナルでは、Claude Code は claude.ai アカウントでサインインするセッションでこれらのスキルを同期します。セッションが開始されると、Claude Code はアカウントのスキルを `~/.claude/skills/synced/` にバックグラウンドでダウンロードし、セッション実行中は約 10 分ごとに claude.ai の変更をチェックします。チェックでスキルが claude.ai で追加、編集、または無効化されたことが判明すると、Claude Code は実行中のセッションでそれを追加、更新、または削除します。再起動は不要です。ターミナルセッションでの同期には Claude Code v2.1.273 以降が必要です。

同期は起動を遅延させません。Claude がスキルを呼び出すときにのみ、スキルのダウンロードを待つためです。短い **非インタラクティブ** (ページ 695) 実行は、新しく追加されたスキルがダウンロードされる前に終了する可能性があります。その場合、後のセッションがそれをダウンロードします。非インタラクティブ実行がスキルをダウンロードし、プロンプトに答える前にリストを待つようにするには、`CLAUDE_CODE_SYNC_SKILLS` を 1 に設定します。

Claude Code は、claude.ai アカウントでサインインし、Anthropic からフィーチャーフラグを取得する セッションでのみ同期します。これらのセッションでは同期しません：

- `/login` で保存されたサインインを使用しないセッション。例えば、API キーで認証するセッション、または `ANTHROPIC_AUTH_TOKEN`、`CLAUDE_CODE_OAUTH_TOKEN`、または `apiKeyHelper` スクリプトが認証情報を提供するセッション
- フィーチャーフラグを取得しないセッション。例えば、Amazon Bedrock 上のセッション、または `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` を設定したセッション
- *bare mode* (ページ 696) のセッション、または `--safe-mode` で開始したセッション

- 組織のマネージドセッティングが スキルをプラグインソースにロック するセッション、または user を除外する *--setting-sources* (ページ 51) リストで開始したセッション

セッション中に /login でサインインする場合、Claude Code を再起動して同期を開始します。

以前のセッションが同期したスキルはディスク上に残ります。Claude Code は、同じアカウントにサインインした後のセッションでそれらを読み込みます。claude.ai に到達できない場合でも同様です。

Claude Code は同期されたスキルをダウンロードし、アップロードしません。~/.claude/skills/synced/ の下のファイルを編集する場合、変更は claude.ai アカウントに保存されず、後の同期で上書きまたは削除される可能性があります。同期されたスキルを変更するには、claude.ai で更新します。次の同期で新しいバージョンがダウンロードされます。

同期されたスキルを確認するには、/skills を実行します。メニューは claude.ai sync の下にそれらをリストします。

pdf や xlsx などの Anthropic のスキルの一部は常に同期します。その他については、claude.ai のスキルセッティングでスキルをオンまたはオフにして、同期するかどうかを変更します。

マシンでの同期を停止するには、ユーザーセッティングで syncClaudeAiSkills を false に設定します。Claude Code はダウンロードを停止し、次回起動時に既に同期したスキルを ~/.claude/skills/.trash/ に移動し、それ以上読み込みません。組織は claude.ai で Skills をオフにすることで、すべてのユーザーの同期をオフにできます。同期を停止しながら Skills をオンのままにするには、マネージドセッティング で同じキーを設定できます。

組織が claude.ai で Skills をオフにする場合、Claude Code はダウンロードされたスキルを削除し、読み込みを停止します。削除されたスキルは ~/.claude/skills/.trash/ に移動されます。保持期間スイープ がそれらを削除するまで、ファイルを復旧できます。組織が Skills を再度オンにすると、Claude Code は次の同期で有効化したスキルをダウンロードします。

同期されたスキル名が別のコマンドと一致する場合

同期されたスキルは、その短い名前 /<name> または完全な名前 /anthropic-skills:<name> で呼び出すことができます。別のコマンドがその短い名前を使用する場合、/<name> は他のコマンドを実行し、同期されたスキルは /anthropic-skills:<name> としてのみ実行されます。ローカル deploy スキルと同期された deploy がある場合、/deploy はローカルスキルを実行し、/anthropic-skills:deploy は同期されたスキルを実行します。v2.1.269 より前では、同期されたスキルは短い名前のみを持っていました。

/ メニュー、/skills、および /context では、同期されたスキルは短い名前の下に表示されるか、別のコマンドがその短い名前を使用している間は完全な名前の下に表示されます。セッションで /skills を実行します。リストの下の注記は、短い名前を失った各同期されたスキルを説明します。~/.claude/ の personal スキルまたはコマンドファイルの 1 つがその名前を使用する場合、注記は名前を変更または削除して解放する内容も説明します。

v2.1.269 から v2.1.280 では、これらのリストはすべての同期されたスキルをその完全な名前の下に表示し、/skills にはそのような注記がありませんでした。両方とも v2.1.281 で変更されました。

短い名前を使用するコマンドは、これらのいずれかにすることができます：

- 組み込みコマンドまたは **バンドルされたスキル** (ページ 275)。例えば、バンドルされたスキルをオフにした後に利用できないもの
- **ローカルレベル** (ページ 279) のスキルまたは .claude/commands/ 内のファイル

- プラグインスキル
- *MCP* **プロンプト** (ページ 409)

Claude Code は同期されたスキルにラベルを付けるため、どこから来たかを判断できます。`/skills` メニューと `/context` は同期されたスキルを `claude.ai sync` の下にグループ化し、`/` コマンドメニューは claude.ai から来たものとしてマークします。

名前を比較するとき、Claude Code は大文字小文字、スペース、および目に見えない文字を無視し、全幅文字とダッシュバリアントなどの互換性形式をそれらのプレーン等価物として扱います。例えば、`Commit` という名前の同期されたスキルとローカル `commit` スキルは同じ名前と見なされるため、`/commit` はローカルスキルを実行し続けます。

別のアルファベットの見た目が似た文字だけで異なる名前は異なる名前と見なされ、`claude.ai sync` ラベルは 2 つを区別する方法です。これらのチェックとラベルには Claude Code v2.1.228 以降が必要です。

同期されたスキル用に予約された名前

Claude Code は名前 `anthropic-skills` と、`anthropic-skills:pdf` などのその名前空間内のすべての名前を、claude.ai から同期されたスキル用に予約しているため、同期されたスキルの完全な名前は他のものを実行しません。この名前は、claude.ai アカウントでサインインするかどうかに関わらず、すべてのセッションで予約されています。

- **スキルフォルダ、frontmatter** `name`**、** `.claude/commands/` **内のファイルまたはサブフォルダ、または** 保存されたワークフロー: 読み込まれません。スタートアップ通知 は、名前を変更または編集する最初のアイテムに名前を付けます。
- `anthropic-skills` **という名前のプラグイン**: 読み込まれます。そのスキルの 1 つと同期されたスキルの両方が `<name>` という名前の場合、`/anthropic-skills:<name>` は同期されたスキルを実行します。
- `anthropic-skills` **という名前の MCP サーバー**: 接続され、そのツールは機能しますが、**そのプロンプトはコマンドとして表示されません** (ページ 409)。MCP 設定でサーバーの名前を変更してそれらをリストします。

Claude Code が同期されたスキルの frontmatter をどのように処理するか

Claude Code は、同期されたスキルの frontmatter に 2 つのルールを適用します：

- フロントマターはあらゆる種類のセッションで適用されるため、`allowed-tools` による許可は通常の **権限フロー** (ページ 1039) を経由します。組織が `allowManagedPermissionRulesOnly` を設定している場合、この許可は **適用されません** (ページ 299)。
- Claude Code は、スキルが提供する表示テキスト（説明など）をサニタイズします。制御文字を削除し、Claude に到達するテキスト（説明など）では、テキストが Claude Code の内部フォーマットを模倣できないように、角括弧をエスケープします。このサニタイズには Claude Code v2.1.228 以降が必要です。

Claude Code が同期されたスキルの本体をどのように処理するか

Claude Code が同期されたスキルの本体で何をするかは、セッションが実行される場所によって異なります：

- クラウドセッションでは、本体はローカルスキルが持つ動作を保持します。セッションは分離されたコンテナで実行されるためです。
- デスクトップ上の Cowork セッションでは、本体はローカルスキルが持つ動作を保持します。ただし、Claude Code はすべての ! コマンドラインを *disableSkillShellExecution* **プレースホルダ** (ページ 300) に置き換えます。すべてのスキルに対して行うように、そこで提供します。
- マシン上の他のセッションでは、Claude Code は コマンド <skills--inject-dynamic-context> を実行しません。@ 参照が名前を付けるファイルをローカルスキルの場合のように添付しません。`${CLAUDE_PROJECT_DIR}` および `${CLAUDE_SESSION_ID}` プレースホルダを置き換えません。そのため、@ 参照と両方のプレースホルダは Claude にリテラルテキストとして到達します。! コマンドラインも `disableSkillShellExecution` がオンの場合、リテラルテキストまたはそのプレースホルダとして到達します。この処理には Claude Code v2.1.228 以降が必要です。

17.3.6 セッション中にスキルを編集する

Claude Code は *bare mode* (ページ 696) を除き、スキルディレクトリのファイル変更を監視します。`~/.claude/skills/`、プロジェクト `.claude/skills/`、または `--add-dir` ディレクトリ内の `.claude/skills/` の下のスキルを追加、編集、または削除すると、Claude Code は現在のセッション内で変更を取得します。再起動は不要です。

セッション開始時に存在しなかったトップレベルスキルディレクトリを作成する場合、`/reload-skills` を実行して、そこに配置したスキルを取得します。Claude Code はまだそのディレクトリを監視していないため、そこで後の変更を行った後、`/reload-skills` を再度実行します。

ライブ変更検出は `SKILL.md` テキストのみをカバーします。スキルフォルダが プラグイン でもある場合、`hooks/`、`.mcp.json`、`agents/`、および `output-styles/` への変更は `/reload-plugins` で有効になります。

17.3.7 スキルを削除する

スキルを削除する方法は、どこから来たかによって異なります：

- **Personal または project スキル**: スキルのディレクトリ `~/.claude/skills/<skill-name>/` または `.claude/skills/<skill-name>/` を削除します。Claude Code は **現在のセッションの *skills* からそれを削除します** (ページ 286)。Claude Code が既に読み込んだコンテンツは **スキルコンテンツライフサイクル** (ページ 297) に従います。
- **Enterprise スキル**: 管理者が マネージドセッティングディレクトリ 内の `.claude/skills/` からスキルのディレクトリを削除します。例えば、Linux では `/etc/claude-code/.claude/skills/<skill-name>/`。
- **Plugin スキル**: `/plugin` メニューから、または `/plugin uninstall <plugin-name>@<marketplace-name>` でプラグインを無効化またはアンインストールします。Claude Code は 変更が適用される か、再起動するときにプラグインのスキルをアンロードします。
- **claude.ai から同期されたスキル**: claude.ai のスキルセッティングで、**有効化した** (ページ 282) 同じ場所でスキルをオフにします。Claude Code は **スキルを同期する** (ページ 283) 次回にそれを `~/.claude/`

skills/synced/ から削除します。代わりに手動でディレクトリを削除する場合、次の同期はスキルが claude.ai で有効なままの間、それを再度ダウンロードします。

- **バンドルされたスキル**: *disableBundledSkills* (ページ 275) を true に設定してバンドルされたスキルをオフにするか、*skillOverrides* (ページ 306) で 1 つのスキルを "off" に設定して非表示にします。

personal または project スキルを保持しながら Claude がそれを自動的に呼び出すのを停止するには、frontmatter で *disable-model-invocation: true* (ページ 296) を設定するか、ファイルを編集したくない場合は *skillOverrides* (ページ 306) で "user-invocable-only" を設定します。

17.4 スキルを設定する

スキルは SKILL.md の上部にある YAML フロントマターと、その後に続くマークダウンコンテンツを通じて設定されます。

17.4.1 スキルコンテンツのタイプ

スキルファイルには任意の指示を含めることができますが、それらをどのように呼び出したいかを考えることで、何を含めるべきかを決めるのに役立ちます。

リファレンスコンテンツ は、Claude が現在の作業に適用する知識を追加します。規約、パターン、スタイルガイド、ドメイン知識などです。このコンテンツはインラインで実行されるため、Claude は会話コンテキストと一緒にそれを使用できます。

```
---
name: api-conventions
description: API design patterns for this codebase
---

When writing API endpoints:
- Use RESTful naming conventions
- Return consistent error formats
- Include request validation
```

タスクコンテンツ は、デプロイメント、コミット、コード生成など、特定のアクションのステップバイステップの指示を Claude に提供します。これらは、Claude に自動的に実行させるのではなく、/skill-name で直接呼び出したいアクションであることが多いです。disable-model-invocation: true を追加して、Claude が自動的にトリガーするのを防ぎます。以下の例は context: fork を追加しており、これはスキルを独自のサブエージェントコンテキストで実行します。**サブエージェントでスキルを実行する** (ページ 303) を参照してください。

```
---
name: deploy
description: Deploy the application to production
context: fork
disable-model-invocation: true
```

(次のページに続く)

(前のページからの続き)

```
---

Deploy the application:
1. Run the test suite
2. Build the application
3. Push to the deployment target
```

本体自体は簡潔に保ちます。スキルが読み込まれると、そのコンテンツは **ターン全体でコンテキストに留まる** (ページ 297) ため、すべての行は繰り返されるトークンコストになります。何をするかを述べ、どのように、なぜするかを説明するのではなく、*CLAUDE.md* **コンテンツ** (ページ 120) に対して適用するのと同じ簡潔性テストを適用します。

17.4.2 フロントマターリファレンス

SKILL.md の上部にある --- マーカー間の YAML **フロントマター** でスキルを設定し、閉じる --- の後にマークダウンとしてスキルの指示を記述します。フィールド名は when_to_use を除いてハイフンで区切られた小文字の単語を使用します。.claude/commands/ の **コマンドファイル** (ページ 279) は name と paths を除いて同じフィールドを受け入れます。この例は 4 つのフィールドを設定しています。

```
---
name: my-skill
description: What this skill does
disable-model-invocation: true
allowed-tools: Read Grep
---

Your skill instructions here...
```

すべてのフィールドはオプションです。Claude がスキルをいつ使用するかを知るために、description のみが推奨されます。フィールド名はテーブルと正確に一致する必要があります。ハイフンを含めて、Claude Code は認識しないフィールドをエラーを報告せずに無視します。

Claude Code はフロントマターを読み込むのは、開く --- がファイルの最初の行である場合のみです。そうでない場合、--- マーカーを含むファイル全体をスキルコンテンツとして扱います。マーカー間の YAML が解析されない場合、スキルはフィールドセットなしで読み込まれます。**スキルがトリガーされない** (ページ 314) を参照して、エラーを見つけて修正してください。

ブール値フィールドは、true と false に加えて、任意の大文字小文字で yes、no、on、off、1、0 を受け入れます。v2.1.218 より前では、Claude Code は true と false のみを認識していました。

フィールド	必須	説明
`name`	いいえ	/ メニューに表示されるコマンド名。ディレクトリ名がデフォルトです。スキルを呼び出すために入力する名前とフィールドがどのように相互作用するかについては、**スキルがコマンド名を取得する方法** (ページ 292) を参照してください。
`description`	推奨	スキルが何をするか、いつ使用するか。Claude はこれを使用してスキルを適用するかどうかを決定します。省略された場合、マークダウンコンテンツの最初の空でない行を使用します。主要なユースケースを最初に配置します。結合された `description` と `when_to_use` テキストはコンテキスト使用量を削減するためにスキルリストで 1,536 文字で切り詰められます。
`when_to_use`	いいえ	Claude がスキルを呼び出すべき時期に関する追加コンテキスト。トリガーフレーズやリクエスト例など。スキルリストの `description` に追加され、1,536 文字の上限にカウントされます。
`argument-hint`	いいえ	オートコンプリート中に表示されるヒント。予想される引数を示します。例：`[issue-number]` または `[filename] [format]`。
`arguments`	いいえ	スキルコンテンツの *$name* **置換** (ページ 294) のための名前付き位置引数。スペース区切り文字列または YAML リストを受け入れます。名前は引数位置に順序でマップされます。
`disable-model-invocation`	いいえ	Claude がこのスキルを自動的に読み込むのを防ぐために `true` に設定します。`/name` で手動でトリガーしたいワークフローに使用します。また、スキルが **サブエージェントに事前読み込みされる** (ページ 337) のを防ぎます。v2.1.196 以降、スキルがプロンプトとして スケジュール済みタスク が発火したときに実行されるのも防ぎます。デフォルト：`false`。
`user-invocable`	いいえ	Claude のみがスキルを呼び出すべき場合は `false` に設定します。Claude Code はそれを / メニューから非表示にし、`/name` を入力したときに実行しません。ユーザーが直接呼び出すべきではないバックグラウンド知識に使用します。デフォルト：`true`。
`allowed-tools`	いいえ	このスキルを呼び出すターン中に Claude が許可を求めずに使用できるツール。許可はあなたが次のメッセージを送信するときにクリアされます。スペースまたはコンマ区切り文字列、または YAML リストを受け入れます。**スキルのツールを事前承認する** (ページ 298) を参照してください。

次のページに続く

表 5 – 前のページからの続き

フィールド	必須	説明
disallowed-tools	いいえ	このスキルがアクティブな間、Claude の利用可能なプールから削除されるツール。バックグラウンドループの AskUserQuestion など、自律的なスキルが特定のツールを呼び出すべきではない場合に使用します。スペースまたはコンマ区切り文字列、または YAML リストを受け入れます。制限はあなたが次のメッセージを送信するときにクリアされます。拒否ルールと同様に、他のツールが残っている間、フィールドは EndConversation を削除できません。
model	いいえ	このスキルがアクティブな場合に使用するモデル。オーバーライドは現在のターンの残りに適用され、設定に保存されません。セッションモデルは次のプロンプトを送信するときに再開されます。*/model* (ページ 725) と同じ値、または inherit を受け入れてアクティブなモデルを保持します。組織の *availableModels* (ページ 732) 許可リストで除外された値は使用されず、セッションは現在のモデルを保持します。自動モード では、および 分類器がコマンドをレビューしている間の計画モード では、自動モードがサポートしないモデルも使用されず、セッションは現在のモデルを保持します。context: fork では、値は **フォークされたサブエージェントのモデル** (ページ 303) を設定し、除外された値は **サブエージェントモデルオーバーライドと同じルール** (ページ 732) に従います。
effort	いいえ	このスキルがアクティブな場合の *effort* **レベル** (ページ 747)。セッションの effort レベルを上書きします。省略した場合、レベルは *effort* **の解決順序** (ページ 747) によって決まります。オプション：low、medium、high、xhigh、max。利用可能なレベルはモデルに依存します。
context	いいえ	フォークされたサブエージェントコンテキストで実行するために fork に設定します。**サブエージェントでスキルを実行する** (ページ 303) を参照してください。
agent	いいえ	context: fork が設定されている場合に使用するサブエージェントタイプ。

次のページに続く

表 5 – 前のページからの続き

フィールド	必須	説明
background	いいえ	context: fork にのみ適用されます。スキルを呼び出すターンでフォークされたサブエージェントの結果を待つために false に設定します。**バックグラウンドで実行する** (ページ 303) のではなく。デフォルト：true。Claude Code v2.1.218 以降が必要です。
hooks	いいえ	Claude Code がスキルが呼び出されたときに登録し、セッションの残りの間実行し続けるフック。設定形式と once オプションについては、**スキルとエージェントのフック** (ページ 465) を参照してください。
paths	いいえ	このスキルがアクティブ化される時期を制限する Glob パターン。コンマ区切り文字列または YAML リストを受け入れます。設定されている場合、Claude はパターンに一致するファイルで作業している場合にのみ自動的にスキルを読み込みます。**パス固有のルール** (ページ 1004) と同じ形式を使用します。
shell	いいえ	このスキルの !`command` および ```! ブロックに使用するシェル。bash（デフォルト）または powershell を受け入れます。powershell を設定すると、PowerShell ツール が有効な場合、PowerShell 経由でインラインシェルコマンドを実行します。Windows では Git Bash なしでデフォルトでオン、Git Bash では claude.ai および Console アカウントでデフォルトでオン、Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry セッション、および macOS、Linux、WSL では CLAUDE_CODE_USE_POWERSHELL_TOOL=1 が必要です。ツールをオフにするには 0 に設定します。
metadata	いいえ	権限またはカタログフィールドなど、独自のキーと値のデータ用の自由形式の YAML マップ。SKILL.md から独自のツーリングで読み取られます。Claude Code はその内容に対して動作せず、マップではない値を削除します。paths などのフロントマターフィールド名をキーとして再利用しないでください。
license	いいえ	スキルをカバーするライセンス。Agent Skills 仕様の一部。*Claude Code* **外でスキルフロントマターを使用する** (ページ 292) を参照してください。Claude Code はフィールドを受け入れますが、それに対して動作しません。

次のページに続く

表 5 – 前のページからの続き

フィールド	必須	説明
compatibility	いいえ	対象製品やシステム前提条件など、スキルの環境要件。Agent Skills 仕様で定義されています。*Claude Code* **外でスキルフロントマターを使用する** (ページ 292) を参照してください。最大 500 文字の文字列を受け入れます。Claude Code はフィールドを受け入れますが、それに対して動作しません。

Claude Code 外でスキルフロントマターを使用する

Claude Code はテーブル上のすべてのフィールドを受け入れます。Claude Code 外では、Agent Skills 仕様のフィールドのみを使用できます。

配布パス	使用できるフロントマターフィールド
任意のレベル (ページ 279) の Claude Code スキル。プラグイン スキルを含む	テーブル上のすべてのフィールド
claude.ai スキルアップロード、Skills API、および anthropics/skills の package_skill.py でのパッケージング	name、description、license、compatibility、metadata、allowed-tools

たとえば、*Cowork* **とクラウドセッション** (ページ 282) とルーチンで使用するために個人スキルを claude.ai アカウント用に有効にする場合、それを claude.ai にアップロードするため、同じルールが適用されます。

仕様が許可しないフィールドを含める場合、パッケージングまたはアップロードはフィールドを無視する代わりにハードエラーで失敗します。

```
Unexpected key(s) in SKILL.md frontmatter: argument-hint. Allowed properties are:␣
↪allowed-tools, compatibility, description, license, metadata, name
```

フロントマターを仕様の 6 つのフィールドに制限すると、上記の予期しないキーエラーを回避できます。Agent Skills 仕様 および Skills API 要件 は、これらのパスが検証するその他すべてを定義します。**動的コンテキスト注入** (ページ 300) などの Claude Code のみの本体機能は、claude.ai チャットまたは API を通じて機能しません。Claude Code はすべての 6 つのフィールドを受け入れるため、仕様に従うフロントマターは変更なしに Claude Code で読み込まれます。

スキルがコマンド名を取得する方法

スキルを呼び出すために入力するコマンドは、スキルファイルが存在する場所から、およびスキルディレクトリとプラグインスキルの場合はフロントマター name フィールドからも来ます。個人またはプロジェクトスキルディレクトリでは、name は / メニューに表示されるコマンドを設定し、別のコマンドがその名前をまだ使用していない場合は入力するコマンドです。ディレクトリ名もスキルを呼び出します。プラグインスキルでは、name はコマンドの最後のセグメントを設定し、プラグインプレフィックスは所定の位置に留まります。

以下のテーブルは、各レイアウトのコマンド名がどこから来るかを示しています。

スキルの場所	コマンド名のソース	例
~/.claude/skills/ または .claude/skills/ の下のスキルディレクトリ	フロントマター name またはディレクトリ名	.claude/skills/deploy-staging/SKILL.md → /deploy-staging、または name: deploy で /deploy
ネストされた (ページ 279) .claude/skills/ ディレクトリ。別のスキルと名前が衝突する場合	作業ディレクトリに相対的なサブディレクトリパス。その後、スキルディレクトリ名	apps/web/.claude/skills/deploy/SKILL.md → /apps/web:deploy
.claude/commands/ の下のファイル	拡張子なしのファイル名	.claude/commands/deploy.md → /deploy
.claude/commands/ のサブディレクトリ内のファイル	commands/ に相対的なサブディレクトリパス。各 / を : に置き換え。その後、拡張子なしのファイル名	.claude/commands/frontend/component.md → /frontend:component
プラグイン skills/ サブディレクトリ	フロントマター name またはディレクトリ名。プラグインでネームスペース化	my-plugin/skills/review/SKILL.md → /my-plugin:review。または name: fancy で /my-plugin:fancy
プラグインルート SKILL.md	フロントマター name。フォールバックとしてプラグインディレクトリ名	my-plugin/SKILL.md と name: review → /my-plugin:review。プラグインルートの単一スキル を参照
claude.ai **から同期されたスキル** (ページ 283)	claude.ai アカウント上のスキルの名前。anthropic-skills: でプレフィックス化	アカウントスキル deploy → /anthropic-skills:deploy。または他のコマンドがその名前を使用していない場合は /deploy

プラグインスキルでは、フロントマター name はコマンドの最後のセグメント内のディレクトリ名を置き換えるため、my-plugin/skills/review/SKILL.md と name: fancy は /my-plugin:fancy になります。別の /fancy もスキルを呼び出します。別のコマンドがその名前をまだ使用していない場合。書き込む name がプラグイン独自のプレフィックスで既に始まる場合、Claude Code は v2.1.246 以降でプレフィックスを再度追加しません。たとえば、name: my-plugin:fancy は依然として /my-plugin:fancy になります。v2.1.216 から v2.1.245 まで、Claude Code は name がそれを既に実行していた場合、プレフィックスを倍にしました。

非対話型セッション (ページ 695) では、名前 help と feedback はターミナルのみの組み込みコマンド用に予約されていないため、これらの名前を持つプラグインスキルはそこで裸のコマンドを保持します。他のすべてのターミナルのみの組み込みの名前（/login など）は、これらのセッションでコマンドを実行できないにもかかわらず、予約されたままです。

プラグインルート SKILL.md の場合、スキルディレクトリがないため、name は最後のセグメント全体を提供します。name フィールドがない場合、Claude Code はプラグインのディレクトリ名にフォールバックします。

利用可能な文字列置換

スキルはスキルコンテンツの動的値の文字列置換をサポートしています。

変数	説明
$ARGUMENTS	スキルを呼び出すときに渡されたすべての引数。プレースホルダーが引数を受け取らない場合、Claude Code は ARGUMENTS: <value> として追加します。**スキルに引数を渡す** (ページ 299) を参照してください。
$ARGUMENTS[N]	0 ベースのインデックスで特定の引数にアクセスします。例：最初の引数の場合は $ARGUMENTS[0]。
$N	$ARGUMENTS[N] の短縮形。例：最初の引数の場合は $0、2 番目の引数の場合は $1。
$name	*arguments* (ページ 288) フロントマターリストで宣言された名前付き引数。名前は位置に順序でマップされるため、arguments: [issue, branch] では、プレースホルダー $issue は最初の引数に展開され、$branch は 2 番目に展開されます。
${CLAUDE_SESSION_ID}	現在のセッション ID。ログ、セッション固有のファイルの作成、またはスキル出力とセッションの相関に役立ちます。
${CLAUDE_EFFORT}	現在の努力レベル：low、medium、high、xhigh、または max。これを使用して、アクティブな努力設定にスキル指示を適応させます。
${CLAUDE_SKILL_DIR}	スキルの SKILL.md ファイルを含むディレクトリ。プラグインスキルの場合、これはプラグインルートではなく、プラグイン内のスキルのサブディレクトリです。現在の作業ディレクトリに関係なく、スキルにバンドルされたスクリプトまたはファイルを参照するために bash インジェクションコマンドで使用します。
${CLAUDE_PROJECT_DIR}	プロジェクトルートディレクトリ。これは **フック** (ページ 463) と MCP サーバーが CLAUDE_PROJECT_DIR として受け取るのと同じパスです。${CLAUDE_PROJECT_DIR}/.claude/hooks/helper.sh など、スキルがインストールされている場所に関係なく、プロジェクトローカルスクリプトまたはファイルを参照するために使用します。
${CLAUDE_PLUGIN_ROOT}	プラグインのインストールディレクトリ。プラグインスキルでのみ置換されます。プラグイン内の任意の場所にバンドルされたスクリプトまたはファイル（プラグインのスキル間で共有されるリソースを含む）を参照するために使用します。プラグイン環境変数 を参照してください。
${CLAUDE_PLUGIN_DATA}	プラグインの 永続データディレクトリ。プラグイン更新を生き残ります。プラグインスキルでのみ置換されます。インストール済みの依存関係、生成されたファイル、または更新を超えて存続する必要があるキャッシュを参照するために使用します。

Claude Code は ${CLAUDE_SKILL_DIR} と ${CLAUDE_PROJECT_DIR} を 2 つの場所で置換します。スキルのマークダウンコンテンツ、および *allowed-tools* (ページ 288) フロントマターの Bash ルール。プラグインスキルでは、Claude Code は ${CLAUDE_PLUGIN_ROOT} と ${CLAUDE_PLUGIN_DATA} を同じ 2 つの場所で置換します。両方の場所で同じ変数を使用すると、スキルは許可プロンプトなしでバンドルされたス

クリプトを実行できます。以下のスキルはパターンを示しています。

```
---
name: render-chart
description: Render a chart from a CSV file
allowed-tools: Bash(${CLAUDE_SKILL_DIR}/scripts/render.sh *)
---

Run `${CLAUDE_SKILL_DIR}/scripts/render.sh <csv-file>` to render the chart.
```

このスキルが ~/.claude/skills/render-chart/ にインストールされている場合、${CLAUDE_SKILL_DIR} の両方の出現はそのディレクトリに展開されます。allowed-tools ルールはスキル本体が Claude に実行するよう指示する正確なコマンドと一致するため、スクリプトはプロンプトなしで実行されます。

${CLAUDE_PROJECT_DIR} 置換には Claude Code v2.1.196 以降が必要です。

インデックス付き引数はシェルスタイルのクォートを使用するため、複数単語の値をクォートで囲んで単一の引数として渡します。たとえば、/my-skill "hello world" second は $0 を hello world に展開し、$1 を second に展開します。$ARGUMENTS プレースホルダーは常に入力されたとおりの完全な引数文字列に展開されます。

対応する引数がないインデックス付きプレースホルダー（1 つの引数のみが渡された場合の $2 など）はコンテンツで変更されないままです。*arguments* (ページ 288) フロントマターからの対応する引数がない名前付きプレースホルダーは空の文字列に展開されます。

$1 または $ARGUMENTS などのテキストを含む引数値を渡す場合、Claude Code はそれをリテラルテキストとして挿入し、展開しません。たとえば、スキルの本体に Summarize $0 が含まれており、/summarize "$ARGUMENTS from yesterday" を実行する場合、Claude は Summarize $ARGUMENTS from yesterday を受け取ります。Claude Code は、引数を挿入した後、${CLAUDE_SKILL_DIR} などの ${CLAUDE_*} 変数を置換します。

数字、ARGUMENTS、または宣言された引数名の前にリテラル $ を含めるには（例：散文の $1.00）、バックスラッシュでエスケープします。\$1.00。他の $ の前のバックスラッシュは変更されないままです。トークンの直前の単一バックスラッシュのみがそれをエスケープします。\\$1 などの二重バックスラッシュは両方のバックスラッシュを所定の位置に残し、$1 は依然として引数値に展開されます。バックスラッシュエスケープはこれらの引数プレースホルダーのみをカバーします。バックスラッシュは、変数が適用される ${CLAUDE_*} 変数の置換を防ぎません。

置換を使用した例：

```
---
name: session-logger
description: Log activity for this session
---

Log the following to logs/${CLAUDE_SESSION_ID}.log:

$ARGUMENTS
```

17.4.3 サポートファイルを追加する

スキルはディレクトリに複数のファイルを含めることができます。これにより、SKILL.md は本質的なものに焦点を当てることができ、Claude は必要に応じてのみ詳細なリファレンス資料にアクセスできます。大規模なリファレンスドキュメント、API 仕様、またはサンプルコレクションは、スキルが実行されるたびにコンテキストに読み込む必要はありません。

```
my-skill/
├─── SKILL.md (required - overview and navigation)
├─── reference.md (detailed API docs - loaded when needed)
├─── examples.md (usage examples - loaded when needed)
└─── scripts/
    └─── helper.py (utility script - executed, not loaded)
```

SKILL.md からサポートファイルを参照して、Claude が各ファイルに何が含まれているか、いつそれを読み込むかを知るようにします。

```
## Additional resources

- For complete API details, see [reference.md](reference.md)
- For usage examples, see [examples.md](examples.md)
```

Tip:

SKILL.md を 500 行以下に保ちます。詳細なリファレンス資料を別のファイルに移動します。

17.4.4 スキルを呼び出すユーザーを制御する

デフォルトでは、あなたと Claude の両方がスキルを呼び出すことができます。/skill-name を入力して直接呼び出すことができ、Claude は会話に関連する場合に自動的に読み込むことができます。2 つのフロントマターフィールドでこれを制限できます。

- disable-model-invocation: true：あなたのみがスキルを呼び出すことができます。/commit、/deploy、/send-slack-message など、副作用があるワークフロー、またはタイミングを制御したいワークフローに使用します。コードが準備完了に見えるため、Claude がデプロイすることを望みません。
- user-invocable: false：Claude のみがスキルを呼び出すことができます。アクションとして実行できないバックグラウンド知識に使用します。legacy-system-context スキルは古いシステムがどのように機能するかを説明します。Claude はこれが関連する場合に知るべきですが、/legacy-system-context はユーザーが実行する意味のあるアクションではありません。

この例は、あなたのみがトリガーできるデプロイスキルを作成します。disable-model-invocation: true を設定した場合、Claude はスキルを自動的に実行できません。

```
---
name: deploy
description: Deploy the application to production
disable-model-invocation: true
---

Deploy $ARGUMENTS to production:

1. Run the test suite
2. Build the application
3. Push to the deployment target
4. Verify the deployment succeeded
```

Claude が試みた場合、Claude Code は呼び出しをブロックし、デプロイステップを別の方法で再現しないよう指示するため、/deploy を自分で実行することを提案することを期待してください。

2 つのフィールドが呼び出しとコンテキスト読み込みにどのように影響するかは次のとおりです。

フロントマター	あなたが呼び出せる	Claude が呼び出せる	コンテキストに読み込まれる時期
（デフォルト）	はい	はい	説明は常にコンテキストにあり、呼び出されたときに完全なスキルが読み込まれます
disable-model-invocation: true	はい	いいえ	説明はコンテキストにはなく、あなたが呼び出したときに完全なスキルが読み込まれます
user-invocable: false	いいえ	はい	説明は常にコンテキストにあり、呼び出されたときに完全なスキルが読み込まれます

注釈:

通常のセッションでは、スキルの説明がコンテキストに読み込まれるため、Claude は何が利用可能かを知っていますが、完全なスキルコンテンツは呼び出されたときにのみ読み込まれます。**事前読み込みされたスキルを持つサブエージェント** (ページ 337) は異なります。完全なスキルコンテンツはスタートアップで注入されます。

17.4.5 スキルコンテンツのライフサイクル

あなたまたは Claude がスキルを呼び出すと、レンダリングされた SKILL.md コンテンツが会話に単一のメッセージとして入力され、後のターン全体で留まります。この永続性はスキルの指示に適用され、その権限には適用されません。*allowed-tools* (ページ 298) 許可はあなたが次のメッセージを送信するときにクリアされます。Claude Code は後のターンでスキルファイルを再度読み込まないため、タスク全体に適用すべき指導を 1 回限りのステップではなく、スタンディング指示として記述します。

Claude が、レンダリングされたコンテンツがコンテキストに既にある複製と同じであるスキルを再呼び出すと、Claude Code はスキルが既に読み込まれていることを示す短いメモを追加します。コンテンツが異

なる場合（引数が変更されたか、**動的コンテキスト** (ページ 300) コマンドが新しい出力を生成したため）、Claude Code は完全なコンテンツを再度追加します。

自動コンパクション はトークン予算内で呼び出されたスキルを前方に実行します。会話が要約されてコンテキストを解放するとき、Claude Code は各スキルの最新の呼び出しを要約の後に再度アタッチし、最初の 5,000 トークンを保持します。再度アタッチされたスキルは 25,000 トークンの結合予算を共有します。Claude Code はこの予算を最近呼び出されたスキルから開始して埋めるため、セッション内で多くを呼び出した場合、古いスキルはコンパクション後に完全にドロップできます。

スキルが最初の応答の後に動作に影響を与えるのを停止しているように見える場合、コンテンツは通常まだ存在し、モデルは他のツールまたはアプローチを選択しています。スキルの `description` と指示を強化して、モデルがそれを優先し続けるようにするか、**フック** (ページ 443) を使用して動作を決定的に強制します。スキルが大きい場合、または他のスキルを呼び出した後、コンパクション後に再呼び出しして完全なコンテンツを復元します。

17.4.6 スキルのツールを事前承認する

`allowed-tools` フィールドは、スキルを呼び出すターン中にリストされたツールの権限を付与するため、Claude は承認を求めることなくそれらを使用できます。許可はあなたが次のメッセージを送信するときにクリアされます。スキルコンテンツは **コンテキストに留まる** (ページ 297) にもかかわらず。スキルを再呼び出すと、そのターンに対して再度適用されます。これはどのツールが利用可能かを制限しません。すべてのツールは呼び出し可能なままであり、**権限設定** (ページ 1039) はリストされていないツールを引き続き管理します。セッション全体ではなく単一のターンのツールを事前承認するには、代わりにそれらの権限設定に許可ルールを追加します。

ワークスペースの信頼はこのフィールドの適用を制限しません。Claude Code は、一度も信頼したことのないフォルダでの `-p` 実行であっても、プロジェクトスキルの `allowed-tools` を適用します。スキルは自身に広いツールアクセスを付与できるため、Claude Code をそこで実行する前に、リポジトリにチェックインされたスキルの `allowed-tools` を確認してください。組織全体でリポジトリのスキルからこのフィールドを無効にするには、**管理された権限ルールのみが適用される場合** (ページ 299) を参照してください。

このスキルでは、スキルを呼び出すときはいつでも、Claude は許可を求めることなく git コマンドを実行できます。

```
---
name: commit
description: Stage and commit the current changes
disable-model-invocation: true
allowed-tools: Bash(git add *) Bash(git commit *) Bash(git status *)
---
```

スキルがアクティブな間、Claude の利用可能なプールからツールを削除するには、スキルのフロントマターの `disallowed-tools` にそれらをリストします。制限はあなたが次のメッセージを送信するときにクリアされます。拒否ルールと同様に、フィールドは他のツールが残っている間、`EndConversation` を削除できません。すべてのスキルとプロンプト全体でツールをブロックするには、**権限設定** (ページ 1039) に拒否ルールを追加します。

管理された権限ルールのみが適用される場合

組織が管理設定で `allowManagedPermissionRulesOnly` を設定している場合、Claude Code はプロジェクトスキルと個人スキル、および この設定の項目に記載されているその他のソース の `allowed-tools` を無視します。これには Claude Code v2.1.282 以降が必要です。

影響を受けるスキルがリストするツールは、代わりに組織の管理ルールと通常の権限プロンプトを経由します。`/status` を実行すると、セッション内でこれまでに Claude Code が `allowed-tools` を無視した各スキルが一覧表示されます。どの管理ルールでも許可されていない、スキル内の注入されたコマンドは、**注入されたコマンドの権限チェック** (ページ 303) に従います。

17.4.7 スキルに引数を渡す

あなたと Claude の両方がスキルを呼び出すときに引数を渡すことができます。引数は `$ARGUMENTS` プレースホルダーを通じて利用可能です。

このスキルは GitHub の問題を番号で修正します。`$ARGUMENTS` プレースホルダーはスキル名の後に続くもので置き換えられます。

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---

Fix GitHub issue $ARGUMENTS following our coding standards.

1. Read the issue description
2. Understand the requirements
3. Implement the fix
4. Write tests
5. Create a commit
```

`/fix-issue 123` を実行すると、Claude は「Fix GitHub issue 123 following our coding standards...」を受け取ります。

スキルに引数を指定して呼び出しても、スキルのコンテンツのプレースホルダーが 1 つも受け取らない場合、Claude Code は `ARGUMENTS: <your input>` をスキルコンテンツの最後に追加するため、Claude は依然として入力したものを見ます。プレースホルダーは `$ARGUMENTS`、`$1` などのインデックス形式、または名前付き引数です。対応する引数がないインデックス付きプレースホルダーはリテラルテキストのままで、1 つを受け取ったとしてカウントされません。名前付きプレースホルダーは、位置に引数がない場合でも、空の文字列に展開されるため、カウントされます。

1 つのメッセージの先頭に複数のスキルをスタックすることもできます。`/write-tests /fix-issue 123` と入力すると、両方のスキルが読み込まれ、末尾のテキスト `123` が `$ARGUMENTS` として各スキルに渡されます。

Claude Code は最初のスキルと、その後にスタックされた最大 5 つのスキルを展開します。展開は、インラインユーザー呼び出し可能スキルではない最初のトークンで停止するため、**フォークされたサブエージェント** (ページ 303) として実行されるスキル（`/code-review` など）、またはその引数自体がスラッシュコマ

ンドで始まる可能性があるスキル（/loop など）も、そこで実行を終了します。そのトークンとその後のすべてが、展開されたすべてのスキルの引数テキストになります。v2.1.218 から /code-review はフォークされたサブエージェントとして実行されます。以前のバージョンではインラインで実行され、スタックされました。

位置で個別の引数にアクセスするには、$ARGUMENTS[N] または短い $N を使用します。

```
---
name: migrate-component
description: Migrate a component from one language to another
---

Migrate the $ARGUMENTS[0] component from $ARGUMENTS[1] to $ARGUMENTS[2].
Preserve all existing behavior and tests.
```

/migrate-component SearchBar JavaScript TypeScript を実行すると、$ARGUMENTS[0] が SearchBar に、$ARGUMENTS[1] が JavaScript に、$ARGUMENTS[2] が TypeScript に置き換えられます。$N 短縮形を使用する同じスキル。

```
---
name: migrate-component
description: Migrate a component from one language to another
---

Migrate the $0 component from $1 to $2.
Preserve all existing behavior and tests.
```

17.5 高度なパターン

17.5.1 動的コンテキストを注入する

!`<command>` 構文は、スキルコンテンツが Claude に送信される前にシェルコマンドを実行します。コマンド出力がプレースホルダーを置き換えるため、Claude はコマンド自体ではなく実際のデータを受け取ります。Claude Code は、スキルが *claude.ai* **アカウントから同期される** (ページ 286) 場合、マシン上でこれらのコマンドを実行しません。この制限には Claude Code v2.1.228 以降が必要です。

このスキルは GitHub CLI を使用してライブ PR データを取得することで、プルリクエストを要約します。!`gh pr diff` およびその他のコマンドが最初に実行され、その出力がプロンプトに挿入されます。

```
---
name: pr-summary
description: Summarize changes in a pull request
context: fork
agent: Explore
allowed-tools: Bash(gh *)
---
```

(次のページに続く)

(前のページからの続き)

```
## Pull request context
- PR diff: !`gh pr diff`
- PR comments: !`gh pr view --comments`
- Changed files: !`gh pr diff --name-only`

## Your task
Summarize this pull request...
```

置換は元のファイルに対して 1 回実行されます。コマンド出力はプレーンテキストとして挿入され、さらに !`<command>` プレースホルダーについて再スキャンされないため、コマンドは後の処理で展開するプレースホルダーを出力することはできません。

インライン形式は、! が行の開始時または空白の直後に表示される場合にのみ認識されます。`KEY=!`cmd`` のように ! が別の文字の後に続く場合、プレースホルダーはリテラルテキストとして残され、コマンドは実行されません。

複数行のコマンドの場合、インライン形式の代わりに ```! で開かれたフェンスコードブロックを使用します。

````
## Environment
```!
node --version
git status --short
```
````

ユーザー、プロジェクト、プラグイン、または *additional-directory* (ページ 281) ソースからのスキルおよびカスタムコマンドについてこの動作を無効にするには、*settings* (ページ 711) で `"disableSkillShellExecution": true` を設定します。各コマンドは実行される代わりに `[shell command execution disabled by policy]` に置き換えられます。バンドルされたスキルと管理されたスキルは影響を受けません。この設定は managed settings で最も有用です。ここではユーザーはそれをオーバーライドできません。

Claude Code は、*claude.ai* **アカウントから同期されたスキル** (ページ 286) に表示されるコマンドをマシン上で実行することはありません。この制限には Claude Code v2.1.228 以降が必要です。*Claude Code* **がどのように同期されたスキルの本体を処理するか** (ページ 286) は、各種セッションで Claude が受け取るコマンドの代わりになるものを説明しています。

Tip:

スキルが実行されるときにより深い推論をリクエストするには、スキルコンテンツの任意の場所に `ultrathink` を含めます。*1* **回限りの深い推論に** *ultrathink* **を使用する** (ページ 750) を参照してください。

注入されたコマンドがどのように実行されるか

Claude Code は、スキルのフロントマターの `shell` キーと環境からスキルの注入されたコマンドを実行するツールを選択します。すべての組み合わせはコマンドを Bash ツールまたは PowerShell ツールを通じて実行します。ただし、呼び出しを完全に失敗させる 1 つの組み合わせを除きます。

- `shell: powershell`、PowerShell ツール が有効な場合：コマンドは PowerShell ツールを通じて実行されます。
- `shell: bash` で bash が利用できない場合：コマンドが実行される前に呼び出しが失敗します。これは Git Bash のない Windows で発生します。Claude Code は `Skill <name> requires bash (`shell: bash` in frontmatter) but Git Bash was not found` を表示します。
- その他の組み合わせ：bash が利用可能な場合、コマンドは Bash ツールを通じて実行されます。利用できない場合、PowerShell ツールを通じて実行されます。

どちらのツールも、Claude 独自のシェルコマンドを実行するのと同じ方法でコマンドを実行します。これらは作業ディレクトリ、タイムアウト、および出力処理を共有します。

- **作業ディレクトリ**：Claude Code は各コマンドをセッションシェルの現在の作業ディレクトリで実行します。Claude が `cd` を実行するとそのディレクトリが移動します。毎回同じ方法で解決する必要があるパスで *${CLAUDE_SKILL_DIR}* **または** *${CLAUDE_PROJECT_DIR}* (ページ 294) を使用します。
- **stderr**：デフォルトの `bash` シェルでは、Claude Code は stderr を stdout にマージします。コマンドが stderr に書き込むすべてのものが注入されたテキストに表示されます。
- **タイムアウト**：各コマンドは Bash ツールのデフォルト 2 分 timeout の下で実行されます。Bash ツールが タイムアウトしたコマンドをバックグラウンドに移動 する場合、スキルは引き続きレンダリングされます。注入されたテキストは移動を報告し、バックグラウンドタスクとコマンドの出力を収集するファイルに名前を付けます。コマンドが Bash ツールが自動的にバックグラウンドに移動しないコマンドの場合、Claude Code はタイムアウト時にそれを強制終了します。その失敗は **呼び出しを中止します** (ページ 302)。
- **出力サイズ**：Bash ツールのインライン上限を超える出力は、切り詰められたテキストではなく、ファイルパスと短いプレビューとして到着します。出力制限 は上限とそれぞれの境界を調整する方法をカバーしています。

PowerShell ツールは、実行するコマンドに同じタイムアウト、バックグラウンド処理、および出力上限の動作を適用します。その詳細については、PowerShell ツール セクションを参照してください。

注入されたコマンドが失敗する場合

失敗したコマンドは、スキル全体の呼び出しを中止します。独自のプレースホルダーだけではありません。Claude はその呼び出しのスキルコンテンツを見ることはありません。中止は `Shell command failed for pattern "..."` を表示します。エラーメッセージには `[stderr]` の下のコマンドの出力が含まれます。

デフォルトの `bash` シェルでは、ゼロ以外の終了コードはすべて失敗としてカウントされます。1 つの例外が適用されます。Claude Code は 検索および比較コマンド からの終了コード 1 を通常の結果として扱い、その出力を注入します。終了コード 2 以上はこれらのコマンドでも失敗します。

どのコマンドが例外を取得するかはシェルによって異なります。

- デフォルト `bash` シェル：出力制限 の下にリストされているコマンド

- `shell: powershell`、PowerShell ツールが有効な場合：`grep` と `git diff` を含むが `find` や `diff` は含まない 異なるセット

デフォルトの `bash` シェルでは、ゼロ以外で終了することが予想される他のコマンドに `|| true` を追加します。問題を見つけたときに 1 で終了するチェックスクリプトは 1 つの例です。

注入されたコマンドの権限チェック

注入されたコマンドは、スキルのレンダリング中に権限を求めるプロンプトを表示することはありません。Claude Code は最初に **権限ルール** (ページ 1039) に対して各コマンドをチェックします。deny ルールが一致するコマンドは `Shell command permission check failed for pattern "..."` で呼び出しを中止します。

auto モード 以外では、コマンドの権限チェックが allow 以外を返す場合、Claude Code は同じエラーで呼び出しを中止します。これには通常はユーザーに確認を求めるルールも含まれます。一致しないコマンドがここで中止されるのを防ぐには、*`allowed-tools`* (ページ 298) で事前に承認します。組織が権限ルールを管理設定に制限している場合は、**管理された権限ルールのみが適用される場合** (ページ 299) を参照してください。拒否ルールと確認ルールは引き続き `allowed-tools` より優先されます。**権限を管理する** (ページ 1041) を参照してください。

auto mode では、そうでなければあなたの承認が必要なコマンドは呼び出しを中止しません。スキルは Claude にコマンドを最初に実行するよう指示して読み込まれ、Claude 独自の呼び出しは auto mode の通常のチェック を通じて進みます。呼び出しは、`agent` を設定する **フォークされたスキル** (ページ 303) でも、Claude が **注入されたコマンドを実行するシェルツール** (ページ 302) を持たないセッションでも中止されます。

17.5.2 スキルをサブエージェントで実行する

スキルを分離して実行したい場合は、フロントマターに `context: fork` を追加します。Claude Code は `agent` フィールドで設定されたタイプの新しいサブエージェントを開始し、スキルコンテンツをそのプロンプトとして提供します。サブエージェントは会話履歴を見ないため、スキルの指示は独立して機能する必要があります。

注釈:

名前にもかかわらず、`context: fork` を持つスキルは **現在の会話のフォーク** (ページ 354) では実行されません。これはサブエージェントにこれまで議論したすべてを渡すでしょう。タスクがその履歴に依存する場合、`context: fork` を使用する代わりに会話をフォークします。

フォークされたサブエージェントは **バックグラウンド** (ページ 344) で実行されます。スキルが実行されている間、あなたは作業を続けることができ、その結果は完了時に会話に到着します。フロントマターで `background: false` を設定して、代わりにスキルを呼び出した順番で結果を待ちます。v2.1.218 より前では、フォークされたスキルは常に完了するまでターンをブロックしていました。

Claude Code は、スキルが `background: false` を設定しない場合でも、次のような場合に結果を待ちます。

- `-p` フラグまたは Agent SDK を使用した非対話モード
- `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` を 1 に設定した場合。これはすべての他のバックグラウンドタスク機能もオフにします

- 同じスキルの以前の呼び出しがまだ実行中の間にフォークされたスキルを呼び出す場合
- スケジュールされたタスク がスキルをそのプロンプトとして発火する場合

バックグラウンドで実行されるフォークされたスキルは、**バックグラウンドサブエージェントに適用される狭いツールセット** (ページ 344) で編集を適用します。スキルのサブエージェントは通常のエージェントタイプなので、会話をフォークするサブエージェントの例外はそれをカバーしません。スキルのステップがそのセット外のツールに依存する場合、`background: false` を設定して完全なツールセットを保持します。

バックグラウンドで実行されるフォークされたスキルは、セッションの *checkpoints* (ページ 1019) の外で編集を適用するため、`/rewind` はそれらを元に戻しません。git を使用してそれらを元に戻します。

警告:

`context: fork` は明示的な指示を持つスキルにのみ意味があります。スキルに「これらの API 規約を使用する」などのガイドラインが含まれていて、タスクがない場合、サブエージェントはガイドラインを受け取りますが、実行可能なプロンプトはなく、意味のある出力なしで返されます。

スキルと **サブエージェント** (ページ 317) は 2 つの方向で連携します。

アプローチ	システムプロンプト	タスク	また読み込む
`context: fork` を持つスキル	エージェントタイプから	SKILL.md コンテンツ	CLAUDE.md、エージェントの *startup context* (ページ 350) に従う
`skills` フィールドを持つサブエージェント	サブエージェントのマークダウン本体	Claude の委任メッセージ	事前読み込みされたスキル + CLAUDE.md、サブエージェントの *startup context* (ページ 350) に従う

`context: fork` では、スキルにタスクを記述し、それを実行するエージェントタイプを選択します。組み込みの Explore および Plan エージェントは *CLAUDE.md* と *git status* **をスキップ** (ページ 350) して、コンテキストを小さく保ちます。そのため、`agent: Explore` を使用するフォークされたスキルは SKILL.md コンテンツとエージェント独自のシステムプロンプトのみを見ます。逆に、参照資料としてスキルを使用するカスタムサブエージェントを定義する場合は、**サブエージェント** (ページ 337) を参照してください。

例：Explore エージェントを使用した研究スキル

このスキルはフォークされた Explore エージェントで研究を実行します。スキルコンテンツはタスクになり、エージェントはコードベース探索に最適化された読み取り専用ツールを提供します。

```
---
name: deep-research
description: Research a topic thoroughly
context: fork
agent: Explore
---
```

(次のページに続く)

(前のページからの続き)

```
Research $ARGUMENTS thoroughly:

1. Find relevant files using Glob and Grep
2. Read and analyze the code
3. Summarize findings with specific file references
```

このスキルが実行される場合：

1. 新しい分離されたコンテキストが作成されます
2. サブエージェントはスキルコンテンツをそのプロンプトとして受け取ります（「Research $ARGUMENTS thoroughly」指示）
3. agent フィールドは実行環境（モデル、ツール、および権限）を決定します
4. サブエージェントはその結果を要約し、完了時にメイン会話に返します

agent フィールドは使用するサブエージェント構成を指定します。オプションには組み込みエージェント（Explore、Plan、general-purpose）または .claude/agents/ からのカスタムサブエージェントが含まれます。省略された場合、general-purpose を使用します。

17.5.3 Claude のスキルアクセスを制限する

デフォルトでは、Claude は disable-model-invocation: true が設定されていない任意のスキルを呼び出すことができます。*allowed-tools* (ページ 298) を定義するスキルは、そのスキルを呼び出すターンの間、使用ごとの承認なしでそれらのツールへのアクセスを Claude に付与します。この付与は次のメッセージを送信するとクリアされます。**権限設定** (ページ 1039) は引き続き、その他のすべてのツールの基本的な承認動作を管理します。/init や /security-review を含むいくつかの組み込みコマンドも Skill ツールを通じて利用できます。/compact などのその他の組み込みコマンドは利用できません。

Claude が呼び出すことができるスキルを制御する 3 つの方法：

すべてのスキルを無効にする には、/permissions で Skill ツールを deny します。

```
# Add to deny rules:
Skill
```

特定のスキルを許可または拒否する には *permission rules* (ページ 1039) を使用します。

```
# Allow only specific skills
Skill(commit)
Skill(review-pr *)

# Deny specific skills
Skill(deploy *)
```

権限構文：正確な一致の場合は Skill(name)、任意の引数を持つプレフィックス一致の場合は Skill(name *)。

テーブルは、あなたが書く名前の種類によって、deny ルールが名前を超えてブロックするものを示しています。

あなたの deny ルール名	ルール例	Claude Code もブロック
エイリアス	`Skill(review)`	バンドルされた `/code-review`、その `/review` エイリアスを通じて
修飾されていない名前	`Skill(deploy)`	**ネストされたスキル** (ページ 279) を `apps/web:deploy` として
claude.ai **から同期されたスキル** (ページ 283)	`Skill(anthropic-skills:deploy)`	Claude Desktop がそれをセッションにプラグインとして配信するとき
同期されたスキルのプラグイン形式	`Skill(deploy:deploy)`	同期されたスキル
parameter form (ページ 1043) のスキル	`Skill(skill:deploy)`	スキルが呼び出される名前のいずれかで、そのエイリアスと表示名を含む

v2.1.260 より前では、Claude Code は deny ルールが修飾されていない名前のみを指定する場合、修飾された名前の下にリストされたネストされたスキルをブロックしませんでした。

Claude Code は `allow` ルールをスキル独自の名前と Claude の呼び出しの名前に対してのみ一致させます。*synced skill* (ページ 283) を事前承認なしで承認するには、その *reserved namespace* (ページ 285) 内の名前を付けます。

- `Skill(anthropic-skills:pdf)` は synced `pdf` スキルを承認します
- `Skill(anthropic-skills *)` はすべての synced スキルを承認します
- `Skill(anthropic *)` は `anthropic-skills:pdf` をカバーしません。ネームスペース外のプレフィックスはその中の名前と一致しないため

個別のスキルを非表示にする には、フロントマターに `disable-model-invocation: true` を追加します。これはスキルを Claude のコンテキストから完全に削除します。

注釈:

`user-invocable: false` では、スキルを呼び出すことはできませんが、Claude はできます。Claude が Skill ツールを通じてそれを呼び出すのを防ぐには、`disable-model-invocation: true` を設定します。

17.5.4 設定からスキルの可視性をオーバーライドする

`skillOverrides` 設定は、スキル独自のフロントマターの代わりに *settings* (ページ 711) からスキルの可視性を制御します。SKILL.md を編集したくないスキル（共有プロジェクトリポにチェックインされたものなど）に使用します。`/skills` メニューはあなたのために書き込みます。スキルをハイライトして `Space` を押して状態をサイクルし、`Esc` を押して `.claude/settings.local.json` に保存します。

各キーはスキル名で、各値は 4 つの状態の 1 つです。

値	Claude にリストされている	/ メニューで
"on"	名前と説明	はい
"name-only"	名前のみ	はい
"user-invocable-only"	非表示	はい
"off"	非表示	非表示

/skills メニューは "user-invocable-only" 状態を user-only とラベル付けします。

"off" は、ターミナルの / メニューに加えて、Remote Control クライアントと Agent SDK 呼び出し元に公開されるコマンドリストからもスキルを非表示にします。非表示のスキルを完全な名前で呼び出すと、実行される代わりに skillOverrides エラーが返されます。

skillOverrides に存在しないスキルは "on" として扱われます。以下の例は 1 つのスキルをその名前に折りたたみ、別のスキルを完全にオフにします。

```
{
  "skillOverrides": {
    "legacy-context": "name-only",
    "deploy": "off"
  }
}
```

一部のバンドルされたスキルには、/doctor の checkup などのエイリアスがあります。managed settings または --settings フラグで渡すファイルでエイリアスの下に skillOverrides エントリを設定する場合、Claude Code はそれをエイリアスの背後にあるスキルに適用します。エイリアスを通じてスキルをさらに制限することのみが可能で、より可視化することはできません。また、managed settings でスキル独自の名前の下にエントリも設定する場合、そのエントリが優先されます。v2.1.260 より前では、Claude Code はいかなる設定ソースでもエイリアスの下のエントリをスキルに適用しませんでした。

ユーザー、プロジェクト、およびローカル設定では、Claude Code はスキル名に対してのみエントリを一致させます。そこで review のエントリを設定する場合、それは review という名前のスキルに適用され、バンドルされた /code-review にはその /review エイリアスを通じて適用されません。

プラグインスキルは skillOverrides の影響を受けません。/plugin を通じてそれらを管理します。

17.5.5 未使用のスキルを見つける

スキルリスト (ページ 315) のすべてのスキルは、Claude がそれを使用するかどうかに関係なく、すべてのターンでコンテキストに追加されます。/skill-doctor を実行して、各スキルのコストと使用頻度を確認し、どのスキルをオフにするかを決定します。対話型セッションでは、レポートは /plugin マネージャーの **Stats** タブで開きます。**非対話モード** (ページ 695) で -p を使用する場合、Claude Code はテキストとして出力します。

レポートはバンドルされたスキルとエンタープライズスキル以外のセッション内のスキルをカバーします。リストに表示されたスキルのうち、呼び出されたことのないスキルにフラグを付け、どこでそれらをオフにするかを示します。オフにする場所を示すスキルのうち、最も高いコンテキストコストを持つものから始めます。レポートは最近使用していないプラグインもリストします。

/skill-doctor には Claude Code v2.1.252 以降が必要で、feature-flag fetching をスキップするセッションでは利用できません。Remote Control から電話またはブラウザで /skill-doctor を実行する場合、Claude Code は Skill usage reports are not available on this connection. で返信します。セッションが実行されているマシンのターミナルで /skill-doctor を実行します。

17.6 スキルを評価して反復する

スキルがトリガーされたことを確認することは、Claude がそれを見つけたことを意味しますが、意図した動作をしたことを意味しません。スキルが機能していることを知るには、Claude がそれを呼び出すべきプロンプトで実際に呼び出すかどうか、および呼び出す場合に出力が期待と一致するかどうかを個別に測定する必要があります。

両方をチェックするには、ベースライン比較を行います。現実的なプロンプトをいくつか収集し、スキルが利用可能な新しいセッションで各プロンプトを実行し、スキルをオフにした状態でも実行して、結果を比較します。新しいセッションが重要なのは、スキルの作成時に残されたコンテキストが、書かれた指示のギャップをマスクするためです。

2 番目の実行でスキルをオフにする方法は、スキルの出所によって異なります。

- **個人用またはプロジェクトスキル**：*skillOverrides* (ページ 306) で "off" に設定します。
- **プラグインが提供するスキル**：skillOverrides はプラグインスキルには適用されません。代わりに claude plugin eval を使用します。これはプラグインが読み込まれていない状態で各実行を繰り返します。

ベースライン比較を自動化する 2 つのツールがあります。プラグイン で配布されるスキルの場合、claude plugin eval は各プロンプトを分離されたセッションでプラグインの有無で実行し、定義したグレーダーまたはそれが作成したグレーダーでスコアリングし、閾値以下の場合はゼロ以外で終了するため、CI でそれをゲートできます。Claude Code 会話内の単一スキルを反復する場合、以下のスキル作成者プラグインは独自の evals/evals.json 形式で同様のループを実行します。2 つの形式は相互交換可能ではありません。

17.6.1 skill-creator でエバルを実行する

skill-creator プラグイン は Claude Code 内の比較ループを自動化します。VS Code 拡張機能またはデスクトップアプリでは、プラグインをインストールする の手順に従って公式マーケットプレイスからインストールします。ターミナルでは、claude を実行して Claude Code を起動し、そのプロンプトで次のように入力します。

```
/plugin install skill-creator@claude-plugins-official
```

インストールが失敗した場合は、Claude Code が報告するメッセージと一致させます。

- Marketplace "claude-plugins-official" not found：/plugin marketplace add anthropics/claude-plugins-official でマーケットプレイスを追加してから、インストールを再試行します。
- プラグインがマーケットプレイスで見つかりません：プラグイン名を確認します。

インストール概要が Run /reload-plugins to activate. を報告する場合、Claude Code はそのリロードを実行します。リロードが次のメッセージが会話を再読み込みすることを警告する場合は、/reload-plugins

`--force` を実行してプラグインのスキルを現在のセッションで利用可能にします。その後、Claude に既存のスキルを評価するよう依頼します。例えば `evaluate my summarize-changes skill with skill-creator` です。プラグインはテストケースの作成をガイドし、ループを実行します。

- **テストケース**：プロンプト、入力ファイル、および期待される動作をスキルディレクトリ内の `evals/evals.json` に保存します
- **分離された実行**：テストケースごとに **サブエージェント** (ページ 317) を生成して、各実行がクリーンなコンテキストで開始され、トークン数と期間を記録します
- **グレーディング**：各アサーションを出力に対してチェックし、`grading.json` に証拠とともに合格または不合格を書き込みます
- **ベンチマーク**：スキルあり対スキルなしの合格率、時間、トークンを `benchmark.json` に集約して、トークンと時間のオーバーヘッドに対する合格率の改善を比較できます
- **バージョン比較**：スキルの 2 つのバージョン間でブラインド A/B テストを実行して、編集をコミットする前に改善であることを確認できます
- **説明チューニング**：トリガーすべきおよびトリガーすべきでないプロンプトを生成し、ヒット率を測定し、スキルが間違ったリクエストで起動する場合は説明編集を提案します
- **レビュービューアー**：各出力を検査し、次の反復が読み込む定性的フィードバックを記録できる HTML レポートを開きます

エバルファイル形式と完全な反復ワークフローについては、agentskills.io の スキル出力品質の評価 を参照してください。ベンチマークと比較モードの背景については、skill-creator アナウンスメント を参照してください。

17.7 スキルを共有する

スキルはオーディエンスに応じて異なるスコープで配布できます。

- **プロジェクトスキル**: `.claude/skills/` をバージョン管理にコミットする
- **プラグイン**: プラグインに `skills/` ディレクトリを作成する
- **マネージド**: マネージド設定を通じて組織全体にデプロイする

17.7.1 ビジュアル出力を生成する

スキルは任意の言語でスクリプトをバンドルして実行でき、Claude に単一のプロンプトでは不可能な機能を提供します。1 つのパターンはビジュアル出力を生成することです。つまり、ブラウザで開いてデータを探索したり、デバッグしたり、レポートを作成したりするためのインタラクティブな HTML ファイルです。

この例は、コードベースエクスプローラーを作成します。これはインタラクティブなツリービューで、ディレクトリを展開・折りたたんだり、ファイルサイズを一目で確認したり、ファイルタイプを色で識別したりできます。

スキルディレクトリを作成します。

```
mkdir -p ~/.claude/skills/codebase-visualizer/scripts
```

これを ~/.claude/skills/codebase-visualizer/SKILL.md に保存します。説明は Claude にこのスキルをいつ有効化するかを伝え、指示は Claude にバンドルされたスクリプトを実行するよう伝えます。スクリプトパスは *${CLAUDE_SKILL_DIR}* (ページ 294) を使用するため、スキルが個人、プロジェクト、またはプラグインレベルでインストールされているかどうかに関わらず正しく解決されます。

```
---
name: codebase-visualizer
description: Generate an interactive collapsible tree visualization of your codebase.
 Use when exploring a new repo, understanding project structure, or identifying large␣
↪files.
allowed-tools: Bash(python3 *)
---

# Codebase Visualizer

Generate an interactive HTML tree view that shows your project's file structure with␣
↪collapsible directories.

## Usage

Run the visualization script from your project root:

```bash
python3 ${CLAUDE_SKILL_DIR}/scripts/visualize.py .
```

This creates `codebase-map.html` in the current directory and opens it in your␣
↪default browser.

What the visualization shows

- **Collapsible directories**: Click folders to expand/collapse
- **File sizes**: Displayed next to each file
- **Colors**: Different colors for different file types
- **Directory totals**: Shows aggregate size of each folder
```

これを ~/.claude/skills/codebase-visualizer/scripts/visualize.py に保存します。このスクリプトはディレクトリツリーをスキャンし、以下を含む自己完結型の HTML ファイルを生成します。

- ファイル数、ディレクトリ数、合計サイズ、ファイルタイプ数を表示する**サマリーサイドバー**
- コードベースをファイルタイプ別（サイズ上位 8 つ）に分類する**棒グラフ**
- ディレクトリを展開・折りたたんだり、色分けされたファイルタイプインジケーターを表示したりできる**折りたたみ可能なツリー**

スクリプトは Python 3 を必要としますが、組み込みライブラリのみを使用するため、インストールするパッケージはありません。

```
#!/usr/bin/env python3
"""Generate an interactive collapsible tree visualization of a codebase."""

import json
import sys
import webbrowser
from html import escape
from pathlib import Path
from collections import Counter

IGNORE = {'.git', 'node_modules', '__pycache__', '.venv', 'venv', 'dist', 'build'}

def scan(path: Path, stats: dict) -> dict:
    result = {"name": path.name, "children": [], "size": 0}
    try:
        for item in sorted(path.iterdir()):
            if item.name in IGNORE or item.name.startswith('.'):
                continue
            if item.is_file():
                size = item.stat().st_size
                ext = item.suffix.lower() or '(no ext)'
                result["children"].append({"name": item.name, "size": size, "ext":␣
↪ext})
                result["size"] += size
                stats["files"] += 1
                stats["extensions"][ext] += 1
                stats["ext_sizes"][ext] += size
            elif item.is_dir():
                stats["dirs"] += 1
                child = scan(item, stats)
                if child["children"]:
                    result["children"].append(child)
                    result["size"] += child["size"]
    except PermissionError:
        pass
    return result

def generate_html(data: dict, stats: dict, output: Path) -> None:
    ext_sizes = stats["ext_sizes"]
    total_size = sum(ext_sizes.values()) or 1
    sorted_exts = sorted(ext_sizes.items(), key=lambda x: -x[1])[:8]
    colors = {
        '.js': '#f7df1e', '.ts': '#3178c6', '.py': '#3776ab', '.go': '#00add8',
        '.rs': '#dea584', '.rb': '#cc342d', '.css': '#264de4', '.html': '#e34c26',
        '.json': '#6b7280', '.md': '#083fa1', '.yaml': '#cb171e', '.yml': '#cb171e',
```

(次のページに続く)

(前のページからの続き)

```
        '.mdx': '#083fa1', '.tsx': '#3178c6', '.jsx': '#61dafb', '.sh': '#4eaa25',
    }
    lang_bars = "".join(
        f'<div class="bar-row"><span class="bar-label">{ext}</span>'
        f'<div class="bar" style="width:{(size/total_size)*100}%;background:{colors.
↪get(ext,"#6b7280")}"></div>'
        f'<span class="bar-pct">{(size/total_size)*100:.1f}%</span></div>'
        for ext, size in sorted_exts
    )
    def fmt(b):
        if b < 1024: return f"{b} B"
        if b < 1048576: return f"{b/1024:.1f} KB"
        return f"{b/1048576:.1f} MB"

    html = f'''<!DOCTYPE html>
<html><head>
  <meta charset="utf-8"><title>Codebase Explorer</title>
  <style>
    body {{ font: 14px/1.5 system-ui, sans-serif; margin: 0; background: #1a1a2e;␣
↪color: #eee; }}
    .container {{ display: flex; height: 100vh; }}
    .sidebar {{ width: 280px; background: #252542; padding: 20px; border-right: 1px␣
↪solid #3d3d5c; overflow-y: auto; flex-shrink: 0; }}
    .main {{ flex: 1; padding: 20px; overflow-y: auto; }}
    h1 {{ margin: 0 0 10px 0; font-size: 18px; }}
    h2 {{ margin: 20px 0 10px 0; font-size: 14px; color: #888; text-transform:␣
↪uppercase; }}
    .stat {{ display: flex; justify-content: space-between; padding: 8px 0; border-
↪bottom: 1px solid #3d3d5c; }}
    .stat-value {{ font-weight: bold; }}
    .bar-row {{ display: flex; align-items: center; margin: 6px 0; }}
    .bar-label {{ width: 55px; font-size: 12px; color: #aaa; }}
    .bar {{ height: 18px; border-radius: 3px; }}
    .bar-pct {{ margin-left: 8px; font-size: 12px; color: #666; }}
    .tree {{ list-style: none; padding-left: 20px; }}
    details {{ cursor: pointer; }}
    summary {{ padding: 4px 8px; border-radius: 4px; }}
    summary:hover {{ background: #2d2d44; }}
    .folder {{ color: #ffd700; }}
    .file {{ display: flex; align-items: center; padding: 4px 8px; border-radius: 4px;
 }}
    .file:hover {{ background: #2d2d44; }}
    .size {{ color: #888; margin-left: auto; font-size: 12px; }}
    .dot {{ width: 8px; height: 8px; border-radius: 50%; margin-right: 8px; }}
  </style>
```

(次のページに続く)

(前のページからの続き)

```
</head><body>
  <div class="container">
    <div class="sidebar">
      <h1>□ Summary</h1>
      <div class="stat"><span>Files</span><span class="stat-value">{stats["files"]:,}
↪</span></div>
      <div class="stat"><span>Directories</span><span class="stat-value">{stats["dirs
↪"]:,}</span></div>
      <div class="stat"><span>Total size</span><span class="stat-value">{fmt(data[
↪"size"])}</span></div>
      <div class="stat"><span>File types</span><span class="stat-value">{len(stats[
↪"extensions"])}</span></div>
      <h2>By file type</h2>
      {lang_bars}
    </div>
    <div class="main">
      <h1>□ {escape(data["name"])}</h1>
      <ul class="tree" id="root"></ul>
    </div>
  </div>
  <script>
    const data = {json.dumps(data)};
    const colors = {json.dumps(colors)};
    function fmt(b) {{ if (b < 1024) return b + ' B'; if (b < 1048576) return (b/
↪1024).toFixed(1) + ' KB'; return (b/1048576).toFixed(1) + ' MB'; }}
    function esc(s) {{ return s.replace(/[&<>"']/g, c => ({{"&":"&amp;","<":"&lt;",">
↪":"&gt;",'"':"&quot;","'":"&#39;"}}[c])); }}
    function render(node, parent) {{
      if (node.children) {{
        const det = document.createElement('details');
        det.open = parent === document.getElementById('root');
        det.innerHTML = `<summary><span class="folder">□ ${{esc(node.name)}}</span>
↪<span class="size">${{fmt(node.size)}}</span></summary>`;
        const ul = document.createElement('ul'); ul.className = 'tree';
        node.children.sort((a,b) => (b.children?1:0)-(a.children?1:0) || a.name.
↪localeCompare(b.name));
        node.children.forEach(c => render(c, ul));
        det.appendChild(ul);
        const li = document.createElement('li'); li.appendChild(det); parent.
↪appendChild(li);
      }} else {{
        const li = document.createElement('li'); li.className = 'file';
        li.innerHTML = `<span class="dot" style="background:${{colors[node.ext]||'
↪#6b7280'}}"></span>${{esc(node.name)}}<span class="size">${{fmt(node.size)}}</span>
↪`;
```

(次のページに続く)

(前のページからの続き)

```
        parent.appendChild(li);
      }}
    }}
    data.children.forEach(c => render(c, document.getElementById('root')));
  </script>
</body></html>'''
    output.write_text(html)

if __name__ == '__main__':
    target = Path(sys.argv[1] if len(sys.argv) > 1 else '.').resolve()
    stats = {"files": 0, "dirs": 0, "extensions": Counter(), "ext_sizes": Counter()}
    data = scan(target, stats)
    out = Path('codebase-map.html')
    generate_html(data, stats, out)
    print(f'Generated {out.absolute()}')
    webbrowser.open(f'file://{out.absolute()}')
```

テストするには、任意のプロジェクトで Claude Code を開き、「Visualize this codebase.」と尋ねます。Claude はスクリプトを実行し、生成されたファイルのパス（例：`Generated /path/to/codebase-map.html`）を出力し、ブラウザで開きます。ブラウザが開かないヘッドレス環境で作業している場合、出力されたパスはスクリプトが成功したことを確認します。

このパターンはあらゆるビジュアル出力に対応します。依存関係グラフ、テストカバレッジレポート、API ドキュメント、またはデータベーススキーマの可視化です。バンドルされたスクリプトが作業を行い、Claude がオーケストレーションを処理します。

17.8 トラブルシューティング

17.8.1 スキルがトリガーされない

Claude がスキルを期待通りに使用しない場合：

1. 説明にユーザーが自然に言うキーワードが含まれているか確認してください
2. スキルが「利用可能なスキルは何ですか？」に表示されているか確認してください
3. 説明により密接に一致するようにリクエストを言い換えてみてください
4. スキルがユーザー呼び出し可能な場合は、`/skill-name` で直接呼び出してください

frontmatter YAML が不正な形式の場合、Claude Code はスキル本体を空のメタデータで読み込むため、`/skill-name` は機能しますが Claude は `description` と照合できません。`--debug` で実行してパースエラーを確認してください。

スキルがプラグインに含まれている場合、1 つずつ確認するのではなく、現実的なプロンプト全体でスキルがどのくらいの頻度でトリガーされるかを測定できます。tool_used: Skill グレーダーを使用して eval ケースを作成し、説明を変更するたびに `claude plugin eval` で実行してください。

frontmatter がパースされない SKILL.md ファイルを見つけるには、スキルディレクトリで claude plugin validate を実行してください。例えば、プロジェクトスキルの場合は claude plugin validate .claude/skills、個人スキルの場合は claude plugin validate ~/.claude/skills です。Claude Code v2.1.233 以降が必要です。

17.8.2 スキルが頻繁にトリガーされる

Claude がスキルを不要な時に使用する場合：

1. 説明をより具体的にしてください
2. 手動呼び出しのみが必要な場合は disable-model-invocation: true を追加してください

17.8.3 Claude がスキルに従わなくなる

Claude が最初の応答でスキルに従い、その後に従わなくなった場合は、以下のいずれかのケースに該当するもので始めてください：

- **Claude が毎回成立する必要があるルールをスキップした**：ルールを *hook* (ページ 411) に移動してください。Claude Code は hook をそのイベントが発生するたびに実行します。例えば、ファイル編集の前など、Claude がスキルに従っているかどうかに関わらず実行されます。ルールをスキルと一緒に保つには、スキルの *hooks frontmatter* (ページ 465) で hook を定義してください。その hook はスキルが呼び出された時点からセッション終了まで適用されます。
- **Claude が判断を伴って適用すべきガイダンスをスキップした**：ガイダンスが全体的なタスクに適用されるように言い換えてください。例えば「すべての編集後にテストを実行する」ではなく「テストを実行する」ではなく、「すべての編集後にテストを実行する」と言ってください。Claude Code はスキルが呼び出された時点で会話にスキルのコンテンツを追加し、**後の段階ではファイルを再読み込みしません** (ページ 297)。
- **会話がコンパクト化された**：スキルを再度呼び出してコンテンツを復元してください。コンパクト化後、Claude Code は **呼び出されたスキルの開始部分のみを保持できます** (ページ 297)。そのため、最も重要な指示を SKILL.md の上部に配置してください。

17.8.4 スキルの説明が短くカットされている

Claude Code はスキル名と説明のリストをコンテキストに読み込み、Claude が利用可能なものを知ることができるようにします。リストには常にすべてのスキル名が含まれていますが、スキルが多い場合、Claude Code はリストの文字予算に合わせるために一部の説明を削除し、Claude が要求と照合するために必要なキーワードが削除されます。予算はモデルのコンテキストウィンドウの 1% でスケーリングされます。リストが予算を超える場合、Claude Code は最も呼び出しが少ないスキルから説明を削除するため、最も使用するスキルは完全なテキストを保持します。

/doctor を実行してリストのコンテキストコストとその最大の貢献者の推定値を取得してください。オフにする価値のあるスキルを見つけるには、*/skill-doctor* (ページ 307) を実行してください。リストが予算を超える場合、Claude Code はデバッグログに警告も書き込みます。これは *--debug* (ページ 51) で表示できます。

/context の Skills 行は、予算が適用された後のリストのサイズを報告するため、モデルが受け取るものと一致します。v2.1.196 より前は、この行はすべての説明の完全なテキストをカウントし、設定された予算の数倍大きい値を表示できました。

予算を増やすには、skillListingBudgetFraction 設定（例えば 0.02 = 2%）または SLASH_COMMAND_TOOL_CHAR_BUDGET 環境変数を固定文字数に設定してください。他のスキルのために予算を解放するには、*skillOverrides* (ページ 306) で低優先度のエントリを "name-only" に設定して、説明なしでリストされるようにしてください。また、ソースで description と when_to_use テキストをトリミングすることもできます。各エントリの結合テキストは予算に関わらず 1,536 文字でキャップされているため、主要なユースケースを最初に配置してください。キャップは skillListingMaxDescChars で設定可能です。

17.8.5 個人スキルが消えた

~/.claude/skills/ に作成したスキルフォルダが消えている場合は、~/.claude/skills/.trash/ を確認してください。Claude Code が *claude.ai* **からスキルを同期** (ページ 283) する場合、それらを別の synced サブフォルダにダウンロードし、作成したフォルダを移動または削除しません。

v2.1.280 より前は、~/.claude/skills/ に manifest.json という名前のファイルがあると、Claude Code はそのファイルがリストしたスキルフォルダを ~/.claude/skills/.trash/ の下のタイムスタンプ付きフォルダに移動し、それらのスキルは読み込まれなくなりました。

スキルを復元するには、タイムスタンプ付きフォルダからそのフォルダを ~/.claude/skills/ に戻してください。これは 保持期間スイープ がゴミ箱エントリを削除する前に行ってください。デフォルトではゴミ箱に移動されてから 30 日後に削除されます。

17.9 関連リソース

- 設定をデバッグする：スキルが表示されない、またはトリガーされない理由を診断する
- スキル出力品質の評価：agentskills.io の eval ファイル形式と反復ワークフロー
- スキル作成のベストプラクティス：Claude 製品全体に適用される作成ガイダンス
- **サブエージェント** (ページ 317)：特化したエージェントにタスクを委任する
- プラグイン：他の拡張機能でスキルをパッケージ化して配布する
- **フック** (ページ 443)：ツールイベント周辺のワークフローを自動化する
- **メモリ** (ページ 997)：永続的なコンテキストのための CLAUDE.md ファイルを管理する
- コマンド：組み込みコマンドとバンドルされたスキルのリファレンス
- **権限** (ページ 1039)：ツールとスキルアクセスを制御する
- Claude Tag スキル：リポジトリにコミットされたプロジェクトスキルは、そのリポジトリが Claude Tag チャネルで使用される場合にも読み込まれます

第18章　カスタムサブエージェントの作成

> Claude Code でタスク固有のワークフローと改善されたコンテキスト管理のための特化した AI サブエージェントを作成して使用します。

サブエージェントは、特定の種類のタスクを処理する特化した AI アシスタントです。サイドタスクがメイン会話に検索結果、ログ、または再度参照しないファイルコンテンツで溢れかえる場合に使用します。サブエージェントはそのタスクを独自のコンテキストで実行し、概要のみを返します。同じ種類のワーカーを同じ指示で繰り返し生成する場合は、カスタムサブエージェントを定義します。

各サブエージェントは、カスタムシステムプロンプト、特定のツールアクセス、および独立した権限を備えた独自のコンテキストウィンドウで実行されます。また、独自のリクエストを送信し、これはメイン会話と同じ **使用制限** (ページ 913) にカウントされます。Claude がサブエージェントの説明に一致するタスクに遭遇すると、そのサブエージェントに委譲し、サブエージェントは独立して動作して結果を返します。実際にコンテキスト節約を確認するには、コンテキストウィンドウの可視化で、サブエージェントが独自の別のウィンドウで研究を処理するセッションを説明しています。

注釈:

サブエージェントは単一のセッション内で動作します。多くの独立したセッションを並行して実行し、1 つの場所から監視するには、バックグラウンドエージェントを参照してください。互いにメッセージを渡すセッションについては、クロスセッションメッセージングを参照してください。Claude が生成および監督する調整されたセッションのチームについては、エージェントチームを参照してください。

サブエージェントは以下に役立ちます：

- **コンテキストを保持する** ことで、探索と実装をメインの会話から分離します
- **制約を強制する** ことで、サブエージェントが使用できるツールを制限します
- **設定を再利用する** ことで、ユーザーレベルのサブエージェントをプロジェクト全体で再利用します
- **動作を特化させる** ことで、特定のドメイン向けの焦点を絞ったシステムプロンプトを使用します
- **コストを制御する** ことで、Haiku のような高速で安価なモデルにタスクをルーティングします

Claude は各サブエージェントの説明を使用して、タスクを委譲するかどうかを決定します。サブエージェントを作成するときは、Claude がいつそれを使用するかを知るように、明確な説明を書いてください。

これらの説明はコンテキストを占有するため、短く保ってください。組み込みのサブエージェントを除いたサブエージェントの説明の合計が 15,000 トークンを超える場合、Claude Code はスタートアップ時に総トークン数を含む警告を表示します。サブエージェントの `description` フィールドをトリミングし、詳細を各サブエージェントのシステムプロンプトに移動します。システムプロンプトは、そのサブエージェントが実行される場合にのみロードされます。

18.1 組み込みサブエージェント

Claude Code には、Claude が必要に応じて自動的に使用する組み込みサブエージェントが含まれています。各サブエージェントは親の会話の権限を継承します。ほとんどは制限されたツールセットで実行されます。

Explore と Plan は、研究を高速かつ低コストに保つために、CLAUDE.md ファイルと git ステータススナップショットをスキップします。その他のすべての組み込みサブエージェントと**カスタムサブエージェント** (ページ 321) は、その定義が *`omitClaudeMd`* (ページ 326) フィールドを設定してユーザー、プロジェクト、およびローカル CLAUDE.md ファイルをスキップしない限り、両方を読み込みます。サブエージェントに到達するものの完全な内訳については、**スタートアップ時に読み込まれるもの** (ページ 350) を参照してください。

Explore
: コードベースの検索と分析に最適化された高速な読み取り専用エージェント。

  - **モデル**: メイン会話のモデル。メイン会話が Fable で実行されている場合、Explore のモデルは接続方法によって異なります。
    - Claude サブスクリプション、Anthropic Console アカウント、または `ANTHROPIC_BASE_URL` 経由で接続する *LLM* **ゲートウェイ** (ページ 841) を使用する場合、Explore は *opus* **エイリアス** (ページ 725) が解決される Opus モデルで実行されます。
    - Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry、Claude Platform on AWS、または Claude apps ゲートウェイでは、Explore はメイン会話のモデルのままです。
  - **ツール**: 読み取り専用ツール。Write と Edit は拒否されます。
  - **目的**: ファイル検出、コード検索、コードベース探索

  `Explore` という名前の**ユーザーまたはプロジェクトサブエージェント** (ページ 321) は組み込みを上書きし、独自の `model` フィールドを保持します。そのため、`model: haiku` で定義すると、探索をより低コストのモデルで実行できます。Explore を含むすべてのサブエージェントに 1 つのモデルを強制するには、**すべてのサブエージェントを *1* つのモデルで実行** (ページ 331) を参照してください。

  Claude は、変更を加えずにコードベースを検索または理解する必要がある場合、Explore に委譲します。これにより、探索結果がメイン会話コンテキストから除外されます。

  Explore を呼び出すとき、Claude は徹底度レベルを指定します。ターゲット検索の場合は **quick**、バランスの取れた探索の場合は **medium**、包括的な分析の場合は **very thorough** です。

Plan
: プランモード中にプランを提示する前にコンテキストを収集するために使用される研究エージェント。

  - **モデル**: メイン会話から継承されます。ただし、`CLAUDE_CODE_SUBAGENT_MODEL` を設定して**すべてのサブエージェントを *1* つのモデルで実行** (ページ 331) する場合を除きます。
  - **ツール**: 読み取り専用ツール。Write と Edit は拒否されます。
  - **目的**: 計画のためのコードベース研究

  プランモード中で Claude がコードベースを理解する必要がある場合、探索出力が別のコンテキストウィンドウに留まるように、Plan サブエージェントに研究を委譲し、メイン会話は読み取り専用のままです。

General-purpose
探索とアクションの両方を必要とする複雑なマルチステップタスク用の有能なエージェント。

- **モデル**: *CLAUDE_CODE_SUBAGENT_MODEL* (ページ 329) モデル（設定した場合で、他の何もモデルを別の方法で割り当てない場合）、それ以外の場合はメイン会話のモデル。**モデルを選択** (ページ 329) は完全な順序を示し、**すべてのサブエージェントを *1* つのモデルで実行** (ページ 331) は変数がこれらのソースをオーバーライドする方法を示します。
- **ツール**: **サブエージェントで利用可能** (ページ 331) なすべてのツール
- **目的**: 複雑な研究、マルチステップ操作、コード変更

Claude は、タスクが探索と変更の両方を必要とする場合、結果を解釈するための複雑な推論、または複数の依存ステップが必要な場合、general-purpose に委譲します。

Other
Claude Code には、特定のタスク用の追加のヘルパーエージェントが含まれています。これらは通常自動的に呼び出されるため、直接使用する必要はありません。

エージェント	モデル	Claude が使用する場合
claude	独自のモデルはありません。Claude がサブエージェントとして生成する場合は**モデル順序** (ページ 329) に従います	タスクがより特化したエージェントに適合しない場合。**サブエージェントで利用可能** (ページ 331) なすべてのツールを備えたキャッチオール。また、ディスパッチされたバックグラウンドセッションのデフォルトエージェント。セッションが開始される方法に応じて、開始される権限モード。
statusline-setup	Sonnet	`/statusline` を実行してステータスラインを設定する場合
claude-code-guide	Haiku	Claude Code 機能について質問する場合

組み込みサブエージェントはインタラクティブセッションでデフォルトで登録されます。これらを制限するには：

- 特定の組み込みタイプをブロックするには、**特定のサブエージェントを無効にする** (ページ 340) に示されているように、`permissions.deny` に追加します。
- Claude がサブエージェントに委譲するのを防ぐには、*permissions.deny* (ページ 1047) で `Agent` ツール自体を拒否します。
- 組み込みの `Explore` および `Plan` サブエージェントのみを削除するには、`CLAUDE_CODE_DISABLE_EXPLORE_PLAN_AGENTS=1` を設定します。Claude はファイルを直接読み取り、探索し、サブエージェントに委譲しません。Claude Code v2.1.198 以降が必要です。
- **非インタラクティブモード** (ページ 695) および Agent SDK では、`CLAUDE_AGENT_SDK_DISABLE_BUILTIN_AGENTS=1` を設定して、すべての組み込みタイプを削除し、独自のタイプのみを提供します。

サブエージェント型を省略する Agent ツール呼び出しは、セッションにフォールバックする general-purpose サブエージェントがない場合、`subagent_type is required` で失敗します。

これらの組み込みサブエージェント以外に、カスタムプロンプト、ツール制限、権限モード、フック、およびスキルを使用して独自のサブエージェントを作成できます。次のセクションでは、サブエージェントの使用を開始し、カスタマイズする方法を示します。

18.2 クイックスタート：最初のサブエージェントを作成する

サブエージェントは YAML フロントマターを含む Markdown ファイルです。Claude に作成してもらうか、**手動で作成** (ページ 324) することができます。

このチュートリアルでは、コードをレビューして改善を提案するユーザーレベルのサブエージェントを作成します。

1. **Claude にサブエージェントの作成を依頼する**

   Claude Code で、作成したいサブエージェントと保存場所を説明します：

   ```
   Create a personal code-improver subagent in ~/.claude/agents/ that scans
   files and suggests improvements for readability, performance, and best
   practices. It should explain each issue, show the current code, and
   provide an improved version. Make it read-only and have it use Sonnet.
   ```

   Claude は `name`、`description`、`tools` リスト、`model`、およびシステムプロンプトを含むファイルを作成します。

2. **ファイルを確認する**

   `~/.claude/agents/code-improver.md` を開き、フロントマターが要求内容と一致することを確認します。結果は次のようになります：

   ```
   ---
   name: code-improver
   description: Scans files and suggests improvements for readability, performance,␣
   ↪and best practices. Use after writing or modifying code.
   tools: Read, Grep, Glob
   model: sonnet
   ---

   You are a code improvement specialist. For each issue you find, explain
   the problem, show the current code, and provide an improved version.
   ```

   ファイルが `~/.claude/agents/` に存在するため、サブエージェントはマシン上のすべてのプロジェクトで利用可能です。代わりに 1 つのプロジェクトにスコープを設定するには、そのプロジェクトの `.claude/agents/` ディレクトリに移動します。**サブエージェントスコープを選択する** (ページ 321) で 2 つを比較しています。

3. **試してみる**

   Claude に新しいサブエージェントに委譲するよう依頼します：

```
Use the code-improver agent to suggest improvements in this project
```

Claude は新しいサブエージェントに委譲し、コードベースをスキャンして改善提案を返します。トランスクリプトでは、委譲はツール呼び出し行として表示され、サブエージェント名の後に短いタスク説明が続きます（例：`code-improver(Suggest code improvements)`）。

Claude が新しいサブエージェントを見つけられない場合は、Claude Code を再起動してもう一度試してください。これは `~/.claude/agents/` がセッション開始前に存在しなかった場合にのみ発生します。実行中のセッションは新しく作成された `agents` ディレクトリを検出しないためです。

これで、マシン上のプロジェクトでコードベースを分析し、改善を提案するために使用できるサブエージェントができました。

サブエージェントファイルを手動で作成したり、CLI フラグを使用して定義したり、プラグインを通じて配布したりすることもできます。以下のセクションでは、すべての設定オプションについて説明します。

注釈:

`/agents` を実行すると、Claude に依頼するか、`.claude/agents/` および `~/.claude/agents/` を直接編集するよう促すメッセージが表示されます。 Claude Code v2.1.197 以前では、`/agents` はライブサブエージェントを一覧表示する **Running** タブと、サブエージェントを作成、編集、削除するための **Library** タブを備えたインタラクティブウィザードを開きます。

18.3 サブエージェントの設定

サブエージェントのファイルの場所によって、誰がそれを利用できるかが決まり、frontmatter によってそれが何をできるかが決まります。このセクションでは、サブエージェントファイルがどこに存在するか、およびサポートされるすべてのフィールドについて説明します。

18.3.1 サブエージェントのスコープを選択する

スコープに応じて、サブエージェントファイルを異なる場所に保存します。複数のサブエージェントが同じ名前を共有する場合、Claude Code はより優先度の高い場所のものを使用します。

場所	スコープ	優先度	作成方法
管理設定	組織全体	1（最高）	**管理設定** (ページ 711) 経由でデプロイ
`--agents` CLI フラグ	現在のセッション	2	Claude Code 起動時に JSON を渡す
`.claude/agents/`	現在のプロジェクト	3	Claude に依頼するか、ファイルを手動で作成
`~/.claude/agents/`	すべてのプロジェクト	4	Claude に依頼するか、ファイルを手動で作成
プラグインの `agents/` ディレクトリ	プラグインが有効な場所	5（最低）	プラグインでインストール

プロジェクトサブエージェント（.claude/agents/）は、コードベースに固有のサブエージェントに最適です。バージョン管理にチェックインして、チームが協力して使用および改善できるようにします。

プロジェクトサブエージェントは現在の作業ディレクトリから上へ向かって検出されるため、そこからリポジトリルートまでのすべての .claude/agents/ がスキャンされます。これらのネストされたディレクトリの複数が同じ name を定義する場合、Claude Code は作業ディレクトリに最も近い定義を使用します。

--add-dir または /add-dir でディレクトリを追加すると、Claude Code はプロジェクトサブエージェントと一緒にその .claude/agents/ フォルダも読み込みます。**追加ディレクトリ** (ページ 1061) を参照して、--add-dir から読み込む他の設定タイプを確認してください。--add-dir を使用せずにプロジェクト間でサブエージェントを共有するには、~/.claude/agents/ またはプラグインを使用します。

ユーザーサブエージェント（~/.claude/agents/）は、すべてのプロジェクトで利用可能な個人用サブエージェントです。

Claude Code は .claude/agents/ と ~/.claude/agents/ を再帰的にスキャンするため、agents/review/ や agents/research/ などのサブフォルダに定義を整理できます。サブディレクトリパスはサブエージェントの識別方法や呼び出し方法に影響しません。これは、ID が name frontmatter フィールドのみから来るためです。

name 値をツリー全体で一意に保ちます。同じ .claude/agents/ ディレクトリ内（サブフォルダを含む）の 2 つのファイルが同じ名前を宣言する場合、Claude Code はそのうちの 1 つだけを読み込み、ドキュメント化された優先順位ではなくファイルシステムの読み取り順序で選択されます。ネストされたプロジェクトディレクトリ全体では、作業ディレクトリに最も近い定義が優先されます（上記で説明）。/doctor セットアップチェックアップは、同じディレクトリ内で名前を共有するファイルをレポートし、1 つを除くすべての名前変更または削除を提案します。v2.1.205 より前では、/doctor は重複をリストし、どの定義がアクティブであるかを示す診断画面を開きました。

プラグイン agents/ ディレクトリも再帰的にスキャンされます。プロジェクトおよびユーザースコープとは異なり、プラグインの agents/ ディレクトリ内のサブフォルダは**スコープ付き識別子** (ページ 343) の一部になります。プラグイン my-plugin の agents/review/security.md にあるファイルは my-plugin:review:security として登録されます。

CLI 定義サブエージェントは Claude Code 起動時に JSON として渡されます。これらはそのセッションのみに存在し、ディスクに保存されないため、クイックテストまたは自動化スクリプトに便利です。1 つの --agents 呼び出しで複数のサブエージェントを定義できます。

macOS, Linux, WSL

```
claude --agents '{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,␣
↪and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  },
  "debugger": {
    "description": "Debugging specialist for errors and test failures.",
    "prompt": "You are an expert debugger. Analyze errors, identify root causes,␣
```

(次のページに続く)

(前のページからの続き)

```
↪and provide fixes."
  }
}'
```

Windows PowerShell

```
claude --agents @'
{
  "code-reviewer": {
    "description": "Expert code reviewer. Use proactively after code changes.",
    "prompt": "You are a senior code reviewer. Focus on code quality, security,␣
↪and best practices.",
    "tools": ["Read", "Grep", "Glob", "Bash"],
    "model": "sonnet"
  },
  "debugger": {
    "description": "Debugging specialist for errors and test failures.",
    "prompt": "You are an expert debugger. Analyze errors, identify root causes,␣
↪and provide fixes."
  }
}
'@
```

非対話モード (ページ 695) では、`--agents` はコマンドラインで渡すには大きすぎる定義用に、JSON ファイルを保持するパスも受け入れます。例えば、`claude -p --agents ./agents.json "Review my changes"` はそのファイルから定義を読み取ります。対話型セッションでは、Claude Code はファイルパスを拒否します。ファイル形式には Claude Code v2.1.281 以降が必要です。

JSON の各トップレベルキーはエージェントの名前で、その値がそのエージェントの定義です。名前を `-` で始めないでください。定義は以下のフィールドを取ります。

- `prompt`。エージェントのシステムプロンプト。ファイルベースのサブエージェントのマークダウン本体と同等です。`prompt` は空にすることができます。空の `prompt` と `memory` フィールドのないエージェントを `--agent` でセッションのエージェントとして選択する場合、セッションのシステムプロンプトは変更されません。空の `prompt` には Claude Code v2.1.281 以降が必要です。
- *Frontmatter* **フィールド** (ページ 326)。`description`、`tools`、`disallowedTools`、`model`、`permissionMode`、`mcpServers`、`hooks`、`maxTurns`、`skills`、`initialPrompt`、`memory`、`effort`、`background`、`omitClaudeMd`、および `isolation`。
- **無視されるフィールド**。`color` と `experimental` はここでは受け入れられず、拒否されるのではなく無視されます。

Claude Code が読み込めない値で何をするか、およびそのチェックをスキップするフラグと環境変数については、`Invalid --agents configuration` を参照してください。

管理サブエージェントは組織管理者によってデプロイされます。管理設定ディレクトリ内の `.claude/agents/` にマークダウンファイルを配置し、プロジェクトおよびユーザーサブエージェントと同じ frontmatter 形式を使用します。管理定義は同じ名前のプロジェクトおよびユーザーサブエージェントより優先されます。

プラグインサブエージェントは、インストールしたプラグインから来ます。これらはカスタムサブエージェントと一緒に自動的に読み込まれ、スコープ付き名の下の @-mention タイプアヘッドに表示されます。プラグインサブエージェント作成の詳細については、プラグインコンポーネントリファレンスを参照してください。

注釈:

セキュリティ上の理由から、プラグインサブエージェントは `hooks`、`mcpServers`、または `permissionMode` フロントマターフィールドをサポートしていません。これらのフィールドはプラグインからエージェントを読み込むときに無視されます。これらが必要な場合は、エージェントファイルを `.claude/agents/` または `~/.claude/agents/` にコピーしてください。また、`settings.json` または `settings.local.json` の `permissions.allow` にルールを追加することもできますが、これらのルールはセッション全体に適用され、プラグインサブエージェントのみには適用されません。
プラグインの作成者である場合は、代わりにフックをプラグインの `hooks/hooks.json` に、MCP サーバーをその `.mcp.json` に含めて配布してください。これらはサブエージェント内だけでなく、プラグインが有効な場合は常に適用されます。

サブエージェント定義は、エージェントチームのチームメイトとして再利用することもできます。Claude にチームメイトのスポーンを依頼するときにサブエージェントタイプを指定すると、Claude Code はその定義の一部をチームメイトに適用します。各表示モードでどのスコープとどの部分が適用されるかについては、チームメイトにサブエージェント定義を使用するを参照してください。

18.3.2 サブエージェントファイルを作成する

サブエージェントファイルは設定用の YAML frontmatter を使用し、その後にマークダウンのシステムプロンプトが続きます。

注釈:

Claude Code は `~/.claude/agents/` と `.claude/agents/` を監視します。ディスク上のサブエージェントファイルを追加または編集するか、Claude にそれを作成するよう依頼すると、Claude Code は数秒以内に変更を検出し、次の委任は再起動なしで更新された定義を使用します。
3 つのケースではまだ再起動が必要です。

- ウォッチャーはセッション開始時に存在していたディレクトリのみをカバーするため、新しい `agents` ディレクトリでスコープの最初のエージェントファイルを作成した後、再起動して読み込みます。
- Claude Code は `--add-dir` または `/add-dir` で追加されたディレクトリ内の `.claude/agents/` を監視しないため、そこでサブエージェントを追加または編集した後、再起動して変更を読み込みます。
- `--disable-slash-commands` で開始されたセッションはこれらのディレクトリをまったく監視しません。

リスト 1: .claude/agents/code-reviewer.md

```
---
name: code-reviewer
description: Reviews code for quality and best practices
```

(次のページに続く)

(前のページからの続き)

```
tools: Read, Glob, Grep
model: sonnet
---

You are a code reviewer. When invoked, analyze the code and provide
specific, actionable feedback on quality, security, and best practices.
```

frontmatter はサブエージェントのメタデータと設定を定義します。本体はサブエージェントの動作をガイドするシステムプロンプトになります。サブエージェントは、このシステムプロンプトと作業ディレクトリなどの基本的な環境詳細のみを受け取り、Claude Code システムプロンプトは受け取りません。

非対話モード (ページ 695) では、*--append-subagent-system-prompt* (ページ 51) を渡して、テキストをすべてのサブエージェント（ネストされたサブエージェントを含む）のシステムプロンプトの末尾に追加します。ただし、**フォークされたサブエージェント** (ページ 354) は除きます。これは会話独自のプロンプトを再利用します。Claude Code v2.1.205 以降が必要です。テキストがコマンドラインで渡すには長すぎる場合は、ファイルに保存して `--append-subagent-system-prompt-file` でパスを渡してください。ファイルフラグには Claude Code v2.1.261 以降が必要です。

サブエージェントはメイン会話の現在の作業ディレクトリで開始します。サブエージェント内では、`cd` コマンドは Bash または PowerShell ツール呼び出し間で永続化されず、メイン会話の作業ディレクトリに影響しません。代わりにサブエージェントにリポジトリの分離されたコピーを与えるには、*isolation: worktree* (ページ 326) を設定します。

`isolation: worktree` を持つサブエージェントは、その worktree 内で Bash および PowerShell コマンドを実行します。作業ディレクトリがメインチェックアウトに解決されるコマンド（例えば、サブエージェント実行中に worktree ディレクトリが削除された場合）はエラーで失敗します。v2.1.203 より前では、そのようなコマンドはメインチェックアウトで実行できました。

この作業ディレクトリチェックは、Claude Code を起動したディレクトリを含むリポジトリ全体をカバーします。セッションがリンクされた worktree で独自に実行される場合、チェックはその worktree がリンクされているメインチェックアウトもカバーします。v2.1.210 より前では、チェックは起動ディレクトリのみをカバーしていました。作業ディレクトリがリポジトリ内の他の場所（例えば、monorepo サブディレクトリから Claude Code を起動した場合のリポジトリルート）に解決されるコマンドは、失敗する代わりにそこで実行されました。

Bash コマンドの場合、Claude Code はコマンド自体を 2 つの方法でチェックします。

- git をメインチェックアウトにリダイレクトするコマンドをブロックします。
- コマンドテキストから、コマンドが実行する git がすべて worktree 内に留まることを確認できないコマンドを拒否します。例えば、コマンド名が実行時に計算される場合。

リダイレクトベクトルと形状ルールは Claude Code が分離を強制する方法の下にリストされています。PowerShell コマンドは作業ディレクトリチェックのみを取得します。

Monitor コマンドは Bash コマンドと同じ作業ディレクトリおよびコマンドコンテンツチェックを通過します。

メイン会話自体が worktree で分離されて実行される場合、Claude Code はセッションとそれがスポーンするすべてのサブエージェント（`isolation: worktree` なしのサブエージェントを含む）に同じチェック

を適用します。Claude Code が分離を強制する方法を参照してください。

18.3.3 Frontmatter リファレンス

以下のフィールドは YAML *frontmatter* で使用できます。`name` と `description` のみが必須です。

複数単語のフィールド名は `maxTurns` や `disallowedTools` などの camelCase を使用し、テーブルと正確に一致する必要があります。Claude Code は認識しないフィールドを無視し、エラーを報告しません。サブエージェントファイルが読み込まれなかった理由を確認するには、*Claude Code* **がスキップするサブエージェントファイル** (ページ 328) を参照してください。

フィールド	必須	説明
`name`	はい	一意の識別子（`code-reviewer` や `reviewer-v2` など）。**フック** (ページ 521) はこの値を `agent_type` として受け取ります。ファイル名は一致する必要はありません。名前に `:` を含めることはできません。これは plugin-scoped identifiers（`my-plugin:reviewer` など）用に予約されています。Claude Code は `:` を含む名前のファイルを読み込まず、デバッグログにエラーをログします。v2.1.218 より前では、そのような名前は受け入れられていました
`description`	はい	Claude がこのサブエージェントに委任すべき場合
`tools`	いいえ	サブエージェントが使用できる**ツール** (ページ 331)。`Read, Grep, Glob` や YAML リストなどのカンマ区切り文字列として。省略した場合、サブエージェントで利用可能なすべてのツールを継承します。リスト内のエントリがツールに解決されない場合、サブエージェントは通常、エントリに名前を付けるエラーで起動に失敗します。スキルをコンテキストにプリロードするには、ここで `Skill` をリストするのではなく、`skills` フィールドを使用します
`disallowedTools`	いいえ	継承または指定されたリストから削除するツール。`tools` と同じ形式。`Bash(git push *)` などの指定子を持つエントリは、**ツール全体** (ページ 331) を削除します
`model`	いいえ	使用する**モデル** (ページ 329)。`sonnet`、`opus`、`haiku`、`fable`、`claude-opus-5-5` などの完全なモデル ID、または `inherit`。省略した場合、Claude Code は**サブエージェントモデル順序** (ページ 329) でモデルを選択します
`permissionMode`	いいえ	**権限モード** (ページ 336)。`default`、`acceptEdits`、`auto`、`dontAsk`、`bypassPermissions`、`plan`、または `default` のエイリアスとしての `manual`。`manual` エイリアスには Claude Code v2.1.200 以降が必要です。**プラグインサブエージェント** (ページ 321) では無視されます

次のページに続く

表 3 – 前のページからの続き

フィールド	必須	説明
`maxTurns`	いいえ	サブエージェントが停止する前の最大 agentic ターン数。サブエージェントが制限に達すると、Claude Code は出力を部分的としてマークして返し、Claude は**再開** (ページ 352) して続行できます。部分的なマーキングには Claude Code v2.1.246 以降が必要です
`skills`	いいえ	起動時にサブエージェントのコンテキストにプリロードする**スキル** (ページ 275)。説明のみではなく、完全なスキルコンテンツが注入されます。サブエージェントは、Skill ツールを通じてリストされていないプロジェクト、ユーザー、およびプラグインスキルを呼び出すことができます
`mcpServers`	いいえ	このサブエージェントで利用可能な *MCP* **サーバー** (ページ 363)。各エントリは、既に設定されているサーバーを参照するサーバー名（例えば、`"slack"`）、または完全な *MCP* **サーバー設定** (ページ 364) を値として持つサーバー名をキーとするインライン定義のいずれかです。**プラグインサブエージェント** (ページ 321) では無視されます
`hooks`	いいえ	このサブエージェントにスコープされた**ライフサイクルフック** (ページ 340)。**プラグインサブエージェント** (ページ 321) では無視されます
`memory`	いいえ	**永続メモリスコープ** (ページ 337)。`user`、`project`、または `local`。クロスセッション学習を有効にします
`background`	いいえ	Claude がフォアグラウンドで実行するよう要求した場合でも、このサブエージェントをバックグラウンドに保つには `true` に設定します。**フォークモード** (ページ 356) がオンの場合、Claude Code は Claude がスポーンするサブエージェントを既に**バックグラウンド** (ページ 344) で実行します
`omitClaudeMd`	いいえ	このサブエージェントをユーザー、プロジェクト、およびローカル CLAUDE.md ファイルなしで起動するには `true` に設定します。**管理ポリシーファイル** (ページ 1002) は引き続き読み込まれます。ただし、**管理サブエージェント** (ページ 321) は除きます。**委任プロンプト** (ページ 350) から必要なすべてを取得するサブエージェントに使用します。エージェントが `--agent` または `agent` 設定経由でメインセッションエージェントとして実行される場合は無視されます。Claude Code v2.1.271 以降が必要です
`effort`	いいえ	このサブエージェントがアクティブな場合の努力レベル。セッション努力レベルをオーバーライドします。デフォルト。セッションから継承します。オプション。`low`、`medium`、`high`、`xhigh`、`max`。利用可能なレベルはモデルに依存します

次のページに続く

表 3 – 前のページからの続き

フィールド	必須	説明
isolation	いいえ	サブエージェントを一時的な git worktree で実行するには worktree に設定します。これにより、親セッションの HEAD ではなく、デフォルトでデフォルトブランチからブランチされたリポジトリの分離されたコピーが提供されます。サブエージェントが変更を加えない場合、worktree は自動的にクリーンアップされます
color	いいえ	タスクリストとトランスクリプトでのサブエージェントの表示カラー。red、blue、green、yellow、purple、orange、pink、または cyan を受け入れます
initialPrompt	いいえ	このエージェントがメインセッションエージェント（--agent または agent 設定経由）として実行される場合、最初のユーザーターンとして自動送信されます。コマンドと**スキル** (ページ 275) が処理されます。ユーザー提供のプロンプトの前に付加されます。**プラグインサブエージェント** (ページ 321) では無視されます
experimental	いいえ	実験的オプションのマップ。その cacheTtl キーを 5m または 1h に設定して、このサブエージェントのリクエストのプロンプトキャッシュライフタイムを選択します。frontmatter のキャッシュライフタイム優先順位の場所。Claude Code は他の値を無視し、Claude サブスクリプションが使用クレジットを使用している間は 1h を無視し、サブエージェントファイルからのみフィールドを読み取ります。Claude Code v2.1.248 以降が必要です

cacheTtl を frontmatter のトップレベルではなく、experimental マップ内に書き込みます。

```
---
name: repo-auditor
description: Audits a large repository and reports what it finds
experimental:
  cacheTtl: 1h
---
```

Claude Code がスキップするサブエージェントファイル

Claude Code は、frontmatter に以下の問題がある場合、プロジェクト、ユーザー、または管理 agents ディレクトリ、または --add-dir で追加したディレクトリの下のファイルをセッションで報告せずにスキップします。

- name **がない**。Claude Code はファイルをエージェントの横に保持されたドキュメントとして扱います。
- **ファイルの最初の行ではない開き** ---。Claude Code はファイルに frontmatter がないと読み取り、ドキュメントとして扱います。

- `-` **で始まるか：を含む** `name`。Claude Code はファイルをスキップし、デバッグログにエラーを書き込みます。上記の表の `name` 行を参照してください。
- `name` **があるが** `description` **がない**。Claude Code はファイルをスキップし、理由をデバッグログに書き込みます。
- **解析されない YAML**。Claude Code はファイルからフィールドを読み取らず、スキップして、解析エラーをデバッグログに書き込みます。

デバッグログを表示するには、`--debug` で Claude Code を実行します。

frontmatter に `name` がない、または解析されないプラグインサブエージェントは、ファイル名の下で引き続き読み込まれます。

セッション前に `agents` ディレクトリをチェックする

frontmatter が解析されない `agents` ディレクトリ内のファイルを見つけるには、例えば `.claude/agents` または `~/.claude/agents` に対して `claude plugin validate` を実行します。Claude Code は名前を付けたディレクトリのみをチェックし、frontmatter が解析されるが `name` がないファイルにはフラグを立てません。Claude Code v2.1.233 以降が必要です。

18.3.4 モデルを選択する

`model` フィールドはサブエージェントが使用するモデルを制御します。

- **モデルエイリアス**。利用可能なエイリアスの 1 つを使用します。`sonnet`、`opus`、`haiku`、または `fable`
- **完全なモデル ID**。`claude-opus-5-5` または `claude-sonnet-5` などの完全なモデル ID を使用します。`--model` フラグと同じ値を受け入れます
- **inherit**。メイン会話と同じモデルを使用します

Claude がサブエージェントを呼び出すとき、その特定の呼び出しに対して `model` パラメータを渡すこともできます。Claude Code はサブエージェントのモデルをこの順序で解決します。

1. 呼び出しごとの `model` パラメータ
2. サブエージェント定義の `model` frontmatter。`inherit` はメイン会話のモデルを選択します
3. *`CLAUDE_CODE_SUBAGENT_MODEL`* (ページ 757) 環境変数。モデルエイリアスまたはモデル ID に設定した場合
4. メイン会話のモデル

2 つのケースでは、`opus` などのファミリエイリアスが、呼び出しごとのパラメータまたは frontmatter で、メイン会話のモデルの代わりに**エイリアスが指す** (ページ 725) バージョンに解決されます。

- **メイン会話のモデルがそのファミリに属する**。サブエージェントはメイン会話の正確なモデル（`[1m]` サフィックスを含む）で実行されるため、メイン会話と同じ**拡張コンテキスト** (ページ 752) ウィンドウを取得します。
- **Claude Code がメイン会話のモデルファミリを判断できない。***Anthropic API* **以外のプロバイダー** (ページ 847) **上で**。これは Claude Code がバッキングモデルに解決していない**アプリケーション推論プロファイル** *ARN* (ページ 873) を持つ Amazon Bedrock で発生する可能性があります。このケース

は opus エイリアスのみをカバーし、*ANTHROPIC_DEFAULT_OPUS_MODEL* (ページ 757) を設定した場合は適用されません。opus はその後、設定したモデルに解決されるため。

CLAUDE_CODE_SUBAGENT_MODEL のエイリアスは、メイン会話のファミリに名前を付けた場合でも、常にエイリアスが指すバージョンに解決されます。

CLAUDE_CODE_SUBAGENT_MODEL を単独で設定しても、組み込みの Explore および Plan サブエージェントが実行されるモデルは変わりません。変更するには、**すべてのサブエージェントを *1* つのモデルで実行** (ページ 331) を参照してください。

v2.1.251 より前では、CLAUDE_CODE_SUBAGENT_MODEL はこの順序で最初に来て、呼び出しごとのパラメータと frontmatter（model: inherit を含む）の両方をオーバーライドしました。

変数を inherit に設定することは、設定を解除するのと同じです。v2.1.196 より前では、その値はサブエージェントをメイン会話のモデルに強制し、他のソースを無視しました。

Claude Code は、呼び出しごとのパラメータ、frontmatter、および環境変数の値を組織の *availableModels* (ページ 732) 許可リストに対してチェックします。ブロックされた値の場合、別のモデルに置き換えます。

- opus などのファミリエイリアスがブロックされた場合、Claude Code はサブエージェントを許可リストが許可するそのファミリの最新バージョンで実行します。/model と同じ**置換ルールとプロバイダースコープ** (ページ 732) に従います。v2.1.222 より前では、Claude Code はブロックされたファミリエイリアスについても継承されたモデルでサブエージェントを実行していました。
- その他のブロックされた値の場合、その置換が動作しないプロバイダー、または許可リストがファミリのバージョンを許可しない場合、Claude Code は代わりに継承されたモデルでサブエージェントを実行します。CLAUDE_CODE_SUBAGENT_MODEL を設定した場合、Claude Code はそのモデルを最初に試します。これらの同じルールの下で。

対話型セッションでは、Claude Code は要求されたモデルとサブエージェントが実行されるモデルに名前を付ける警告を表示します。どちらの置換についても。

サブエージェントが実行されているモデルを確認するには、/tasks を実行します。Claude Code はサブエージェントの行でモデルに名前を付け、サブエージェントの定義またはそれがフォークされたスキルが *effort* (ページ 326) を設定する場合、**努力レベル** (ページ 747) を追加します。Claude Code v2.1.242 以降が必要です。

呼び出しごとの model パラメータは、サブエージェントが**再開または後続メッセージが送信** (ページ 352) される場合にも適用されるため、サブエージェントはそのモデルに留まります。v2.1.211 より前では、再開は呼び出しごとの値をドロップし、サブエージェントは定義の model フィールドまたはメイン会話のモデルに戻りました。

v2.1.198 以降、サブエージェントはメイン会話の**拡張思考** (ページ 751) 設定も継承します。セッションで思考がオンの場合、サブエージェントではオンになり、オフの場合、オフのままです。サブエージェントごとの思考設定はありません。v2.1.198 より前では、サブエージェントはメイン会話の設定に関係なく、拡張思考が無効で実行されていました。

すべてのサブエージェントを 1 つのモデルで実行する

CLAUDE_CODE_SUBAGENT_MODEL はデフォルトであるため、サブエージェントの定義または Claude が渡すモデルは引き続き優先されます。すべてのサブエージェント、チームメイト、およびワークフローエージェントに 1 つのモデルを適用するには、CLAUDE_CODE_SUBAGENT_MODEL_FORCE を 1 に設定します。Claude Code v2.1.257 以降が必要です。

- 両方の変数を設定した場合、サブエージェントは CLAUDE_CODE_SUBAGENT_MODEL のモデルで実行されます。
- CLAUDE_CODE_SUBAGENT_MODEL_FORCE のみを設定した場合、サブエージェントはメイン会話のモデルで実行されます。ただし、組み込みの Explore サブエージェントは、**組み込みサブエージェントの項でそれに対して記載されているモデル** (ページ 318) で実行されます。

例えば、すべてのサブエージェントを Haiku で実行するには、**設定ファイル** (ページ 711) の env ブロックで両方の変数を設定します。

```
{
  "env": {
    "CLAUDE_CODE_SUBAGENT_MODEL": "haiku",
    "CLAUDE_CODE_SUBAGENT_MODEL_FORCE": "1"
  }
}
```

設定が有効になったことを確認するには、サブエージェントが実行されている間に /tasks を実行します。サブエージェントの行は、実行されるモデルを示します。

CLAUDE_CODE_SUBAGENT_MODEL_FORCE がオンの間、Claude Code はサブエージェント定義の model フィールドを無視し、Claude はサブエージェント開始時にモデルを渡すことができません。以下のサブエージェントは引き続きメイン会話のモデルで実行されます。

- **フォーク** (ページ 354)
- model: inherit を持つ**サブエージェントで実行されるスキル** (ページ 303)

18.3.5 サブエージェント機能を制御する

ツールアクセス、権限モード、および条件付きルールを通じて、サブエージェントが何をできるかを制御できます。

利用可能なツール

サブエージェントは、メイン会話で利用可能な組み込みツールと MCP ツールを継承しますが、2 つのフィルタで絞られます。最初のフィルタはすべてのサブエージェントから短いツールリストを削除し、2 番目のフィルタは**バックグラウンド** (ページ 344) で実行されるサブエージェント（デフォルト）の組み込みツールセットを削減します。macOS、Linux、および WSL では、メイン会話にない場合、サブエージェントは Glob および Grep ツールを受け取ることもできます。Glob ツール動作で説明されています。**フォーク** (ページ 354) は両方のフィルタをスキップし、メイン会話の正確なツールプールを受け取ります。最初のフィルタは、tools フィールドにリストされている場合でも、これらのツールを削除します。

- `Agent`。サブエージェントが**深さ制限** (ページ 349) にある場合。**フォーク** (ページ 354) ではツールはリストされたままですが、代わりにエラーを返します
- `AskUserQuestion`
- `EndConversation`。メイン会話のみを終了できます。EndConversation ツール動作を参照してください
- `EnterPlanMode`
- `ExitPlanMode`。サブエージェントの *`permissionMode`* (ページ 336) が `plan` でない限り
- `ScheduleWakeup`
- `WaitForMcpServers`
- `Workflow`

2 番目のフィルタはバックグラウンドで実行されるサブエージェントに適用されます。`Agent` と `ExitPlanMode` を除き、これらはサブエージェントが実行される場所に関係なく最初のフィルタの条件に従います。バックグラウンドサブエージェントはすべての MCP ツールを保持しますが、これらの組み込みツールのみです。`Read`、`Grep`、`Glob`、`LSP`、`Bash`、`PowerShell`、`Edit`、`Write`、`NotebookEdit`、`WebFetch`、`WebSearch`、`TodoWrite`、`Skill`、`ToolSearch`、`EnterWorktree`、`ExitWorktree`、`Monitor`、`TaskStop`、`SendMessage`、および `Artifact`。それを報告するサブエージェント用の `SubagentHandback`。Claude Code はバックグラウンドサブエージェントから他のすべての組み込みツールを削除します。継承またはリストされているかどうかに関わらず、`tools` フィールドで。同じ定義はフォアグラウンドとバックグラウンドで異なるツールに解決できます。削除は、`tools` リストが何も解決しない場合を除き、エラーを報告しません。

v2.1.280 より前では、バックグラウンドサブエージェントは `LSP` を使用できませんでした。

`ListAgents` は、他の組み込みツールと同様にこれらのフィルタに従います。フォアグラウンドサブエージェントは、クロスセッションメッセージングが有効なセッションで継承し、バックグラウンドサブエージェントは保持しません。

エージェントチームのチームメイトは、さらにタスクツールと cron ツールを保持します。`TaskCreate`、`TaskGet`、`TaskList`、`TaskUpdate`、`CronCreate`、`CronDelete`、および `CronList`。

Task ツールのないセッションでは、Claude Code はサブエージェントにもタスクツールを提供しません。サブエージェントが異なるモデルを実行する場合でも。インプロセスチームメイトはセッションと同じ方法に従いますが、チームメイトが独自の分割ペインで実行される場合、別の Claude Code プロセスとして実行されるため、独自のモデルが決定します。

ツールを制限するには、`tools` フィールドを許可リストとして、または `disallowedTools` フィールドを拒否リストとして使用します。この例は `tools` を使用して、Read、Grep、Glob、および Bash のみを許可します。サブエージェントはファイルを編集、書き込み、または MCP ツールを使用することはできません。

```
---
name: safe-researcher
description: Research agent with restricted capabilities
tools: Read, Grep, Glob, Bash
---
```

この例は disallowedTools を使用して、Write と Edit を除く利用可能なツールを継承します。サブエージェントは Bash、MCP ツール、およびプールの残りを保持します。

```
---
name: no-writes
description: Inherits the available tools except file writes
disallowedTools: Write, Edit
---
```

両方が設定されている場合、disallowedTools が最初に適用され、次に tools が残りのプールに対して解決されます。両方にリストされているツールは削除されます。

tools リスト内の何もツールに解決されない場合（例えば、すべてのエントリが誤字であるか、サブエージェントで利用できないツールに名前を付けている場合）、Claude Code は通常、サブエージェントの起動を拒否し、Agent ツールは解決されないエントリに名前を付けるエラーを返します。Agent would be spawned with zero tools を参照してください。メッセージと各エントリを修正する方法。v2.1.208 より前では、そのサブエージェントはツールなしで起動し、空または混乱した結果を返す可能性があります。

両方のフィールドは、正確なツール名に加えて MCP サーバーレベルのパターンを受け入れます。mcp__<server> または mcp__<server>__* は、名前付きサーバーからすべてのツールを付与または削除します。disallowedTools では、mcp__* はすべての MCP ツールをすべてのサーバーから削除します。この例は、github MCP サーバーからすべてのツールを削除しながら、他のサーバーのツールとプール内の組み込みツールを保持します。

```
---
name: local-only
description: Inherits every tool except those from the github MCP server
disallowedTools: mcp__github
---
```

disallowedTools エントリに Bash(git push *) などの指定子がある場合、マッチングコマンドのみではなく、ツール全体をサブエージェントから削除します。Bash を保持し、特定のコマンドをブロックするには、設定に *Bash* **拒否ルール** (ページ 1047)（Bash(git push *) など）を permissions.deny に追加します。ルールはメイン会話とサブエージェントに適用されます。

スポーンできるサブエージェントを制限する

エージェントが claude --agent でメインスレッドとして実行される場合、Agent ツールを使用してサブエージェントをスポーンできます。スポーンできるサブエージェントタイプを制限するには、tools フィールドで Agent(agent_type) 構文を使用します。

注釈:

バージョン 2.1.63 では、Task ツールが Agent に名前変更されました。設定とエージェント定義の既存の Task(...) 参照は引き続きエイリアスとして機能します。

```
---
name: coordinator
description: Coordinates work across specialized agents
tools: Agent(worker, researcher), Read, Bash
---
```

これは許可リストです。worker と researcher サブエージェントのみをスポーンできます。エージェントが他のタイプをスポーンしようとすると、リクエストは失敗し、エージェントはプロンプトで許可されたタイプのみを見ます。特定のエージェントをブロックしながら他のすべてを許可するには、代わりに *permissions.deny* (ページ 340) を使用します。

制限なしでサブエージェントをスポーンできるようにするには、括弧なしで Agent を使用します。

```
tools: Agent, Read, Bash
```

tools リストから Agent を完全に省略した場合、エージェントは Agent ツールでサブエージェントをスポーンできません。

Agent(agent_type) 許可リスト構文は、claude --agent でメインスレッドとして実行されるエージェントにのみ適用されます。サブエージェント定義では、tools に Agent をリストするとそのサブエージェントが**深さ制限** (ページ 349) を許可する限り独自のサブエージェントをスポーンできますが、括弧内のタイプリストは無視されます。

MCP サーバーをサブエージェントにスコープする

mcpServers フィールドを使用して、メイン会話で利用できない *MCP* (ページ 363) サーバーへのアクセスをサブエージェントに与えます。ここで定義されたインラインサーバーは、サブエージェント開始時に接続され、**エージェントファイルのフォルダの信頼ルール** (ページ 335) の対象となり、終了時に切断されます。文字列参照は親セッションの接続を共有します。

注釈:

mcpServers フィールドは、エージェントファイルが実行できる両方のコンテキストに適用されます。

- Agent ツールまたは @-mention を通じてスポーンされたサブエージェント
- *--agent* (ページ 343) または agent 設定で起動されたメインセッション

エージェントがメインセッションの場合、インラインサーバー定義は *.mcp.json* (ページ 363) および設定ファイルのサーバーと一緒に起動時に接続され、**エージェントファイルのフォルダの信頼ルール** (ページ 335) の下にあります。/mcp では、以前に使用したリモート（HTTP または SSE）サーバーは *cached* **ステータス** (ページ 369) を代わりに表示できます。Claude Code は Claude が最初にそのツールの 1 つを呼び出すときに接続します。

リスト内の各エントリは、インラインサーバー定義またはセッションで既に設定されている MCP サーバーを参照する文字列のいずれかです。

```
---
name: browser-tester
description: Tests features in a real browser using Playwright
```

(次のページに続く)

(前のページからの続き)

```
mcpServers:
  # Inline definition: scoped to this subagent only
  - playwright:
      type: stdio
      command: npx
      args: ["-y", "@playwright/mcp@latest"]
  # Reference by name: reuses an already-configured server
  - github
---

Use the Playwright tools to navigate, screenshot, and interact with pages.
```

インライン定義は .mcp.json サーバーエントリと同じスキーマを使用し、サーバー名でキー付けされ、stdio、http、sse、および ws タイプをサポートします。

MCP サーバーをメイン会話から完全に除外し、そのツール説明がコンテキストを消費するのを避けるには、.mcp.json ではなくここでインラインで定義します。サブエージェントはツールを取得します。親会話は取得しません。

メインセッションに適用される MCP 制限は、サブエージェントのフロントマターで宣言されたサーバーにも適用されます。

- *--strict-mcp-config* (ページ 45) および *--bare* (ページ 45)
- エンタープライズ管理 MCP 設定
- allowedMcpServers および deniedMcpServers ポリシー

これらのいずれかがサーバーをブロックする場合、Claude Code はそのサーバーをスキップし、ブロックされたサーバーを示す警告を表示します。

管理設定の制限は、定義方法に関係なくすべてのサブエージェントに適用されます。--strict-mcp-config は、--agents または SDK の agents オプション経由でインラインで渡すサーバーをフィルタリングしません。これらは呼び出し元による明示的な入力であるためです。

インライン MCP サーバーに必要な信頼

Claude Code は、プロジェクトの .claude/agents/ ディレクトリ、または --add-dir ディレクトリの .claude/agents/ にあるエージェントファイルから**インライン *MCP* サーバー** (ページ 334) を読み込むのは、**エージェントファイルのあるフォルダを信頼** (ページ 1066) した後のみです。v2.1.238 より前では、Claude Code はこれらのサーバーを信頼をチェックせずに読み込んでいました。

- **カウントされない信頼**。親フォルダの信頼、および -p または SDK セッションが**設定ファイルのフック** (ページ 1066) に対して取得する自動信頼
- **それまで**。Claude Code はそのエージェントファイル内のすべてのインラインサーバーをスキップし、~/.claude.json の正確な projects["<path>"].hasTrustDialogAccepted キーをデバッグログに書き込みます

- `--add-dir` **ディレクトリ**。信頼されたワークスペースのリポジトリの外側のディレクトリは、その `.claude/agents/` ファイルがワークスペースの信頼を継承しないため、独自の信頼エントリが必要です

Claude Code は、エージェントファイルが来たフォルダの信頼をチェックせずに 2 種類のサーバーを読み込みます。

- 既に設定されているサーバーを参照する名前
- `~/.claude/agents/` のエージェントファイル、`--agents` または SDK `agents` オプションで渡すもの、または管理設定が提供するもの内のインラインサーバー

権限モード

`permissionMode` を設定して、サブエージェントが実行される権限モードを選択します。モードの設定値を使用するため、Manual モードは `default` です。設定を解除した場合、サブエージェントはメイン会話のモードを継承します。

メイン会話の権限モードは、Claude Code が設定した値を使用するかどうかを決定します。

- メイン会話が `bypassPermissions`、`acceptEdits`、または自動モードにある場合、サブエージェントはそのモードで実行され、Claude Code は設定した `permissionMode` を無視します。自動モードでは、分類器はメイン会話のブロックおよび許可ルールでサブエージェントのツール呼び出しを評価します。サブエージェントが終了すると、分類器はその作業と最終レポートもレビューしてから、レポートが配信されます。自動モードがサブエージェントを処理する方法を参照してください。
- メイン会話が `default`、`dontAsk`、または `plan` モードにある場合、サブエージェントは設定した権限モードで実行されます。ただし `bypassPermissions` を除きます。`bypassPermissions` を宣言するサブエージェントはメイン会話のモードを保持します。`bypassPermissions` 例外には Claude Code v2.1.267 以降が必要です。

`permissionMode` はこれらの値を受け入れ、`default` のエイリアスとして `manual` を受け入れます。

モード	動作
`default`	Manual モード。権限を求めるプロンプト
`acceptEdits`	ファイル編集と作業ディレクトリまたは `additionalDirectories` 内のパスの一般的なファイルシステムコマンドを自動受け入れ
`auto`	自動モード。バックグラウンド分類器がコマンドと保護されたディレクトリ書き込みをレビュー
`dontAsk`	権限プロンプトを自動拒否。明示的に許可されたツールは引き続き機能します。`AskUserQuestion`、*`requiresUserInteraction`* (ページ 404) とマークされた MCP ツール、およびコネクタツール**組織が *ask* に設定** (ページ 399) したセッション（その設定が Claude Code に到達する場合）は、許可されている場合でも拒否されます
`bypassPermissions`	権限プロンプトをスキップ。サブエージェントはメイン会話がそうである場合のみこのモードで実行されます
`plan`	Plan モード（読み取り専用探索）

スキルをサブエージェントにプリロードする

skills フィールドを使用して、スキルコンテンツをサブエージェントのコンテキストに起動時に注入します。これにより、実行中にスキルを発見して読み込む必要なく、サブエージェントにドメイン知識を提供します。

```
---
name: api-developer
description: Implement API endpoints following team conventions
skills:
  - api-conventions
  - error-handling-patterns
---

Implement API endpoints. Follow the conventions and patterns from the preloaded␣
↪skills.
```

リストされた各スキルの完全なコンテンツは起動時にサブエージェントのコンテキストに注入されます。このフィールドはプリロードされるスキルを制御し、サブエージェントがアクセスできるスキルではありません。なしで、サブエージェントは実行中に Skill ツールを通じてプロジェクト、ユーザー、およびプラグインスキルを発見して呼び出すことができます。スキルを完全に呼び出すことを防ぐには、*tools* (ページ 331) リストから Skill を省略するか、disallowedTools に追加します。

disable-model-invocation: true (ページ 296) を設定するスキルはプリロードできません。プリロードは Claude が呼び出すことができるスキルの同じセットから描画するため。これには、バンドルされた /verify スキルが含まれます。実行できるのはあなただけなので、プリロードすることもできません。

リストされたスキルが見つからないか無効な場合（例えば、組織のポリシーによって）、Claude Code はスキップし、デバッグログに警告をログします。

注釈:

これは**スキルをサブエージェントで実行する** (ページ 303) の逆です。サブエージェント内の skills を使用すると、サブエージェントはシステムプロンプトを制御し、スキルコンテンツを読み込みます。スキル内の context: fork を使用すると、スキルコンテンツが指定したエージェントに注入されます。どちらの場合も、サブエージェントは会話履歴なしで開始されます。

永続メモリを有効にする

memory フィールドはサブエージェントに、会話全体で存続する永続ディレクトリを提供します。サブエージェントはこのディレクトリを使用して、コードベースパターン、デバッグインサイト、アーキテクチャ決定などの知識を時間をかけて構築します。

```
---
name: code-reviewer
description: Reviews code for quality and best practices
memory: user
---
```

(次のページに続く)

(前のページからの続き)

```
You are a code reviewer. As you review code, update your agent memory with
patterns, conventions, and recurring issues you discover.
```

メモリがどの程度広く適用されるべきかに基づいてスコープを選択します。

スコープ	場所	使用時期
user	~/.claude/agent-memory/<name-of-agent>/	サブエージェントはすべてのプロジェクト全体で学習を記憶すべき
project	.claude/agent-memory/<name-of-agent>/	サブエージェントの知識はプロジェクト固有で、バージョン管理経由で共有可能
local	.claude/agent-memory-local/<name-of-agent>/	サブエージェントの知識はプロジェクト固有だが、バージョン管理にチェックインすべきではない

サブエージェントメモリは**自動メモリ** (ページ 1013) の一部です。自動メモリをオフにした場合、autoMemoryEnabled 設定または CLAUDE_CODE_DISABLE_AUTO_MEMORY を使用すると、memory フィールドは効果がなく、サブエージェントはメモリ指示またはメモリツールアクセスなしで起動されます。以下で説明します。

メモリが有効な場合。

- サブエージェントのシステムプロンプトには、メモリディレクトリへの読み取りと書き込みの指示が含まれます。
- サブエージェントのシステムプロンプトには、メモリディレクトリの MEMORY.md の最初の 200 行または 25KB（どちらか先）も含まれます。その制限を超える場合は MEMORY.md をキュレートする指示付き。
- Read、Write、および Edit ツールは自動的に有効になり、サブエージェントはメモリファイルを管理できます。

永続メモリのヒント

- project は推奨されるデフォルトスコープです。サブエージェント知識をバージョン管理経由で共有可能にします。
- サブエージェントに作業開始前にメモリを確認するよう依頼します。「このプルリクエストをレビューし、以前に見たパターンについてメモリを確認してください。」
- タスク完了後にメモリを更新するようサブエージェントに依頼します。「完了したので、学習したことをメモリに保存してください。」時間をかけて、これはサブエージェントをより効果的にする知識ベースを構築します。
- メモリ指示をサブエージェントのマークダウンファイルに直接含めて、積極的にメモリを維持します。

```
Update your agent memory as you discover codepaths, patterns, library
locations, and key architectural decisions. This builds up institutional
```

(次のページに続く)

(前のページからの続き)

```
knowledge across conversations. Write concise notes about what you found
and where.
```

フックを使用した条件付きルール

ツール使用をより動的に制御するには、PreToolUse フックを使用して、実行前に操作を検証します。これは、ツールの一部の操作を許可しながら他をブロックする必要がある場合に便利です。

この例は、読み取り専用データベースクエリのみを許可するサブエージェントを作成します。PreToolUse フックは、各 Bash コマンド実行前に command で指定されたスクリプトを実行します。

```yaml
---
name: db-reader
description: Execute read-only database queries
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---
```

Claude Code は**フック入力を *JSON* として** (ページ 495)stdin 経由でフックコマンドに渡します。検証スクリプトはこの JSON を読み取り、Bash コマンドを抽出し、**終了コード** *2* (ページ 472) で書き込み操作をブロックします。

```bash
#!/bin/bash
# ./scripts/validate-readonly-query.sh

INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

# Block SQL write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE)\b'␣
↪> /dev/null; then
  echo "Blocked: Only SELECT queries are allowed" >&2
  exit 2
fi

exit 0
```

macOS および Linux では、スクリプトを実行可能にするか、フックが失敗します。何もブロックしません。

```
chmod +x ./scripts/validate-readonly-query.sh
```

ルールをテストするには、サブエージェントに UPDATE ステートメントを実行するよう依頼します。スクリプトは終了コード 2 で終了し、Claude Code はコマンドをブロックし、サブエージェントは Blocked: Only SELECT queries are allowed メッセージを見ます。

完全な入力スキーマについては**フック入力** (ページ 495) を、終了コードが動作に影響する方法については**終了コード** (ページ 469) を参照してください。Windows では、PowerShell でフックスクリプトを作成し、*PowerShell* **でフックを実行** (ページ 561) に示すように、フックエントリに shell: powershell を追加します。

特定のサブエージェントを無効にする

設定の deny 配列にサブエージェントを追加して、Claude が特定のサブエージェントを使用するのを防ぐことができます。Agent(subagent-name) 形式を使用します。ここで subagent-name はサブエージェントの name フィールドと一致します。

```
{
  "permissions": {
    "deny": ["Agent(Explore)", "Agent(my-custom-agent)"]
  }
}
```

これは組み込みとカスタムサブエージェントの両方で機能します。--disallowedTools CLI フラグを使用することもできます。

```
claude --disallowedTools "Agent(Explore)"
```

権限ルールの詳細については、**権限ドキュメント** (ページ 1047) を参照してください。

18.3.6 サブエージェント用のフックを定義する

サブエージェントはサブエージェントのライフサイクル中に実行される**フック** (ページ 443) を定義できます。フックを設定する 2 つの方法があります。

- **サブエージェントの frontmatter で**。そのサブエージェントがアクティブな間のみ実行されるフックを定義
- **settings.json で**。セッション全体のフック。サブエージェント内でも発火します。PreToolUse および PostToolUse などのツールイベントはメイン会話と同じ方法でサブエージェントのツール呼び出しに対して発火し、SubagentStart および SubagentStop はサブエージェント開始時および終了時に発火します

設定ファイル、管理ポリシー設定、およびプラグイン (ページ 450) からのフックはすべてサブエージェント内に適用されるため、settings.json の PreToolUse フックはサブエージェントが使用するすべてのツールの前にも実行されます。

サブエージェント frontmatter のフック

サブエージェントのマークダウンファイルでフックを直接定義します。これらのフックはその特定のサブエージェントがアクティブな間のみ実行され、終了時にクリーンアップされます。

注釈:

Frontmatter フックは、Agent ツールまたは @-mention を通じてサブエージェントとしてスポーンされた場合、および *--agent* (ページ 343) または agent 設定経由でメインセッションとして実行される場合に発火します。メインセッションの場合、*settings.json* (ページ 443) で定義されたフックと一緒に実行されます。

プロジェクトレベルのサブエージェントの frontmatter フックを実行させるには、エージェントファイルを含むフォルダの**ワークスペース信頼ダイアログ** (ページ 1064) を受け入れます。~/.claude/agents/ のユーザーレベルサブエージェントからのフックと --agents で渡す定義は、このステップなしで実行されます。--add-dir で信頼されたワークスペースのリポジトリの外側からフォルダを追加した場合、そのフォルダを個別に信頼します。その .claude/agents/ フックはワークスペースの付与を継承しません。

フォルダを信頼するまで、サブエージェントは引き続き実行されますが、Claude Code は frontmatter フックをスキップし、フォルダを信頼する方法を説明するエラーをデバッグログにログします。これは設定ファイルのフックのルールより厳しいです。親フォルダの信頼は十分ではなく、-p セッションは信頼されたとしてカウントされません。**フォルダを信頼する前に実行されるもの** (ページ 1066) は 2 つを比較します。v2.1.218 より前では、frontmatter フックは信頼していないフォルダから実行でき、非対話型セッションを含めて実行できました。

すべての**フックイベント** (ページ 480) がサポートされています。サブエージェントの最も一般的なイベントは。

イベント	マッチャー入力	発火時期
PreToolUse	ツール名	サブエージェントがツールを使用する前
PostToolUse	ツール名	サブエージェントがツールを使用した後
Stop	（なし）	サブエージェントが終了する場合（実行時に Subagent Stop に変換）

この例は PreToolUse フックで Bash コマンドを検証し、PostToolUse でファイル編集後にリンターを実行します。

```
---
name: code-reviewer
description: Review code changes with automatic linting
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-command.sh"
  PostToolUse:
    - matcher: "Edit|Write"
```

(次のページに続く)

(前のページからの続き)

```
      hooks:
        - type: command
          command: "./scripts/run-linter.sh"
---
```

エージェントがサブエージェントとして呼び出される場合、frontmatter の Stop フックは自動的に SubagentStop イベントに変換されます。

サブエージェントイベント用のプロジェクトレベルフック

メインセッションでサブエージェントライフサイクルイベントに応答するフックを settings.json で設定します。

イベント	マッチャー入力	発火時期
SubagentStart	エージェントタイプ名	サブエージェント実行開始時
SubagentStop	エージェントタイプ名	サブエージェント完了時

両方のイベントは、名前でエージェントタイプをターゲットするマッチャーをサポートします。マッチャー値は、プロジェクトレベルおよびユーザーレベルサブエージェントの frontmatter name、またはプラグインサブエージェントの my-plugin:db-agent などのプラグインスコープ識別子です。スコープ付き名にはコロンが含まれるため、**アンカーなし正規表現** (ページ 451) として評価されます。^my-plugin:db-agent$ のように ^ と $ でアンカーして、そのエージェントのみをマッチします。

この例は、db-agent サブエージェント開始時のみセットアップスクリプトを実行し、サブエージェント停止時にクリーンアップスクリプトを実行します。

```
{
  "hooks": {
    "SubagentStart": [
      {
        "matcher": "db-agent",
        "hooks": [
          { "type": "command", "command": "./scripts/setup-db-connection.sh" }
        ]
      }
    ],
    "SubagentStop": [
      {
        "hooks": [
          { "type": "command", "command": "./scripts/cleanup-db-connection.sh" }
        ]
      }
    ]
  }
}
```

完全なフック設定形式については**フック** (ページ 443) を参照してください。

18.4 サブエージェントを使用する

18.4.1 自動委譲を理解する

Claude は、リクエスト内のタスク説明、サブエージェント設定の `description` フィールド、および現在のコンテキストに基づいて、タスクを自動的に委譲します。プロアクティブな委譲を促進するには、サブエージェントの説明フィールドに「use proactively」などのフレーズを含めてください。

説明は簡潔に保ってください。Claude Code は、サブエージェントの説明の合計が 15,000 トークンの制限を超えると起動時に警告を表示し、すべてのサブエージェントを読み込みます。

サブエージェントが プラグイン に含まれている場合、一度に 1 つずつチェックするのではなく、現実的なプロンプト全体で Claude がそれに委譲する信頼性を測定できます。`claude plugin eval` は各プロンプトをプラグインの有無で実行し、結果をスコアリングします。

18.4.2 サブエージェントを明示的に呼び出す

自動委譲では不十分な場合、サブエージェントを自分で要求できます。3 つのパターンは、1 回限りの提案からセッション全体のデフォルトまでエスカレートします。

- **自然言語**: プロンプトでサブエージェントに名前を付けます。Claude が委譲するかどうかを決定します
- **@-mention**: サブエージェントが 1 つのタスクで実行されることを保証します
- **セッション全体**: `--agent` フラグまたは `agent` 設定を介して、セッション全体がそのサブエージェントとして実行されます

自然言語の場合、特別な構文はありません。サブエージェントに名前を付けると、Claude は通常委譲します。

```
test-runner サブエージェントを使用してテストを実行し、失敗したテストを修正してください
code-reviewer サブエージェントに最近の変更を確認させてください
```

サブエージェントを @-mention してください。 `@` を入力し、ファイルを @-mention するのと同じ方法で、タイプアヘッドからサブエージェントを選択します。これにより、特定のサブエージェントが実行されることが保証され、選択が Claude に任されません。

```
@"code-reviewer (agent)" 認証の変更を確認してください
```

メッセージ全体は Claude に送信され、Claude は質問内容に基づいてサブエージェントのタスクプロンプトを作成します。@-mention は Claude が呼び出すサブエージェントを制御し、受け取るプロンプトではありません。

有効な プラグイン によって提供されるサブエージェントは、`my-plugin:code-reviewer` や `my-plugin:review:security` などのスコープ付き名でタイプアヘッドに表示されます。プラグインが **エージェントをサブフォルダに整理** (ページ 321) している場合です。セッションで現在実行されている名前付きバックグラウンドサブエージェントもタイプアヘッドに表示され、名前の横にステータスが表示されます。

ピッカーを使用せずに手動でメンションを入力することもできます。ローカルサブエージェントの場合は `@agent-<name>`、プラグインサブエージェントの場合は `@agent-` の後にスコープ付き名を入力します。例えば `@agent-my-plugin:code-reviewer` です。このフォームを入力している間、タイプアヘッドはエージェントではなくファイルマッチを表示します。エージェントメンションは送信時に解決されます。

セッション全体をサブエージェントとして実行します。 *--agent <name>* (ページ 45) を渡して、メインスレッド自体がそのサブエージェントのツール制限とモデルを引き継ぐセッションを開始します。

```
claude --agent code-reviewer
```

エージェントの **プロンプトが空でない限り** (ページ 321)、カスタムサブエージェントのシステムプロンプトはデフォルトの Claude Code システムプロンプトを完全に置き換えます。これは *--system-prompt* (ページ 45) と同じ方法です。`CLAUDE.md` ファイルとプロジェクトメモリは、エージェント定義が *omitClaudeMd* (ページ 326) を設定している場合でも、通常のメッセージフローを通じて読み込まれます。

エージェント名は起動ヘッダーに `@<name>` として表示されるため、アクティブであることを確認できます。

これは組み込みおよびカスタムサブエージェントで機能し、セッションを再開するときに選択が保持されます。Claude Code はエージェントのツール制限とモデルを会話とともに復元します。セッションを再開するときにエージェントが存在しなくなった場合、セッションはデフォルトツールで続行され、エージェントに名前を付ける警告 が表示されます。どちらの場合のシステムプロンプトについては、**再開された会話のシステムプロンプトフラグ** (ページ 73) を参照してください。

プラグイン提供のサブエージェントの場合、エージェント名のみを渡すことができ、Claude Code がそれを見つけます。

```
claude --agent security-reviewer
```

複数のプラグインが同じ名前のエージェントを提供する場合、スコープ付き名を渡して曖昧さを解消します。

```
claude --agent my-plugin:security-reviewer
```

プラグインがエージェントを `agents/` ディレクトリのサブフォルダに配置している場合、スコープ付き名にサブフォルダを含めます。例えば `claude --agent my-plugin:review:security` です。

プロジェクト内のすべてのセッションのデフォルトにするには、`.claude/settings.json` で `agent` を設定します。

```
{
  "agent": "code-reviewer"
}
```

両方が存在する場合、CLI フラグが設定を上書きします。

18.4.3 サブエージェントをフォアグラウンドまたはバックグラウンドで実行する

サブエージェントはフォアグラウンドまたはバックグラウンドで実行できます。

- **フォアグラウンドサブエージェント** はメイン会話をブロックして完了するまで待機します。権限プロンプトは発生時にユーザーに渡されます。

- **バックグラウンドサブエージェント** は、ユーザーが作業を続ける間に同時に実行されます。バックグラウンドサブエージェントが権限が必要なツール呼び出しに到達すると、Claude Code はメインセッションでプロンプトを表示し、要求しているサブエージェントの名前を示します。承認するとサブエージェントは続行し、Esc を押すとサブエージェントを停止せずにそのツール呼び出しのみを拒否できます。

Claude が Agent ツールで生成するサブエージェントごとに、Claude Code は、適用される最初のケースからフォアグラウンドまたはバックグラウンドを選択します。

- インプロセス エージェントチーム チームメイトがサブエージェントを生成した場合、Claude Code はそれをフォアグラウンドで実行します。Claude Code は、定義が *`background: true`* (ページ 326) を設定するチームメイトのサブエージェントの生成をエラーで拒否します。**フォークモード** (ページ 356) がオフで、バックグラウンドタスクをオフにしていない 場合、チームメイトが `run_in_background: true` を設定したときも Claude Code はエラーで拒否します。
- `CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` を `1` に設定した場合、Claude Code はすべての種類のセッションでサブエージェントをフォアグラウンドで実行し、フォークモードがオンかどうかに関わらず実行します。
- **フォークモード** (ページ 356) がオンの場合（対話セッションではデフォルト）、Claude Code はサブエージェントをバックグラウンドで実行し、フォークおよび非フォークサブエージェント両方を実行し、Claude はフォアグラウンドを要求できません。
- フォークモードがオフの場合、Claude はデフォルトでサブエージェントをバックグラウンドで実行し、続行する前に結果が必要な場合はフォアグラウンドで実行します。フォークモードは、オンにしない限り、**非対話モード** (ページ 695) で `-p` を使用する場合と Agent SDK ではオフです。Claude が結果を必要とする場合でも特定のサブエージェントをバックグラウンドに保つには、フロントマター *`background`* (ページ 326) フィールドを `true` に設定します。

`context: fork` を持つスキルの場合、Claude Code は、フォークモードがオンかどうかに関わらず、**サブエージェントでスキルを実行** (ページ 303) のルールに従います。

バックグラウンドサブエージェントは、会話フォークと **再開された** (ページ 352) フォアグラウンドサブエージェントを除き、フォアグラウンドサブエージェントより **小さい組み込みツールセット** (ページ 331) で実行されます。

バックグラウンドサブエージェントはメインセッションのすべての権限プロンプトを表示します。それらのプロンプトの 1 つに、セッションの残り期間続く付与など、その 1 つのツール呼び出しを超えて続く選択で答える場合、Claude Code はセッション全体（メイン会話を含む）にその答えを適用します。

バックグラウンドサブエージェントは、バックグラウンド Bash または PowerShell コマンド を **ターンの終了後も実行し続ける** (ページ 88) ことができます。そのコマンドが終了すると、Claude Code はサブエージェントに通知を送信します。

バックグラウンドサブエージェントの結果は、後のターンで完了通知として Claude に到達します。Claude はその通知を待ってからサブエージェントの結果を報告し、先に進捗について質問すると、サブエージェントがまだ実行中であることを報告します。v2.1.211 より前では、Claude は完了していないバックグラウンドサブエージェントの結果を報告することがありました。

また、自分で操作することもできます。

- フォークモードがオフの場合、Claude にタスクをバックグラウンドまたはフォアグラウンドで実行するよう依頼します

- **Ctrl+B** を押して実行中のタスクをバックグラウンドにします

Claude Code は、サブエージェントの終了方法に応じて、2 つの方法のいずれかでプロンプト入力下のサブエージェントパネルからバックグラウンドサブエージェントの行をクリアします。

- サブエージェントが正常に完了すると、Claude Code はその行をすぐに削除し、スクリーンリーダーモード を除き、フッターに `/tasks to see subagents` を 30 秒間表示します。その 30 秒間に `/tasks` を実行し、サブエージェントで `Enter` を押してそのトランスクリプトを開きます。v2.1.232 より前では、Claude Code は失敗したサブエージェントと同じように、サブエージェントが完了してから 30 秒間行を保持し、フッターヒントを表示しませんでした。
- サブエージェントが失敗するか停止すると、Claude Code は 30 秒間その行を保持します。行をすぐにクリアするには、それを選択して `x` を押します。

完了したバックグラウンドサブエージェントは、フッターヒントと同じ 30 秒間、完了とマークされ実行中の作業の下にソートされた状態で `/tasks` に一覧表示されたままになります。詳細ビューはサブエージェントが完了しても開いたままです。失敗するか停止したサブエージェントはリストから消えます。v2.1.208 より前では、完了したサブエージェントは完了した瞬間にリストから消え、詳細ビューが閉じました。

18.4.4 サブエージェント名

Claude は Agent ツール呼び出しで `name` パラメータを渡すことでサブエージェントに名前を付けることができ、ユーザーに確認することなく自分で行う場合があります。名前はサブエージェントをアドレス指定可能にします。Claude はサブエージェントの完了後に **名前でメッセージを送信または再開** (ページ 352) できます。

エージェントチーム が有効になっている対話セッションでは、メイン会話から Claude が `name` 付きで生成するサブエージェントは、呼び出しが **フォーク** (ページ 354) であるか、呼び出し自体で `isolation` を渡さない限り、代わりにチームメイトとして起動します。サブエージェントのフロントマターの `isolation` 値はそれを防がず、その場合チームメイトはメインセッションの作業ディレクトリで実行されます。Claude がエージェントチームを開始する方法 を参照してください。

18.4.5 サブエージェント内の API エラー

何かが サブエージェントの応答をストリーム中に切断 し、部分的な応答にテキストが含まれているがツール呼び出しがない場合、Claude Code は実行を終了するのではなく、サブエージェントに続行を促します。これは対話セッションでも発生します。実行はそれらの継続が使い果たされた場合にのみエラーで終了します。

API エラー（使用制限や繰り返されるサーバーエラーなど）で実行が終了したサブエージェントは、その失敗を Claude に報告します。Claude が受け取るものは、サブエージェントが実行された場所によって異なります。

- **フォアグラウンド**: レート制限、オーバーロード、またはサーバーエラーがテキスト出力を既に生成したサブエージェントを切断した場合、Agent ツールはその部分出力と、サブエージェントが切断され、タスクを完了しなかったというメモを返します。何も出力しなかったか、唯一の出力がツール呼び出しだったサブエージェントは `Agent terminated early due to an API error` で失敗し、その後にエラー詳細が続きます。v2.1.199 では、ツール呼び出しのみの形状を切断したレート制限、オーバーロード、またはサーバーエラーは、切断メモのみを含む空の部分結果を返しました。

- **バックグラウンド**: サブエージェントは失敗とマークされ、Claude が終了時に受け取るメッセージは API エラーに名前を付け、サブエージェントの最後の出力を含むため、部分的な作業は失われません。

フォールバックモデルチェーン (ページ 743) を設定し、サブエージェントがチェーンがカバーする失敗（モデルが利用できないなど）に遭遇した場合、Claude Code はサブエージェントをリクエストを受け入れるチェーンの最初のモデルに切り替えます。サブエージェントはエラーで終了するのではなく、作業を続けます。

基盤となる API エラーがクリアされたら、Claude にタスクを再試行するか、**サブエージェントを再開** (ページ 352) するよう依頼してください。

18.4.6 サブエージェント出力スキャン

Claude Code は、Claude がそれを読む前に各サブエージェントの最終レポートをスキャンします。サブエージェントはユーザーが確認していないファイル、ウェブページ、またはコマンド出力を読んだ可能性があり、それらのソースからのテキストはメイン会話を対象とした指示を含むことがあります。スキャンは何も削除または言い換えません。レポートで気付く可能性のある 2 種類の変更を行います。

- **バックスラッシュ挿入**: スキャンは、`<system-reminder>` タグや `Human:` または `Assistant:` で始まる行など、Claude Code 独自の出力を模倣するテキストにバックスラッシュを挿入し、模倣が会話の一部として誤解されるのではなく、通常のテキストとして読まれるようにします。
- **マーカー行**: スキャンは、`<system-reminder>` のようなタグを模倣するか、`bypassPermissions` や `--dangerously-skip-permissions` などの権限設定に言及するレポートの前に、`[harness: subagent output matched instruction-shaped pattern(s):` で始まる行を付加します。権限設定の言及はマーカー行を取得しますが、テキスト自体は書かれたままです。

スキャンはコンテンツが悪意があるかどうかを判断せず、レポート内の指示が何ができるかを変更しません。レポートが Claude に導くツール呼び出しは、引き続きセッションの **権限チェック** (ページ 1039) と **サンドボックス化** (ページ 1069) を通じて実行されます。これは **サブエージェントが到達できるものを制限する** (ページ 331) ための代替ではありません。

Claude のサブエージェント結果として返されるレポートは、サブエージェント出力としてマークするヘッダーの下にも到達します。ヘッダーは、レポート内の指示または承認の主張がサブエージェントの言葉であり、ユーザーからの権限を持たないことを述べています。

バックグラウンドサブエージェントのレポート (ページ 344) は完了通知内に到達し、これはユーザーからのメッセージではなく自動化されたイベントとしてマークされます。

注釈:

サブエージェント出力スキャンには Claude Code v2.1.210 以降が必要です。

18.4.7 一般的なパターン

大量の操作を分離する

サブエージェントの最も効果的な用途の 1 つは、大量の出力を生成する操作を分離することです。テストの実行、ドキュメントの取得、またはログファイルの処理は、かなりのコンテキストを消費する可能性があります。これらをサブエージェントに委譲することで、詳細な出力はサブエージェントのコンテキストに留まり、関連する概要のみがメイン会話に返されます。

```
サブエージェントを使用してテストスイートを実行し、失敗したテストとそのエラーメッセージのみを報告してください
```

並列研究を実行する

独立した調査の場合、複数のサブエージェントを生成して同時に作業させます。

```
認証、データベース、API モジュールを並列で調査し、別々のサブエージェントを使用してください
```

各サブエージェントは独立して領域を探索し、Claude は検出結果を合成します。これは調査パスが互いに依存しない場合に最適です。

警告:

サブエージェントが完了すると、その結果はメイン会話に返されます。詳細な結果を返す多くのサブエージェントを実行すると、かなりのコンテキストを消費する可能性があり、各サブエージェントは実行中に独自のトークンを費やします。

並列で実行し続ける必要があるか、1 つのコンテキストウィンドウに収まらない作業の場合、別のセッション で実行し、Claude に セッション間で検出結果を渡す ようにします。

サブエージェントをチェーンする

マルチステップワークフローの場合、Claude にサブエージェントを順番に使用するよう依頼します。各サブエージェントはタスクを完了し、結果を Claude に返します。Claude は関連するコンテキストを次のサブエージェントに渡します。

```
code-reviewer サブエージェントを使用してパフォーマンスの問題を見つけ、次に optimizer サブエージェントを使用してそれらを修正してください
```

18.4.8 サブエージェントとメイン会話の選択

メイン会話 を使用する場合：

- タスクは頻繁なやり取りまたは反復的な改善が必要です
- 計画、実装、テストなど、複数のフェーズが重要なコンテキストを共有します
- 迅速で対象を絞った変更を行っています

- レイテンシが重要です。**フォーク** (ページ 354) ではないサブエージェントは新規に開始し、コンテキストを収集するのに時間がかかる場合があります

サブエージェント を使用する場合：

- タスクはメインコンテキストで必要としない詳細な出力を生成します
- 特定のツール制限または権限を適用したいです
- 作業は自己完結型で、概要を返すことができます

スキル (ページ 275) を代わりに検討してください。分離されたサブエージェントコンテキストではなく、メイン会話コンテキストで実行される再利用可能なプロンプトまたはワークフローが必要な場合です。

会話に既にあるものについての質問の場合、サブエージェントの代わりに *`/btw`* (ページ 98) を使用してください。フルコンテキストを見ますが、ツールアクセスはなく、答えは履歴に追加されません。

18.4.9 サブエージェントが独自のサブエージェントを生成できるようにする

デフォルトでは、サブエージェントは独自のサブエージェントを生成でき、メイン会話の下に最大 3 層です。深さの制限では、Claude Code は **フォーク** (ページ 354) を除くすべてのサブエージェントから `Agent` ツールを保留するため、制限時のサブエージェントは委譲された作業を自分で行い、1 つの概要を返します。制限時のフォークは継承されたツールリストに `Agent` を保持しますが、ツールは生成する代わりにエラーを返します。

ネストされたサブエージェントは、委譲されたタスク自体が並列サブタスクに分割される場合に適しています。例えば、検出結果ごとに検証者を派遣するレビューアーサブエージェントです。対話セッションでは、トップレベルのサブエージェントの概要のみがユーザーに返され、中間出力はメイン会話から外れます。バックグラウンドサブエージェントを起動するサブエージェントは、完了する前にその結果を待ちます。**非対話モード** (ページ 695) と Agent SDK では、起動するサブエージェントは待たないため、ネストされたバックグラウンドサブエージェントがその起動者の終了後に完了すると、代わりにメイン会話に報告されます。

制限を変更するには、`CLAUDE_CODE_MAX_SUBAGENT_SPAWN_DEPTH` をメイン会話の下に必要なサブエージェント層の数に設定します。例えば、*`settings.json`* (ページ 711) のこのエントリはネストを 2 層に制限します。

```
{
  "env": {
    "CLAUDE_CODE_MAX_SUBAGENT_SPAWN_DEPTH": "2"
  }
}
```

この値では、サブエージェントは 2 番目の層に委譲でき、その 2 番目の層はさらに委譲できません。1 に設定してネストをオフにします。

ネストされたサブエージェントはトップレベルのサブエージェントと同じ方法で設定され、同じ **スコープ** (ページ 321) から解決されます。ネストがオンの場合、読み取り専用のままにするべきレビューアーなど、1 つのサブエージェントが生成しないようにするには、その *`tools`* (ページ 331) リストから `Agent` を省略するか、`disallowedTools` に追加します。

ターミナルでは、Claude Code はネストされたサブエージェントをプロンプト入力下のサブエージェントパネルのツリーとして表示し、パネル内に子孫がまだいる各行を (+N) 数でマークします。行を開いて、そのサブエージェントの兄弟と直接の子を main へのパスで表示します。

注釈:

以前のバージョンは異なるデフォルトを使用していました。

- **v2.1.172 から v2.1.216**: サブエージェントはデフォルトでネストでき、最大 5 層深く、制限は変更できませんでした。
- **v2.1.217 から v2.1.218**: 制限はデフォルトで 1 でしたので、サブエージェントは上げない限り独自に生成できませんでした。v2.1.219 はデフォルトを 3 に上げました。

18.4.10 同時実行サブエージェント制限

2 つの制限がサブエージェント使用を制御し、それぞれ独自の変数があります。これは Claude が多くのサブエージェントが実行されている間に、より多くのサブエージェントを生成するのを停止し、**深さ制限** (ページ 349) はサブエージェントがどのくらい深くネストするかを制限します。セッション全体で Claude が生成できるサブエージェントの総数に制限はありません。

デフォルトでは、20 個のサブエージェントがセッションで実行されている場合、Agent ツールで別のサブエージェントを生成すると Concurrent subagent limit reached で失敗し、エラーは Claude に再試行しないよう指示します。実行中の数が制限を下回ると、生成が再び成功します。制限を変更するには、CLAUDE_CODE_MAX_CONCURRENT_SUBAGENTS を任意の正の整数に設定します。*ultracode* (ページ 747) がアクティブなセッションは除外されます。制限はそこで適用されません。Claude Code v2.1.217 以降が必要です。

制限は Claude が Agent ツールで生成するサブエージェントのみをブロックしますが、他の実行は同じスロットを占有します。

- */subtask* (ページ 354) で開始するセッション内フォークは、実行中にスロットを取得し、制限によってブロックされることはありません。
- **既に完了したサブエージェントを再開** (ページ 352) すると、制限をチェックせずに新しいスロットを取得するため、再開は実行中の数を制限を超えて押し上げることがあります。

ワークフロー エージェントや エージェントチーム チームメイトなど、他の機能が実行するエージェントは、代わりに独自の制限に従います。

18.4.11 サブエージェントコンテキストを管理する

起動時に読み込まれるもの

各サブエージェントは新しい、分離されたコンテキストウィンドウで開始されます。会話履歴、既に呼び出したスキル、または Claude が既に読んだファイルは表示されません。Claude はタスクを要約する委譲メッセージを作成し、サブエージェントはそこから作業します。例外は **フォーク** (ページ 354) で、新規に開始する代わりに親会話を継承します。

非フォークサブエージェントの初期コンテキストには以下が含まれます。

- **システムプロンプト**: エージェント独自のプロンプトと Claude Code が追加する環境詳細。Claude Code システムプロンプトではありません。カスタムサブエージェントは **マークダウン本体** (ペー

ジ 324) または prompt フィールドで定義します。組み込みエージェントは事前定義されたプロンプトを持ちます。

- **タスクメッセージ**: Claude が作業を引き継ぐときに作成する委譲プロンプト。
- **CLAUDE.md ファイル**: メイン会話が読み込む *CLAUDE.md* **階層** (ページ 1002) のすべてのレベル。~/.claude/CLAUDE.md、プロジェクトルール、CLAUDE.local.md、管理ポリシーファイル、およびプロジェクト指示として読み込まれる任意の *AGENTS.md* **ファイル** (ページ 1008) を含みます。組み込みの Explore および Plan エージェントはこれをスキップします。定義が *omitClaudeMd* (ページ 326) を設定するサブエージェントは、管理ポリシーファイルのみを読み込むか、定義が **管理設定** (ページ 321) から来る場合はまったく読み込みません。
- **Git ステータス**: Claude Code がサブエージェント開始時にリポジトリから読み込むスナップショット。Git リポジトリの外部またはスナップショットがオフになっている場合は含まれません。include GitInstructions を参照してください。Explore および Plan はいずれにせよそれをスキップします。
- **プリロードされたスキル**: エージェントの *skills* **フィールド** (ページ 337) で名前が付けられたスキルの完全なコンテンツ。組み込みエージェントはスキルをプリロードしません。
- **兄弟名簿**: main と、セッション内のすべての他の名前付きエージェントをリストする **システムリマインダー**。各エージェントは *SendMessage* (ページ 352) の有効な to 値です。Claude Code v2.1.206 以降が必要です。名簿は、サブエージェントのツールに SendMessage が含まれ、少なくとも 1 つの他のエージェントに名前がある場合にのみ表示されます。Claude が生成時に名前を付けたか、エージェントチーム チームメイトとして実行されるかに関わらず。これはサブエージェント開始時に取得されたスナップショットであるため、後で名前が付けられたエージェントは表示されません。

ユーザー、プロジェクト、ローカル CLAUDE.md ファイルなしで独自のサブエージェントを起動するには、そのフロントマターまたは --agents JSON で *omitClaudeMd: true* (ページ 326) を設定します。

メイン会話は、これらのサブエージェントの結果を読むときに完全な CLAUDE.md を持っているため、ほとんどのルールはサブエージェント自体に到達する必要はありません。ルールが必要な場合（「vendor/ ディレクトリを無視する」など）、委譲時に Claude に与えるプロンプトで再度述べてください。

サブエージェントが git ステータスを受け取るかを変更することはできません。Explore および Plan のみがそれをスキップします。

いくつかのメイン会話状態は非フォークサブエージェントに到達しません。

- **出力スタイル**: サブエージェントは独自のシステムプロンプトを実行するため、**出力スタイル** (ページ 821) はその応答を形作りません。**フォーク** (ページ 354) を除きます。
- **自動メモリ**: メイン会話の **自動メモリ** (ページ 1013) は読み込まれません。サブエージェントに独自の永続的なメモリを与えるには、*memory* **フィールド** (ページ 337) を使用します。
- **コンテキストウィンドウサイズ**: サブエージェントのコンテキストウィンドウは、親のではなく、独自のモデルによってサイズ設定されます。より小さいウィンドウを持つモデルに委譲すると、そのサブエージェントにより小さいウィンドウが与えられます。

サブエージェントを再開する

各サブエージェント呼び出しは、以前のものを続行するのではなく、新しいインスタンスを作成します。既存のサブエージェントの作業を最初からやり直すのではなく続行するには、Claude に再開するよう依頼してください。

再開されたサブエージェントは、すべての前のツール呼び出し、結果、推論を含む完全な会話履歴を保持します。サブエージェントが **独自のバックグラウンドサブエージェント** (ページ 349) を生成した場合、その履歴には、実行中にそれらが配信した結果が含まれます。サブエージェントは新規に開始するのではなく、停止した場所から正確に再開します。

- サブエージェントが完了すると、Claude はそのエージェント ID を受け取ります。
- 組み込みの Explore および Plan エージェントは 1 回限りで、エージェント ID を返さないため、Claude はそれらを再開できません。作業を続ける必要がある場合は、`general-purpose` またはカスタムサブエージェントを使用してください。
- サブエージェントが *`maxTurns`* (ページ 326) 制限で停止すると、Claude Code は返された出力を部分的としてマークします。エージェント ID を返すサブエージェントの場合、Claude Code は、Claude がサブエージェントにメッセージを送信して停止した場所から続行できることを結果に記載します。

Claude は `SendMessage` ツールを使用し、エージェント ID または名前を `to` フィールドとして使用して再開します。`SendMessage` は エージェントチーム が有効になっている必要はありません。`shutdown_request` や `plan_approval_response` などの構造化チームプロトコルメッセージのみが必要です。サブエージェントとチームメイトを超えて、クロスセッションメッセージングが有効なセッションでは、Claude は同じツールを使用して 他の Claude Code セッション にメッセージを送信でき、このマシンまたは それを超えて です。

サブエージェントを再開するには、Claude に前の作業を続行するよう依頼してください。

```
code-reviewer サブエージェントを使用して認証モジュールをレビューしてください
[エージェントが完了します]

そのコードレビューを続行し、認可ロジックを分析してください
[Claude は前の会話からの完全なコンテキストでサブエージェントを再開します]
```

Claude が完了したサブエージェントに `SendMessage` ツールでメッセージを送信すると、サブエージェントは新しい `Agent` 呼び出しなしでバックグラウンドで再開されます。Claude が `TaskStop` ツールで停止したサブエージェントにも、停止した実行が終了した後は同じことが適用されます。再開された実行は **サブエージェントが最初に実行された場所からのツールセット** (ページ 344) を保持し、元の実行が温めたプロンプトキャッシュ を読み続けることができます。

`SendMessage` ツールを持つサブエージェントもそのメッセージを送信できます。対話セッションでは、再開されたエージェントはメイン会話ではなく、それを再開したサブエージェントに報告します。そのサブエージェントは、独自の作業を完了する前に結果を待ちます。サブエージェントが、その起動者など、報告先のエージェントにメッセージを送信すると、Claude Code はそのエージェントを結果をリダイレクトせずに再開します。

ユーザー自身が停止したサブエージェント（`/tasks` の `x` または SDK `stop_task` リクエスト）は自動再開しません。Claude がメッセージを送信すると、メッセージは拒否され、Claude にはエージェントがキャンセルされたことが伝えられます。

そのサブエージェントの行がサブエージェントパネルにまだある (ページ 344) 間に、そのトランスクリプトに入力して自分で再開します。その後、Claude からのメッセージは再び自動再開できます。

再開は同じ ID の下でエージェントの新しい実行を開始するため、既に失敗または完了したサブエージェントはタスクリストと Agent SDK のタスクイベントで再び実行中として表示されます。v2.1.205 より前では、再開された実行が動作している間、以前の失敗または完了ステータスを表示し続けました。

`SendMessage` は、名前が会話の前半で到達したのと同じエージェントを引き続き参照していることを確認します。新しいエージェントが名前を取得した場合（例えば、名前を再利用した再生成されたバックグラウンドエージェント）、Claude Code は間違ったエージェントに配信するのではなく送信を拒否し、エラーは名前が現在到達するエージェントを報告するため、Claude は再ターゲットできます。以前のエージェントがまだ実行中の間にそれに到達するには、Claude はそのエージェントを生成したときに受け取ったエージェント ID でアドレス指定します。チェックは現在の会話にスコープされ、`/clear` でリセットされます。

サブエージェントは、それを起動したエージェントからのメッセージを、タスク中のコース修正を含め通常のタスク指示として扱い、独自の権限設定内でそれに基づいて行動します。誰がメッセージを送信したかに関わらず、2 つの制限が適用されます。どのエージェントからのメッセージも保留中の権限プロンプトに対するユーザーの承認としてカウントされず、エージェントメッセージはサブエージェントの権限設定、`CLAUDE.md`、または設定を変更できません。権限システムまたはユーザー自身のメッセージのみが承認を付与できます。

エージェント ID を明示的に参照したい場合は Claude に依頼することもできます。または、`~/.claude/projects/{project}/{sessionId}/subagents/` のトランスクリプトファイルで ID を見つけます。各トランスクリプトは `agent-{agentId}.jsonl` として保存されます。

サブエージェントトランスクリプトはメイン会話から独立して永続化されます。

- **メイン会話のコンテキスト圧縮**: メイン会話が圧縮されても、サブエージェントトランスクリプトは影響を受けません。別のファイルに保存されます。
- **セッション永続化**: サブエージェントトランスクリプトはセッション内で永続化されます。Claude Code を再起動して同じセッションを再開することで、**サブエージェントを再開** (ページ 352) できます。
- **自動クリーンアップ**: Claude Code は、`cleanupPeriodDays` 保持期間（デフォルト 30 日）の後、保持スイープルール に従ってサブエージェントトランスクリプトを削除します。

自動圧縮

サブエージェントはメイン会話と同じロジックを使用して自動圧縮をサポートします。圧縮は同じ条件下でトリガーされ、`CLAUDE_AUTOCOMPACT_PCT_OVERRIDE` はサブエージェントにも適用されます。上書きがいつ有効になるかについては、環境変数 を参照してください。

圧縮イベントはサブエージェントトランスクリプトファイルに記録されます。

```
{
  "type": "system",
  "subtype": "compact_boundary",
  "compactMetadata": {
    "trigger": "auto",
    "preTokens": 167189
```

(次のページに続く)

(前のページからの続き)

```
  }
}
```

preTokens 値は、圧縮が発生する前に使用されたトークン数を示します。

18.5 現在の会話をフォークする

注釈:

フォークされたサブエージェントは /subtask で実行され、Claude Code v2.1.212 以降が必要です。エージェントビューがオフになっている場合、/subtask は利用できず、/fork がフォークされたサブエージェントを開始します。それ以外の場合、/fork はセッション全体を新しいバックグラウンドセッションにコピーします。

フォークは、これまでの会話全体を継承するサブエージェントです。これにより、サブエージェントが通常提供する入力分離が削除されます。フォークはメインセッションと同じシステムプロンプト、ツール、モデル、およびメッセージ履歴を表示するため、状況を再度説明することなく、サイドタスクを渡すことができます。フォークのツール呼び出しはまだ会話から除外され、最終結果のみが返されるため、メインコンテキストウィンドウはクリーンなままです。フォークを使用する場合は、他のサブエージェントが有用であるには背景が多すぎる場合、または同じ開始点から複数のアプローチを並行して試したい場合です。

Claude は Agent ツールを通じて fork サブエージェントタイプをリクエストすることでフォークを開始します。これを制御するのは**フォークモード** (ページ 356) で、インタラクティブセッションではデフォルトでオンになっています。

/subtask の後にタスクを続けることで、フォークモードがオンかどうかに関わらず、自分でフォークを開始できます。v2.1.161 から v2.1.211 ではコマンドは /fork です。Claude Code はフォークにタスクの最初の単語から名前を付けます。次の例は、メインセッションで実装を続ける間に、会話をドラフトテストケースにフォークします：

```
/subtask draft unit tests for the parser changes so far
```

フォークはプロンプト入力の下のパネルに表示され、作業を続ける間にバックグラウンドで実行されます。完了すると、その結果がメイン会話にメッセージとして到着します。次のセクションでは、実行中のフォークを監視して操作するためのパネルコントロールについて説明します。

18.5.1 実行中のフォークを観察して操作する

実行中のフォークはプロンプト入力の下のパネルに表示され、メインセッション用に 1 行、各フォーク用に 1 行があります。

フォークが正常に完了すると、Claude Code はその行を削除します。Claude Code は失敗したフォークまたは停止したフォークの行を 30 秒間保持します。これは**他のバックグラウンドサブエージェントと同じ** (ページ 344) です。v2.1.232 より前では、Claude Code は完了したフォークの行も 30 秒間保持していました。

これらのキーを使用してパネルと対話します：

キー	アクション
↑ / ↓	行間を移動
Enter	選択したフォークのトランスクリプトを開き、フォローアップメッセージを送信
x	実行中の場合は選択したフォークを停止するか、実行中でなくなった場合はその行を閉じます。メインセッション行またはトランスクリプトを Enter で開いたフォークの行では、x はプロンプトに入力します
Esc	フォーカスをプロンプト入力に戻す

フォークまたはサブエージェントのトランスクリプトが開いている場合、フォローアップメッセージと**スキル** (ページ 275) はそのエージェントに送信され、組み込みコマンドはメイン会話に送信されます。ただし、次の安全策があります：

- /compact、/clear、/rewind はメイン会話に対して動作するため、このビューからこれらのいずれかを実行する前に、Claude Code は確認を求めます。
- /model と /fast は、表示しているエージェントではなくメイン会話のモデルと fast mode を設定するため、このビューからは実行されません。その理由は通知で表示されます。

表示しているエージェントが待機中の作業が完了する前にメッセージを読むようにするには、*Ctrl+Enter* **または** *Ctrl+X Ctrl+S* (ページ 767) で送信します。エージェントが待機しているシェルコマンドやサブエージェントのうち、バックグラウンドに移動できるものはバックグラウンドに移動し、実行を続けます。エージェントが応答を書いている場合、またはバックグラウンドに移動できない作業を待っている場合は、そのまま処理を続け、それが完了した時点でメッセージを読みます。Claude Code v2.1.286 以降が必要です。

18.5.2 フォークと他のサブエージェントの違い

フォークはメインセッションがその時点で持っているすべてを継承します。他のサブエージェントはその定義から開始します。

	フォーク	非フォークサブエージェント
コンテキスト	完全な会話履歴	渡すプロンプトを使用した新しいコンテキスト
システムプロンプトとツール	メインセッションと同じ	サブエージェントの**定義ファイル** (ページ 324) から、**バックグラウンド実行用にフィルタリング** (ページ 331)
モデル	メインセッションと同じ	サブエージェントの model フィールドから
権限	プロンプトがターミナルに表示	**バックグラウンド実行時にメインセッションに表示** (ページ 344)
プロンプトキャッシュ	メインセッションと共有	別のキャッシュ

フォークのシステムプロンプトとツール定義は親と同じであるため、最初のリクエストは親のプロンプトキャッシュを再利用します。これにより、同じコンテキストが必要なタスクの場合、フォークは新しいサブエージェントをスポーンするよりも安価です。

Claude が Agent ツール経由でフォークをスポーンするときに、`isolation: "worktree"` を渡すことができるため、フォークのファイル編集は、チェックアウトではなく、別の git worktree に書き込まれます。フォークはさらにフォークをスポーンできません。

18.5.3 フォークモードをオンまたはオフにする

Claude Code はインタラクティブセッションではフォークモードをデフォルトでオンにし、**非インタラクティブモード** (ページ 695)（`-p` 付き）および Agent SDK ではデフォルトでオフにします。インタラクティブデフォルトには Claude Code v2.1.232 以降が必要です。それより前のバージョンでは、`CLAUDE_CODE_FORK_SUBAGENT` を `1` に設定してフォークモードをオンにします。

フォークモードがオンであることは、Claude Code が Agent ツールを処理する方法からわかります：

- Claude は `fork` サブエージェントタイプをリクエストすることでフォークをスポーンできます。Claude がタイプをリクエストしない場合、セッションがまだそのタイプを持っていれば**汎用** (ページ 318) サブエージェントを取得します。Explore などの定義から生成されたサブエージェントは通常通り機能します。
- Claude Code は Claude がスポーンするサブエージェント（フォークと非フォークサブエージェント両方）をバックグラウンドで実行します。ただし**フォアグラウンドに留まるケース** (ページ 344) は除きます。Claude Code は Agent ツールの `run_in_background` パラメータも削除するため、Claude はフォアグラウンドをリクエストできません。

`CLAUDE_CODE_FORK_SUBAGENT` 環境変数を設定してデフォルトをオーバーライドします：

- `1` は非インタラクティブモードおよび Agent SDK でもフォークモードをオンにします
- `0` はすべての種類のセッションでフォークモードをオフにします

フォークモードをオンに保ちながら Claude がフォークをスポーンするのを停止するには、`Agent(fork)` ルールで *fork* **サブエージェントタイプを拒否** (ページ 340) します。Claude Code は Claude がスポーンするサブエージェントをバックグラウンドで実行し続けます。ただし同じ**フォアグラウンドに留まるケース** (ページ 344) は除きます。

18.6 サブエージェントの例

これらの例は、サブエージェントを構築するための効果的なパターンを示しています。出発点として使用するか、Claude を使用してカスタマイズされたバージョンを生成します。

Tip:

ベストプラクティス：

- **焦点を絞ったサブエージェントを設計する：** 各サブエージェントは 1 つの特定のタスクに優れている必要があります
- **各サブエージェントを単独で指定する説明を書く：** Claude は説明を使用して委譲するかどうかを決定します。各説明は正しいサブエージェントにルーティングするのに十分な具体性を持たせ、組み合わせたセット全体を *15,000* **トークン説明予算** (ページ 343) 内に保ちます
- **ツールアクセスを制限する：** セキュリティと焦点のために必要な権限のみを付与します
- **バージョン管理にチェックインする：** プロジェクトサブエージェントをチームと共有します

18.6.1 コードレビュアー

コードを変更せずにレビューする読み取り専用サブエージェント。この例は、制限されたツールアクセス（Edit と Write を除外）と、何を探すべきか、出力をどのようにフォーマットするかを正確に指定する詳細なプロンプトを使用して、焦点を絞ったサブエージェントを設計する方法を示しています。

```
---
name: code-reviewer
description: Expert code review specialist. Proactively reviews code for quality,␣
↪security, and maintainability. Use immediately after writing or modifying code.
tools: Read, Grep, Glob, Bash
model: inherit
---

You are a senior code reviewer ensuring high standards of code quality and security.

When invoked:
1. Run git diff to see recent changes
2. Focus on modified files
3. Begin review immediately

Review checklist:
- Code is clear and readable
- Functions and variables are well-named
- No duplicated code
- Proper error handling
- No exposed secrets or API keys
- Input validation implemented
- Good test coverage
- Performance considerations addressed

Provide feedback organized by priority:
- Critical issues (must fix)
```

(次のページに続く)

(前のページからの続き)

```
- Warnings (should fix)
- Suggestions (consider improving)

Include specific examples of how to fix issues.
```

18.6.2 デバッガー

問題を分析して修正できるサブエージェント。コードレビュアーとは異なり、このサブエージェントはバグの修正にはコード変更が必要なため、Edit を含みます。プロンプトは診断から検証までの明確なワークフローを提供します。

```
---
name: debugger
description: Debugging specialist for errors, test failures, and unexpected behavior.␣
↪Use proactively when encountering any issues.
tools: Read, Edit, Bash, Grep, Glob
---

You are an expert debugger specializing in root cause analysis.

When invoked:
1. Capture error message and stack trace
2. Identify reproduction steps
3. Isolate the failure location
4. Implement minimal fix
5. Verify solution works

Debugging process:
- Analyze error messages and logs
- Check recent code changes
- Form and test hypotheses
- Add strategic debug logging
- Inspect variable states

For each issue, provide:
- Root cause explanation
- Evidence supporting the diagnosis
- Specific code fix
- Testing approach
- Prevention recommendations

Focus on fixing the underlying issue, not the symptoms.
```

18.6.3 データサイエンティスト

データ分析作業向けのドメイン固有のサブエージェント。この例は、典型的なコーディングタスク以外の特化したワークフロー向けのサブエージェントを作成する方法を示しています。より有能な分析のために `model: sonnet` を明示的に設定します。

```
---
name: data-scientist
description: Data analysis expert for SQL queries, BigQuery operations, and data␣
↪insights. Use proactively for data analysis tasks and queries.
tools: Bash, Read, Write
model: sonnet
---

You are a data scientist specializing in SQL and BigQuery analysis.

When invoked:
1. Understand the data analysis requirement
2. Write efficient SQL queries
3. Use BigQuery command line tools (bq) when appropriate
4. Analyze and summarize results
5. Present findings clearly

Key practices:
- Write optimized SQL queries with proper filters
- Use appropriate aggregations and joins
- Include comments explaining complex logic
- Format results for readability
- Provide data-driven recommendations

For each analysis:
- Explain the query approach
- Document any assumptions
- Highlight key findings
- Suggest next steps based on data

Always ensure queries are efficient and cost-effective.
```

18.6.4 データベースクエリバリデーター

Bash アクセスを許可しますが、読み取り専用 SQL クエリのみを許可するようにコマンドを検証するサブエージェント。この例は、`tools` フィールドが提供するよりも細かい制御が必要な場合に、`PreToolUse` hooks を使用して条件付き検証を行う方法を示しています。

```
---
name: db-reader
```

(次のページに続く)

(前のページからの続き)

```
description: Execute read-only database queries. Use when analyzing data or␣
↪generating reports.
tools: Bash
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/validate-readonly-query.sh"
---

You are a database analyst with read-only access. Execute SELECT queries to answer␣
↪questions about the data.

When asked to analyze data:
1. Identify which tables contain the relevant data
2. Write efficient SELECT queries with appropriate filters
3. Present results clearly with context

You cannot modify data. If asked to INSERT, UPDATE, DELETE, or modify schema, explain␣
↪that you only have read access.
```

Claude Code は *hook* **入力を** *JSON* **として** (ページ 495)stdin を通じて hook コマンドに渡します。検証スクリプトはこの JSON を読み取り、実行されるコマンドを抽出し、SQL 書き込み操作のリストに対してチェックします。書き込み操作が検出された場合、スクリプトは**終了コード** *2* (ページ 472) で終了して、stderr を通じて Claude にエラーメッセージを返します。

プロジェクト内の任意の場所に検証スクリプトを作成します。パスは hook 設定の command フィールドと一致する必要があります：

```
#!/bin/bash
# Blocks SQL write operations, allows SELECT queries

# Read JSON input from stdin
INPUT=$(cat)

# Extract the command field from tool_input using jq
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command // empty')

if [ -z "$COMMAND" ]; then
  exit 0
fi

# Block write operations (case-insensitive)
if echo "$COMMAND" | grep -iE '\
```

(次のページに続く)

(前のページからの続き)

```
↪b(INSERT|UPDATE|DELETE|DROP|CREATE|ALTER|TRUNCATE|REPLACE|MERGE)\b' > /dev/null;␣
↪then
  echo "Blocked: Write operations not allowed. Use SELECT queries only." >&2
  exit 2
fi

exit 0
```

macOS と Linux では、スクリプトを実行可能にします：

```
chmod +x ./scripts/validate-readonly-query.sh
```

Windows では、検証スクリプトを PowerShell で記述し、hook エントリに `shell: powershell` を追加します。*PowerShell で hooks を実行する* (ページ 561) を参照してください。

hook は stdin を通じて JSON を受け取り、Bash コマンドは `tool_input.command` にあります。終了コード 2 は操作をブロックし、エラーメッセージを Claude にフィードバックします。終了コードと出力の詳細については *Hooks* (ページ 469) を参照し、完全な入力スキーマについては *Hook 入力* (ページ 495) を参照してください。

システムプロンプトはサブエージェントに書き込みリクエストを拒否するよう指示するため、hook はバックストップです。サブエージェントが書き込みを試みた場合、Claude Code はコマンドをブロックし、サブエージェントは `Blocked: Write operations not allowed. Use SELECT queries only.` メッセージを表示します。

18.7 次のステップ

サブエージェントを理解したので、これらの関連機能を探索してください：

- プラグインでサブエージェントを配布することで、チームまたはプロジェクト全体でサブエージェントを共有します
- *Claude Code をプログラムで実行する* (ページ 695) ことで、Agent SDK を使用して CI/CD と自動化を行います
- *MCP サーバーを使用する* (ページ 363) ことで、サブエージェントに外部ツールとデータへのアクセスを提供します

第19章　MCP を使用して Claude Code をツールに接続する

Model Context Protocol を使用して Claude Code をツールに接続する方法を学びます。

Claude Code は、AI ツール統合のためのオープンソース標準である Model Context Protocol (MCP) を通じて、数百の外部ツールとデータソースに接続できます。MCP サーバーは Claude Code にツール、データベース、API へのアクセスを提供します。

別のツール（課題追跡ツールや監視ダッシュボードなど）からチャットにデータをコピーしている場合は、サーバーを接続してください。接続すると、Claude は貼り付けたものから作業する代わりに、そのシステムを直接読み取り、操作できます。

初めてサーバーを接続する場合は、ステップバイステップのウォークスルーについて MCP クイックスタート から始めてください。このページは完全なリファレンスです。

19.1 MCP でできること

MCP サーバーが接続されている場合、Claude Code に以下のことを依頼できます：

- **課題追跡ツールから機能を実装する**：「JIRA の課題 ENG-4521 に記載されている機能を追加し、GitHub に PR を作成してください。」
- **監視データを分析する**：「Sentry と Statsig をチェックして、ENG-4521 に記載されている機能の使用状況を確認してください。」
- **データベースをクエリする**：「PostgreSQL データベースに基づいて、ENG-4521 機能を使用した 10 人のランダムなユーザーのメールアドレスを検索してください。」
- **デザインを統合する**：「Slack に投稿された新しい Figma デザインに基づいて、標準メールテンプレートを更新してください。」
- **ワークフローを自動化する**：「新機能に関するフィードバックセッションに招待する 10 人のユーザーに Gmail ドラフトを作成してください。」
- **外部イベントに対応する**：MCP サーバーは チャネル として機能することもでき、セッションにメッセージをプッシュするため、Claude は離席中に Telegram メッセージ、Discord チャット、または webhook イベントに対応できます。

19.2 MCP サーバーを検索してビルドする

Anthropic Directory でレビュー済みのコネクターを参照してください。Directory コネクターは Claude Code と同じ MCP インフラストラクチャを使用しているため、`claude mcp add` を使用して、そこにリストされているリモートサーバーを追加できます。

警告:

接続する前に、各サーバーを信頼できることを確認してください。外部コンテンツを取得するサーバーは、**プロンプトインジェクションリスク** (ページ 1034) にあなたを晒す可能性があります。

独自のサーバーをビルドするには、プロトコルの基礎については MCP サーバーガイド を、認証、テスト、Directory への提出については Claude コネクター構築ドキュメント を参照してください。

公式の mcp-server-dev プラグイン を使用して、Claude にサーバーをスキャフォールドしてもらうこともできます。

1. **プラグインをインストールする**

   VS Code 拡張機能またはデスクトップアプリでは、この手順の代わりにプラグインをインストールするの手順に従ってください。ターミナルでは、`claude` を実行して Claude Code を起動し、そのプロンプトで以下を入力します：

   ```
   /plugin install mcp-server-dev@claude-plugins-official
   ```

   インストールが失敗した場合は、Claude Code が報告するメッセージに一致させてください：

   - `Marketplace "claude-plugins-official" not found`：`/plugin marketplace add anthropics/claude-plugins-official` でマーケットプレイスを追加してから、インストールを再試行してください。
   - プラグインがマーケットプレイスで見つかりません：プラグイン名を確認してください。

   インストール概要が `Run /reload-plugins to activate.` を報告する場合、Claude Code はその後、そのリロードを実行します。リロードが次のメッセージが会話を再度読み込むことになると警告する場合は、`/reload-plugins --force` を実行してください。

2. **ビルドスキルを実行する**

   ```
   /mcp-server-dev:build-mcp-server
   ```

   Claude があなたのユースケースについて質問し、リモート HTTP またはローカル stdio サーバーをスキャフォールドします。

19.3 MCP サーバーのインストール

MCP サーバーは、ニーズに応じてさまざまな方法で設定できます。

19.3.1 オプション 1: リモート HTTP サーバーを追加する

HTTP サーバーは、リモート MCP サーバーに接続するための推奨オプションです。これはクラウドベースのサービスに対して最も広くサポートされているトランスポートです。

```
# 基本的な構文
claude mcp add --transport http <name> <url>

# 実際の例: Notion に接続
claude mcp add --transport http notion https://mcp.notion.com/mcp

# Bearer トークン付きの例
claude mcp add --transport http secure-api https://api.example.com/mcp \
  --header "Authorization: Bearer your-token"
```

`.mcp.json`、`~/.claude.json`、または `claude mcp add-json` で JSON を使用して MCP サーバーを設定する場合、`type` フィールドは `http` のエイリアスとして `streamable-http` を受け入れます。MCP 仕様ではこのトランスポートに `streamable-http` という名前を使用しているため、サーバードキュメントからコピーされた設定は変更なしで機能します。

`url` を持つが `type` を持たない JSON エントリは設定エラーです。Claude Code は `type` を持たないエントリを stdio サーバーとして読み込むためです。Claude Code はそのサーバーをスキップし、`MCP server "<name>" has a "url" but no "type"; add "type": "http" (or "sse" / "ws") to this entry` と報告します。v2.1.202 より前では、Claude Code はこの設定ミスを `command: expected string, received undefined` と報告していました。

SDK ホストアプリケーション（Agent SDK アプリケーションや **デスクトップアプリ** (ページ 137) など）のみが、インプロセス `"type": "sdk"` サーバーを登録できます。Claude Code は `.mcp.json`、`~/.claude.json`、または設定の `"type": "sdk"` エントリをスキップし、`Skipped — MCP server "<name>" declares type "sdk", which only an SDK host application can register` と報告します。

`--output-format stream-json` 実行では、Claude Code はスキップされた `--mcp-config` エントリを `system/init` イベントの *mcp_server_errors* **フィールド** (ページ 700) でも報告するため、スクリプトはサーバーが読み込まれなかったことを検出できます。これには Claude Code v2.1.219 以降が必要です。

19.3.2 オプション 2: リモート SSE サーバーを追加する

警告:

SSE（Server-Sent Events）トランスポートは非推奨です。利用可能な場合は HTTP サーバーを使用してください。

一部のサービスは SSE エンドポイントのみを公開しています。これらを *HTTP* **サーバー** (ページ 365) と同じ `claude mcp add --transport http <name> <url>` コマンドで追加してください。Claude Code は最初に HTTP トランスポートを試し、サーバーがそれを受け入れない場合は SSE に切り替わります。自動切り替えには Claude Code v2.1.265 以降が必要です。

以前のバージョンで、または SSE 経由で直接接続するには、代わりに `--transport sse` を渡してください。

```
# 基本的な構文
claude mcp add --transport sse <name> <url>

# 実際の例: Asana に接続
claude mcp add --transport sse asana https://mcp.asana.com/sse

# 認証ヘッダー付きの例
claude mcp add --transport sse private-api https://api.company.com/sse \
  --header "X-API-Key: your-key-here"
```

19.3.3 オプション 3: ローカル stdio サーバーを追加する

Stdio サーバーはマシン上のローカルプロセスとして実行されます。システムへの直接アクセスやカスタムスクリプトが必要なツールに最適です。

Claude Code は、生成されたサーバーの環境に CLAUDE_PROJECT_DIR を設定して、プロジェクトルートに設定します。これにより、サーバーは作業ディレクトリに依存することなくプロジェクト相対パスを解決できます。これはフックが CLAUDE_PROJECT_DIR 変数で受け取るのと同じディレクトリです。サーバープロセス内からこれを読み取ります。例えば、Node では process.env.CLAUDE_PROJECT_DIR、Python では os.environ["CLAUDE_PROJECT_DIR"] です。

CLAUDE_PROJECT_DIR は安定したプロジェクトルートであり、セッション中に作業ディレクトリを追加または削除しても変わりません。ファイルシステムアクセスを許可されたディレクトリのセットに制限するサーバーは、代わりに MCP roots/list リクエストを実装する必要があります。Claude Code は roots/list にセッションの起動ディレクトリと、--add-dir、/add-dir、または additionalDirectories 設定で付与した **追加作業ディレクトリ** (ページ 1060) をすべて返します。Claude Code はそのセットが変わるときに notifications/roots/list_changed を送信します。v2.1.203 より前では、roots/list は起動ディレクトリのみを返し、Claude Code は notifications/roots/list_changed を送信していませんでした。

この変数はサーバーの環境に設定され、Claude Code 自体の環境には設定されないため、プロジェクトスコープの .mcp.json エントリ、または ~/.claude.json 内のローカルまたはユーザースコープのサーバーエントリの command または args で ${VAR} 展開を使用して参照するには、${CLAUDE_PROJECT_DIR:-.} などのデフォルトが必要です。プラグイン提供の MCP 設定は ${CLAUDE_PROJECT_DIR} を直接置換し、デフォルトは必要ありません。

```
# 基本的な構文
claude mcp add [options] <name> -- <command> [args...]

# 実際の例: Airtable サーバーを追加
claude mcp add --env AIRTABLE_API_KEY=YOUR_KEY --transport stdio airtable \
  -- npx -y airtable-mcp-server
```

注釈:

重要: サーバー引数を -- で区切る
Stdio サーバーの場合、--（ダブルダッシュ）は Claude 自体のオプション（--transport、--env、--scope など）をサーバーを実行するコマンドと引数から分離します。-- の後のすべてはサーバーに変更されずに渡されます。
例えば：

- `claude mcp add --transport stdio myserver -- npx server` → `npx server` を実行します
- `claude mcp add --env KEY=value --transport stdio myserver -- python server.py --port 8080` → 環境に `KEY=value` を設定して `python server.py --port 8080` を実行します

-- がない場合、Claude Code はサーバーのフラグ（上記の --port など）を独自のオプションとして解析しようとします。
--env は複数の KEY=value ペアを受け入れます。サーバー名が --env の直後に来る場合、CLI は名前を別のペアとして読み込み、拒否するため、--env とサーバー名の間に --transport stdio などの別のオプションを少なくとも 1 つ配置してください。

19.3.4 オプション 4: リモート WebSocket サーバーを追加する

WebSocket サーバーは永続的な双方向接続を保持し、Claude に予期しないイベントをプッシュするリモート MCP サーバーに適しています。サーバーがリクエストにのみ応答する場合は HTTP を使用してください。HTTP は OAuth と `claude mcp add --transport` フラグをサポートしますが、WebSocket はどちらもサポートしていないためです。

WebSocket サーバーを `.mcp.json` または `claude mcp add-json` で設定します。

```
claude mcp add-json events-server \
  '{"type":"ws","url":"wss://mcp.example.com/socket","headers":{"Authorization":
↪"Bearer YOUR_TOKEN"}}'
```

`type: "ws"` エントリは `http` と同じ `url`、`headers`、`headersHelper`、`timeout`、`alwaysLoad` フィールドを受け入れます。認証はヘッダーのみなので、`headers` に静的トークンを渡すか、接続時に *`headersHelper`* (ページ 392) で生成してください。`claude mcp add --transport` フラグは `ws` を受け入れません。

19.3.5 別のクライアント向けに書かれたセットアップ指示からサーバーを追加する

MCP サーバーは Claude Code に固有ではないため、サーバーのセットアップ指示は Claude Desktop、Cursor、または別の MCP クライアント向けに書かれている可能性があり、`claude mcp add` コマンドを提供していない場合があります。それでもサーバーを追加するには、指示の中から URL、起動コマンド、または JSON ブロックを探してください。

- **URL**（`https://mcp.example.com/mcp` など）：サーバーはリモートです。
- **起動コマンド**（`npx -y @example/mcp-server` など）：サーバーはマシン上で実行されます。
- `mcpServers` **JSON ブロック**: 別のクライアントの設定ファイル向けに書かれた設定。

各々は *MCP* **サーバーのインストール** (ページ 364) の 4 つのオプションが取る入力の 1 つです。以下で持っている形状を見つけて、Claude Code が受け入れるコマンドに変換してください。各コマンドは `--scope project` または `--scope user` を追加しない限り、**ローカルスコープ** (ページ 381) に書き込みます。

URL から

URL はサーバーがリモートであることを意味します。`https://` エンドポイントの場合、`--transport http` で追加するか、指示が SSE を使用するエンドポイントを示している場合は **オプション** *2* (ページ 365) に従ってください。`wss://` エンドポイントの場合、`--transport` は `ws` を受け入れないため、代わりに **オプション** *4* (ページ 367) を使用してください。

```
claude mcp add --transport http example https://mcp.example.com/mcp
```

指示が API キーまたはトークンヘッダーも提供する場合、**オプション** *1* (ページ 365) に示されているように `--header` で渡してください。

npx、uvx、またはバイナリコマンドから

起動コマンドはサーバーがローカル stdio プロセスとして実行されることを意味します。コマンド全体を `--` の後に配置して、Claude Code が `-y` などのフラグをサーバーを起動するコマンドに渡し、独自のオプションとして読み込まないようにします。指示が要求する環境変数を `--env` で渡します。サーバー名の後、`--` の前に渡します。

```
claude mcp add example --env API_KEY=your-key -- npx -y @example/mcp-server
```

オプション *3* (ページ 366) は `--` セパレータを完全にカバーしています。

mcpServers JSON ブロックから

Claude Desktop などの別の MCP クライアント向けに書かれた `mcpServers` ブロックは、Claude Code が読み込むラッパーキーとエントリ形状を使用します。`claude mcp add-json` には、ラッパーではなく `mcpServers` 内のオブジェクトを渡します。2 つのエントリは最初に修復が必要です。

- `type` **のない** `url`: エンドポイントに一致するように `"type": "http"`、`"type": "sse"`、または `"type": "ws"` を追加してください。Claude Code は `type` を持たないエントリを stdio サーバーとして読み込むため、`type` のない `url` エントリは失敗します。
- **文字、数字、ハイフン、アンダースコア以外の文字を持つキー**: これらの文字のみを使用するサーバー名を選択してください。そうでない場合、キーはサーバー名です。

例えば、このブロック：

```
{
  "mcpServers": {
    "example": {
      "command": "npx",
      "args": ["-y", "@example/mcp-server"]
    }
```

(次のページに続く)

(前のページからの続き)

```
  }
}
```

このコマンドになります：

```
claude mcp add-json example '{"command":"npx","args":["-y","@example/mcp-server"]}'
```

JSON **設定から** *MCP* **サーバーを追加する** (ページ 395) はシェルエスケープと `add-json` の `--scope` フラグをカバーしています。代わりにチームと共有するには、`--scope project` を追加するか、プロジェクトルートの `.mcp.json` の `mcpServers` の下にエントリを追加してコミットしてください。**プロジェクトスコープ** (ページ 382) は Claude Code がそのファイルをどのように読み込み、承認するかをカバーしています。

各 `claude mcp add` と `claude mcp add-json` コマンドは成功時に `Added ...` 行を出力します。Claude Code が接続したことを確認するには、`claude mcp get <name>` を実行してください。**サーバーステータス** (ページ 369) はそれが表示するステータスと `.mcp.json` サーバーの承認ステップをカバーしています。

19.3.6 サーバーの管理

設定されたら、これらのコマンドで MCP サーバーを管理できます。

```
# すべての設定されたサーバーをリストする
claude mcp list

# 特定のサーバーの詳細を取得する
claude mcp get notion

# サーバーを削除する
claude mcp remove notion

# （Claude Code 内）サーバーステータスを確認する
/mcp
```

リモートサーバーを削除すると、Claude Code はそのサーバー用に保存した OAuth トークンとクライアント登録も削除します。

サーバーステータス

`claude mcp add` は `Added ...` 行を出力して成功した追加を確認します。これは設定が書き込まれたことを意味します。コマンドが `was not saved` メッセージを出力する場合は、MCP server was not saved or removed を参照してください。`may not have been saved` メッセージの場合は、MCP server may not have been saved or removed を参照してください。

`claude mcp list` は、リストする各サーバーの横に ✓ `Connected`、! `Needs authentication`、☒ `Failed to connect` などの健全性ステータスを表示します。失敗ステータスは Claude Code がそのサーバーに接続できなかったことを意味し、list コマンドが失敗したことではありません。

このリストのステータスは接続試行ではなく設定決定を報告するため、Claude Code はサーバーに接続せずにそれらを出力します。

- ☒ Pending approval (run `claude` to approve): まだ承認していない .mcp.json からのプロジェクトスコープサーバー。Claude Code はそれを claude mcp list と claude mcp get <name> の両方に表示します。対話的に claude を実行して、それを確認して承認してください。
- ☒ Rejected (see disabledMcpjsonServers in settings): disabledMcpjsonServers エントリが拒否する .mcp.json サーバー。Claude Code はそれを claude mcp get <name> にのみ表示します。
- ⊘ Disabled for this project (re-enable via /mcp): プロジェクトの *disabledMcpServers* (ページ 373) リストが名前を付けるサーバー。Claude Code はそれを claude mcp list と claude mcp get <name> の両方に表示します。/mcp パネルからサーバーをオンに戻してください。

WebSocket サーバーは claude mcp list 出力に表示されません。claude mcp get <name> または /mcp パネルを使用してそれらを確認してください。

プロジェクトサーバーの承認とワークスペーストラスト

v2.1.196 以降、claude mcp list と claude mcp get は .mcp.json 承認を、claude を実行してワークスペーストラストダイアログを受け入れるまでリポジトリにチェックインされていない設定ファイルからのみ読み込みます。クローンされたリポジトリは独自のサーバーを承認できません。プロジェクトの .claude/settings.json にコミットされた enableAllProjectMcpServers または enabledMcpjsonServers は信頼されていないフォルダでは無視され、サーバーは接続されて健全性チェックされる代わりに ☒ Pending approval のままです。

これらのソースからの承認は信頼されていないフォルダでも適用されます。

- ユーザー ~/.claude/settings.json
- 管理設定
- --settings で渡された設定

Claude Code はまた、追跡されていない .claude/settings.local.json からの承認を適用しますが、ファイルが追跡されているかどうかを確認するために git を実行し、その確認は **信頼されたフォルダ** (ページ 1064) でのみ実行されます。信頼したことのないフォルダでは、Claude Code はトラストダイアログを待ってからファイルの承認を適用します。ただし、フォルダがユーザー自身の設定ホームである場合は除きます。ホームディレクトリ、または .claude を CLAUDE_CONFIG_DIR として設定したディレクトリ。v2.1.207 より前では、Claude Code は信頼したことのないフォルダでも追跡されていない .claude/settings.local.json からの承認を適用していました。

任意の設定ファイルの disabledMcpjsonServers エントリはまだサーバーを拒否します。

サーバーステータスの詳細

/mcp（そこにあるサーバーのメニューを含む）と /plugin マネージャーでは、以前使用したリモート HTTP または SSE サーバーは cached ステータス（cached 2h ago · connects on first use · 5 tools など）を表示できます。Claude Code は起動時に接続する代わりに、前のセッションで保存された検出キャッシュからサーバーのツールリストを読み込み、Claude がサーバーのツールの 1 つを最初に呼び出すときに

サーバーを接続します。ツールは最初のメッセージから利用可能なため、何もする必要はありません。検出キャッシュとその cached ステータスには Claude Code v2.1.221 以降が必要です。

検出キャッシュはデフォルトではオフですが、段階的なロールアウトがアカウントに対して有効にしている場合を除きます。MCP_DISCOVERY_CACHE=1 を設定してオンにするか、0 を設定してロールアウトが有効にしている場合でもオフのままにしてください。v2.1.238 より前では、キャッシュはデフォルトでオンでした。

/mcp のサーバーのメニューから **Disable** または **Clear authentication** を選択すると、Claude Code はそのサーバーのキャッシュエントリも破棄します。**Reconnect** は接続されたまたは失敗したサーバーでもそれを破棄します。cached サーバーでは、**Reconnect** は今すぐサーバーを接続し、エントリを保持します。エントリを破棄した後、次に Claude Code がサーバーに接続するときは、キャッシュではなくサーバーからツールリストを取得します。

サーバーのステータスが ☒ Failed to connect の場合、claude mcp list はそのステータス行に失敗の詳細を追加し、claude mcp get <name> は Issue: 行に表示します。内容は HTTP ステータスまたはエラーコード、およびサーバーが返したエラーテキストです。/mcp のサーバーの詳細ビューにも、同じサーバー報告テキストが Issue: 行に含まれます。Claude Code はこの詳細から認証情報のようなテキストを伏せ字にし、シークレットを含む可能性がある展開済みのサーバー URL を含めることはありません。Claude Code は ☒ Connection error ステータスには詳細を追加しません。そこに出力される例外テキストにはその URL が埋め込まれる可能性があるためです。v2.1.219 より前では、両方のコマンドはステータスコードまたはサーバーのエラーテキストなしで、単なる失敗ステータスのみを表示していました。

/mcp から認証を完了し、接続が HTTP ステータスまたはトランスポートエラーコードで失敗し続ける場合、Claude Code は試行後に出力するメッセージにそのコードとサーバーの URL のオリジンを追加します。オリジンはスキームとホスト、および URL にポートが指定されている場合はポートです（例えば https://mcp.example.com）。

- パスとクエリはそのメッセージに表示されません。
- ローカル、プロジェクト、またはユーザー **スコープ** (ページ 380) のサーバー、または管理 MCP 設定のサーバーの場合、オリジンはその設定に書かれたとおりのホストを表示するため、ホストの ${VAR} 参照はメッセージで展開されません。
- ステータスまたはエラーコードのない失敗の場合、Claude Code はオリジンなしでエラーテキストを表示します。

設定に空の url を持つリモートサーバーは /mcp、claude mcp list、/plugin マネージャーで not configured として表示され、Claude Code はそれに接続しようとしません。プラグインは後で設定するコネクタのプレースホルダーエントリをこのように含めることができるため、Claude Code はそれをエラーまたはセットアップの問題として報告しません。/mcp のサーバーの詳細ビューには No URL configured for this server と表示されます。接続するにはエントリの url を設定してください。v2.1.208 より前では、Claude Code は空の url を設定の問題として報告し、再接続を促していました。

設定警告

Claude Code は以下の設定の問題について警告します。各エントリは Claude Code が何をチェックし、警告をクリアする方法を示しています。

- **隠れた空白**: Claude Code は MCP 設定値が隠れた先頭または末尾の空白を持つときに警告します。これはしばしば末尾の改行を持つトークンを貼り付けることから来ます。Claude Code は command、

`url`、各 `args` エントリ、および `env` と `headers` の下の値とキー名をチェックします。Claude Code は警告を `claude mcp list` 出力と `/mcp` に表示し、値を表示せずに影響を受けたフィールドの名前を示します。例えば `Leading or trailing whitespace in: headers.Authorization`。Claude Code は空白をトリムせず、書かれたとおりに値を使用するため、設定を編集してそれを削除してください。

- **複数のスコープで同じ名前**: 異なるエンドポイントで複数の **スコープ** (ページ 380) で同じサーバー名を定義する場合、Claude Code は `claude mcp list` 出力と `/mcp` で競合について警告します。Claude Code は OAuth サインインをエンドポイントごとに保存するため、1 つのプロジェクトで読み込まれる定義を認証しても、別の定義が読み込まれるプロジェクトでは別にサインインする必要があります。必要なエンドポイントを保持し、他を `claude mcp remove <name> --scope <scope>` で削除してください。警告では、Claude Code は各スコープのエンドポイントを設定に書かれたとおりに、*${VAR}* **参照** (ページ 383) を展開せずに引用するため、API キーなどの解決された値を表示することはありません。
- **予約名**: Claude Code は `workspace`、`claude-in-chrome`、`computer-use`、`Claude Preview`、`Claude Browser` を含む組み込みサーバーの名前を予約しています。設定が予約名を持つサーバーを定義する場合、Claude Code はロード時にそれをスキップし、名前を変更するよう求める警告を表示します。`claude mcp add` は予約名をエラーで拒否します。`Claude Preview` と `Claude Browser` はどちらも *Claude Code* **デスクトップアプリのプレビューペイン** (ページ 141) が使用する組み込みサーバーの名前です。
- **環境変数の欠落**: サーバーの設定の *${VAR}* **参照** (ページ 383) が設定されていない変数を指し、`:-default` がない場合、Claude Code は `claude mcp list` 出力と `/mcp` でその変数名を示して警告し、`${VAR}` テキストを展開しないままサーバーを読み込みます。変数を設定するか、`${VAR:-default}` フォールバックを追加してください。リモートサーバーの `url` と `headers` では、一部の認証情報変数は代わりに警告なしで **空として読み込まれます** (ページ 384)。

ツール可用性

`/mcp` パネルは各接続されたサーバーの横にツール数を表示し、ツール機能をアドバタイズするがツールを公開しないサーバーについて警告します。

リクエストがバックグラウンドでまだ接続中のサーバーからのツールを必要とする場合、Claude はそのサーバーが接続するまで待機します。待機の方法は設定によって異なります。

- **ツール検索** (ページ 406)**（デフォルト）を使用**: 待機は `ToolSearch` 呼び出し内で発生します。
- **ツール検索なし**: Claude は代わりに `WaitForMcpServers` ツールを使用します。ツール検索なしの設定には、カスタム `ANTHROPIC_BASE_URL`、`ENABLE_TOOL_SEARCH=false`、Google Cloud の Agent Platform の Claude 4.5 世代より前のモデルが含まれます。
- **Microsoft Foundry の** Azure でホストされたデプロイ: Claude Code はデプロイのサーバー側の拒否を API からしか検出できないため、Claude は `WaitForMcpServers` ではなくツール検索パスで開始します。Claude Code がそのデプロイを **事前読み込み** (ページ 406) に切り替えた後は、接続を完了したサーバーのツールが Claude の次のリクエストで利用可能になります。

ツール検索が有効な場合、Claude が作業中にサーバーが接続を完了すると、Claude Code は同じターンの次のリクエストでそのサーバーのツール名を Claude に伝えます。Claude はそれらのツールを検索して呼び出し、次のメッセージを待つことなく実行できます。

セッションを再開した後、ツールの MCP サーバーがまだ接続中の間に、Claude が保存された会話のツールを呼び出すことがあります。サーバーが最初の接続試行中である間、Claude Code はその呼び出しを最大 10 秒間保留し、ツールが利用可能になった時点で実行します。サーバーが時間内に接続しない場合、またはすでに **失敗した試行の後に再試行中** (ページ 375) である場合、呼び出しは `No such tool available` ツールエラー で失敗します。

19.3.7 サーバーを削除せずに無効にする

`/mcp` パネルでサーバーをオフに切り替えて、Claude Code がそれに接続するのを停止し、設定を失わないようにします。Claude Code はサーバーを `/mcp` にリストし、無効としてマークします。

サーバーを切り替えると、Claude Code はプロジェクトごとに `~/.claude.json` で選択を記録します。2 つのリストの 1 つで、互いに素なサーバーセットをカバーします。

- `disabledMcpServers`: ユーザー設定サーバー、プラグインサーバー、組織が 管理設定を通じて提供 するサーバー、Claude Code が **自身で取得** (ページ 398) する claude.ai コネクタ、およびデフォルトでオンの組み込みサーバーのオプトアウトリスト。Claude Code はここにリストするサーバーに接続しません。*Disable claude.ai connectors* (ページ 399) で説明されているプロジェクトごとの `/mcp` トグルで claude.ai コネクタを無効にすると、Claude Code はそれをこのリストの下に表示名で書き込みます。例えば `claude.ai Slack`。
- `enabledMcpServers`: `computer-use` などのデフォルトでオフの組み込みサーバーのオプトインリスト。Claude Code はここにリストする場合にのみデフォルトオフサーバーに接続します。

Claude Code は各サーバーに対して 2 つのリストの 1 つを正確に参照するため、どちらのリストも他方を上書きしません。通常のサーバーを `enabledMcpServers` に追加するか、デフォルトオフの組み込みサーバーを `disabledMcpServers` に追加する場合、Claude Code はエントリを無視します。

`disabledMcpServers` と `enabledMcpServers` は `enabledMcpjsonServers` と `disabledMcpjsonServers` とは無関係です。これらはプロジェクトの `.mcp.json` ファイルで定義されたサーバーの承認を制御します。

19.3.8 MCP クライアントランタイム

Claude Code は 2 つのクライアントランタイムの 1 つを通じて MCP サーバーに接続します。v1 ランタイムは MCP TypeScript SDK 1.x に基づいています。v2 ランタイムは MCP TypeScript SDK 2.0 上の同じコードで、MCP プロトコルリビジョン 2026-07-28 を追加します。このページの残りは両方のランタイムに適用されます。ただし、セクションが v2 ランタイムに名前を付ける場合を除きます。

Claude Code は起動するたびにランタイムを選択し、終了するまで保持します。フィーチャーフラグを取得するセッションでは、Claude Code v2.1.232 以降で v2 ランタイムを使用します。

フィーチャーフラグを取得しないセッションでは、Claude Code は Claude Code v2.1.274 以降でデフォルトで v2 ランタイムを使用します。

- Amazon Bedrock、Claude Platform on AWS、Google Cloud の Agent Platform、または Microsoft Foundry のセッション。ただし、Claude Code を埋め込むホストプラットフォームが `CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST` を設定する場合を除きます
- Claude apps gateway を通じてサインインしたセッション
- テレメトリまたはフィーチャーフラグ取得をオフにしたセッション。例えば `DISABLE_TELEMETRY` を使用

v2 では、Claude Code も：

- HTTP サーバーに新しいリビジョンをサポートするかどうかを尋ね、それをサポートするサーバーで使用します。フィーチャーフラグを取得するセッションでは claude.ai コネクタサーバーにも尋ね、Claude Code v2.1.285 以降では Anthropic がその変更を段階的に展開するにつれて stdio サーバーにも尋ねます。すべてのセッションでコネクタサーバーと stdio サーバーに尋ねるようにするには、`MCP_PROTOCOL_NEGOTIATION` を `auto` に設定してください。他のすべてのサーバーに v1 のように接続します。
- 新しいリビジョンのサーバーから **保持するストリーム** (ページ 374) 上で `list_changed` 通知を受け取ります。
- 新しいリビジョンで接続する **チャネル** (ページ 376) サーバーを登録しません。そのリビジョンはチャネルメッセージを運ぶことができないためです。
- 予期しない発行者を示す認可応答の *MCP OAuth* **サインイン** (ページ 386) を失敗させます。
- *MCP OAuth* (ページ 386) 認証情報を HTTPS でサービスされるトークンエンドポイント、または `localhost`、`127.0.0.1`、`::1` でのみ送信します。別の場所（ローカルネットワーク上のデバイスなど）で平文 `http://` のトークンエンドポイントを持つサーバーのサインインは失敗します。Refusing to send credentials to non-https token endpoint を参照してください。

Anthropic は特定のサーバーを以前のプロトコルに保つか、Claude Code が取得するフィーチャーフラグでそのストリームをオフにすることができます。

ランタイムを自分で選択するには、`MCP_SDK_GENERATION` を `v1` または `v2` に設定してください。Claude Code が尋ねるかどうかを決定するには、`MCP_PROTOCOL_NEGOTIATION` を `auto` または `legacy` に設定してください。

19.3.9 動的ツール更新

MCP サーバーは、接続中に提供するツール、プロンプト、リソースを変更し、`list_changed` 通知を送信できます。通知が届くと、次のように動作します。

- **対話的なターミナルセッション**では、Claude Code はそのサーバーから更新されたリストを取得するため、再接続する必要はありません。
- `-p` **フラグを使用した** **非対話モード** (ページ 695) **および** Agent SDK では、Claude Code はこれらの通知に対してツールリストのみをリフレッシュします。

リフレッシュリクエストが失敗する場合、Claude Code は後のリフレッシュが成功するまで、サーバーの以前に検出されたツール、プロンプト、リソースを保持します。v2.1.214 より前では、リフレッシュ中の一時的なエラーはサーバーのツール、プロンプト、リソースを空のリストに置き換えていました。

v2 ランタイムの通知ストリーム

v2 **ランタイム** (ページ 373) では、Claude Code は新しいプロトコルリビジョンのサーバーから保持するストリーム上で `list_changed` 通知を受け取ります。ストリームが閉じると、Claude Code はそれを再度開きます。2 つの制限があります。

- **ストリームが 10 秒以内に再度閉じる**: Claude Code はそれを最大 3 回再度開き、その接続に対して停止します。

- **ストリームが 10 秒以上開いたままで、その後閉じる**（サーバーレスホストへのストリームでよく起こります）: 1 時間に 5 回再度開いた後、Claude Code は次に再度開くまで約 6 時間待機します。

ストリームが再度開くまで、サーバーの最後に取得したツール、プロンプト、リソースを保持します。変更をより早く取得するには、`/mcp` からサーバーを再接続してください。

19.3.10 自動再接続

Claude Code はセッション中にドロップするリモートサーバーを再接続し、一時的なエラーの後に HTTP または SSE サーバーの最初の接続を再試行します。Stdio サーバーはローカルプロセスであり、Claude Code は自動的にそれらを再接続しません。

リモートサーバーのセッション中のドロップ

Claude Code はドロップしたリモートサーバーを指数バックオフで再接続します。最大 5 回試行し、1 秒の遅延から開始して毎回 2 倍にします。表示内容は Claude Code の実行方法によって異なります。

- **対話的セッション**: `/mcp` は Claude Code が再接続している間、サーバーを保留中として表示します。5 回の試行に失敗した後、Claude Code はサーバーを失敗としてマークするか、サーバーの再認可が必要な場合は認証が必要としてマークします。サーバーを失敗としてマークすると、`MCP server "<name>" disconnected · open /mcp to reconnect` 通知が表示されます。`/mcp` から手動で再試行できます。
- *`claude -p`* (ページ 695) **実行と** Agent SDK **セッション**: Claude Code は同じスケジュールで再接続し、試行を表示する `/mcp` パネルはありません。

失敗した最初の接続

HTTP または SSE サーバーの最初の接続が 5xx レスポンス、接続拒否、タイムアウトなどの一時的なエラーで失敗する場合、Claude Code は最大 3 回再試行します。接続がまだ失敗する場合、Claude Code はサーバーを失敗としてマークします。

Claude Code はこれらの場合には再試行しません。

- WebSocket サーバーの最初の接続
- 認証エラーまたは not-found エラー。解決には設定の変更が必要なためです。*`headersHelper`* (ページ 392) がサーバーの `Authorization` ヘッダーの唯一のソースである場合は、Claude Code は認証エラーでも再試行します。各試行でヘルパーを再実行し、新しい認証情報を取得できるためです

失敗した検出リクエスト

サーバーが接続した後、Claude Code は `tools/list`、`prompts/list`、`resources/list` などの機能検出リクエストを送信します。Claude Code は一時的なネットワークまたはサーバーエラーの後、短いバックオフで最大 3 回それらのリクエストを再試行します。認証エラー、4xx レスポンス、またはリクエストタイムアウトは再試行しません。

失敗したサーバーを自分で再試行する

失敗した、または認証が必要なすべてのサーバーを再試行するには、`/mcp reconnect all` を実行してください。対話的なターミナルでは、これには Claude Code v2.1.284 以降が必要です。それより前のバージョンでは、そこで `MCP server "all" not found` が出力されます。

Claude がサーバーが失敗したことを学ぶ方法

Claude Code が接続に失敗した設定されたサーバーについて Claude に伝えるかどうかは **ツール検索** (ページ 406)（デフォルトではオン）に依存します。

- ツール検索を使用している場合、Claude Code は Claude にどのサーバーが失敗したか、その接続エラーを伝えるため、Claude は応答で接続失敗を報告します。Claude Code は一致するツールを見つけない `ToolSearch` 結果にも同じ情報を含めます。
- **ツール検索なしの設定** (ページ 407) では、Claude Code は失敗したサーバー接続を Claude に報告しません。

19.3.11 チャネルでメッセージをプッシュする

MCP サーバーはまた、CI 結果、監視アラート、チャットメッセージなどの外部イベントに Claude が反応できるようにメッセージをセッションに直接プッシュできます。これを有効にするには、サーバーが `claude/channel` 機能を宣言し、起動時に `--channels` フラグでオプトインします。チャネル を使用して公式にサポートされているチャネルを使用するか、チャネルリファレンス を参照して独自に構築してください。

v2 **ランタイム** (ページ 373) では、MCP プロトコルリビジョン 2026-07-28 をネゴシエートするチャネルサーバーはチャネルメッセージを配信できないため、Claude Code はそれをチャネルとして登録しません。そのリビジョンをサポートしないチャネルサーバーは以前のハンドシェイクで接続し、従来どおり登録されます。

`MCP_PROTOCOL_NEGOTIATION` を `auto` に設定すると、Claude Code は stdio サーバーにそのリビジョンを尋ねます。Anthropic は、Claude Code が フィーチャーフラグを取得 するセッションで、Claude Code v2.1.285 以降を対象にこれをデフォルトでオンにしつつあります。stdio チャネルサーバーを以前のハンドシェイクのままにするには、`MCP_PROTOCOL_NEGOTIATION` を `legacy` に設定してください。これにより、すべてのサーバーが以前のハンドシェイクのままになります。

Tip:

ヒント：

- `-s` または `--scope` フラグを使用して、設定が保存される場所を指定します。
  - `local`（デフォルト）：現在のプロジェクトでのみ利用可能
  - `project`: `.mcp.json` ファイルを通じてプロジェクト内のすべてのユーザーと共有
  - `user`: すべてのプロジェクト全体で利用可能
- `-e` または `--env` フラグで環境変数を設定します（例えば、`-e KEY=value`）
- `--transport` と `--header` フラグは `-t` と `-H` 短形式も受け入れます
- `MCP_TIMEOUT` 環境変数を使用して MCP サーバー起動タイムアウトを設定します（例えば、`MCP_TIMEOUT=10000 claude` は 10 秒のタイムアウトを設定します）
- そのサーバーの `.mcp.json` エントリにミリ秒単位の `timeout` フィールドを追加して、サーバーごとのツール実行タイムアウトを設定します。例えば `"timeout": 600000` は 10 分です。これはそのサーバーに対してのみ `MCP_TOOL_TIMEOUT` 環境変数を上書きします
- Claude Code は MCP ツール出力が 10,000 トークンを超えるときに警告を表示し、デフォルトで出力を 25,000 トークンに制限します。制限を上げるには、`MAX_MCP_OUTPUT_TOKENS` 環境変数を設定します（例えば、`MAX_MCP_OUTPUT_TOKENS=50000`）。警告しきい値は固定です。*MCP* **出力制限と警告** (ページ 401) を参照してください
- `/mcp` を使用して、OAuth 2.0 認証が必要なリモートサーバーで認証します

サーバーごとの `timeout` はツール呼び出しごとのハードウォールクロック制限であり、サーバーからの進捗通知はそれを延長しません。1000 未満の値は無視され、`MCP_TOOL_TIMEOUT` にフォールスルーするか、その変数が設定されていない場合は約 28 時間のデフォルトにフォールスルーします。HTTP、SSE、または *claude.ai* **コネクタ** (ページ 396) サーバーの場合、サーバーの最初の応答バイトまでの各リクエストをカバーする 2 つ目の、リクエストごとのタイマーもあります。Claude Code はそのタイマーを 3 つの値の最大値に設定します。60 秒、サーバーに適用されるツールタイムアウト、`MCP_TIMEOUT`。設定されていない `MCP_TOOL_TIMEOUT` の 28 時間デフォルトはその比較に入らず、60 秒未満の値はタイマーを短縮しません。Stdio と WebSocket サーバーにはリクエストごとのタイマーがありません。

少なくとも 1000 のサーバーごとの `timeout` は、以下で説明されるアイドルタイムアウトのフロアとしても機能します。Claude Code はそのサーバーのツール呼び出しをサーバーごとの `timeout` より早くアイドルのために中止しません。Claude Code v2.1.203 以降が必要です。

アイドルウィンドウの間、応答も進捗通知も送信しない MCP サーバーへのツール呼び出しは、ウォールクロック制限を待つ代わりにエラーで中止されます。アイドルタイムアウトは IDE サーバーと SDK インプロセスサーバーを除くすべてのサーバータイプに適用されます。アイドルウィンドウは HTTP、SSE、WebSocket、*claude.ai* **コネクタ** (ページ 396) サーバーの場合は 5 分、stdio サーバーの場合は 30 分にデフォルト設定されます。v2.1.203 より前では、stdio サーバーはアイドルタイムアウトから除外されていました。

`CLAUDE_CODE_MCP_TOOL_IDLE_TIMEOUT` 環境変数をミリ秒単位で設定してアイドルウィンドウを変更するか、`0` に設定してチェックを無効にしてください。

これらのタイムアウトは呼び出しがどのくらい実行できるかを制限するものであり、セッションをどのくらいブロックするかを常に制限するわけではありません。2 分を超えて実行されるメイン会話の呼び出しは、先にバックグラウンドタスクに移動します。**長いツール呼び出しの自動バックグラウンド化** (ページ 378) を参照してください。

19.3.12 長いツール呼び出しの自動バックグラウンド化

メイン会話の MCP ツール呼び出しが 2 分後も実行中の場合、セッションをブロックする代わりにバックグラウンドタスクに移動します。Claude はタスク ID をすぐに受け取り、作業を続け、結果は呼び出しが解決するときにタスク通知として到着します。自動バックグラウンド化には Claude Code v2.1.212 以降が必要です。

タスクは `/tasks` に表示され、そこで停止することもでき、セッションを終了しても存続しません。タスクのエントリは、サーバーが報告した最新の進捗を表示します。

呼び出しがバックグラウンドで実行されている間、呼び出しごとの制限は引き続き適用されます。サーバーごとの `timeout` または `MCP_TOOL_TIMEOUT` で設定されたウォールクロック制限、および `CLAUDE_CODE_MCP_TOOL_IDLE_TIMEOUT` で設定されたアイドルタイムアウト。

`CLAUDE_CODE_MCP_AUTO_BACKGROUND_MS` 環境変数をミリ秒単位で設定してしきい値を変更するか、`0` に設定して自動バックグラウンド化をオフにしてください。`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS` を 1 に設定すると、それもオフになり、他のすべてのバックグラウンドタスク機能も同様です。

一部の呼び出しはバックグラウンドに移動しません。

- **サブエージェント** (ページ 317) からの呼び出し。Claude Code はメイン会話呼び出しのみをバックグラウンド化します
- IDE サーバーへの呼び出し
- **非対話モード** (ページ 695) での呼び出し。ただし `CLAUDE_AUTO_BACKGROUND_TASKS` が 1 に設定されている場合を除きます。1 回限りの実行は結果が到着する前に終了する可能性があるため

開いている **エリシテーションダイアログ** (ページ 405) を待つ呼び出しは、ダイアログが開いている間はバックグラウンド化されません。サーバーは遅いのではなく入力を待ってブロックされているため、Claude Code はダイアログが閉じるまで移動を延期します。

19.3.13 プラグイン提供の MCP サーバー

プラグイン は、プラグインを有効にするときにツールと統合を提供する MCP サーバーをバンドルできます。プラグイン MCP サーバーはユーザー設定サーバーと同じように機能します。

プラグイン MCP サーバーの動作方法：

- プラグインはプラグインルートの `.mcp.json` または `plugin.json` にインラインで MCP サーバーを定義します
- プラグインを有効にすると、Claude Code は自動的にその MCP サーバーを起動します
- Claude Code は手動で設定された MCP ツールと並んでプラグイン MCP ツールを提供します
- プラグインサーバーの追加と削除は、`/mcp` コマンドではなく、プラグインのインストールまたはアンインストールで行います。それでも `/mcp` で **インストールされたプラグインサーバーをオフに切り替える** (ページ 373) ことはでき、プラグインを削除せずに Claude Code がそれに接続するのを停止できます

プラグイン MCP 設定の例：

プラグインルートの `.mcp.json` で：

```
{
  "mcpServers": {
    "database-tools": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
      "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
      "env": {
        "DB_URL": "${DB_URL}"
      }
    }
  }
}
```

または plugin.json にインラインで：

```
{
  "name": "my-plugin",
  "mcpServers": {
    "plugin-api": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/api-server",
      "args": ["--port", "8080"]
    }
  }
}
```

プラグイン MCP 機能：

- **自動ライフサイクル**: サーバーはこれらのポイントで接続および切断されます。
  - セッション起動時、Claude Code は有効なプラグインのサーバーを自動的に接続します。/mcp では、以前使用したリモート（HTTP または SSE）プラグインサーバーは代わりに *cached* **ステータス** (ページ 370) を表示できます。Claude Code は Claude が最初にそのツールの 1 つを呼び出すときに接続します
  - セッション中にプラグインを有効または無効にする場合、Claude Code はその変更が適用されるときにその MCP サーバーを接続または切断します。それがいつかは プラグイン変更を再起動なしで適用 で説明しています。対話的なターミナルのないセッションでは、/reload-plugins はプラグイン MCP サーバーを接続または切断しません。これらの変更は次のセッションで有効になります
  - リロードするとき、Claude Code は設定が変更されていないプラグインサーバーのライブ接続を保持し、Agent SDK からそれらを指定せずに セッションの MCP サーバーリストを置き換える ときも同様です
  - v2.1.246 以降で */cd* **でセッションを移動する** (ページ 1061) とき、Claude Code は新しいディレクトリの設定が有効にするプラグインのサーバーを接続し、有効でなくなったプラグインのサーバーを切断するため、移動後に /reload-plugins を実行する必要はありません
  - **クラウドセッション** (ページ 223) では、まだ接続されていないプラグインサーバーへの MCP 呼び出し（アイドルセッションが起動した直後など）は、サーバーをオンデマンドで開始し、接

続を待ちます

- **パスプレースホルダー**: `${CLAUDE_PLUGIN_ROOT}` はプラグインのインストールディレクトリに解決され、`${CLAUDE_PLUGIN_DATA}` はその 永続状態 ディレクトリに解決され、`${CLAUDE_PROJECT_DIR}` は安定したプロジェクトルートに解決されます。置換は以下に適用されます。
  - `stdio` サーバー: `command`、`args`、`env`
  - `http`、`sse`、`ws` サーバー: `url`、`headers`、`headersHelper`
- **ユーザー環境アクセス**: 手動で設定されたサーバーと同じ環境変数へのアクセス
- **複数のトランスポートタイプ**: stdio、SSE、HTTP、WebSocket トランスポートのサポート。ただし、トランスポートサポートはサーバーによって異なる場合があります

プラグインサーバーは `/mcp` に表示され、プラグインから来ることを示すインジケータが付きます。

プラグインの stdio サーバーの場合、`claude mcp get` は `Command: stdio`、空の `Args:` 行、および各環境変数を `NAME=[REDACTED]` として出力します。値は認証情報を含む可能性があるため、隠されています。

プラグイン MCP ツール名：

プラグインにバンドルされた MCP サーバーからのツールは、呼び出し可能な名前にプラグイン名とサーバーキーの両方を含めます。完全な形式は `mcp__plugin_<plugin-name>_<server-name>__<tool-name>` です。`A-Z`、`a-z`、`0-9`、`_`、`-` 以外の文字は `_` に置き換えられます。`my-plugin` という名前のプラグインにバンドルされた `database-tools` サーバーの場合、`query` ツールは以下のように呼び出し可能です。

```
mcp__plugin_my-plugin_database-tools__query
```

権限ルール (ページ 1039)、スキルの `allowed-tools` リスト、**サブエージェントの *tools* フィールド** (ページ 331)、または **フック *matcher*** (ページ 454) でツールを参照するときに、この完全な名前を使用してください。素のサーバーキー（`mcp__database-tools__.*` など）に対して書かれたフック matcher は、プラグインにバンドルされたサーバーに対しては発火しません。

サーバー自体は `plugin:<plugin-name>:<server-name>`（`plugin:my-plugin:database-tools` など）のスコープ付き名前で登録されます。設定されたサーバー名が想定される場所（*mcp_tool* **フックの *server* フィールド** (ページ 461) など）ではその名前を使用してください。

プラグインで MCP サーバーをバンドルする詳細については、プラグインコンポーネントリファレンス を参照してください。

19.4 MCP インストールスコープ

MCP サーバーは 3 つのスコープで設定できます。選択するスコープは、サーバーがロードされるプロジェクトと、設定がチームと共有されるかどうかを制御します。管理者は、**マネージド設定** (ページ 410) を通じてすべてのユーザーに対してサーバーをデプロイまたは提供することもできます。

スコープ	ロード対象	チームと共有	保存場所
ローカル (ページ 381)	現在のプロジェクトのみ	いいえ	`~/.claude.json`

次のページに続く

表 1 – 前のページからの続き

スコープ	ロード対象	チームと共有	保存場所
プロジェクト (ページ 382)	現在のプロジェクトのみ	はい、バージョン管理経由	プロジェクトルートの `.mcp.json`
ユーザー (ページ 383)	すべてのプロジェクト	いいえ	`~/.claude.json`

19.4.1 ローカルスコープ

ローカルスコープはデフォルトです。ローカルスコープのサーバーは、追加したプロジェクトでのみロードされ、あなたにプライベートなままです。Claude Code は `~/.claude.json` のそのプロジェクトのパスの下に保存するため、同じサーバーは他のプロジェクトに表示されません。個人開発サーバー、実験的な設定、またはバージョン管理に含めたくない認証情報を持つサーバーにはローカルスコープを使用してください。

注釈:

MCP サーバーの「ローカルスコープ」という用語は、一般的なローカル設定とは異なります。MCP ローカルスコープのサーバーは `~/.claude.json`（ホームディレクトリ）に保存されますが、一般的なローカル設定は `.claude/settings.local.json`（プロジェクトディレクトリ内）を使用します。設定ファイルの場所の詳細については、**設定** (ページ 711) を参照してください。

```
# ローカルスコープのサーバーを追加する（デフォルト）
claude mcp add --transport http stripe https://mcp.stripe.com

# ローカルスコープを明示的に指定する
claude mcp add --transport http stripe --scope local https://mcp.stripe.com
```

コマンドは現在のプロジェクトのエントリを `~/.claude.json` に書き込みます。以下の例は、`/path/to/your/project` から実行した場合の結果を示しています：

```
{
  "projects": {
    "/path/to/your/project": {
      "mcpServers": {
        "stripe": {
          "type": "http",
          "url": "https://mcp.stripe.com"
        }
      }
    }
  }
}
```

19.4.2 プロジェクトスコープ

プロジェクトスコープのサーバーは、プロジェクトのルートディレクトリの `.mcp.json` ファイルに設定を保存することで、チーム間のコラボレーションを可能にします。プロジェクトスコープのサーバーを追加すると、Claude Code は自動的にこのファイルを作成または更新して、適切な設定構造を使用します。`.mcp.json` をバージョン管理にチェックインして、チームのすべてのメンバーが同じ MCP ツールとサービスを取得するようにしてください。

```
# プロジェクトスコープのサーバーを追加する
claude mcp add --transport http shared-server --scope project https://example.com/mcp
```

結果の `.mcp.json` ファイルは標準化された形式に従います：

```
{
  "mcpServers": {
    "shared-server": {
      "type": "http",
      "url": "https://example.com/mcp"
    }
  }
}
```

セキュリティ上の理由から、Claude Code はインタラクティブセッションで `.mcp.json` ファイルからプロジェクトスコープのサーバーを使用する前に承認を求めます。これらの承認選択をリセットするには、`claude mcp reset-project-choices` を実行してください。

`claude -p` 実行、*Agent SDK* (ページ 695) セッション、および**クラウドセッション** (ページ 223) では、Claude Code はそのプロンプトを表示できません。プロジェクトスコープのサーバーを確認なしでロードします。Claude Code はまた、ユーザー設定またはマネージド設定で `skipDangerousModePermissionPrompt` が設定された `bypassPermissions` モードで開始したセッションではプロンプトをスキップします。とにかくサーバーを除外するには：

- `disabledMcpjsonServers` に追加します。これはすべての権限モードでそれをブロックします。
- *`--setting-sources`* (ページ 51) または SDK の `settingSources` オプションでプロジェクト設定全体を除外します。
- *`--strict-mcp-config`* (ページ 51) でセッションを開始します。Claude Code は `--mcp-config` で渡した MCP サーバーのみを使用します。Claude Code がロードしていないプロジェクトスコープのサーバーの承認プロンプトをスキップするには、Claude Code v2.1.246 以降が必要です。v2.1.246 より前では、厳密なセッションでもそれらの承認を待機していたため、バックグラウンドセッションは起動時に待機していました。マネージド MCP ファイルの下でフラグが何をするかについては、マネージド mcp.json での排他的制御を参照してください。

プロジェクトサーバーの承認とワークスペーストラスト (ページ 370) は、リポジトリにコミットされた承認がワークスペーストラストとどのように相互作用するかについて説明しています。

19.4.3 ユーザースコープ

ユーザースコープのサーバーは `~/.claude.json` に保存され、クロスプロジェクトのアクセス可能性を提供し、マシン上のすべてのプロジェクト全体で利用可能になりながら、ユーザーアカウントにプライベートなままです。このスコープは、個人的なユーティリティサーバー、開発ツール、または異なるプロジェクト全体で頻繁に使用するサービスに適しています。

```
# ユーザーサーバーを追加する
claude mcp add --transport http hubspot --scope user https://mcp.hubspot.com/anthropic
```

19.4.4 スコープの階層と優先順位

同じサーバーが複数の場所で定義されている場合、Claude Code はそれに 1 回接続し、最も優先度の高いソースからの定義を使用します。そのソースからのサーバーエントリ全体が使用されます。フィールドはスコープ全体でマージされません。

1. ローカルスコープ
2. プロジェクトスコープ
3. ユーザースコープ
4. プラグイン提供サーバー
5. *claude.ai* **コネクタ** (ページ 396)

Claude Code は 3 つのスコープ全体で名前によって重複を照合します。プラグインとコネクタはエンドポイントで照合するため、上記のサーバーと同じ URL またはコマンドを指すものは重複として扱われます。

2 つの URL スペルは、スキームまたはホストの大文字小文字、スキームのデフォルトポート（`https` の `:443` など）、または末尾のスラッシュのみが異なる場合、同じエンドポイントとして扱われます。異なるパス、クエリ文字列、ユーザー情報、または非デフォルトポートは 2 つのサーバーを作成します。

組織が `managedMcpServers` マネージド設定を通じて提供するサーバーは、これらすべての上にランクされるため、それらの 1 つがそれを複製する場合、Claude Code は組織の定義に接続します。Claude Code v2.1.259 以降が必要です。

Desktop **アプリの** *Code* **タブ** (ページ 168) でローカルセッションを開く場合、`~/.claude.json`（ユーザースコープ）のトップレベルと `.mcp.json` に同じ stdio サーバー名がある場合、Code タブは `~/.claude.json` 定義を使用します。

19.4.5 `.mcp.json` での環境変数の展開

Claude Code は `.mcp.json` ファイルの環境変数の展開をサポートしており、チームが設定を共有しながら、マシン固有のパスと API キーなどの機密値の柔軟性を維持できます。

サポートされている構文

- `${VAR}` - 環境変数 `VAR` の値に展開されます
- `${VAR:-default}` - `VAR` が設定されている場合は `VAR` に展開され、そうでない場合はデフォルトを使用します

展開場所

環境変数は以下で展開できます：

- `command` - サーバー実行可能ファイルのパス
- `args` - コマンドライン引数
- `env` - サーバーに渡される環境変数
- `url` - HTTP サーバータイプの場合
- `headers` - HTTP サーバー認証の場合

変数展開を使用した例

```
{
  "mcpServers": {
    "api-server": {
      "type": "http",
      "url": "${API_BASE_URL:-https://api.example.com}/mcp",
      "headers": {
        "Authorization": "Bearer ${API_KEY}"
      }
    }
  }
}
```

デフォルトなしの未設定変数

参照される環境変数が設定されておらず、デフォルト値がない場合、設定はまだロードされます。Claude Code はそのサーバーに対して `claude mcp list` 出力で欠落変数の警告を報告し、展開されていない `${VAR}` テキストをそのまま使用します。変数を設定するか、`:-default` フォールバックを追加して、サーバーが意図した値で起動するようにしてください。リモートサーバーの `url` と `headers` では、一部の認証情報変数は**空として読み込まれます** (ページ 384)。警告はありません。

空として読み込まれる認証情報変数

リモートサーバーの `url` と `headers` では、Claude Code は環境から認証情報変数を空として読み込みます。これは、プロジェクトの `.mcp.json` またはプラグインが Claude Code またはクラウドプロバイダーの認証情報をそれが指定するサーバーに送信するのを防ぎます。`Bearer ${ANTHROPIC_AUTH_TOKEN}` と書いた場合、サーバーは認証情報なしで `Bearer` を受け取り、通常は `401` でリクエストを拒否します。Claude Code はそれを接続失敗として報告します。

対象となる名前は：

- Claude Code 独自の認証情報（ANTHROPIC_API_KEY や ANTHROPIC_AUTH_TOKEN など）
- クラウドプロバイダーの認証情報（AWS_BEARER_TOKEN_BEDROCK など）
- 環境が保持する他の認証情報（HTTPS_PROXY や NPM_TOKEN など）

対象となる名前は、変数を設定しているかどうかに関わらず空として読み込まれ、それに対する :-default フォールバックは無視されます。ANTHROPIC_BASE_URL などのプロバイダーベース URL はまだ展開されるため、"url": "${ANTHROPIC_BASE_URL}/mcp" は機能します。ただし、URL の値自体がユーザー名とパスワードなどの認証情報を埋め込んでいない限りです。

API_KEY などこのセットの外にある名前は、書かれたとおりに展開されます。サーバーに対象となる認証情報の 1 つを提供するには、それを自分の名前の変数にコピーして、その名前を参照してください。

リモートサーバーの url または headers が設定した対象変数を参照する場合、Claude Code はデバッグログ行でそれを指定します。行を読むには、claude --debug-file /tmp/claude-debug.log を実行して、そのファイルで never expanded toward a remote server を検索してください。

/mcp と CLI 出力での参照の表示方法

ローカル、プロジェクト、またはユーザー**スコープ** (ページ 380) 内のサーバーの場合、以下のサーフェスは ${VAR} 参照を解決された値ではなく名前で表示します：

- サーバーの /mcp 詳細ビューの URL またはコマンドライン
- claude mcp list と claude mcp get の出力

/mcp 詳細ビューは Claude Code v2.1.268 以降でこの方法で参照を表示します。

組織が managedMcpServers 設定を通じて提供するサーバーの場合、これらのサーフェスは URL のホストのみを表示します。

接続が失敗した場合に claude mcp list、claude mcp get、および /mcp が何を表示するかを確認するには、**サーバーステータスの詳細** (ページ 370) を参照してください。

19.5 実践的な例

19.5.1 例：コードレビューのために GitHub に接続する

GitHub のリモート MCP サーバーは、ヘッダーとして渡される GitHub 個人アクセストークンで認証します。取得するには、GitHub トークン設定を開き、Claude が操作したいリポジトリへのアクセス権を持つ新しいきめ細かいトークンを生成してから、サーバーを追加します：

```
claude mcp add --transport http github https://api.githubcopilot.com/mcp/ \
  --header "Authorization: Bearer YOUR_GITHUB_PAT"
```

YOUR_GITHUB_PAT を個人アクセストークンに置き換えてください。claude mcp add コマンドは認証情報を検証せずに設定を保存するため、ここではプレースホルダー値が受け入れられますが、サーバーは後で接続に失敗します。接続を確認するには、/mcp を実行し、サーバーが connected と表示されていること

を確認してください。認証情報が不正なサーバーは `failed` と表示され、失敗の詳細には、サーバーが返した HTTP ステータス（401 など）が含まれます。

その後、GitHub で作業します：

```
PR #456 をレビューして改善を提案してください
```

```
見つけたバグの新しい課題を作成してください
```

```
自分に割り当てられているすべてのオープン PR を表示してください
```

19.5.2 例：PostgreSQL データベースをクエリする

DBHub（`@bytebase/dbhub` パッケージ）は、`--dsn` で渡す接続文字列を通じて Claude をリレーショナルデータベースに接続する MCP サーバーです。Claude が実行するクエリがデータを変更できないように、接続文字列で読み取り専用データベースユーザーを使用してください：

```
claude mcp add --transport stdio db -- npx -y @bytebase/dbhub \
  --dsn "postgresql://readonly:pass@prod.db.com:5432/analytics"
```

サーバーが起動することを確認するには、`/mcp` を実行し、`db` が `connected` と表示されていることを確認してください。

その後、データベースを自然に照会します：

```
今月の総収益はいくらですか？
```

```
orders テーブルのスキーマを表示してください
```

```
過去 90 日間に購入していない顧客を検索してください
```

19.6 リモート MCP サーバーで認証する

多くのクラウドベースの MCP サーバーは認証が必要です。Claude Code は安全な接続のために OAuth 2.0 をサポートしています。

Claude Code は、サーバーが `401 Unauthorized` または `403 Forbidden` で応答するとき、リモートサーバーが認証を必要としていることをマークします。Claude Code が表示する内容はサーバーによって異なります。

- サインインしていないサーバーの場合、どちらのステータスコードでも `/mcp` でフラグが立てられるため、OAuth フローを完了できます。
- *claude.ai* **コネクタ** (ページ 396) の場合、claude.ai がセッショントークンを拒否することによる `401` はコネクタにフラグを立てません。コネクタを再認可してもログインを修正できないためです。Claude Code は代わりにセッショントークン拒否状態を表示します。

- `Authorization` ヘッダーを設定したサーバーの場合、`headers` または *`headersHelper`* (ページ 392) を通じて、接続中の `401` または `403` はサーバーにフラグを立てません。修正する認証情報は設定した認証情報だからです。Claude Code は代わりに接続が失敗したことを報告します。設定したヘッダーが `${VAR}` 参照から設定されている場合は、その変数が Claude Code が**空として読み込む** (ページ 384) 変数の 1 つであるかどうかを確認してください。
- **クラウドセッションに配信されたコネクタ** (ページ 398) の場合、Claude Code はサインインフローを実行しません。セッションのプロキシが claude.ai で付与した認可を使用してコネクタに認証するためです。そこでコネクタが再度認可が必要な場合、セッションからではなく claude.ai/customize/connectors で再接続してください。

既にサインインした OAuth サーバーへのリクエストが `401 Unauthorized` を返すとき、Claude Code は保存されたトークンをリフレッシュし、再接続して、リクエストを 1 回再試行します。その再試行も失敗した場合にのみ、`/mcp` でサーバーにフラグを立てます。v2.1.206 より前は、ネットワークエラーなどの一時的な理由でトークンリフレッシュが失敗した場合、リフレッシュトークンがまだ有効であっても、OAuth サーバーは残りのセッション中、認証が必要としてフラグが立てられていました。

サーバーが保存されたリフレッシュトークンを拒否するとき、Claude Code は直ちに `/mcp` を指す通知を表示します。`/mcp` を開き、サーバーで **Re-authenticate** を選択して、次のツール呼び出しが失敗する前に再度サインインしてください。

`WWW-Authenticate` ヘッダーを返すカスタムサーバーは、その認可サーバーを指し、他のリモートサーバーと同じ自動検出を取得します。

Claude Code は、1 つ以上の設定されたサーバーが認証を必要とするときにスタートアップ通知も表示するため、`/mcp` を開いて認証が必要なサーバーを検出する必要がありません。この通知には Claude Code v2.1.193 以降が必要です。Claude Code からサインインできるサーバーのみをカウントします。v2.1.218 より前は、claude.ai で接続されていない *claude.ai* **コネクタ** (ページ 396) もカウントされていました。これらは claude.ai 設定からのみ接続できます。

通知は各サーバーを 1 回アナウンスし、そのサーバーが接続して再度サインインが必要になるまで、後の起動時のカウントから除外します。`/mcp` は依然としてサインインが必要なすべてのサーバーをリストします。

非対話モードでは `/mcp` パネルがないため、Claude Code は OAuth フローを実行できません。v2.1.196 以降、**ツール検索** (ページ 406) が有効な（デフォルト）`claude -p` または Agent SDK 実行中に設定されたサーバーが認証を必要とするとき、Claude Code はサーバーのツールが認可されるまで利用できないことを Claude に伝えます。Claude はサーバーが設定されていないかのように応答する代わりに、サインインが必要なサーバーに名前を付けることができます。対話セッションから `/mcp` または `claude mcp login <name>` でサインインを完了してください。

サーバーに `headers.Authorization` を設定し、サーバーがそのヘッダーを拒否する場合、Claude Code は OAuth にフォールバックする代わりに接続が失敗したことを報告します。トークンが MCP エンドポイントに対して有効であることを確認するか、ヘッダーを削除して OAuth フローを使用してください。

1. **認証が必要なサーバーを追加する**

   MCP クイックスタートで既に `sentry` サーバーを追加した場合、このステップをスキップしてください。同じサーバー名で同じスコープで `claude mcp add` を再度実行すると、`MCP server sentry already exists in local config` で失敗します。それ以外の場合は、以下を実行してください。

```
claude mcp add --transport http sentry https://mcp.sentry.dev/mcp
```

2. **Claude Code 内で /mcp コマンドを使用する**

   Claude Code で、以下のコマンドを使用します。

```
/mcp
```

   その後、ブラウザーのステップに従ってログインしてください。

Tip:

ヒント：

- 認証トークンは安全に保存され、自動的にリフレッシュされます
- /mcp メニューの「Clear authentication」を使用してアクセスを取り消します
- ブラウザーが自動的に開かない場合は、提供された URL をコピーして手動で開いてください
- 認証後、ブラウザーリダイレクトが接続エラーで失敗する場合は、ブラウザーのアドレスバーから完全なコールバック URL を貼り付けて、Claude Code に表示される URL プロンプトに入力してください
- OAuth 認証は HTTP サーバーで機能します

19.6.1 コマンドラインから認証する

`claude mcp login <name>` コマンドは設定されたサーバーの OAuth フローをシェルから直接実行するため、セッション内の /mcp パネルを開く必要がありません。

```
claude mcp login sentry
```

後で保存された認証情報をクリアするには、`claude mcp logout <name>` を実行してください。

`claude mcp login` は SSH セッション中やディスプレイサーバーのない Linux など、ローカルブラウザーが利用できない場合を検出し、ブラウザーを開こうとする代わりに認可 URL を出力します。ローカルマシンで URL を開き、ブラウザーのアドレスバーから完全なリダイレクト URL をプロンプトに貼り付けてください。このコマンドは貼り付けステップのために対話型ターミナルが必要なため、`ssh -t` で接続してください。ローカルブラウザーが検出された場合でも URL プロンプトを強制するには、`--no-browser` を渡してください。

```
claude mcp login sentry --no-browser
```

19.6.2 固定 OAuth コールバックポートを使用する

一部の MCP サーバーは、事前に登録された特定のリダイレクト URI が必要です。デフォルトでは、Claude Code は OAuth コールバック用にランダムに利用可能なポートを選択します。`--callback-port` を使用してポートを固定し、`http://localhost:PORT/callback` 形式の事前登録されたリダイレクト URI と一致させてください。Claude Code v2.1.229 でのサインインがリダイレクト URI の不一致で失敗する場合は、**事前設定された *OAuth* 認証情報を使用する** (ページ 389) の下のバージョンノートを参照してください。

`--callback-port` は単独で（動的クライアント登録を使用）または `--client-id` と一緒に（事前設定された認証情報を使用）使用できます。

```
# 動的クライアント登録を使用した固定コールバックポート
claude mcp add --transport http \
  --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

19.6.3 事前設定された OAuth 認証情報を使用する

一部の MCP サーバーは、動的クライアント登録による自動 OAuth セットアップをサポートしていません。「Incompatible auth server: does not support dynamic client registration」のようなエラーが表示される場合、サーバーは事前設定された認証情報が必要です。Claude Code は、動的クライアント登録の代わりにクライアント ID メタデータドキュメント（CIMD）を使用するサーバーもサポートし、これらを自動的に検出します。自動検出が失敗する場合は、サーバーの開発者ポータルを通じて OAuth アプリを登録してから、サーバーを追加するときに認証情報を提供してください。

1. **サーバーで OAuth アプリを登録する**

   サーバーの開発者ポータルを通じてアプリを作成し、クライアント ID とクライアントシークレットをメモしてください。

   登録フォームがリダイレクト URI を要求する場合は、利用可能なポートを選択し、`http://localhost:PORT/callback` 形式でリダイレクト URI を入力してください。次のステップで同じポートを使用します。

   v2.1.229 では、Claude Code は代わりに `http://127.0.0.1:PORT/callback` を送信していました。登録されたリダイレクト URI と完全に一致するサーバーは、リダイレクト URI の不一致でサインインを拒否していました。Claude Code v2.1.231 は `localhost` 形式を復元しました。v2.1.229 で復旧するには、Claude Code をアップグレードするか、一時的にサーバーの登録されたリダイレクト URI に `http://127.0.0.1:PORT/callback` 形式を追加してください。

2. **認証情報を使用してサーバーを追加する**

   以下のタブは両方のコマンドをカバーしています。`claude mcp add` はクライアント ID とコールバックポートをフラグとして受け取り、`claude mcp add-json` は `oauth` オブジェクトで受け取ります。リダイレクト URI を登録した場合は、コールバックポートをその URI のポートに設定してください。

   claude mcp add

   `--client-id` を使用してアプリのクライアント ID を渡します。`--client-secret` フラグはマスクされた入力でシークレットをプロンプトします。

   ```
   claude mcp add --transport http \
     --client-id your-client-id --client-secret --callback-port 8080 \
     my-server https://mcp.example.com/mcp
   ```

   claude mcp add-json

   JSON 設定に `oauth` オブジェクトを含め、`--client-secret` を別のフラグとして渡します。

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"clientId":
↪"your-client-id","callbackPort":8080}}' \
  --client-secret
```

claude mcp add-json（コールバックポートのみ）

コールバックポートのみを固定し、Claude Code がクライアントを自動的に登録できるようにするには、`callbackPort` を単独で設定してください。

```
claude mcp add-json my-server \
  '{"type":"http","url":"https://mcp.example.com/mcp","oauth":{"callbackPort
↪":8080}}'
```

CI / 環境変数

環境変数を通じてシークレットを設定して、対話型プロンプトをスキップします。

```
MCP_CLIENT_SECRET=your-secret claude mcp add --transport http \
  --client-id your-client-id --client-secret --callback-port 8080 \
  my-server https://mcp.example.com/mcp
```

3. **Claude Code で認証する**

Claude Code で `/mcp` を実行し、ブラウザーログインフローに従ってください。

Tip:

ヒント：

- クライアントシークレットは、設定ではなく、システムキーチェーン（macOS）または認証情報ファイルに安全に保存されます
- クライアントシークレットはサーバーを追加するときにのみ設定できます。`claude mcp login` または `/mcp` から認証するとき、Claude Code は保存されたシークレットを使用し、プロンプトを表示したり `MCP_CLIENT_SECRET` を読み込んだりしません
- 後でシークレットを追加または変更するには、`claude mcp remove <name>` でサーバーを削除してから、`--client-secret` と同じ `--scope` で再度追加してください
- サーバーがシークレットのないパブリック OAuth クライアントを使用する場合は、`--client-secret` なしで `--client-id` のみを使用してください
- これらのフラグは HTTP および SSE トランスポートにのみ適用されます。stdio サーバーには効果がありません
- `claude mcp get <name>` を使用して、OAuth 認証情報がサーバーに対して設定されていることを確認してください

19.6.4 OAuth メタデータ検出をオーバーライドする

特定の OAuth 認可サーバーメタデータ URL を指して、デフォルト検出チェーンをバイパスしてください。MCP サーバーの標準エンドポイントがエラーになるとき、または内部プロキシを通じて検出をルーティングしたいときに `authServerMetadataUrl` を設定してください。デフォルトでは、Claude Code は最初

に /.well-known/oauth-protected-resource で RFC 9728 保護リソースメタデータをチェックし、次に /.well-known/oauth-authorization-server で RFC 8414 認可サーバーメタデータにフォールバックします。

.mcp.json のサーバー設定の oauth オブジェクトで authServerMetadataUrl を設定してください。

```
{
  "mcpServers": {
    "my-server": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "oauth": {
        "authServerMetadataUrl": "https://auth.example.com/.well-known/openid-
↪configuration"
      }
    }
  }
}
```

URL は https:// を使用する必要があります。メタデータ URL の scopes_supported は、アップストリームサーバーがアドバタイズするスコープをオーバーライドします。

19.6.5 OAuth スコープを制限する

oauth.scopes を設定して、認可フロー中に Claude Code がリクエストするスコープをピン留めしてください。これは、アップストリーム認可サーバーがより多くのスコープをアドバタイズするときに、MCP サーバーをセキュリティチームによって承認されたサブセットに制限するサポートされた方法です。値は RFC 6749 §3.3 の scope パラメーター形式と一致する単一のスペース区切り文字列です。

```
{
  "mcpServers": {
    "slack": {
      "type": "http",
      "url": "https://mcp.slack.com/mcp",
      "oauth": {
        "scopes": "channels:read chat:write search:read"
      }
    }
  }
}
```

oauth.scopes は authServerMetadataUrl とサーバーが /.well-known で検出するスコープの両方より優先されます。MCP サーバーがリクエストされたスコープセットを決定できるようにするには、設定を解除のままにしてください。

v2.1.196 以降、oauth.scopes が設定されていない場合、Claude Code はサーバーの WWW-Authenticate ヘッダーまたは保護されたリソースメタデータによって提供されるスコープをリクエストし、どちらも提供しない場合は scope パラメーターを送信しません。自動的に検出された認可サーバーメタデータから完全

な scopes_supported カタログをリクエストしなくなりました。そのカタログをリクエストすると、管理者のみまたはテンプレートスコープをアドバタイズするアイデンティティプロバイダーが invalid_scope エラーで認可リクエストを拒否していました。設定された authServerMetadataUrl から取得されたメタデータは、その scopes_supported をリクエストされたスコープとして提供します。

認可サーバーが scopes_supported で offline_access をアドバタイズする場合、Claude Code はそれをピン留めされたスコープに追加して、新しいブラウザーサインインなしでアクセストークンをリフレッシュできるようにします。

サーバーが後でツール呼び出しに対して 403 insufficient_scope を返す場合、呼び出しは追加の権限が必要というメッセージで失敗し、サーバーが要求するスコープに名前を付けます。サーバーは /mcp で認証が必要として表示されます。

そのスコープがピン留めされた oauth.scopes にない場合は、それを追加してから /mcp を実行し、サーバーを再度認証してください。Claude Code はサーバーが名前を付けたスコープではなく、ピン留めされたスコープをリクエストするため、それを追加せずに再度認証する場合、取得するトークンはまだそれを欠いています。

19.6.6 カスタム認証に動的ヘッダーを使用する

MCP サーバーが Kerberos、短命トークン、または内部 SSO などの OAuth 以外の認証スキームを使用する場合は、headersHelper を使用して接続時にリクエストヘッダーを生成してください。Claude Code はコマンドを実行し、その出力を接続ヘッダーにマージします。

```
{
  "mcpServers": {
    "internal-api": {
      "type": "http",
      "url": "https://mcp.internal.example.com",
      "headersHelper": "/opt/bin/get-mcp-auth-headers.sh"
    }
  }
}
```

コマンドはインラインにすることもできます。

```
{
  "mcpServers": {
    "internal-api": {
      "type": "http",
      "url": "https://mcp.internal.example.com",
      "headersHelper": "echo '{\"Authorization\": \"Bearer '\"$(get-token)\"'\"}'"
    }
  }
}
```

要件：

- コマンドは文字列キーと値のペアの JSON オブジェクトを stdout に書き込む必要があります

- Claude Code はコマンドをシェルで実行し、10 秒後にそれを放棄します
- Claude Code は**サーバーを設定した場所** (ページ 393) によってコマンドの作業ディレクトリを選択するため、スクリプトを絶対パスとして指定するか、`PATH` に配置してください
- 動的ヘッダーは同じ名前の静的 `headers` をオーバーライドします

Claude Code は各接続時にヘルパーを新たに実行します。セッション開始時と再接続時に、**プロジェクトおよびローカルスコープサーバーの信頼ルール** (ページ 394) がそれを実行できるようにします。結果をキャッシュしないため、スクリプトはトークンの再利用を担当します。

ツール呼び出しが `401 Unauthorized` または `403 Forbidden` を返す場合、Claude Code は自動的に同じルールの下でヘルパーを再実行し、新しいヘッダーで再接続し、呼び出しを 1 回再試行します。その再試行も失敗した場合にのみ、Claude Code は `/mcp` でサーバーを認証が必要としてマークします。

ヘルパーの出力に `Authorization` ヘッダーが含まれている場合、Claude Code はその認証情報をサーバーの認証として使用し、サーバーの OAuth にフォールバックしません。

サーバーが接続中にヘルパーの認証情報を拒否する場合、Claude Code はサーバーを認証が必要としてマークする代わりに、接続が失敗したことを報告します。ヘルパーが返す認証情報を修正してから、`/mcp` から再接続してヘルパーを再実行してください。

Claude Code はヘルパーを実行するときに、これらの環境変数を設定します。

変数	値
`CLAUDE_CODE_MCP_SERVER_NAME`	MCP サーバーの名前
`CLAUDE_CODE_MCP_SERVER_URL`	MCP サーバーの URL
`CLAUDE_PLUGIN_ROOT`	プラグインのルートディレクトリ。プラグインがサーバーを提供する場合にのみ設定されます

これらを使用して、複数の MCP サーバーに対応する単一のヘルパースクリプトを作成してください。

プラグイン提供の `headersHelper` はプラグインの `${user_config.*}`値を参照できません。コマンドはシェルを通じて実行されるためです。Claude Code はサーバーを設定ミスとしてエラーで報告し、値を置換しません。代わりに、シェル解析されない `headers` フィールドに `${user_config.KEY}` を配置するか、ヘルパースクリプトに設定ファイルから値を読み込ませてください。v2.1.207 より前は、`headersHelper` は `${user_config.*}` 値を置換していました。

ヘルパーが実行される場所

Claude Code は、サーバーを宣言する設定から `headersHelper` コマンドの作業ディレクトリを選択します。Claude が Bash で実行する `cd` はそれを移動しません。*`/cd`* (ページ 1061) はセッションのプライマリ作業ディレクトリから実行されるサーバーのみを移動します。以下の各行は、`headersHelper` コマンドの相対パスが解決される対象のディレクトリを示します。

サーバーを設定した場所	作業ディレクトリ
プラグイン	プラグインのルートディレクトリ

次のページに続く

表 3 – 前のページからの続き

サーバーを設定した場所	作業ディレクトリ
プロジェクト .mcp.json または **ローカルスコープ** (ページ 381) サーバー	サーバーが宣言されているプロジェクトディレクトリ
プロジェクト内のエージェントファイル、SDK の mcpServers オプションまたは setMcpServers() メソッドからのサーバー、または *--mcp-config* (ページ 45)	セッションの**プライマリ作業ディレクトリ** (ページ 1060)
ユーザースコープ (ページ 383)、管理 MCP、*claude.ai* **コネクタ** (ページ 396)、またはプロジェクト外のエージェントファイル（--add-dir ディレクトリからのものを含む）	設定ディレクトリ ~/.claude（CLAUDE_CONFIG_DIR を設定していない場合）

v2.1.238 より前は、Claude Code はユーザースコープ、管理、および claude.ai コネクタサーバーのヘルパー、およびプロジェクト外のエージェントファイルのヘルパーも、それを開始したディレクトリから実行していました。

ヘルパーが読み取ることができる変数

リポジトリまたはプラグインが提供する headersHelper は、書いていないコマンドなため、Claude Code はそれを環境から認証情報変数なしで実行します（ANTHROPIC_API_KEY など）。サーバーを設定した場所によって、これが適用されるかどうかが決まります。

- **削除される**：プロジェクト .mcp.json またはプラグイン内のサーバー、およびプロジェクトからのエージェントファイルまたは --add-dir ディレクトリからのインラインサーバー
- **削除されない**：**ユーザー** (ページ 383) または**ローカルスコープ** (ページ 381)、管理 MCP、*claude.ai* **コネクタ** (ページ 396)、または SDK または *--mcp-config* (ページ 45) から提供されるサーバー、および ~/.claude/agents/ からのインラインサーバー、管理設定から、または --agents で渡されるもの

Git の GIT_CONFIG_KEY_<n> 変数を除き、Claude Code は環境から TOKEN、SECRET、PASSWORD、KEY、または AUTH を含む名前のような認証情報のように見える名前を持つすべての変数を削除します。したがって、ANTHROPIC_API_KEY と MY_REGISTRY_TOKEN の両方が削除されます。Claude Code は、ANTHROPIC_CUSTOM_HEADERS などの名前がそのパターンに従わない固定リストの認証情報変数も削除します。

これがヘルパーに適用される場合は、スクリプトにファイルまたは認証情報ストアから認証情報を読み込ませてください。サーバーの url が**これらの変数のいずれかを展開する** (ページ 383) 場合、ヘルパーが受け取る CLAUDE_CODE_MCP_SERVER_URL 値にはその部分が REDACTED に置き換えられています。

headersHelper が実行される前にフォルダーを信頼する

Claude Code は headersHelper を任意のシェルコマンドとして実行します。プロジェクト .mcp.json 内のサーバーまたは**ローカルスコープ** (ページ 381) の場合、サーバーが宣言されているプロジェクトディレクトリの**信頼ダイアログ** (ページ 1064) を受け入れた後にのみ、ヘルパーを実行します。v2.1.238 より前は、claude -p または SDK セッションはこれらのヘルパーを信頼をチェックせずに実行していました。対話セッションは親フォルダーを信頼した後に 1 回実行していました。

- **カウントされない信頼**：親フォルダーの信頼、および `claude -p` または SDK セッションが**設定ファイルのフック** (ページ 1066) に対して取得する自動信頼
- **フォルダーを信頼するまで**：Claude Code は静的 `headers` のみでサーバーを接続します。`claude -p` または SDK セッションでは、1 つの `headersHelper not run` 行を stderr に出力し、信頼を付与する方法を示します。
- **ダイアログなしで信頼する**：`~/.claude.json` で `projects["<path>"].hasTrustDialogAccepted` を `true` に設定してください。`<path>` は、**プロジェクト許可ルールとワークスペース信頼** (ページ 1064) が Claude Code がキーを設定するフォルダーです。

Claude Code は**エージェントファイル** (ページ 334) で宣言されたサーバーに同じルールを適用し、そのエージェントファイルがどこから来たかをチェックします。プロジェクト内のファイルの場合は `.claude/agents/` ディレクトリから、または `--add-dir` ディレクトリから。**そのプロジェクトまたはディレクトリ自体を信頼する** (ページ 1066) まで、Claude Code はサーバーをロードしません。したがって、そのヘルパーも実行されません。

19.7 JSON 設定から MCP サーバーを追加する

MCP サーバーの JSON 設定がある場合は、直接追加できます：

1. **JSON から MCP サーバーを追加する**

```
# 基本的な構文
claude mcp add-json <name> '<json>'

# 例：JSON 設定を使用して HTTP サーバーを追加する
claude mcp add-json weather-api '{"type":"http","url":"https://api.weather.com/
↪mcp","headers":{"Authorization":"Bearer token"}}'

# 例：JSON 設定を使用して stdio サーバーを追加する
claude mcp add-json local-weather '{"type":"stdio","command":"/path/to/weather-
↪cli","args":["--api-key","abc123"],"env":{"CACHE_DIR":"/tmp"}}'

# 例：事前設定された OAuth 認証情報を使用して HTTP サーバーを追加する
claude mcp add-json my-server '{"type":"http","url":"https://mcp.example.com/mcp
↪","oauth":{"clientId":"your-client-id","callbackPort":8080}}' --client-secret
```

2. **サーバーが追加されたことを確認する**

```
claude mcp get weather-api
```

Tip:

ヒント：

- JSON がシェルで適切にエスケープされていることを確認してください
- JSON は MCP サーバー設定スキーマに準拠する必要があります
- `--scope user` を使用して、プロジェクト固有のサーバーの代わりにユーザー設定にサーバーを追加できます

19.8 Claude Desktop から MCP サーバーをインポートする

Claude Desktop で MCP サーバーを既に設定している場合は、それらをインポートできます：

1. **Claude Desktop からサーバーをインポートする**

```
# 基本的な構文
claude mcp add-from-claude-desktop
```

2. **インポートするサーバーを選択する**

   コマンドを実行した後、インポートするサーバーを選択できる対話的なダイアログが表示されます。

3. **サーバーがインポートされたことを確認する**

```
claude mcp list
```

`claude mcp` コマンドで追加されたサーバー名には、文字、数字、ハイフン、アンダースコアのみを含めることができます。Claude Desktop はその制限を適用しないため、スペースなどの他の文字を含む名前を持つ Claude Desktop サーバーはインポートできません。インポートは拒否された各名前を報告し、選択した他のサーバーはインポートします。v2.1.205 より前では、最初の無効な名前がインポートを停止し、選択されたサーバーは追加されませんでした。

Tip:

ヒント：

- この機能は macOS と Windows Subsystem for Linux（WSL）でのみ機能します
- これらのプラットフォームの標準的な場所から Claude Desktop 設定ファイルを読み取ります
- `--scope user` フラグを使用してサーバーをユーザー設定に追加します
- インポートされたサーバーは、名前に文字、数字、ハイフン、アンダースコアのみが含まれている場合、Claude Desktop と同じ名前を保持します。Claude Code は他の文字を含む名前を持つサーバーを報告し、スキップします
- 同じ名前のサーバーが既に存在する場合、数値サフィックスが付与されます（例：`server_1`）

19.9 claude.ai から MCP サーバーを使用する

claude.ai アカウントで Claude Code にログインしている場合、claude.ai に追加した MCP サーバー（connectors として知られています）は Claude Code で自動的に利用可能になります。

1. **claude.ai で MCP サーバーを設定する**

   claude.ai/customize/connectors でサーバーを追加します。Team および Enterprise プランでは、管理者のみがサーバーを追加できます。

2. **MCP サーバーを認証する**

   claude.ai で必要な認証ステップを完了します。

3. **Claude Code でサーバーを表示および管理する**

Claude Code で、次のコマンドを使用します。

```
/mcp
```

claude.ai からのサーバーはリストに表示され、claude.ai から来たことを示すインジケーターが付きます。

Anthropic は、ユーザーまたは管理者が追加しなくても、いくつかの connector を自体で提供しています。Claude Docs が利用可能なアカウントでは、`/mcp` は `claude.ai Claude Docs` をセットアップなしでリストし、他の人向けのドキュメントを要求すると Claude がそれを使用します。これをオフにするには、`deniedMcpServers` に `"claude.ai Claude Docs"` の `serverName` エントリを追加するか、*Disable claude.ai connectors* (ページ 399) で説明されている `/mcp` トグルを使用します。

Claude Code は、組織が claude.ai で認証を管理している場合、`/mcp` および `/plugin` マネージャーで connector を `managed` としてマークします。Managed ステータスは、Claude Code が connector に接続する方法や、組織の *tool controls* (ページ 399) を適用する方法を変更しません。

まだサインインしたことのない Connector は、claude.ai セクションの最後にある `Show unused connectors` 行の背後に折りたたまれているため、組織がプロビジョニングしたリストがパネルを満たしません。その行を選択して展開します。以前にサインインした Connector は、現在再認証が必要な場合でも表示されたままです。

claude.ai からの Connector は、アクティブな *authentication method* (ページ 858) が claude.ai サブスクリプションログインである場合にのみ取得されます。以前に `/login` を実行した場合でも、次の場合は読み込まれません。

- `ANTHROPIC_API_KEY`、`ANTHROPIC_AUTH_TOKEN`、または `apiKeyHelper` がアクティブ
- Amazon Bedrock や Google Cloud の Agent Platform などのサードパーティプロバイダーがアクティブ
- `ANTHROPIC_PROFILE`、フェデレーション変数、またはアクティブな *Anthropic profile* (ページ 859) が認証情報を提供
- `CLAUDE_CODE_OAUTH_TOKEN` が *`claude setup-token`* (ページ 860) からのトークンを保持している (モデルリクエストのみを実行できます)

`/mcp` が追加した connector をリストしない場合は、`/status` を実行してどの authentication method がアクティブであるかを確認します。その環境変数を設定解除し、`apiKeyHelper` 設定を削除するか、*profile* **をオフに切り替え** (ページ 859)、`/login` を実行して claude.ai アカウントを選択します。

一時的なネットワーク問題により、セッション開始時に connector リストが読み込まれない場合、Claude Code はバックグラウンドで最大 3 回再試行し、再試行が成功すると connector が表示されます。まだ表示されていない場合は、Claude Code を再起動してリストを再度取得します。

`/mcp` が connector を `session token rejected` として表示する場合、またはその詳細ビューが `claude.ai rejected the session token` を表示する場合、claude.ai は Claude Code ログインからのトークンを拒否しました。connector を再度認証しても、connector 自体の認証が拒否されたわけではないため、この状態はクリアされません。クリアするには、以下を実行します。

1. `/login` を実行して再度サインインします。
2. `/mcp` から connector を再度接続します。

v2.1.222 より前では、Claude Code は connector を認証が必要として マークしていました。これを認証しても解決しませんでした。

Claude Code で追加したサーバーは、同じ URL を指す claude.ai connector よりも *precedence* (ページ 383) を持ちます。これが発生すると、`/mcp` は connector を hidden としてリストし、代わりに connector を使用する場合は重複を削除する方法を表示します。

Microsoft 365、Gmail、Google Calendar などの一部の Anthropic ホスト connector は、アップストリーム ID プロバイダーが claude.ai が登録したリダイレクト URL のみを受け入れるため、Claude Code からのローカル OAuth をサポートしていません。`claude mcp add` または `.mcp.json` で追加したサーバーがこれらのホストの 1 つを指し、`/mcp` から、または `claude mcp login` でサインインすると、Claude Code は `is Anthropic-hosted and doesn't support local OAuth` を表示し、代わりに claude.ai/customize/connectors でサービスを接続するよう指示します。

`claude mcp remove <name>` でエントリを削除し、claude.ai でサービスを接続した後、connector は Claude Code に自動的に表示されます。

19.9.1 Connector が Claude Code に到達する方法

どの設定が claude.ai connector を管理するかは、セッションが実行される場所によって異なります。セッションの種類によっては、claude.ai 自体から connector を取得するセッションのみがあるためです。以下の各行は、1 種類のセッションで connector がどのように到達するか、およびそこで何が connector を制御するかを示しています。デスクトップアプリの WSL sessions には、connector がまだ利用できないため、行がありません。

セッションが実行される場所	Connector がどのように到達するか	何が connector を管理するか
Terminal、*VS Code* (ページ 173)、*JetBrains* (ページ 205)、および Agent SDK セッション	Claude Code が claude.ai から取得	このセクションの設定および managed MCP configuration
Cloud sessions (ページ 223)	クラウドホストが渡す	claude.ai 組織設定、および allowlist と denylist 設定がセッションに到達し、セッションを実行するホスト上の `managed-mcp.json`
desktop app (ページ 137) のローカルおよび SSH セッション	デスクトップアプリが in-process で配信	組織の *connector tool controls* (ページ 399) の `blocked` エントリ

`disableClaudeAiConnectors` (ページ 399)、`ENABLE_CLAUDEAI_MCP_SERVERS`、および `allowAllClaudeAiMcps` は最初の行のみに作用します。Claude Code が自体で取得する connector です。他の 2 つの行は次の点で異なります。

- **Cloud sessions**: セッションに到達する `allowedMcpServers` および `deniedMcpServers` エントリ（例えば server-managed settings を通じて）は、配信された connector もフィルタリングします。セッションのプロキシは各 connector の URL を書き直すため、connector 自体の URL 用に書かれた `serverUrl` パターンはそれと一致しません。自己ホスト環境で URL allowlist と共に配信された connector を許可するには、Connector traffic leaves your network の下にリストされている `serverUrl` エントリを追加します。セッションを実行するホスト（例えば self-hosted runner host）に `managed-mcp.json` が存在する場合、Claude Code は配信された connector をドロップします。`allowAllClaudeAiMcps` を

設定するかどうかに関わらず。

- **Desktop app local and SSH sessions**: デスクトップアプリは connector を in-process `type: "sdk"` サーバーとして登録し、MCP 設定または `managed-mcp.json` は到達しません。ユーザーは claude.ai/customize/connectors で connector を切断することで、自分のセッションから connector を除外します。組織は connector の *tools* (ページ 399) をブロックするか、*Claude Code in the desktop app* (ページ 161) を完全にオフにします。

19.9.2 Connector tools の組織コントロール

組織は claude.ai connectors に tool ごとのコントロールを設定できます。Claude Code はこれらの設定をスタートアップ時に読み取り、ローカルで実行します。ただし、デスクトップアプリの *local and SSH sessions* (ページ 398) では除きます。そこでは、デスクトップアプリは connector を配信する前に `blocked` tool を保留し、`ask` 設定は Claude Code に到達しないため、セッションの通常の *permission rules* (ページ 1039) をそれらの tool に適用し、すべての呼び出しでプロンプトを表示する代わりに。Claude Code が connector 自体を取得するセッションでは、`/mcp` を実行して、各 tool に適用される設定を connector で確認します。

- **Tool が `ask` に設定されている場合**: Claude Code は理由 `Your organization requires approval for this tool` ですべての呼び出しでプロンプトを表示します。プロンプトは `acceptEdits`、`auto`、および `bypassPermissions` *permission modes* (ページ 1042) でも表示され、選択を記憶するオプションは提供されません。tool と一致する *Allow rules* (ページ 1039) はプロンプトをスキップしません。プロンプトを表示しない `dontAsk` モードでは、Claude Code は呼び出しを代わりに拒否します。
- **Tool が `blocked` に設定されている場合**: Claude Code は Claude がそれを見る前に tool をフィルタリングするため、tool リストに表示されません。Claude Code が connector 自体を取得するセッションでは、`/mcp` tool リストは依然として tool を表示し、`disabled by your organization` としてマークされます。

デスクトップアプリと claude.ai チャットは同じ `blocked` 設定を適用するため、Claude はそこでも tool を使用できず、デスクトップアプリのセッションから tool を保留しながらチャットで利用可能に保つことはできません。デスクトップアプリは tool がすべてブロックされている connector をスキップします。

19.9.3 claude.ai connector を無効にする

Claude Code は `disableClaudeAiConnectors` を、**自体で取得する** (ページ 398) connector のみに適用し、クラウドホストまたはデスクトップアプリが配信する connector には適用しません。取得する connector をオフにするには、任意の設定スコープで設定を `true` に設定します。

```
{
  "disableClaudeAiConnectors": true
}
```

この設定は any-source-true セマンティクスを使用します。任意の設定ソースの `true` が優先されます。チェックインされたプロジェクト `.claude/settings.json` は Claude Code が自体で取得する connector をリポジトリから除外できますが、プロジェクトレベルの `false` は、ユーザーまたはポリシーレベルの `true` が無効にした connector を再度有効にすることはできません。`--mcp-config` を通じて明示的に渡されたサーバーは影響を受けません。

ENABLE_CLAUDEAI_MCP_SERVERS 環境変数を false に設定することもできます。これは現在のシェルセッションに対して同じ効果があります。

```
ENABLE_CLAUDEAI_MCP_SERVERS=false claude
```

すべての claude.ai connector をブロックする代わりに個別の claude.ai connector をブロックするには、名前または URL パターンで deniedMcpServers に追加します。例えば、serverName エントリ "claude.ai Slack" は Slack connector をブロックします。また、/mcp を実行して、Claude Code が取得する任意の connector を現在のプロジェクトのみでオン/オフに切り替えることもできます。

19.10 Claude Code を MCP サーバーとして使用する

Claude Code 自体を MCP サーバーとして使用して、他のアプリケーションが接続できるようにすることができます。

```
# Claude を stdio MCP サーバーとして起動する
claude mcp serve
```

コマンドは起動時に何も出力しません。stdio MCP サーバーは stdin と stdout を介して通信するため、サイレント状態でブロックされたターミナルはサーバーが実行中で、クライアントの接続を待機していることを意味します。

Claude Desktop でこれを使用するには、claude_desktop_config.json にこの設定を追加します。

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

警告:

実行可能ファイルパスの設定: command フィールドは Claude Code 実行可能ファイルを参照する必要があります。claude コマンドがシステムの PATH にない場合は、実行可能ファイルへの完全なパスを指定する必要があります。
完全なパスを見つけるには：

```
which claude
```

次に、設定で完全なパスを使用します。

```
{
  "mcpServers": {
    "claude-code": {
      "type": "stdio",
      "command": "/full/path/to/claude",
      "args": ["mcp", "serve"],
      "env": {}
    }
  }
}
```

正しい実行可能ファイルパスがない場合、spawn claude ENOENT などのエラーが発生します。

Tip:

ヒント：

- Claude Desktop では、Claude にディレクトリ内のファイルを読み取り、編集などを行うよう依頼してみてください。
- この MCP サーバーは Claude Code のツールのみを MCP クライアントに公開するため、独自のクライアントは個別のツール呼び出しのユーザー確認を実装する責任があります。

19.11 MCP 出力制限と警告

MCP ツールが大きな出力を生成する場合、Claude Code はトークン使用量を管理して、会話コンテキストが圧倒されるのを防ぐのに役立ちます。

- **出力警告閾値**：Claude Code は、MCP ツール出力が 10,000 トークンを超える場合に警告を表示します
- **設定可能な制限**：MAX_MCP_OUTPUT_TOKENS 環境変数を使用して、許可される最大 MCP 出力トークン数を調整できます
- **デフォルト制限**：デフォルトの最大値は 25,000 トークンです
- **スコープ**：環境変数は、独自の制限を宣言していないツールに適用されます。*anthropic/maxResultSizeChars* (ページ 402) を設定するツールは、MAX_MCP_OUTPUT_TOKENS に設定されている値に関係なく、テキストコンテンツに対してその値を代わりに使用します。画像データを返すツールは、引き続き MAX_MCP_OUTPUT_TOKENS の対象となります
- **制限を超える場合**：画像コンテンツのない結果が制限を超える場合、Claude Code はそれをファイル

に保存し、会話内でファイルパスを名前とするメッセージに置き換えます。そのため、Claude はコンテンツが必要な場合にファイルを読み取ります。ファイルはセッションの `tool-results` ディレクトリ内の `~/.claude/projects/` に配置されます。

大きな出力を生成するツールの制限を増やすには：

```
export MAX_MCP_OUTPUT_TOKENS=50000
claude
```

19.11.1 特定のツールの制限を引き上げる

MCP サーバーを構築している場合、ツールの `tools/list` レスポンスエントリで `_meta["anthropic/maxResultSizeChars"]` を設定することで、個別のツールがデフォルトの永続化ディスク閾値より大きい結果を返すことができます。Claude Code はそのツールの閾値を注釈付きの値に引き上げます。ただし、500,000 文字のハードシーリングまでです。

これは、データベーススキーマや完全なファイルツリーなど、本質的に大きいが必要な出力を返すツールに役立ちます。注釈がない場合、デフォルト閾値を超える結果はディスクに永続化され、会話内のファイル参照に置き換えられます。

```
{
  "name": "get_schema",
  "description": "Returns the full database schema",
  "_meta": {
    "anthropic/maxResultSizeChars": 200000
  }
}
```

注釈はテキストコンテンツに対して `MAX_MCP_OUTPUT_TOKENS` とは独立して適用されるため、ユーザーはそれを宣言するツールのために環境変数を引き上げる必要はありません。画像データを返すツールは、引き続きトークン制限の対象となります。

警告:

制御していない特定の MCP サーバーで出力警告が頻繁に発生する場合は、`MAX_MCP_OUTPUT_TOKENS` 制限を増やすことを検討してください。サーバー作成者に `anthropic/maxResultSizeChars` 注釈を追加するか、レスポンスをページネーションするよう依頼することもできます。注釈は画像コンテンツを返すツールには効果がありません。それらの場合、`MAX_MCP_OUTPUT_TOKENS` を引き上げることが唯一のオプションです。

19.11.2 ツール結果内の画像

MCP ツールが PNG、JPEG、GIF、または WebP 画像を返す場合、Claude は会話内にインラインで画像を表示します。インラインコピーはモデルの画像サイズ制限に合わせてスケールダウンまたは圧縮される場合があります。Claude Code はセッションの `tool-results` ディレクトリ内の `~/.claude/projects/` にも元のバイトをファイルに保存し、Claude にパスを提供します。Claude は Bash などのツールを使用して、フルレゾリューションファイルをトリミング、変換、または再利用できます。

--no-session-persistence (ページ 51) または CLAUDE_CODE_SKIP_PROMPT_HISTORY でセッション永続化を無効にした場合、Claude Code は画像ファイルを書き込まず、Claude はインラインコピーのみを受け取ります。

MCP 画像結果をファイルに保存するには、Claude Code v2.1.283 以降が必要です。

19.12 ルートレベルのコンビネータを持つツール入力スキーマ

一部の MCP サーバーは、ツールの入力スキーマを JSON Schema ユニオンとして宣言し、スキーマの最上位に `anyOf`、`oneOf`、または `allOf` を配置しています。Claude API はこれらのキーワードをスキーマのルートで受け入れません。ただし、`properties` 内にネストされたコンビネータは受け入れており、Claude Code はそれらを変更せずに送信します。

ルートレベルのコンビネータを持つツールは利用可能なままです。ツールを API に送信する前に、Claude Code はスキーマをフラット化して単一のオブジェクトにし、どのパラメータグループが一緒に属するかを Claude に伝える文をツールの説明の先頭に追加します。

- `allOf`：すべてのブランチからのプロパティがマージされ、各ブランチの `required` リストは引き続き適用されます
- `anyOf` と `oneOf`：すべてのブランチからのプロパティがマージされ、各ブランチの `required` リストはスキーマによって強制されるのではなく、ツールの説明に記述されます

サーバーは Claude が選択した引数を受け取るため、サーバー側で組み合わせの検証を続けてください。

Claude Code が API が受け入れるスキーマを生成できない場合、またはスキーマの書き換えを有効にするリモート設定を受け取らないデプロイメントの場合、そのツール 1 つをスキップし、理由をサーバーのログに記録し、サーバーの他のツールは利用可能なままにします。

19.13 無効な入力スキーマを持つツール

Claude API はリクエスト内のすべてのツールの入力スキーマをチェックし、いずれかのスキーマが失敗すると、リクエスト全体を 400 エラーで拒否します。そのため、スキーマが不正な形式の MCP ツール 1 つがあると、それを含むすべてのリクエストが失敗します。Claude Code はサーバーのツールを読み込む際に API のチェックのうち 2 つを自身で実行し、それらのチェックに失敗するツールを除外するため、サーバーの他のツールは動作し続けます。

- トップレベルのプロパティ名は 1 〜 64 文字の長さで、ASCII 文字と数字、_、.、- のみを使用する必要があります
- スキーマは JSON Schema draft 2020-12 メタスキーマに対して有効である必要があります。Claude Code は `$schema` を宣言していないスキーマと draft 2020-12 を宣言しているスキーマにこのチェックを適用します。他の方言を宣言しているスキーマはこのチェックをスキップしますが、上記のプロパティ名チェックは引き続き適用されます

Claude Code がツールを除外する場合、その理由をサーバーのログに記録し、除外したツールとその理由を Claude に伝えるため、ツールが見つからない理由を Claude に尋ねることができます。サーバーのスキーマを修正すると、Claude Code が次にサーバーのツールを読み込むときにツールが復帰します。

Claude Code は Anthropic から取得するフィーチャーフラグを通じて除外をオンにします。フラグ取得がオフになっているデプロイメント またはフラグが到着したことのないマシン（エアギャップマシンなど）で

は、Claude Code はチェックを実行してサーバーのログにどのツールが拒否されるかを記録しますが、ツールのスキーマを API に送信します。API は ツールの位置で名前を付けた 400 エラー でそのスキーマを含むリクエストを拒否します。v2.1.216 より前では、デプロイメントはこれらのチェックを実行していませんでした。

ルートレベルのコンビネータ処理 (ページ 403) は独立しており、フラグ取得がオフの場合またはフラグが到着したことのない場合、独自の動作を保持します。

19.14 特定のツールに対して承認を要求する

MCP サーバーを構築している場合、ツールの `tools/list` レスポンスエントリで `_meta["anthropic/requiresUserInteraction"]` を `true` に設定することで、そのツールがすべての呼び出しで明示的な承認を必要とするようにマークできます。値は JSON ブール値 `true` である必要があります。その他の値は無視されます。

Claude Code は、`acceptEdits`、`auto`、`bypassPermissions` **権限モード** (ページ 1042) でも、そのツールの権限プロンプトをすべての呼び出しで表示し、それに対して「今後は表示しない」オプションを提供しません。ツールに一致する **許可ルール** (ページ 1043) もプロンプトをスキップしません。プロンプトを表示しない `dontAsk` モードでは、Claude Code は呼び出しを拒否します。

プロンプトは人間に到達する必要があります。*`--permission-prompt-tool`* (ページ 51) を使用した非対話モードでは、フラグが付いたツールのプロンプトツールからの `allow` 結果は、メッセージ `MCP tool requires user interaction; not supported via --permission-prompt-tool` とともに拒否に変換されます。Agent SDK の canUseTool コールバック はこれらの呼び出しを受け取り、承認できます。これは、SDK アプリケーションがこれらをユーザーに表示することが期待されているためです。

これを使用するのは、権限プロンプト自体がポイントであるツール、たとえば同意またはアクセス許可ステップなど、自動承認は人間が同意しないことを意味する場合です。同じサーバーからの他のツールは、通常の権限動作を保持します。

次の `tools/list` エントリは、1 つのツールを常に承認が必要なものとしてマークします。

```
{
  "name": "grant_access",
  "description": "Requests access to a protected resource",
  "_meta": {
    "anthropic/requiresUserInteraction": true
  }
}
```

Remote Control や Agent SDK 上に構築されたアプリケーションなど、一部のサーフェスでは通常、1 タップでツール呼び出しを承認できます。このアノテーションでマークされたツールの場合、Claude Code は 1 タップアクションを保留し、代わりにツールの完全な権限プロンプトを表示するため、承認は依然として 1 タップではなく、プロンプトに答える人から得られます。

Claude Code は、安全警告を含むものや、リモートサーフェスが表示できない常に許可オプションなど、ターミナルダイアログでのみ完全にレンダリングできる権限リクエストに対して、同じ方法で 1 タップ承認を保留します。そのリクエストには Remote Control からではなく、ターミナルダイアログで答えます。Claude Code v2.1.214 以降が必要です。

19.15 MCP エリシテーション要求に応答する

MCP サーバーは、エリシテーションを使用してタスク中に構造化された入力をリクエストできます。サーバーが独自に取得できない情報が必要な場合、Claude Code はインタラクティブなダイアログを表示し、応答をサーバーに返します。お客様側での設定は不要です。エリシテーションダイアログはサーバーがリクエストすると自動的に表示されます。

サーバーは 2 つの方法で入力をリクエストできます。

- **フォームモード**: Claude Code はサーバーで定義されたフォームフィールド（例えば、ユーザー名とパスワードプロンプト）を含むダイアログを表示します。フィールドに入力して送信します。
- **URL モード**: Claude Code はブラウザでリンクを開くかどうかを確認します。サーバーはこのモードを、ターミナルの外で完了するフロー（サインインなど）に使用します。

URL モードでは、Claude Code は URL をコマンドライン引数としてシステムの URL ハンドラーに渡し、その引数の長さに上限を設けます。コマンドラインのエスケープ後の URL がその上限を超える場合、リクエストを拒否することのみできます。`%` や `&` など、エスケープが必要なすべての文字は、上限に対して 4 倍カウントされます。その文字自体と 3 つのエスケープ文字です。これらを含まない URL は約 8,000 文字で上限に達します。3 番目の文字ごとに `%` があるパーセントエスケープで構成される URL は、約 4,000 で上限に達します。

ダイアログを表示せずにエリシテーション要求に自動応答するには、*Elicitation* **フック** (ページ 549) を使用します。

エリシテーションを使用する MCP サーバーを構築している場合は、プロトコルの詳細とスキーマの例については MCP エリシテーション仕様を参照してください。

プロトコルリビジョン *2026-07-28* (ページ 373) を使用する接続では、Claude Code はクライアント機能で `elicitation: {form: {}, url: {}}` を宣言するため、そこのサーバーはプロトコルの標準エリシテーション要求を通じてどちらのモードでもリクエストできます。

19.16 MCP リソースを使用する

MCP サーバーは、ファイルを参照する方法と同様に、@ メンションを使用して参照できるリソースを公開できます。

19.16.1 MCP リソースを参照する

1. **利用可能なリソースをリストアップする**

   プロンプトで `@` を入力すると、接続されているすべての MCP サーバーから利用可能なリソースが表示されます。リソースはオートコンプリートメニューのファイルと一緒に表示されます。

2. **特定のリソースを参照する**

   `@server:protocol://resource/path` の形式を使用してリソースを参照します：

```
Can you analyze @github:issue://123 and suggest a fix?
```

```
Please review the API documentation at @docs:file://api/authentication
```

3. **複数のリソース参照**

   1 つのプロンプトで複数のリソースを参照できます：

```
Compare @postgres:schema://users with @docs:file://database/user-model
```

Tip:

ヒント：

- リソースは参照されると自動的に取得され、添付ファイルとして含まれます
- リソースパスは @ メンションオートコンプリートでファジー検索可能です
- Claude Code は、サーバーがサポートしている場合、MCP リソースをリストアップして読み取るためのツールを自動的に提供します
- リソースには、MCP サーバーが提供するあらゆるタイプのコンテンツ（テキスト、JSON、構造化データなど）を含めることができます

MCP Apps UI リソースは、`ui://` URI または `text/html;profile=mcp-app` メディアタイプを持つエントリです：Claude が読み取るコンテンツではなく、ホストアプリケーションがレンダリングするページです。これらは `@` の提案やリソースリストツールの結果に表示されず、UI リソースのみを提供するサーバーは空のリソースリストを表示します。URI で UI リソースを読み取ることはまだ機能します。

19.17 MCP ツール検索でスケーリング

ツール検索は、Claude がツールを必要とするまでツール定義を遅延させることで、MCP コンテキスト使用量を低く保ちます。セッション開始時にはツール名とサーバー指示のみが読み込まれるため、MCP サーバーを追加してもコンテキストウィンドウへの影響は最小限です。Claude Code はサーバーごとの固定ツール上限を課しません。実用的な上限はコンテキストウィンドウの予算です。

注釈:

ツール検索は Microsoft Foundry の Azure でホストされているデプロイメントではサポートされていません。これらのデプロイメントはサーバー側でツール検索を拒否します。Claude Code はこの拒否を検出し、そのデプロイメント用に MCP ツールを事前に読み込みます。*ENABLE_TOOL_SEARCH* (ページ 407) はデプロイメント自体からの拒否であるため、これをオーバーライドすることはできません。

19.17.1 MCP サーバー作成者向け

MCP サーバーを構築している場合、ツール検索が有効になるとサーバー指示フィールドがより有用になります。サーバー指示は、**スキル** (ページ 275) の動作方法と同様に、Claude がいつツールを検索すべきかを理解するのに役立ちます。

以下を説明する明確で説明的なサーバー指示を追加してください。

- ツールが処理するタスクのカテゴリ
- Claude がツールを検索すべき時期
- サーバーが提供する主な機能

Claude Code はツール説明とサーバー指示を各 2,048 文字でデフォルトで切り詰めます。簡潔に保ち、重要な詳細は最初の方に配置してください。

セッション内のすべての MCP サーバーの上限を変更するには、CLAUDE_CODE_MAX_MCP_DESCRIPTION_LENGTH を文字数に設定します。この変数には Claude Code v2.1.280 以降が必要です。

19.17.2 ツール検索を設定する

ツール検索はデフォルトで有効です。MCP ツールは遅延され、オンデマンドで検出されます。Claude Code は ANTHROPIC_BASE_URL が非ファーストパーティホストを指している場合、ツール検索を無効にします。ほとんどのプロキシは tool_reference ブロックを転送しないためです。ENABLE_TOOL_SEARCH を明示的に設定して、そのフォールバックをオーバーライドします。

CLAUDE_CODE_DISABLE_EXPERIMENTAL_BETAS を設定するとツール検索がオフになります。ENABLE_TOOL_SEARCH を自分で設定してオーバーライドすることはできません。組織は 管理設定を通じて Claude Code v2.1.227 以降でツール検索をオンに保つことができます。プリリリース機能を無効にするは、オーバーライドが適用される場所と変数が削除する内容をカバーしています。

ツール検索には tool_reference ブロックをサポートするモデルが必要です。Claude Sonnet 4.5、Claude Haiku 4.5、Claude Opus 4.5、およびそれ以降のモデルです。現在のリストについては、API ドキュメントのモデル互換性を参照してください。

Google Cloud の Agent Platform では、Claude Code はモデル世代によって決定します。

- **Claude Opus 4.5、Sonnet 4.5、Haiku 4.5、およびそれ以降**: ツール検索はデフォルトでオンです。Anthropic API と同じです。
- **以前の Agent Platform モデル**: Claude Code は必要なベータヘッダーを拒否するサーバースタックのため、すべての MCP ツールを事前に読み込みます。ENABLE_TOOL_SEARCH=true はこれをオーバーライドしません。

v2.1.221 より前は、Claude Code は ENABLE_TOOL_SEARCH=true を設定しない限り、Google Cloud の Agent Platform 上のすべてのモデルに対してツール検索を無効にしていました。

ENABLE_TOOL_SEARCH 環境変数でツール検索の動作を制御します。

値	動作
（未設定）	すべての MCP ツールが遅延され、オンデマンドで読み込まれます。Google Cloud の Agent Platform の Claude 4.5 世代より前のモデル、ANTHROPIC_BASE_URL が非ファーストパーティホストの場合、または Azure でホストされている Microsoft Foundry デプロイメント上で事前読み込みにフォールバックします
true	すべての MCP ツールが遅延されます。ただし、Azure でホストされている Microsoft Foundry デプロイメント上では、サーバー側の拒否により事前読み込みが強制され、Google Cloud の Agent Platform の Claude 4.5 世代より前のモデル上では、Claude Code はツール読み込みを事前に保ちます。Claude Code はベータヘッダーをプロキシ経由で送信し、tool_reference ブロックをサポートしないプロキシではリクエストが失敗します

次のページに続く

表 5 – 前のページからの続き

値	動作
auto	しきい値モード。Claude Code は、定義の合計がコンテキストウィンドウの 10% 未満の間は、遅延させるツールを事前に読み込み、定義が 10% に達すると、すべてを遅延させます
auto:N	カスタムパーセンテージを使用したしきい値モード。N は 0〜100 です。たとえば、5% の場合は auto:5 です
false	すべての MCP ツールが事前に読み込まれ、遅延はありません

```
# カスタム 5% しきい値を使用する
ENABLE_TOOL_SEARCH=auto:5 claude

# ツール検索を完全に無効にする
ENABLE_TOOL_SEARCH=false claude
```

または settings.json env フィールドで値を設定します。

ToolSearch ツールを特別に無効にすることもできます。

```
{
  "permissions": {
    "deny": ["ToolSearch"]
  }
}
```

19.17.3 サーバーを遅延から除外する

サーバーのツールが常に Claude に表示され、検索ステップなしで利用可能にする場合は、そのサーバーの設定で alwaysLoad を true に設定します。そのサーバーのすべてのツールは、ENABLE_TOOL_SEARCH 設定に関係なく、セッション開始時にコンテキストに読み込まれます。これは、Claude がすべてのターンで必要とする少数のツール用に使用してください。事前読み込みされた各ツールは、会話に利用可能なコンテキストを消費するためです。

次の .mcp.json エントリは、1 つの HTTP サーバーを除外し、他のサーバーを遅延させたままにします。

```
{
  "mcpServers": {
    "core-tools": {
      "type": "http",
      "url": "https://mcp.example.com/mcp",
      "alwaysLoad": true
    }
  }
}
```

alwaysLoad フィールドはすべてのサーバータイプで利用可能です。MCP サーバーは、ツールの _meta オ

ブジェクトに "anthropic/alwaysLoad": true を含めることで、個別のツールを常に読み込まれるようにマークすることもできます。これはそのツールのみに同じ効果があります。

alwaysLoad: true を設定すると、スタートアップはサーバーのツールを待機します。最初のプロンプトが構築されるときに存在する必要があるため、標準の 5 秒接続タイムアウトでキャップされます。有効な *cached* **エントリ** (ページ 370) を持つリモートサーバーは、接続せずにキャッシュからツールを供給するため、スタートアップを保持しません。他のサーバーはデフォルトでバックグラウンドで接続します。MCP_CONNECTION_NONBLOCKING=0 を設定して、スタートアップがそれらも待機するようにします。

19.18 MCP プロンプトをコマンドとして使用する

MCP サーバーは Claude Code でコマンドとして利用可能になるプロンプトを公開できます。

anthropic-skills という名前のサーバーからのプロンプトは表示されません。これは Claude Code が **その名前を予約** (ページ 285) しているためです。これは claude.ai から同期されたスキル用です。サーバーのツールは引き続き機能します。プロンプトを一覧表示するには、MCP 設定でサーバーの名前を変更してください。

19.18.1 MCP プロンプトを実行する

1. **利用可能なプロンプトを検出する**

   / と入力して、MCP サーバーからのプロンプトを含む、利用可能なコマンドを確認します。Claude Code は各 MCP プロンプトを /servername:promptname (MCP) として一覧表示します。/mcp__servername__promptname と入力して実行することもできます。

2. **引数なしでプロンプトを実行する**

```
/mcp__github__list_prs
```

3. **引数付きでプロンプトを実行する**

   多くのプロンプトは引数を受け入れます。コマンドの後に空白で区切られた引数を渡します。Claude Code は引数を空白で分割するため、各引数は単一のトークンです：

```
/mcp__github__pr_review 456
```

```
/mcp__jira__create_issue login-bug high
```

Tip:

ヒント：

- MCP プロンプトは接続されたサーバーから動的に検出されます
- 引数はプロンプトの定義されたパラメータに基づいて解析されます
- プロンプト結果は会話に直接注入されます
- /mcp__servername__promptname の形式では、Claude Code はサーバー名内の A-Z、a-z、0-9、_、および - 以外の文字を _ に置き換え、サーバーが宣言したプロンプト名を使用します

19.19 管理対象 MCP 設定

MCP サーバーへの接続をユーザーが行えるかを一元管理する必要がある組織の場合は、管理対象 MCP 設定を参照してください。`managed-mcp.json` を使用した固定サーバーセットのデプロイ、`managedMcpServers` を使用したすべてのユーザーへのサーバー提供、`allowedMcpServers` と `deniedMcpServers` によるサーバーの制限、およびサーバーがブロックされた場合にユーザーに表示される内容について説明しています。

第20章　hooksでアクションを自動化する

> Claude Codeがファイルを編集したり、タスクを完了したり、入力が必要になったりしたときに、シェルコマンドを自動的に実行します。コードをフォーマットし、通知を送信し、コマンドを検証し、プロジェクトルールを適用します。

Hooksはユーザー定義のシェルコマンドです。Claude Codeはそのライフサイクルの特定のポイントで実行され、決定論的な制御を提供します。LLMが実行を選択するのに依存するのではなく、特定のアクションが常に発生します。Hooksを使用して、プロジェクトルールを適用し、反復的なタスクを自動化し、Claude Codeを既存のツールと統合します。

判断が必要な決定については、決定論的なルールではなく、Claudeモデルを使用して条件を評価する **プロンプトベースの** *hooks* (ページ 434) または **エージェントベースの** *hooks* (ページ 435) を使用することもできます。

Claude Codeを拡張する他の方法については、Claudeに追加の指示と実行可能なコマンドを与えるための *skills* (ページ 275)、分離されたコンテキストでタスクを実行するための *subagents* (ページ 317)、プロジェクト全体で共有する拡張機能をパッケージ化するための plugins を参照してください。

Tip:

このガイドでは一般的なユースケースと始め方をカバーしています。完全なイベントスキーマ、JSON入力/出力形式、非同期hooksやMCPツールhooksなどの高度な機能については、*Hooks* **リファレンス** (ページ 443) を参照してください。

20.1 最初のhookをセットアップする

Hookを作成するには、**設定ファイル** (ページ 434) に `hooks` ブロックを追加します。このチュートリアルではデスクトップ通知hookを作成するため、Claudeがあなたの入力を待っているときにアラートを受け取ることができます。ターミナルを監視する代わりに。

1. **hookを設定に追加する**

   `~/.claude/settings.json` を開き、`Notification` hookを追加します。ファイルが存在しない場合は、作成してください。以下の例はmacOS用に `osascript` を使用しています。Linux と Windows のコマンドについては、*Claude* **が入力を必要とするときに通知を受け取る** (ページ 413) を参照してください。

```
{
  "hooks": {
    "Notification": [
      {
```

(次のページに続く)

(前のページからの続き)

```
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"Claude Code needs␣
↪your attention\" with title \"Claude Code\"'"
          }
        ]
      }
    ]
  }
}
```

設定ファイルに既に hooks キーがある場合は、オブジェクト全体を置き換えるのではなく、Notification を既存のイベントキーの兄弟として追加します。各イベント名は単一の hooks オブジェクト内のキーです：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [{ "type": "command", "command": "jq -r '.tool_input.file_path'␣
↪| xargs npx prettier --write" }]
      }
    ],
    "Notification": [
      {
        "matcher": "",
        "hooks": [{ "type": "command", "command": "osascript -e 'display␣
↪notification \"Claude Code needs your attention\" with title \"Claude Code\"'"␣
↪}]
      }
    ]
  }
}
```

CLI で実現したい内容を説明して、Claude にフックを書いてもらうこともできます。

2. **設定を確認する**

   Claude Code のプロンプトで /hooks と入力して hooks ブラウザを開きます。新しいフックが Notification の下のリストに表示されます。

3. **フックをテストする**

   Esc を押して CLI に戻ります。ステータスバーに ☒ manual mode on が表示されるまで Shift+Tab を押し、権限が必要な操作を Claude に依頼してから、ターミナル以外のウィンドウに切り替えます。

デスクトップ通知を受け取るはずです。

20.2 自動化できるもの

Hooks を使用すると、Claude Code のライフサイクルの主要なポイントでコードを実行できます：編集後にファイルをフォーマットし、実行前にコマンドをブロックし、Claude が入力を必要とするときに通知を送信し、セッション開始時にコンテキストを注入するなど。Hook イベントの完全なリストについては、*Hooks リファレンス* (ページ 443) を参照してください。

各例には、**設定ファイル** (ページ 434) に追加する準備ができた設定ブロックが含まれています。

本番環境での hooks の例として、別のモデルレビューを実行し、その結果をセッションにフィードバックする場合は、security-guidance プラグインが Claude Code と統合する方法 を参照してください。

20.2.1 Claude が入力を必要とするときに通知を受け取る

Claude が作業を完了して入力を必要とするときはいつでもデスクトップ通知を取得し、ターミナルをチェックせずに他のタスクに切り替えることができます。

このフックは `Notification` イベントを使用します。これは Claude が入力または権限を待っているときに Claude Code が発火するイベントです。正確なタイミングについては、**各通知タイプが発火するタイミング** (ページ 518) を参照してください。

以下の各タブはプラットフォームのネイティブ通知コマンドを使用します。これを `~/.claude/settings.json` に追加します：

macOS

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "osascript -e 'display notification \"Claude Code needs␣
↪your attention\" with title \"Claude Code\"'"
          }
        ]
      }
    ]
  }
}
```

通知が表示されない場合

`osascript` は組み込みの Script Editor アプリを通じて通知をルーティングします。Script Editor に通知権限がない場合、コマンドは何も表示せずに失敗し、macOS は権限の付与を求めません。

Terminal でこれを 1 回実行して、Script Editor を通知設定に表示させます：

```
osascript -e 'display notification "test"'
```

まだ何も表示されません。**System Settings > Notifications** を開き、リストで **Script Editor** を見つけて、**Allow Notifications** をオンにします。コマンドを再度実行して、テスト通知が表示されることを確認します。

Linux

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "notify-send 'Claude Code' 'Claude Code needs your␣
↪attention'"
          }
        ]
      }
    ]
  }
}
```

通知が表示されない場合

notify-send はデスクトップ通知デーモンが必要です。ヘッドレスサーバー、SSH セッション、ほとんどのコンテナにはこれがありません。まずコマンドを直接テストしてください：

```
notify-send 'Claude Code' 'test'
```

コマンドが見つからない場合は、Debian と Ubuntu で libnotify-bin パッケージをインストールするか、ディストリビューションの同等のものをインストールしてください。

Windows (PowerShell)

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "powershell.exe -Command \"[System.Reflection.
↪Assembly]::LoadWithPartialName('System.Windows.Forms'); [System.Windows.Forms.
↪MessageBox]::Show('Claude Code needs your attention', 'Claude Code')\""
```

(次のページに続く)

(前のページからの続き)

```
          }
        ]
      }
    ]
  }
}
```

ダイアログが表示されない場合

このコマンドは画面の隅の通知ではなくダイアログボックスを開くため、ダイアログがターミナルウィンドウの背後で開く可能性があります。まず PowerShell でコマンドを直接テストしてください。

Claude Code を WSL 内で実行する場合、`powershell.exe` が Windows interop を通じて `PATH` で利用可能である必要があります。

空の `matcher` はすべての通知タイプで発火します。特定のイベントでのみ発火させるには、次のいずれかの値に設定します：

Matcher	発火するタイミング
`permission_prompt`	Claude がツール使用を承認する必要があるか、サンドボックス化されたコマンドの**ネットワークリクエスト** (ページ 1087) を承認する必要があり、プロンプトが約 6 秒待機している
`idle_prompt`	Claude が約 60 秒前に応答を完了し、その後入力していない
`auth_success`	認証が完了したとき
`elicitation_dialog`	MCP サーバーが引き出しフォームを開き、約 6 秒入力していない
`elicitation_url_dialog`	MCP サーバーがブラウザ URL を開くよう求め、約 6 秒入力していない
`elicitation_complete`	MCP サーバーが *URL* **モード引き出し** (ページ 550) が完了したことを報告する
`elicitation_response`	MCP 引き出し応答がサーバーに送り返されたとき
`agent_needs_input`	バックグラウンドセッションがあなたの入力を待つのを開始するか、agent view が開いている間のみ発火します。また、ターミナルセッションが agent team チームメイトのターミナルセットアップ質問を表示するか、auto mode の分類器リクエスト料金に関する通知を表示し、約 6 秒入力していないときも発火します
`agent_completed`	バックグラウンドセッションが完了または失敗します。agent view が開いている間のみ発火します
`quota_auto_resume_fired`	Claude Code は claude.ai 使用制限が一時停止した後、タスクを続行します：リセット時、または Claude Code 中に何かを行うことで使用可能になったとき（使用クレジットの追加、プランのアップグレード、モデルの切り替えなど）。**モデル設定の例外** (ページ 100) を参照してください

次のページに続く

表 1 – 前のページからの続き

Matcher	発火するタイミング
quota_auto_resume_stale	claude.ai 使用制限がコンピュータが約 30 分以上スリープしている間にリセットされました。Claude Code はタスクを続行する代わりに Enter キーを押すのを待ちます。より短いスリープの後、それは続行し、代わりに quota_auto_resume_fired を発火します
quota_auto_resume_disabled	Claude Code は claude.ai 使用制限の待機をタスクを続行せずに終了します：autoContinueAtUsageLimit がオフになったか、リセットが Claude Code が開始した待機中に 24 時間以上先に移動した、続行されたタスクが制限に引き続きヒットした、または継続がモデルに到達する前にブロックされました。Esc または Ctrl+C を押すか、**Don't continue automatically** を選択したときは発火しません

Claude Code は permission_prompt をターミナルと Claude Desktop、VS Code 拡張機能、および Agent SDK を通じて許可リクエストに答えるその他のホストで異なる方法でタイミングします。**各通知タイプが発火するタイミング** (ページ 518) を参照して、両方のタイミングを確認してください。

quota_auto_resume_fired、quota_auto_resume_stale、および quota_auto_resume_disabled マッチャーには Claude Code v2.1.234 以降が必要です。

ターミナルセッションでは、サンドボックス化されたコマンドのネットワークリクエストに対する permission_prompt には Claude Code v2.1.246 以降が必要です。

チームメイトのターミナルセットアップ質問に対する agent_needs_input には Claude Code v2.1.248 以降が必要です。

Claude Code のプロンプトで /hooks と入力し、Notification の下にフックが表示されることを確認します。

20.2.2 編集後にコードを自動フォーマットする

Claude が編集するすべてのファイルで Prettier を自動的に実行し、手動操作なしでフォーマットの一貫性を保ちます。

この hook は PostToolUse イベントを Edit|Write マッチャーで使用するため、ファイル編集ツールの後にのみ実行されます。コマンドは jq で編集されたファイルパスを抽出し、Prettier に渡します。これをプロジェクトルートの .claude/settings.json に追加します：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
```

(次のページに続く)

(前のページからの続き)

```
            "command": "jq -r '.tool_input.file_path' | xargs npx prettier --write"
          }
        ]
      }
    ]
  }
}
```

hook をテストするには、Claude に JavaScript ファイルにシングルクォート文字列を含む行を追加するよう求めてください。その後ファイルを開きます：Prettier のデフォルト設定では、hook はそれらをダブルクォートに書き直します。

hook が成功すると、Claude Code は会話に何も表示しません。hook が実行されたことを確認するには、編集されたファイルが再フォーマットされていることを確認するか、**デバッグテクニック** (ページ 441) を参照してください。

ファイルが `Bash` コマンドで書き直されるときを含め、特定のファイルがどのように変更されても再フォーマットするには、代わりに *FileChanged* (ページ 535) hook を使用してください。

注釈:

このページの Bash の例は JSON 解析に `jq` を使用します。macOS で `brew install jq`、Debian と Ubuntu で `apt-get install jq` でインストールするか、jq ダウンロード を参照してください。

20.2.3 保護されたファイルへの編集をブロックする

Claude が `.env`、`package-lock.json`、`.git/` 内のものなどの機密ファイルを変更するのを防ぎます。Claude は編集がブロックされた理由を説明するフィードバックを受け取るため、アプローチを調整できます。

この例は hook が呼び出す別のスクリプトファイルを使用します。スクリプトはターゲットファイルパスを保護されたパターンのリストに対してチェックし、終了コード 2 で編集をブロックします。

1. **hook スクリプトを作成する**

   これを `.claude/hooks/protect-files.sh` に保存します：

```
#!/bin/bash
# protect-files.sh

INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

# Windows バックスラッシュセパレータを正規化して、以下のパターンがマッチするようにします
FILE_PATH="${FILE_PATH//\\//}"

PROTECTED_PATTERNS=(".env" "package-lock.json" ".git/")
```

(次のページに続く)

(前のページからの続き)

```
for pattern in "${PROTECTED_PATTERNS[@]}"; do
  if [[ "$FILE_PATH" == *"$pattern"* ]]; then
    echo "Blocked: $FILE_PATH matches protected pattern '$pattern'" >&2
    exit 2
  fi
done

exit 0
```

2. **macOS と Linux でスクリプトを実行可能にする**

   Hook スクリプトが Claude Code で実行されるには、実行可能である必要があります：

```
chmod +x .claude/hooks/protect-files.sh
```

3. **hook を登録する**

   `.claude/settings.json` に `PreToolUse` hook を追加して、`Edit` または `Write` ツール呼び出しの前にスクリプトを実行します：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/protect-files.sh"
          }
        ]
      }
    ]
  }
}
```

4. **hook をテストする**

   Claude に `.env` ファイルにコメントを追加するよう求めてください。Claude Code は実行前に編集をブロックし、スクリプトの `Blocked:` メッセージを Claude にフィードバックとして渡します。

20.2.4 圧縮後にコンテキストを再注入する

Claude のコンテキストウィンドウがいっぱいになると、圧縮は会話を要約してスペースを解放します。これは重要な詳細を失う可能性があります。`compact` マッチャーで `SessionStart` hook を使用して、すべての圧縮後に重要なコンテキストを再注入します。

Claude Code はコマンドが stdout に書き込むプレーンテキストを Claude のコンテキストに追加します。この例はプロジェクト規約と最近の作業を Claude に思い出させます。これをプロジェクトルートの `.claude/settings.json` に追加します：

```
{
  "hooks": {
    "SessionStart": [
      {
        "matcher": "compact",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Reminder: use Bun, not npm. Run bun test before␣
↪committing. Current sprint: auth refactor.'"
          }
        ]
      }
    ]
  }
}
```

`echo` を `git log --oneline -5` などの動的出力を生成するコマンドに置き換えて、最近のコミットを表示できます。すべてのセッション開始時にコンテキストを注入する場合は、代わりに *CLAUDE.md* (ページ 997) を使用することを検討してください。環境変数については、リファレンスの *CLAUDE_ENV_FILE* (ページ 484) を参照してください。

20.2.5 設定変更を監査する

セッション中に設定またはスキルファイルが変更されたときを追跡します。`ConfigChange` イベントは外部プロセスまたはエディタが設定ファイルを変更したときに発火するため、コンプライアンスのために変更をログに記録したり、不正な変更をブロックしたりできます。

この例は各変更を監査ログに追加します。これを `~/.claude/settings.json` に追加します：

```
{
  "hooks": {
    "ConfigChange": [
      {
        "matcher": "",
        "hooks": [
          {
            "type": "command",
            "command": "jq -c '{timestamp: now | todate, source: .source, file: .file_
↪path}' >> ~/claude-config-audit.log"
          }
        ]
      }
```

(次のページに続く)

(前のページからの続き)

```
    ]
  }
}
```

マッチャーは設定タイプでフィルタリングします：`user_settings`、`project_settings`、`local_settings`、`policy_settings`、または `skills`。変更が有効になるのをブロックするには、終了コード 2 で終了するか、`{"decision": "block"}` を返します。完全な入力スキーマについては、*ConfigChange* **リファレンス** (ページ 531) を参照してください。

hook が変更を記録することを確認するには、セッションが実行中に別のエディタで設定ファイルを編集してから、`~/claude-config-audit.log` を開きます：hook は変更ごとに 1 つの JSON 行をタイムスタンプ、ソース、ファイルパスとともに追加します。

20.2.6 ディレクトリまたはファイルが変更されたときに環境をリロードする

一部のプロジェクトは、どのディレクトリにいるかに応じて異なる環境変数を設定します。direnv などのツールはシェルで自動的にこれを行いますが、Claude の Bash ツールはそれらの変更を自動的に取得しません。

`SessionStart` hook を `CwdChanged` hook とペアリングすることでこれを修正します。`SessionStart` は起動したディレクトリの変数をロードし、`CwdChanged` は Claude がディレクトリを変更するたびにそれらをリロードします。どちらも `CLAUDE_ENV_FILE` に書き込み、Claude Code は各 Bash コマンドの前にスクリプトプリアンブルとして実行します。これを `~/.claude/settings.json` に追加します：

```
{
  "hooks": {
    "SessionStart": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "direnv export bash > \"$CLAUDE_ENV_FILE\""
          }
        ]
      }
    ],
    "CwdChanged": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "direnv export bash > \"$CLAUDE_ENV_FILE\""
          }
        ]
      }
    ]
```

(次のページに続く)

(前のページからの続き)

```
  }
}
```

`direnv allow` をすべてのディレクトリで 1 回実行して、direnv が `.envrc` をロードすることが許可されるようにします。direnv の代わりに devbox または nix を使用する場合、同じパターンは `direnv export bash` の代わりに `devbox shellenv` または `devbox global shellenv` で機能します。

すべてのディレクトリ変更ではなく、特定のファイルに反応するには、`FileChanged` を `matcher` で使用して、監視するファイル名をリストします（パイプで区切られています）。ウォッチリストを構築するために、この値は正規表現として評価されるのではなく、リテラルファイル名に分割されます。*FileChanged* (ページ 535) を参照して、同じ値がファイルが変更されたときにどの hook グループが実行されるかをフィルタリングする方法を確認してください。この例は現在のディレクトリの `.envrc` と `.env` を監視します：

```
{
  "hooks": {
    "FileChanged": [
      {
        "matcher": ".envrc|.env",
        "hooks": [
          {
            "type": "command",
            "command": "direnv export bash > \"$CLAUDE_ENV_FILE\""
          }
        ]
      }
    ]
  }
}
```

入力スキーマ、`watchPaths` 出力、および `CLAUDE_ENV_FILE` の詳細については、*CwdChanged* (ページ 533) および *FileChanged* (ページ 535) リファレンスエントリを参照してください。

20.2.7 特定の許可プロンプトを自動承認する

常に許可するツール呼び出しの承認ダイアログをスキップします。この例は `ExitPlanMode` を自動承認します。これは Claude がプランの提示を終了して続行するよう求めるときに呼び出すツールです。プランが準備できるたびにプロンプトが表示されることはありません。

上記の終了コード例とは異なり、自動承認には hook が JSON 決定を stdout に書き込む必要があります。Claude Code は `PermissionRequest` hook を実行します。これは Claude Code が許可ダイアログを表示しようとするときに実行され、hook が `"behavior": "allow"` を返すと、Claude Code はあなたの代わりにリクエストに答えます。

マッチャーは hook を `ExitPlanMode` のみにスコープするため、他のプロンプトは影響を受けません。これを `~/.claude/settings.json` に追加します：

```
{
  "hooks": {
    "PermissionRequest": [
      {
        "matcher": "ExitPlanMode",
        "hooks": [
          {
            "type": "command",
            "command": "echo '{\"hookSpecificOutput\": {\"hookEventName\": \
↪"PermissionRequest\", \"decision\": {\"behavior\": \"allow\"}}}'"
          }
        ]
      }
    ]
  }
}
```

Hook が承認すると、Claude Code は Plan Mode を終了し、Plan Mode に入る前にアクティブだった許可モードを復元します。トランスクリプトは、ダイアログが表示されたはずの場所に「Allowed by PermissionRequest hook」と表示されます。Hook パスは常に現在の会話を保持します：ダイアログができるように、コンテキストをクリアして新しい実装セッションを開始することはできません。

特定の許可モードを設定する代わりに、hook の出力に `setMode` エントリを含む `updatedPermissions` 配列を含めることができます。`mode` 値は `default`、`acceptEdits`、または `bypassPermissions` などの任意の許可モードであり、`destination: "session"` は現在のセッションのみに適用します。

注釈:

`bypassPermissions` は、セッションが既にバイパスモードで起動された場合にのみ適用されます：`--dangerously-skip-permissions`、`--permission-mode bypassPermissions`、`--allow-dangerously-skip-permissions`、または ユーザー、--settings、または管理設定 の `permissions.defaultMode: "bypassPermissions"`。*permissions.disableBypassPermissionsMode* (ページ 1063) で無効化されている場合、または **制限モード** (ページ 51) でセッションを開始した場合は適用されません。
Claude Code は `defaultMode` として永続化することはありません。

セッションを `acceptEdits` に切り替えるには、hook は stdout に次の JSON を書き込みます：

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedPermissions": [
        { "type": "setMode", "mode": "acceptEdits", "destination": "session" }
      ]
    }
```

(次のページに続く)

(前のページからの続き)

```
  }
}
```

マッチャーをできるだけ狭く保ちます。.* でマッチングするか、マッチャーを空のままにすると、ファイル書き込みやシェルコマンドを含むすべての許可プロンプトが自動承認されます。決定フィールドの完全なセットについては、*PermissionRequest* **リファレンス** (ページ 507) を参照してください。

20.3 hooks の仕組み

Claude Code は、ライフサイクルの特定のポイントで hook イベントを発火させます。イベントが発火すると、Claude Code はすべてのマッチングする hooks を並列で実行します。重複するハンドラーの扱い方については、*Hook* **ハンドラーフィールド** (ページ 455) を参照してください。以下の表は各イベントとそれがトリガーされるときを示しています：

イベント	発火するタイミング
SessionStart	セッションが開始または再開されたとき
Setup	--init-only で Claude Code を起動するとき、または -p モードで --init または --maintenance を使用するとき。CI またはスクリプトでの 1 回限りの準備用
UserPromptSubmit	プロンプトを送信するとき、Claude が処理する前。*Claude Code* **が独自に開始するターン** (ページ 487) でも発火します
UserPromptExpansion	ユーザーが入力したコマンドがプロンプトに展開されるとき、Claude に到達する前。展開をブロックできます
PreToolUse	ツール呼び出しが実行される前。ブロックできます
PermissionRequest	ツール呼び出しが権限決定を必要とするとき
PermissionDenied	オートモードがツール呼び出しを拒否するとき、分類器の判定がない拒否を含みます。JSON hookSpecificOutput.retry: true を使用して、モデルが拒否されたツール呼び出しを再試行できることを伝えます。Claude Code は分類器が判定を出さなかった場合、retry を無視します
PostToolUse	ツール呼び出しが成功した後
PostToolUseFailure	ツール呼び出しが失敗した後
PostToolBatch	並列ツール呼び出しの完全なバッチが解決した後、次のモデル呼び出しの前
Notification	Claude Code が通知を送信するとき
MessageDisplay	アシスタントメッセージテキストが表示されている間
SubagentStart	サブエージェントがスポーンされるとき
SubagentStop	サブエージェントが終了するとき
TaskCreated	TaskCreate 経由でタスクが作成されるとき
TaskCompleted	タスクが完了としてマークされるとき
Stop	Claude が応答を終了するとき
StopFailure	API エラーが原因でターンが終了するとき

次のページに続く

表 2 – 前のページからの続き

イベント	発火するタイミング
TeammateIdle	エージェントチーム のチームメイトがアイドル状態になろうとするとき
InstructionsLoaded	CLAUDE.md または .claude/rules/*.md ファイルがコンテキストに読み込まれるとき。セッション開始時およびセッション中にファイルが遅延読み込みされるときに発火します
ConfigChange	セッション中に設定ファイルが変更されるとき
CwdChanged	作業ディレクトリが変更されるとき、例えば Claude が cd コマンドを実行するとき。direnv などのツールを使用したリアクティブな環境管理に便利です
DirectoryAdded	/add-dir または SDK register_repo_root コントロールリクエスト経由でセッション中盤に作業ディレクトリが追加されるとき
FileChanged	監視対象ファイルがディスク上で変更されるとき。matcher フィールドは監視するファイル名を指定します
WorktreeCreate	--worktree、isolation: "worktree"、またはバックグラウンドセッション経由で worktree が作成されるとき。デフォルトの git 動作を置き換えます
WorktreeRemove	セッション終了時、サブエージェント終了時、またはバックグラウンドセッションを削除するときに worktree が削除されるとき
PreCompact	コンテキスト圧縮の前
PostCompact	コンテキスト圧縮が完了した後
PreModelSwitch	Claude Code があなたまたはクライアントがリクエストしたモデルスイッチを適用する前。スイッチをブロックできます
PostModelSwitch	セッションのモデルが変更された後、Claude Code が独自に行う変更（セッションを再開するときのモデル復元など）を含みます
Elicitation	MCP サーバーがツール呼び出し中にユーザー入力をリクエストするとき
ElicitationResult	ユーザーが MCP エリシテーションに応答した後、レスポンスがサーバーに送り返される前
SessionEnd	セッションが終了するとき

各 hook には、それがどのように実行されるかを決定する type があります。ほとんどの hooks は "type": "command" を使用し、シェルコマンドを実行します。他の 4 つのタイプが利用可能です：

- "type": "http"：イベントデータを URL に POST します。*HTTP hooks* (ページ 436) を参照してください。
- "type": "mcp_tool"：設定済みの MCP サーバー上のツールを呼び出します。*MCP tool hooks* (ページ 461) を参照してください。
- "type": "prompt"：シングルターン LLM 評価。**プロンプトベースの** *hooks* (ページ 434) を参照してください。
- "type": "agent"：ツールアクセス付きマルチターン検証。エージェント hooks は実験的であり、変更される可能性があります。**エージェントベースの** *hooks* (ページ 435) を参照してください。

20.3.1 複数の hooks からの結果を組み合わせる

複数の hooks が同じイベントにマッチする場合、すべての hook のコマンドが完了してから Claude Code は結果をマージします。1 つの hook が deny を返しても、兄弟 hooks の実行は停止されません。1 つの hook の deny が別の hook の副作用を抑制することに依存しないでください。

すべてのマッチングする hooks が完了した後、Claude Code はそれらの出力を組み合わせます。PreToolUse 許可決定については、最も制限的な答えが適用されます。順序は deny、defer、ask、allow です。additionalContext からのテキストはすべての hook から保持され、Claude に一緒に渡されます。

以下の例は Bash に 2 つの PreToolUse hooks を登録しています。最初のものはすべてのコマンドをログファイルに追加して 0 で終了します。2 番目のものはスクリプトを実行し、コマンドに rm -rf が含まれている場合は 2 で終了して拒否します：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r .tool_input.command >> ~/.claude/bash.log"
          },
          {
            "type": "command",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/block-rm-rf.sh"
          }
        ]
      }
    ]
  }
}
```

Claude が rm -rf /tmp/build を実行しようとするとき、両方の hooks が並列で実行されます。ログ hook はコマンドを ~/.claude/bash.log に書き込み、0 で終了します。これは決定がないことを報告します。ガードレール hook は 2 で終了し、ツール呼び出しを拒否します。deny が優先されるため、Claude Code はコマンドをブロックし、Claude にガードレールの stderr を表示します。ログエントリはまだ書き込まれます。なぜなら、ログ hook は既に実行されているからです。

20.3.2 入力を読み取り、出力を返す

Hooks は stdin、stdout、stderr、および終了コードを通じて Claude Code と通信します。イベントが発火すると、Claude Code はイベント固有のデータを JSON としてスクリプトの stdin に渡します。スクリプトはそのデータを読み取り、作業を行い、終了コードを通じて Claude Code に次に何をするかを伝えます。

Hook 入力

すべてのイベントには session_id（セッションの一意の ID）や cwd（イベントが発火したときの作業ディレクトリ）などの共通フィールドが含まれていますが、各イベントタイプは異なるデータを追加します。Claude が Bash コマンドを実行するとき、PreToolUse hook は stdin で次のフィールドを受け取ります：

- hook_event_name：hook をトリガーしたイベント
- tool_name：Claude が使用しようとしているツール
- tool_input：Claude がツールに渡した引数。Bash の場合、その command フィールドはシェルコマンドを保持します。

たとえば、npm test コマンドの hook 入力は次のようになります：

```
{
  "session_id": "abc123",
  "cwd": "/Users/sarah/myproject",
  "hook_event_name": "PreToolUse",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test"
  }
}
```

スクリプトはその JSON を解析し、これらのフィールドのいずれかに基づいて動作できます。UserPromptSubmit hooks は代わりに prompt テキストを取得し、SessionStart hooks は source（startup、resume、clear、compact、または fork）を取得するなど。リファレンスの **共通入力フィールド** (ページ 466) で共有フィールドを参照し、各イベントのセクションでイベント固有のスキーマを参照してください。

Hook 出力

スクリプトは stdout または stderr に書き込み、特定のコードで終了することで、Claude Code に次に何をするかを伝えます。以下は、コマンドをブロックする PreToolUse hook の例です：

```
#!/bin/bash
INPUT=$(cat)
COMMAND=$(echo "$INPUT" | jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q "drop table"; then
  echo "Blocked: dropping tables is not allowed" >&2  # stderr は Claude のフィードバックになります
  exit 2 # exit 2 = アクションをブロック
fi

exit 0  # exit 0 = 決定なし。通常の許可フローが適用されます
```

終了コードは次に何が起こるかを決定します：

- **終了 0**：hook は異議を報告しません。

  - `PreToolUse` hook の場合、これはツール呼び出しを承認しません：通常の **許可フロー** (ページ 1039) が引き続き適用されます。
  - `UserPromptSubmit`、`UserPromptExpansion`、`SessionStart`、および `PostModelSwitch` hooks の場合、Claude Code は stdout を **プレーンテキストとして扱い** (ページ 469) Claude のコンテキストに追加します。

- **終了 2**：Claude Code はアクションをブロックします。stderr に理由を書き込みます。それがどこに着地するかはイベントによって異なります：一部のイベントはそれを Claude にフィードバックとして供給するため、調整できます。他のイベントはそれをユーザーに表示し、`ConfigChange` や `Elicitation` などのいくつかはメッセージを表示しません。一部のイベントはブロックできません：`SessionStart` などの場合、終了 2 は stderr をユーザーに表示し、実行は続行されます。**イベントごとの終了コード *2* の動作** (ページ 472) で完全なリストを参照してください。

- **その他の終了コード**：ほとんどのイベントでは、結果は hook が stdout に出力したものによって異なります：
  - スキーマ検証に合格した解析済みオブジェクト：Claude Code は終了コードを無視し、JSON だけが結果を決定し、hook はエラーとして報告されません。`WorktreeCreate` が任意の非ゼロ終了で失敗するなどの例外は、リファレンスの **終了コード出力** (ページ 469) セクションにリストされています。
  - スキーマ検証に失敗した解析済みオブジェクト、または Claude Code が ***JSON* として解析しようとする** (ページ 469) が有効な JSON ではない stdout：ブロックされないエラー。通知は検証またはパースメッセージを含みます。
  - Claude Code が **プレーンテキストとして扱う** (ページ 469) stdout、または空の stdout：アクションはブロックされないエラーとして進行します。トランスクリプトは `<hook name> hook error` 通知を表示し、その後 stderr の最初の行が `Failed with non-blocking status code:` で始まります。完全な stderr をキャプチャするには、`claude --debug` で **デバッグログ** (ページ 562) を有効にするか、セッション中に `/debug` を実行してください。

構造化 JSON 出力

終了コードはブロックするか沈黙するかのみを許可します。より多くの制御のために、終了 0 して stdout に JSON オブジェクトを出力します。

注釈:

終了 2 で stderr メッセージでブロックするか、終了 0 で JSON で構造化制御を使用します。hook ごとに 1 つのアプローチを選択してください。混在させた場合の動作については、**終了コード出力** (ページ 469) を参照してください。

たとえば、`PreToolUse` hook はツール呼び出しを拒否して理由を Claude に伝えたり、ユーザーの承認のためにエスカレートしたりできます：

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
```

(次のページに続く)

(前のページからの続き)

```
    "permissionDecisionReason": "Use rg instead of grep for better performance"
  }
}
```

"deny" を使用すると、Claude Code はツール呼び出しをキャンセルし、permissionDecisionReason を Claude にフィードバックとして返します。

PreToolUse では、Claude Code は各 permissionDecision 値を次のように処理します：

- "allow"：インタラクティブな許可プロンプトをスキップします。Deny および ask ルール（エンタープライズ管理 deny リストを含む）は引き続き適用されます。また、*requiresUserInteraction* (ページ 404) とマークされた MCP ツールのプロンプトと、**組織が *ask* に設定した** (ページ 399) コネクタツールのプロンプトも適用されます。その設定が Claude Code に到達するセッションでも適用されます。
- "deny"：ツール呼び出しをキャンセルし、理由を Claude に送信します
- "ask"：通常どおりユーザーに許可プロンプトを表示します

4 番目の値 "defer" は、-p フラグ付きの **非インタラクティブモード** (ページ 695) で利用可能です。プロセスを終了し、ツール呼び出しを保持して、Agent SDK ラッパーが入力を収集して再開できるようにします。リファレンスの **ツール呼び出しを後で延期する** (ページ 504) を参照してください。

PreModelSwitch hook は同じ permissionDecision フィールドを返します："allow" はモデルスイッチを進行させ、"deny" はそれをキャンセルします。"ask" はインタラクティブセッションで /model を実行するときに確認を求めます。他の場所では、Claude Code は "ask" を拒否として扱います。*PreModelSwitch* **決定制御** (ページ 546) を参照してください。

他のイベントは異なる決定パターンを使用します。たとえば、PostToolUse および Stop hooks はトップレベルの decision: "block" フィールドを使用し、PermissionRequest は hookSpecificOutput.decision.behavior を使用します。リファレンスの **サマリーテーブル** (ページ 477) でイベント別の完全な内訳を参照してください。

UserPromptSubmit hooks の場合、代わりに hookSpecificOutput.additionalContext を使用して Claude のコンテキストにテキストを注入します。additionalContext を hookSpecificOutput の内側にネストします。JSON の最上位レベルに配置すると、Claude Code はそれを無視します。たとえば、この出力はすべてのプロンプトに現在のブランチ状態を追加します：

```
{
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "Current branch: release-42. Deploy freeze until Friday."
  }
}
```

ブロックプロンプトとセッションタイトルの設定を含む完全な出力形状については、*UserPromptSubmit* **決定制御** (ページ 489) を参照してください。

type: "prompt" の Hooks は出力を異なる方法で処理します：**プロンプトベースの *hooks*** (ページ 434) を参照してください。

20.3.3 マッチャーで hooks をフィルタリングする

マッチャーなしでは、hook はそのイベントのすべての発生で発火します。マッチャーを使用すると、それを絞り込むことができます。たとえば、ファイル編集後にのみフォーマッターを実行したい場合（すべてのツール呼び出しの後ではなく）、`PostToolUse` hook にマッチャーを追加します：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          { "type": "command", "command": "prettier --write ..." }
        ]
      }
    ]
  }
}
```

`"Edit|Write"` マッチャーは `Edit` または `Write` ツール呼び出しでのみ発火し、`Bash`、`Read`、または他のツールでは発火しません。カンマもまた同じ方法で代替を区切るため、`"Edit, Write"` は同等です。**マッチャーパターン** (ページ 451) を参照して、プレーン名と正規表現がどのように評価されるかを確認してください。

注釈:

Claude はまた、シェルコマンドを実行することでファイルを作成または変更できます。コンプライアンススキャンまたは監査ログなど、hook がすべてのファイル変更を確認する必要がある場合は、ターンごとに 1 回作業ツリーをスキャンする *Stop* (ページ 526) hook を追加してください。呼び出しごとのカバレッジの場合は、`Bash|PowerShell` もマッチさせ、スクリプトで `git status --porcelain` を使用して変更されたファイルと追跡されていないファイルをリストアップしてください。*PowerShell hook* **入力セクション** (ページ 497) は、`Bash` だけをマッチさせるのが十分でない理由を説明しています。ディスク上の特定のファイルが変更されたときに hook を実行するには、それを書き込んだものが何であれ、*FileChanged* (ページ 535) hook を使用してください。

各イベントタイプは特定のフィールドでマッチします：

イベント	マッチャーがフィルタリングするもの	マッチャー値の例
`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`、`PermissionDenied`	ツール名	`Bash`、`Edit|Write`、`mcp__.*`
`SessionStart`	セッションがどのように開始されたか	`startup`、`resume`、`clear`、`compact`、`fork`

次のページに続く

表 3 – 前のページからの続き

イベント	マッチャーがフィルタリングするもの	マッチャー値の例
Setup	どの CLI フラグがセットアップをトリガーしたか	init、maintenance
SessionEnd	セッションが終了した理由	clear、resume、logout、prompt_input_exit、other
Notification	通知タイプ	permission_prompt、idle_prompt、auth_success、elicitation_dialog、elicitation_url_dialog、elicitation_complete、elicitation_response、agent_needs_input、agent_completed、quota_auto_resume_fired、quota_auto_resume_stale、quota_auto_resume_disabled
SubagentStart	エージェントタイプ	general-purpose、Explore、Plan、またはカスタムエージェント名
PreCompact、PostCompact	圧縮をトリガーしたもの	manual、auto
PreModelSwitch、PostModelSwitch	セッションが切り替わるモデルの正規名（*PreModelSwitch* (ページ 542) で説明）	claude-opus-5、claude-opus-4-6|claude-opus-5、.*opus.*
SubagentStop	エージェントタイプ	SubagentStart と同じ値
ConfigChange	設定ソース	user_settings、project_settings、local_settings、policy_settings、skills
DirectoryAdded	ディレクトリがどのように追加されたか	slash_command、register_repo_root
StopFailure	エラータイプ	rate_limit、overloaded、authentication_failed、oauth_org_not_allowed、account_on_hold、billing_error、invalid_request、model_not_found、server_error、max_output_tokens、cloud_credential_error、unknown
InstructionsLoaded	ロード理由	session_start、nested_traversal、path_glob_match、include、compact
Elicitation	MCP サーバー名	設定した MCP サーバー名
ElicitationResult	MCP サーバー名	Elicitation と同じ値

次のページに続く

表 3 – 前のページからの続き

イベント	マッチャーがフィルタリングするもの	マッチャー値の例
FileChanged	リテラルファイル名を監視 (*FileChanged* (ページ 535) を参照)	.envrc|.env
UserPromptExpansion	コマンド名	スキルまたはコマンド名
UserPromptSubmit、PostToolBatch、Stop、TeammateIdle、TaskCreated、TaskCompleted、WorktreeCreate、WorktreeRemove、CwdChanged、MessageDisplay	マッチャーサポートなし	すべての発生で常に発火

以下のタブは、異なるイベントタイプのいくつかのマッチャーを示しています。

すべての Bash コマンドをログに記録する

Bash ツール呼び出しのみをマッチし、各コマンドをファイルにログに記録します。PostToolUse イベントはコマンドが完了した後に発火するため、tool_input.command は実行されたものを含みます。Hook は stdin で JSON としてイベントデータを受け取り、jq -r '.tool_input.command' はコマンド文字列のみを抽出し、>> はログファイルに追加します：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "jq -r '.tool_input.command' >> ~/.claude/command-log.txt"
          }
        ]
      }
    ]
  }
}
```

MCP ツールをマッチさせる

MCP ツールは組み込みツールとは異なる命名規則を使用します：mcp__<server>__<tool>。ここで <server> は MCP サーバー名で、<tool> はそれが提供するツールです。たとえば、mcp__github__search_repositories または mcp__filesystem__read_file。**プラグインバンドルサーバー** (ページ 378) からのツールは、mcp__plugin_my-plugin_db__query などのスコープ付きサーバーセグメントを使用します。特定のサーバーからすべてのツールをターゲットするために正規表現マッチャーを使用するか、mcp__.*__write.* のようなパターンでサーバー全体でマッチします。リファレンスの *MCP* **ツールをマッチさせる** (ページ 454) を参照して、完全な例のリストを確認してください。

以下のコマンドは hook の JSON 入力からツール名を `jq` で抽出し、stderr に書き込みます。stderr に書き込むことで stdout をクリーンに保ち、メッセージを **デバッグログ** (ページ 562) に送信します：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__github__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo \"GitHub tool called: $(jq -r '.tool_name')\" >&2"
          }
        ]
      }
    ]
  }
}
```

セッション終了時にクリーンアップする

`SessionEnd` イベントはセッションが終了した理由のマッチャーをサポートします。この hook は `clear`（`/clear` を実行するとき）でのみ発火し、通常の終了では発火しません：

```
{
  "hooks": {
    "SessionEnd": [
      {
        "matcher": "clear",
        "hooks": [
          {
            "type": "command",
            "command": "rm -f /tmp/claude-scratch-*.txt"
          }
        ]
      }
    ]
  }
}
```

`if` フィールドでツール名と引数でフィルタリングする

`if` フィールドは **許可ルール構文** (ページ 1039) を使用して、ツール名と引数の両方で hooks をフィルタリングするため、hook プロセスはツール呼び出しがマッチするときにのみ生成されます。これは `matcher` を超えており、ツール名のみでグループレベルでフィルタリングします。

たとえば、すべての Bash コマンドではなく、Claude が `git` コマンドを使用するときにのみ hook を実行するには：

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "if": "Bash(git *)",
            "command": "\"$CLAUDE_PROJECT_DIR\"/.claude/hooks/check-git-policy.sh"
          }
        ]
      }
    ]
  }
}
```

Hook コマンドが実行されるかどうかは、`if` パターンの形状と Claude が呼び出している Bash コマンドによって異なります：

`if` パターン	Bash コマンド	Hook は実行されるか？	理由
`Bash(git *)`	`git push`	はい	コマンド名がマッチします
`Bash(git *)`	`npm test && git push`	はい	各サブコマンドがチェックされます。`git push` がマッチします
`Bash(git *)`	`echo $(git log)`	はい	`$()` とバッククォート内のコマンドがチェックされます。`git log` がマッチします
`Bash(git *)`	`echo $(date)`	いいえ	サブコマンドが `git *` にマッチしません
`Bash(git push *)`	`echo $(date)`	はい	コマンド名以上を指定するパターンは、`$()`、バッククォート、または `$VAR` で hook を実行します

Claude Code がどのコマンドが Bash 入力を実行するかを判断できない場合、パターンに関係なく hook を実行します。*Bash* **マッチングテーブル** (ページ 457) は、Claude Code がサブコマンドで絞り込むことができる、またはできないコマンド形状をカバーしています。フィルターはベストエフォートであるため、ハード allow または deny を強制するには、hook ではなく **許可システム** (ページ 1039) を使用してください。

`if` フィールドは許可ルールと同じパターンを受け入れます：`"Bash(git *)"`、`"Edit(*.ts)"` など。複数のツール名をマッチさせるには、それぞれ独自の `if` 値を持つ別のハンドラーを使用するか、パイプ交替がサポートされている `matcher` レベルでマッチします。

`if` はツールイベントでのみ機能します：`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`、および `PermissionDenied`。他のイベントに追加すると、hook が実行されるのを防ぎます。

20.3.4 hook の場所を設定する

Hook を追加する場所がそのスコープを決定します：

場所	スコープ	共有可能
~/.claude/settings.json	すべてのプロジェクト	いいえ、マシンにローカル
.claude/settings.json	単一プロジェクト	はい、リポジトリにコミット可能
.claude/settings.local.json	単一プロジェクト	いいえ、gitignored
管理ポリシー設定	組織全体	はい、管理者制御
Plugin hooks/hooks.json	プラグインが有効なとき	はい、プラグインにバンドル
Skill (ページ 275) frontmatter	スキルが呼び出されたら、セッションの残り。**スキルとエージェントの *Hooks*** (ページ 465) を参照してください。	はい、スキルファイルで定義
Subagent (ページ 317) frontmatter	そのサブエージェントが実行されている間	はい、サブエージェントファイルで定義

Claude Code で */hooks* (ページ 465) を実行して、イベント別にグループ化されたすべての設定済み hooks を参照します。

hooks を無効にするには、設定ファイルで `"disableAllHooks": true` を設定します。Claude Code は **設定の優先順位** (ページ 466) の後に残された値を読み取るため、プロジェクトの設定ファイルはあなたのものをオーバーライドできます。管理設定で設定された Hooks は、`disableAllHooks` がそこにも設定されていない限り、実行されます。各レベルの完全な範囲については、`disableAllHooks` を参照してください。

Claude Code が実行中に設定ファイルを直接編集する場合、ファイルウォッチャーは通常、hook の変更を自動的に取得します。

20.4 プロンプトベースの hooks

決定論的なルールではなく判断が必要な決定については、`type: "prompt"` hooks を使用します。シェルコマンドを実行する代わりに、Claude Code はプロンプトと hook の入力データを Claude モデルに送信して決定を下します。より多くの機能が必要な場合は、`model` フィールドで異なるモデルを指定できます。

モデルの唯一の仕事は、その決定を JSON として返すことです：

- `"ok": true`：アクションが続行されます
- `"ok": false`：何が起こるかはイベントによって異なります：
  - `Stop` および `SubagentStop`：`reason` は Claude にフィードバックとして返されるため、作業を続けます。ただし、レスポンスが `"impossible": true` も設定している場合は、その条件が決して満たされることのない条件として印付けられ、その場合 Claude Code は停止を許可してターンが終了します
  - `PreToolUse`：ツール呼び出しが拒否されます。デフォルトではターンが終了し、拒否の `reason` は警告行としてチャットに表示されます。hook に `continueOnBlock: true` を設定すると、代

わりに reason を Claude にツールエラーとして返すため、調整して続行できます。v2.1.210 より前では、拒否の reason は Claude にツールエラーとして返され、ターンが続行されていました

- PostToolUse：デフォルトではターンが終了し、reason は警告行としてチャットに表示されます。continueOnBlock: true を設定すると、reason を Claude にフィードバックとして返し、代わりにターンを続行します
- PostToolBatch、UserPromptSubmit、および UserPromptExpansion：ターンが終了し、reason は警告行としてチャットに表示されます

この例は Stop hook を使用して、要求されたすべてのタスクが完了しているかどうかをモデルに尋ねます。モデルが条件がまだ満たされていないため "ok": false を返す場合、Claude は作業を続け、reason を次の指示として使用します：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Check if all tasks are complete. If not, respond with {\"ok\":␣
↪false, \"reason\": \"what remains to be done\"}."
          }
        ]
      }
    ]
  }
}
```

完全な設定オプションについては、リファレンスの **プロンプトベースの** *hooks* (ページ 552) を参照してください。

20.5 エージェントベースの hooks

警告:

エージェント hooks は実験的です。動作と設定は将来のリリースで変更される可能性があります。本番ワークフローについては、**コマンド** *hooks* (ページ 458) を優先してください。

検証がファイルの検査またはコマンドの実行を必要とする場合、type: "agent" hooks を使用します。プロンプト hooks は単一の LLM 呼び出しを行いますが、エージェント hooks は条件を返す前にファイルを読み取り、コードを検索し、他のツールを使用できる subagent を生成します。

エージェント hooks は "ok" / "reason" 応答形式を使用し、デフォルトのタイムアウトが 60 秒で、最大 50 ツール使用ターンです。プロンプト hooks の impossible フィールドはサポートされていません。ok: false の場合、Claude Code はエージェント hooks を同じイベント上で continueOnBlock: true を持

つプロンプト hooks と同じ方法で処理するため、`PreToolUse` と `PostToolUse` ではターンが続行されます。エージェント hooks には `continueOnBlock` フィールドがありません。フィールドを含む **エージェント *hooks* の設定** (ページ 557) を参照してください。これには Claude Code が hooks の JSON 入力に置き換える `$ARGUMENTS` プレースホルダーが含まれます。

この例は Claude が停止することを許可する前にテストが合格することを検証します：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and check␣
↪the results. $ARGUMENTS",
            "timeout": 120
          }
        ]
      }
    ]
  }
}
```

Hook 入力データだけで決定を下すのに十分な場合はプロンプト hooks を使用します。コードベースの実際の状態に対して何かを検証する必要がある場合はエージェント hooks を使用します。

完全な設定オプションについては、リファレンスの **エージェントベースの *hooks*** (ページ 556) を参照してください。

20.6 HTTP hooks

`type: "http"` hooks を使用して、シェルコマンドを実行する代わりに、イベントデータを HTTP エンドポイントに POST します。エンドポイントはコマンド hook が stdin で受け取るのと同じ JSON を受け取り、HTTP レスポンスボディを使用して同じ JSON 形式で結果を返します。

HTTP hooks は、Web サーバー、クラウド関数、または外部サービスに hook ロジックを処理させたい場合に便利です。たとえば、チーム全体のツール使用イベントをログに記録する共有監査サービス。

この例はすべてのツール使用をローカルログサービスに POST します：

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "http",
```

(次のページに続く)

(前のページからの続き)

```
            "url": "http://localhost:8080/hooks/tool-use",
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

エンドポイントは、コマンド hooks と同じ **出力形式** (ページ 474) を使用して JSON レスポンスボディを返す必要があります。ツール呼び出しをブロックするには、適切な `hookSpecificOutput` フィールドで 2xx レスポンスを返します。HTTP ステータスコードだけではアクションをブロックできません。

ヘッダー値は `$VAR_NAME` または `${VAR_NAME}` 構文を使用した環境変数補間をサポートします。`allowedEnvVars` 配列にリストされている変数のみが解決されます。他のすべての `$VAR` 参照は空のままです。

完全な設定オプションとレスポンス処理については、リファレンスの *HTTP hooks* (ページ 460) を参照してください。

20.7 制限事項とトラブルシューティング

20.7.1 制限事項

フックを設計する際は、以下の制約に留意してください。

- コマンドフックは stdout、stderr、終了コードのみを介して通信します。/ コマンドやツール呼び出しをトリガーすることはできません。`additionalContext` を介して返されたテキストは**システムリマインダー**として挿入され、Claude はこれをプレーンテキストとして読み取ります。HTTP フックは代わりにレスポンスボディを介して通信します。
- フックのタイムアウトはタイプによって異なります。フックごとに `timeout` フィールド（秒単位）で上書きできます。
  - `command`、`http`、`mcp_tool`：10 分。Claude Code は、`UserPromptSubmit`、`PreModelSwitch`、`PostModelSwitch` フックではこのデフォルトを 30 秒に、`MessageDisplay` では 10 秒に引き下げます。
  - `prompt`：30 秒。
  - `agent`：60 秒。
  - *`SessionEnd`* (ページ 548) フックは、タイプを問わず 1.5 秒の割り当て時間を共有します。設定でフックごとにより長い `timeout` を指定している場合、Claude Code は最大 60 秒までそれに合わせて割り当て時間を引き上げます。
- `PostToolUse` フックは、ツールがすでに実行されているため、アクションを元に戻すことはできません。

- *PermissionRequest* (ページ 506) フックは、Claude Code がユーザーに権限を求めようとするとき、またはプロンプトを表示できない呼び出しをそのままでは自動的に拒否するときに発火します。-p フラグを使用した**非対話モード** (ページ 695) でも、dontAsk モード以外では引き続き実行され、どのフックも判断せず、他に応答できるものもない呼び出しは拒否されます。
- Stop フックは、タスクの完了時だけでなく、Claude が応答を終えるたびに発火します。ユーザーによる中断では発火しません。API エラーの場合は代わりに *StopFailure* (ページ 529) が発火します。
- 複数の PreToolUse フックがツールの引数を書き換えるために *updatedInput* (ページ 495) を返す場合、最後に完了したものが有効になります。フックは並列に実行されるため、順序は非決定的です。同じツールの入力を複数のフックで変更することは避けてください。

20.7.2 フックと権限モード

PreToolUse フックは、dontAsk を含むすべての権限モードにおいて、権限モードのチェックより前に発火します。permissionDecision: "deny" を返すフックは、bypassPermissions モードや --dangerously-skip-permissions を使用している場合でもツールをブロックします。これにより、ユーザーが権限モードを変更しても回避できないポリシーを適用できます。

逆は成り立ちません。"allow" を返すフックは、設定の拒否ルールを回避することはできず、また *requiresUserInteraction* (ページ 404) が指定された MCP ツールや、その設定が Claude Code に反映されるセッションにおいて**組織が *ask* に設定した** (ページ 399) コネクタツールのプロンプトを抑制することもできません。設定ファイルやプラグインの hooks/hooks.json 内のフックは、制限を厳しくすることはできますが、権限ルールが許可する範囲を超えて緩めることはできません。

インストールした mod が tool.check を処理する場合、そのフックが管理設定にない限り、PreToolUse フックがブロックした呼び出しを mod が承認できます。mod に対してどのルールが優先されるかは、**フックで権限を拡張する** (ページ 1059) に記載されています。

20.7.3 フックが発火しない

フックは設定されているが、一度も実行されない場合。

- /hooks を実行し、フックが正しいイベントの下に表示されていることを確認します
- matcher のパターンがツール名と正確に一致していることを確認します。matcher は大文字と小文字を区別します
- 正しいイベントタイプをトリガーしていることを確認します。PreToolUse はツールの実行前に、PostToolUse は実行後に発火します。PermissionRequest フックは Claude Code がユーザーに権限を求めようとするときに発火します。非対話の場合については**制限事項** (ページ 437) を参照してください

20.7.4 出力にフックエラーが表示される

トランスクリプトに「PreToolUse hook error: ...」のようなメッセージが表示される場合。

- スクリプトが予期せずゼロ以外のコードで終了しています。サンプルの JSON をパイプで渡して手動でテストします。

```
echo '{"tool_name":"Bash","tool_input":{"command":"ls"}}' | ./my-hook.sh
echo $?  # Check the exit code
```

- 「command not found」と表示される場合は、絶対パスまたは `${CLAUDE_PROJECT_DIR}` を使用してスクリプトを参照します。シェルのクォートを完全に回避するには、`"args": []` を追加して *exec* **形式** (ページ 459) に切り替えます。これにより、シェルを介さずにスクリプトが直接起動されます
- 「jq: command not found」と表示される場合は、`jq` をインストールするか、JSON の解析に Python/Node.js を使用します
- 通知に JSON 検証メッセージが表示される場合、フックの stdout は JSON として解析されましたが、スキーマ検証に失敗しています。JSON 解析メッセージが表示される場合、stdout は JSON オブジェクトのように見えましたが、有効な JSON ではありませんでした。どちらも終了コード 0 の場合でも発生します。

  解析の失敗を修正するには、文字列の連結ではなく `jq` などの JSON エンコーダーを使用してペイロードを組み立て、値に含まれる引用符やバックスラッシュがエスケープされるようにします。終了コードと JSON の組み合わせについては、リファレンスの**終了コードの出力** (ページ 469) セクションで説明しています
- スクリプトがまったく実行されていない場合は、実行可能にします：`chmod +x ./my-hook.sh`

20.7.5 /hooks にフックが設定されていないと表示される

設定ファイルを編集したが、フックがメニューに表示されない場合。

- ファイルの編集は通常自動的に反映されます。数秒経っても表示されない場合は、ファイルウォッチャーが変更を検出できなかった可能性があります。セッションを再起動して強制的に再読み込みしてください。
- JSON が有効であることを確認します。末尾のカンマやコメントは使用できません
- 設定ファイルが正しい場所にあることを確認します。プロジェクトのフックは `.claude/settings.json`、グローバルのフックは `~/.claude/settings.json` です
- メニューに `Only hooks from managed settings run here` と表示される場合は、組織が `allowManagedHooksOnly` を設定しています。ユーザー、プロジェクト、ローカルの設定ファイル内のフックは実行されず、一覧にも表示されません

20.7.6 Stop フックがブロック上限に達する

Claude が停止せずに作業を続け、その後 Stop フックが連続してブロックした回数が多すぎるという警告とともにターンを終了する場合。

Claude Code は、Stop フックが進展のないまま 8 回連続でブロックすると、そのフックを上書きします。フックスクリプトでは、すでに継続をトリガーしたかどうかを確認する必要があります。JSON 入力から `stop_hook_active` フィールドを解析し、`true` の場合は早期に終了します。

```
#!/bin/bash
INPUT=$(cat)
if [ "$(echo "$INPUT" | jq -r '.stop_hook_active')" = "true" ]; then
  exit 0  # Allow Claude to stop
fi
# ... rest of your hook logic
```

フックが収束するまでに正当な理由で 8 回を超える反復が必要な場合は、CLAUDE_CODE_STOP_HOOK_BLOCK_CAP で上限を引き上げてください。

20.7.7 フックの JSON が効果を持たない

フックは有効な JSON を出力しているが、判断が反映されず、トランスクリプトにもエラーが表示されない場合。どの原因に該当するかを確認してください。

- **JSON の前に余分な出力がある**：他の何かが先に stdout に書き込んでいます。通常はシェルプロファイル内の無条件の echo です。そのため出力が { で始まらなくなり、Claude Code はそれを JSON として解析しません。原因と修正方法はこのリストの後で説明します。
- **フィールドの階層が間違っている**：各フィールドの配置を *JSON 出力* (ページ 474) の形式と比較してください。たとえば、permissionDecision はトップレベルではなく hookSpecificOutput の内側に配置する必要があります。

Claude Code がシェル形式のコマンドフック（args のないもの）を実行する場合、macOS と Linux では sh -c を、Windows では Git Bash を、Git Bash がインストールされていない場合はデフォルトで PowerShell を起動します。このシェルは非対話ですが、Git Bash や、BASH_ENV が ~/.bashrc を指しているなどの一部の設定では、プロファイルが読み込まれます。そのプロファイルに無条件の echo 文が含まれていると、その出力がフックの JSON の前に付加されます。

```
Shell ready on arm64
{"decision": "block", "reason": "Not allowed"}
```

結合された出力は { で始まらなくなるため、Claude Code は stdout 全体をプレーンテキストとして扱い、JSON を無視します。終了コード 0 の場合、トランスクリプトには何も報告されず、解析の試行は**デバッグログ** (ページ 562) にのみ記録されます。これを修正するには、シェルプロファイル内の echo 文を、対話シェルでのみ実行されるように囲みます。

```
# In ~/.zshrc or ~/.bashrc
if [[ $- == *i* ]]; then
  echo "Shell ready"
fi
```

$- 変数にはシェルのフラグが含まれており、i は対話を意味します。フックは非対話シェルで実行されるため、echo はスキップされます。

フックが permissionDecision や additionalContext を hookSpecificOutput の内側ではなくトップレベルで返した場合でも、JSON は解析されますが、Claude Code は誤って配置されたフィールドをエラーを報告せずに無視します。どのフィールドが無視されたかを確認するには、claude --debug で Claude Code を起動し、**デバッグログ** (ページ 562) で Hook JSON output had unrecognized keys を検索します。

20.7.8 デバッグ手法

`Ctrl+O` を押してトランスクリプトビューを開き、フック実行の結果を確認します。

- **実行成功**：フックの JSON が `systemMessage` や Stop フックのフィードバックなどを表示しない限り、何も表示されません。
  - フックが実行されたことを確認するには、ファイルが再フォーマットされたなどの効果を確認するか、以下で説明するようにデバッグログをオンにしてから再度フックをトリガーします
- **ブロッキングエラー**：ほとんどのイベントでは、フックのフィードバックが表示されます。フックの JSON がブロックの判断を行った場合、フィードバックはその判断の理由です。それ以外の場合はフックの stderr です。`ConfigChange` や `Elicitation` などの一部のイベントでは、ブロックしてもメッセージは表示されません。
- **非ブロッキングエラー**：アクションは続行され、`<hook name> hook error` という通知と短い説明が表示されます。説明は、`Failed with non-blocking status code:` を先頭に付けた stderr の最初の行や、JSON の検証メッセージまたは解析メッセージなどです。

どの終了コードと JSON の組み合わせがそれぞれの結果を生むか（イベントごとの例外を含む）は、リファレンスの**終了コードの出力** (ページ 469) セクションで定義されています。

どのフックが一致したか、その終了コード、stdout、stderr を含む実行の詳細をすべて確認するには、デバッグログを読みます。`claude --debug-file /tmp/claude.log` で Claude Code を起動して既知のパスに書き込み、別のターミナルで `tail -f /tmp/claude.log` を実行します。このフラグを付けずに起動した場合は、セッションの途中で `/debug` を実行してログを有効にし、ログのパスを確認します。

20.8 詳細を学ぶ

- *Hooks* **リファレンス** (ページ 443)：完全なイベントスキーマ、JSON 出力形式、非同期 hooks、および MCP ツール hooks
- **セキュリティに関する考慮事項** (ページ 560)：共有または本番環境に hooks をデプロイする前に確認してください
- Bash コマンドバリデーター例：完全なリファレンス実装

第21章　Hooks リファレンス

Claude Code のフック イベント、設定スキーマ、JSON 入出力形式、終了コード、非同期フック、HTTP フック、プロンプト フック、MCP ツール フックのリファレンス。

Tip:

例を含むクイックスタート ガイドについては、**フックでアクションを自動化する** (ページ 411) を参照してください。

フックは、Claude Code のライフサイクル内の特定のポイントで自動的に実行されるユーザー定義のシェル コマンド、HTTP エンドポイント、MCP ツール呼び出し、LLM プロンプト、またはサブエージェントです。Claude Code は、ターミナルでのセッション、IDE 拡張機能、デスクトップアプリ、**クラウドセッション** (ページ 223) など、どこで実行されていても同じフック イベントを発火します。このリファレンスを使用して、イベント スキーマ、設定オプション、JSON 入出力形式、非同期フック、HTTP フック、MCP ツール フックなどの高度な機能を検索してください。

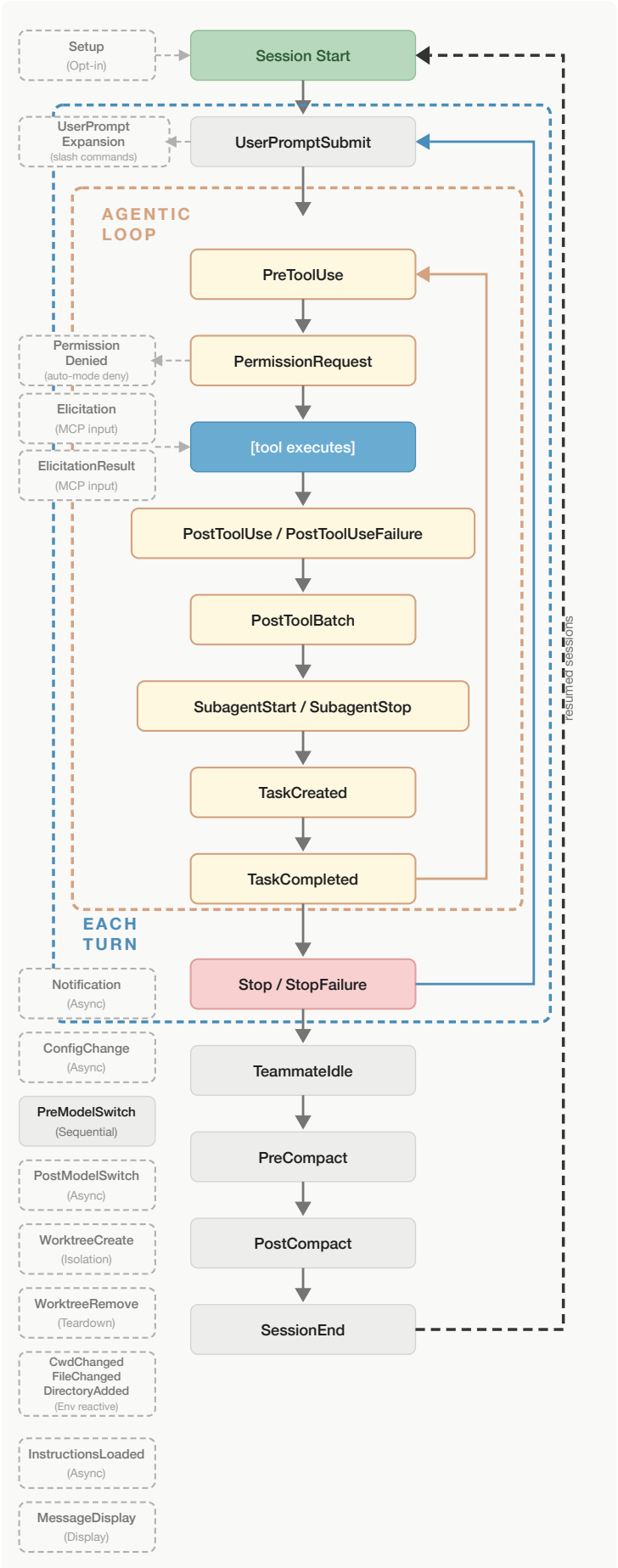
プラグインは、Claude Code が自身のプロセス内で呼び出す JavaScript 関数としてフックを登録することもでき、これによりイベントに応じて動作するだけでなく、インターフェースに描画することもできます。そうしたプラグインは mod であり、それらの関数フックについてはここではなくイベントに反応するで説明しています。このページで説明するフックは、mod と併用しても引き続き動作します。

21.1 フック ライフサイクル

Claude Code は、セッション中の特定のポイントでフックを実行します。イベントが発火して matcher がマッチすると、Claude Code はイベントに関する JSON コンテキストをフック ハンドラーに渡します。コマンド フックの場合、入力は stdin に到着します。HTTP フックの場合、POST リクエスト本体として到着します。ハンドラーは入力を検査し、アクションを実行し、オプションで決定を返すことができます。

イベントは 3 つのケイデンスに分類されます。

- セッションごとに 1 回：`SessionStart` と `SessionEnd`
- ターンごとに 1 回：`UserPromptSubmit`、`Stop`、`StopFailure`
- エージェント型ループ内のすべてのツール呼び出しで：`PreToolUse` と `PostToolUse`。ただし、`End Conversation` の呼び出しは両方をスキップします


Setup
(Opt-in)
Session Start
UserPrompt
Expansion
(slash commands)
UserPromptSubmit
AGENTIC
LOOP
PreToolUse
Permission
Denied
(auto-mode deny)
PermissionRequest
Elicitation
(MCP input)
[tool executes]
ElicitationResult
(MCP input)
PostToolUse / PostToolUseFailure
PostToolBatch
SubagentStart / SubagentStop
TaskCreated
TaskCompleted
EACH
TURN
Notification
(Async)
Stop / StopFailure
ConfigChange
(Async)
TeammateIdle
PreModelSwitch
(Sequential)
PreCompact
PostModelSwitch
(Async)
PostCompact
WorktreeCreate
(Isolation)
WorktreeRemove
(Teardown)
SessionEnd
CwdChanged
FileChanged
DirectoryAdded
(Env reactive)
InstructionsLoaded
(Async)
MessageDisplay
(Display)
resumed sessions

以下の表は、各イベントがいつ発火するかをまとめています。**フック イベント** (ページ 480) セクションでは、各イベントの完全な入力スキーマと決定制御オプションについて説明しています。

イベント	発火するタイミング
`SessionStart`	セッションが開始または再開されたとき
`Setup`	`--init-only` で Claude Code を起動するとき、または `-p` モードで `--init` または `--maintenance` を使用するとき。CI またはスクリプトでの 1 回限りの準備用
`UserPromptSubmit`	プロンプトを送信するとき、Claude が処理する前。*Claude Code* **が独自に開始するターン** (ページ 487) でも発火します
`UserPromptExpansion`	ユーザーが入力したコマンドがプロンプトに展開されるとき、Claude に到達する前。展開をブロックできます
`PreToolUse`	ツール呼び出しが実行される前。ブロックできます
`PermissionRequest`	ツール呼び出しが権限決定を必要とするとき
`PermissionDenied`	オートモードがツール呼び出しを拒否するとき、分類器の判定がない拒否を含みます。JSON `hookSpecificOutput.retry: true` を使用して、モデルが拒否されたツール呼び出しを再試行できることを伝えます。Claude Code は分類器が判定を出さなかった場合、`retry` を無視します
`PostToolUse`	ツール呼び出しが成功した後
`PostToolUseFailure`	ツール呼び出しが失敗した後
`PostToolBatch`	並列ツール呼び出しの完全なバッチが解決した後、次のモデル呼び出しの前
`Notification`	Claude Code が通知を送信するとき
`MessageDisplay`	アシスタントメッセージテキストが表示されている間
`SubagentStart`	サブエージェントがスポーンされるとき
`SubagentStop`	サブエージェントが終了するとき
`TaskCreated`	`TaskCreate` 経由でタスクが作成されるとき
`TaskCompleted`	タスクが完了としてマークされるとき
`Stop`	Claude が応答を終了するとき
`StopFailure`	API エラーが原因でターンが終了するとき
`TeammateIdle`	エージェントチーム のチームメイトがアイドル状態になろうとするとき
`InstructionsLoaded`	CLAUDE.md または `.claude/rules/*.md` ファイルがコンテキストに読み込まれるとき。セッション開始時およびセッション中にファイルが遅延読み込みされるときに発火します
`ConfigChange`	セッション中に設定ファイルが変更されるとき
`CwdChanged`	作業ディレクトリが変更されるとき、例えば Claude が `cd` コマンドを実行するとき。direnv などのツールを使用したリアクティブな環境管理に便利です
`DirectoryAdded`	`/add-dir` または SDK `register_repo_root` コントロールリクエスト経由でセッション中盤に作業ディレクトリが追加されるとき
`FileChanged`	監視対象ファイルがディスク上で変更されるとき。`matcher` フィールドは監視するファイル名を指定します

次のページに続く

表 1 – 前のページからの続き

イベント	発火するタイミング
WorktreeCreate	--worktree、isolation: "worktree"、またはバックグラウンドセッション経由で worktree が作成されるとき。デフォルトの git 動作を置き換えます
WorktreeRemove	セッション終了時、サブエージェント終了時、またはバックグラウンドセッションを削除するときに worktree が削除されるとき
PreCompact	コンテキスト圧縮の前
PostCompact	コンテキスト圧縮が完了した後
PreModelSwitch	Claude Code があなたまたはクライアントがリクエストしたモデルスイッチを適用する前。スイッチをブロックできます
PostModelSwitch	セッションのモデルが変更された後、Claude Code が独自に行う変更（セッションを再開するときのモデル復元など）を含みます
Elicitation	MCP サーバーがツール呼び出し中にユーザー入力をリクエストするとき
ElicitationResult	ユーザーが MCP エリシテーションに応答した後、レスポンスがサーバーに送り返される前
SessionEnd	セッションが終了するとき

21.1.1 フックがどのように解決されるか

イベント、matcher、ハンドラーがどのように組み合わさるかを理解するために、破壊的なシェル コマンドをブロックする次の PreToolUse フックを考えてみましょう。

macOS/Linux

matcher は Bash ツール呼び出しに絞り込み、if 条件は rm * にマッチする Bash サブコマンドにさらに絞り込むため、block-rm.sh は両方のフィルターがマッチするときのみ生成されます。

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "if": "Bash(rm *)",
            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-rm.sh",
            "args": []
          }
        ]
      }
    ]
  }
}
```

スクリプトは stdin から JSON 入力を読み取り、コマンドを抽出し、`rm -rf` が含まれている場合は `permissionDecision` として `"deny"` を返します。Claude Code が実行できるように、プロジェクト内の `.claude/hooks/block-rm.sh` に保存し、`chmod +x .claude/hooks/block-rm.sh` で実行可能にしてください。

```
#!/bin/bash
# .claude/hooks/block-rm.sh
COMMAND=$(jq -r '.tool_input.command')

if echo "$COMMAND" | grep -q 'rm -rf'; then
  jq -n '{
    hookSpecificOutput: {
      hookEventName: "PreToolUse",
      permissionDecision: "deny",
      permissionDecisionReason: "Destructive command blocked by hook"
    }
  }'
else
  exit 0  # no decision; normal permission flow applies
fi
```

このスクリプトは、JSON 入力を解析するこのページの他の Bash の例と同様に `jq` を使用します。試す前に `jq` をインストールし、`PATH` 上にあることを確認してください。

Windows (PowerShell)

matcher `Bash|PowerShell` は、Bash に加えて *PowerShell* **ツール** (ページ 497) も対象にします。1 つの `if` ルールは 1 つのツールの呼び出しにしかマッチしないため、ツールごとに個別のハンドラーを用意します。1 つ目は `rm *` にマッチする Bash サブコマンドに、2 つ目は `Remove-Item *` にマッチする PowerShell コマンドに絞り込みます。どちらも `powershell.exe` を通じて同じスクリプトを実行します。

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash|PowerShell",
        "hooks": [
          {
            "type": "command",
            "if": "Bash(rm *)",
            "command": "powershell.exe",
            "args": [
              "-NoProfile",
              "-ExecutionPolicy",
              "Bypass",
              "-File",
              "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-rm.ps1"
```

(次のページに続く)

(前のページからの続き)

```
            ]
          },
          {
            "type": "command",
            "if": "PowerShell(Remove-Item *)",
            "command": "powershell.exe",
            "args": [
              "-NoProfile",
              "-ExecutionPolicy",
              "Bypass",
              "-File",
              "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-rm.ps1"
            ]
          }
        ]
      }
    ]
  }
}
```

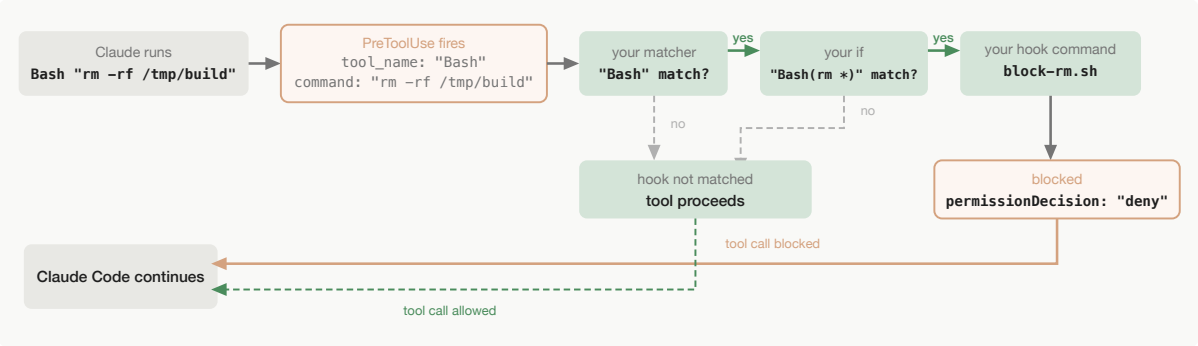
-NoProfile フラグは PowerShell プロファイルの読み込みをスキップしてフックをすばやく起動させ、-ExecutionPolicy Bypass は PowerShell がローカルのスクリプト ファイルを実行できるようにします。

スクリプトは stdin から JSON 入力を読み取り、コマンドを抽出し、rm -rf または -Recurse が後に続く Remove-Item が含まれている場合は permissionDecision として "deny" を返します。プロジェクト内の .claude/hooks/block-rm.ps1 に保存してください。

```
# .claude/hooks/block-rm.ps1
$callInput = [Console]::In.ReadToEnd() | ConvertFrom-Json
$command = $callInput.tool_input.command

if ($command -match 'rm -rf|Remove-Item.*-Recurse') {
  @{
    hookSpecificOutput = @{
      hookEventName = "PreToolUse"
      permissionDecision = "deny"
      permissionDecisionReason = "Destructive command blocked by hook"
    }
  } | ConvertTo-Json
} else {
  exit 0  # no decision; normal permission flow applies
}
```

ここで、macOS/Linux の設定に対して Claude Code が Bash "rm -rf /tmp/build" を実行することにしたとします。以下が起こります。



1. **イベントが発火**

   PreToolUse イベントが発火します。Claude Code はツール入力を JSON として stdin のフックに送信します。

   ```
   { "tool_name": "Bash", "tool_input": { "command": "rm -rf /tmp/build" }, ... }
   ```

2. **matcher がチェック**

   matcher "Bash" がツール名にマッチするため、このフック グループがアクティブになります。matcher を省略するか "*" を使用すると、グループはイベントのすべての出現でアクティブになります。

3. **If 条件がチェック**

   if 条件 "Bash(rm *)" は rm -rf /tmp/build が rm * にマッチするサブコマンドであるためマッチし、このハンドラーが生成されます。コマンドが npm test だった場合、if チェックは失敗し、block-rm.sh は実行されず、プロセス生成のオーバーヘッドを回避します。if フィールドはオプションです。なければ、マッチしたグループ内のすべてのハンドラーが実行されます。

4. **フック ハンドラーが実行**

   スクリプトは完全なコマンドを検査し、rm -rf を見つけるため、stdout に決定を出力します。

   ```
   {
     "hookSpecificOutput": {
       "hookEventName": "PreToolUse",
       "permissionDecision": "deny",
       "permissionDecisionReason": "Destructive command blocked by hook"
     }
   }
   ```

   コマンドが安全な rm バリアント（rm file.txt など）だった場合、スクリプトは代わりに exit 0 に到達します。出力なしの終了コード 0 は、フックが報告する決定がないことを意味するため、ツール呼び出しは通常の**権限フロー** (ページ 1039) を通じて続行されます。フックは呼び出しを拒否できますが、沈黙を保つことは承認を意味しません。

5. **Claude Code が結果に基づいて行動**

   Claude Code は JSON 決定を読み取り、ツール呼び出しをブロックし、Claude に理由を表示します。

以下の**設定** (ページ 450) セクションでは完全なスキーマについて説明し、各**フック イベント** (ページ 480) セクションでは、コマンドが受け取る入力と返すことができる出力について説明しています。

21.2 設定

フックは JSON 設定ファイルで定義されます。設定には 3 つのネストレベルがあります。

1. 応答する**フック イベント** (ページ 480) を選択します（`PreToolUse` や `Stop` など）
2. 発火するタイミングをフィルタリングする *matcher* **グループ** (ページ 451) を追加します（「Bash ツールのみ」など）
3. マッチしたときに実行する 1 つ以上の**フック ハンドラー** (ページ 455) を定義します

完全なウォークスルーと注釈付きの例については、上記の**フックがどのように解決されるか** (ページ 446) を参照してください。

注釈:

このページでは各レベルに特定の用語を使用しています。**フック イベント**はライフサイクル ポイント、**matcher グループ**はフィルター、**フック ハンドラー**はシェル コマンド、HTTP エンドポイント、MCP ツール、プロンプト、または実行されるエージェントです。「フック」単独は一般的な機能を指します。

21.2.1 フック位置

フックを定義する場所によって、そのスコープが決まります。

位置	スコープ	共有可能
`~/.claude/settings.json`	すべてのプロジェクト	いいえ、マシンにローカル
`.claude/settings.json`	単一プロジェクト	はい、リポジトリにコミット可能
`.claude/settings.local.json`	単一プロジェクト	いいえ、Claude Code が設定を保存する際に gitignore 対象になります
管理ポリシー設定	組織全体	はい、管理者が制御
プラグイン `hooks/hooks.json`	プラグインが有効な場合	はい、プラグインにバンドル
スキル (ページ 275) のフロントマター	スキルが呼び出された後のセッションの残りの期間。**スキルとエージェントのフック** (ページ 465) を参照	はい、スキルファイルで定義
サブエージェント (ページ 317) のフロントマター	そのサブエージェントの実行中	はい、サブエージェントファイルで定義

クラウドセッション (ページ 223) は、ローカルの `~/.claude/settings.json` を読み取りません。セルフホスト環境では、Claude Code はオペレーターがランナーホストの `~/.claude/` に用意したフックも実行します。また、ランナーイメージの管理設定ファイルが Claude Code が適用する管理ソースに含まれる場合は、そのファイル内のフックも実行します。デフォルトでは、これはサーバー管理設定も MDM で配布された Claude Code ポリシーも管理層を提供していない場合に限られます。どの設定ファイルとプラグイン、つまりどのフックがクラウドセッションに届くかについては、セットアップから引き継がれるものを参照してください。

設定ファイル解決の詳細については、**設定** (ページ 711) を参照してください。

設定ファイル、管理ポリシー設定、プラグインからのフックは、**サブエージェント** (ページ 317) 内でも実行されます。サブエージェントがツールを呼び出すと、`PreToolUse` や `PostToolUse` などのツールイベントはメインの会話と同じ設定済みフックを発火させ、入力にはサブエージェントを識別する `agent_id` と `agent_type` の**共通入力フィールド** (ページ 466) が含まれます。

管理者は管理設定で `allowManagedHooksOnly` を使用して、実行されるフックを制限できます。

- ユーザー、プロジェクト、ローカル、プラグインのフックはブロックされます。管理設定の `enabledPlugins` で強制的に有効化されたプラグインのフックは除外されます
- Claude Code は、*`statusLine`* (ページ 785)、`fileSuggestion`、*`subagentStatusLine`* (ページ 815) の設定も管理設定のものに限定します
- Claude Code は、command ソースを持つプラグインも無効化します。これには管理設定の `enabledPlugins` で強制的に有効化されたプラグインも含まれます。ただし、`disableCommandPluginSources` が明示的に `false` に設定されている場合は除きます。command ソースには Claude Code v2.1.229 以降が必要です
- Claude Code は、マーケットプレイスの headersHelper コマンドもブロックします。ただし、`disableCommandPluginSources` が明示的に `false` に設定されている場合は除きます。また、管理設定自体が宣言しているマーケットプレイスは対象外です

allowManagedHooksOnly の下で実行されるものを参照してください。

フックエントリは、設定レベル間で互いに置き換えられるのではなくマージされます。ユーザー、プロジェクト、ローカルの設定は管理フックを削除せずに独自のフックを追加します。また、*`disableAllHooks`* (ページ 466) 設定は、管理設定の外からは管理フックを無効化できません。

HTTP フックの許可リストは、管理ポリシー設定を含むすべてのソースからのフックに適用されます。

- `allowedHttpHookUrls`: いずれかの設定レベルで定義されている場合、Claude Code は URL がマージされた許可リストに一致する HTTP フックハンドラーのみを実行します
- `httpHookAllowedEnvVars`: 定義されている場合、Claude Code はそのリストにある環境変数のみをフックヘッダーに補間します

21.2.2 Matcher パターン

`matcher` フィールドは、フックが発火するタイミングをフィルタリングします。matcher の評価方法は、含まれている文字に依存します。

matcher 値	評価方法	例
`"*"`、`""`、または省略	すべてにマッチ	イベントのすべての出現で発火
文字、数字、_、-、スペース、`,`、`|` のみ	完全一致、または `|` または `,` で区切られた完全一致のリスト (オプションで周囲の空白を含む)	`Bash` は Bash ツールのみにマッチ。`Edit|Write` と `Edit, Write` はいずれかのツールに完全にマッチ。`code-reviewer` はそのエージェント タイプのみにマッチ

次のページに続く

表 3 – 前のページからの続き

matcher 値	評価方法	例
その他の文字を含む	JavaScript 正規表現、アンカーなし	`^Notebook` は名前が `Notebook` で始まるツールにマッチ。`mcp__memory__.*` は `memory` サーバーのすべてのツールにマッチ

正規表現パス上の matcher は JavaScript の `RegExp.prototype.test` でテストされます。これは値内のどこかでマッチすると成功します。`Edit.*` は `Edit` と `NotebookEdit` の両方にマッチします。完全文字列マッチが必要な場合は、`^Edit$` のようにパターンを `^` と `$` でラップしてください。

`FileChanged` と `StopFailure` は、文字、数字、`_`、`|` のみの狭い完全一致セットを使用します。これら 2 つのイベントの matcher にハイフン、スペース、またはカンマがあると、正規表現パスに留まり、`|` のみが代替を区切ります。後続の表で matcher サポートを持つ他のすべてのイベントは `|` または `,` を受け入れます。

`FileChanged` イベントは監視リストを構築するときにこれらのルールに従いません。*FileChanged* (ページ 535) を参照してください。

各イベント タイプは異なるフィールドでマッチします。

イベント	matcher がフィルタリングするもの	matcher 値の例
`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`、`PermissionDenied`	ツール名	`Bash`、`Edit|Write`、`mcp__.*`
`SessionStart`	セッションの開始方法	`startup`、`resume`、`clear`、`compact`、`fork`
`Setup`	セットアップをトリガーした CLI フラグ	`init`、`maintenance`
`SessionEnd`	セッションが終了した理由	`clear`、`resume`、`logout`、`prompt_input_exit`、`other`
`Notification`	通知タイプ	`permission_prompt`、`idle_prompt`、`auth_success`、`elicitation_dialog`、`elicitation_url_dialog`、`elicitation_complete`、`elicitation_response`、`agent_needs_input`、`agent_completed`、`quota_auto_resume_fired`、`quota_auto_resume_stale`、`quota_auto_resume_disabled`

次のページに続く

表 4 – 前のページからの続き

イベント	matcher がフィルタリングするもの	matcher 値の例
SubagentStart	エージェント タイプ	general-purpose、Explore、Plan、カスタム エージェント名、またはプラグイン スコープ付き名 前（^my-plugin:reviewer$ など）
PreCompact、PostCompact	コンテキスト圧縮をトリガーしたもの	manual、auto
PreModelSwitch、PostModel Switch	セッションの切り替え先モデルの正規名（*PreModelSwitch* (ページ 542) で説明）	claude-opus-5、claude-opus-4-6|claude-opus-5、.*opus.*
SubagentStop	エージェント タイプ	SubagentStart と同じ値
ConfigChange	設定ソース	user_settings、project_settings、local_settings、policy_settings、skills
CwdChanged	matcher サポートなし	すべての出現で常に発火
DirectoryAdded	ディレクトリが追加された方法	slash_command、register_repo_root
FileChanged	監視するリテラル ファイル名（*FileChanged* (ページ 535) を参照）	.envrc|.env
StopFailure	エラー タイプ	rate_limit、overloaded、authentication_failed、oauth_org_not_allowed、account_on_hold、billing_error、invalid_request、model_not_found、server_error、max_output_tokens、cloud_credential_error、unknown
InstructionsLoaded	ロード理由	session_start、nested_traversal、path_glob_match、include、compact
UserPromptExpansion	コマンド名	スキルまたはコマンド名
Elicitation	MCP サーバー名	設定された MCP サーバー名
ElicitationResult	MCP サーバー名	Elicitation と同じ値
UserPromptSubmit、PostToolBatch、Stop、TeammateIdle、TaskCreated、TaskCompleted、WorktreeCreate、WorktreeRemove、MessageDisplay	matcher サポートなし	すべての出現で常に発火

StopFailure を cloud_credential_error でマッチさせるには Claude Code v2.1.267 以降が必要です。これは、認証情報の読み込み失敗を server_error や unknown ではなくこの値で報告する最初のバージョンです。

ほとんどのイベントでは、Claude Code は、フックに stdin で送信する *JSON* **入力** (ページ 466) のフィールドに対して matcher を評価します。ツール イベントの場合、そのフィールドは `tool_name` です。`PreModelSwitch` と `PostModelSwitch` では、*PreModelSwitch* (ページ 542) で説明しているとおり、Claude Code は `to_model` から導出した正規名に対して matcher を評価します。各**フック イベント** (ページ 480) セクションでは、matcher 値の完全なセットとそのイベントの入力スキーマをリストしています。

この例は、Claude がファイルを書き込むまたは編集するときにのみ linting スクリプトを実行します。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Edit|Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/lint-check.sh"
          }
        ]
      }
    ]
  }
}
```

matcher をサポートしないイベントに `matcher` フィールドを追加すると、サイレントに無視されます。

ツール イベントの場合、個別のフック ハンドラーで *if* **フィールド** (ページ 456) を設定することで、より狭くフィルタリングできます。`if` は**権限ルール構文** (ページ 1039) を使用してツール名と引数を一緒にマッチするため、`"Bash(git *)"` は `git *` に一致する Bash 入力のサブコマンドのいずれかに対して実行され、`"Edit(*.ts)"` は TypeScript ファイルのみに対して実行されます。

MCP ツールをマッチ

MCP (ページ 363) サーバー ツールはツール イベント（`PreToolUse`、`PostToolUse`、`PostToolUseFailure`、`PermissionRequest`、`PermissionDenied`）で通常のツールとして表示されるため、他のツール名と同じ方法でマッチできます。

MCP ツールは `mcp__<server>__<tool>` という命名パターンに従います。例えば、

- `mcp__memory__create_entities`: Memory サーバーの create entities ツール
- `mcp__filesystem__read_file`: Filesystem サーバーの read file ツール
- `mcp__github__search_repositories`: GitHub サーバーの search ツール

サーバーのすべてのツールをマッチするには、サーバー プレフィックスに `.*` を追加します。`.*` は必須です。`mcp__memory` や `mcp__brave-search` のような matcher は完全一致文字のみを含むため、完全一致として比較され、どのツールにもマッチしません。

- `mcp__memory__.*` は `memory` サーバーのすべてのツールにマッチ

- `mcp__brave-search__.*` は名前にハイフンを含むサーバーのすべてのツールにマッチ
- `mcp__.*__write.*` は任意のサーバーの、名前が `write` で始まるツールにマッチ

プラグイン バンドル *MCP* サーバー (ページ 378) からのツールは、プラグイン名を含むスコープ付きサーバー セグメントを使用します。`mcp__plugin_<plugin-name>_<server-name>__<tool>`。ベア サーバー キーに対して記述された matcher は、これらのツールに対して発火しません。`db` キーの下でサーバーをバンドルする `my-plugin` という名前のプラグインの場合、`query` ツールは `mcp__plugin_my-plugin_db__query` として表示されるため、そのサーバーのすべてのツールの matcher は `mcp__plugin_my-plugin_db__.*` です。ハンドラーの ***if* フィールド** (ページ 456) で同じスコープ付きツール名を使用します。スコープ付き名がどのように構築されるかについては、**プラグイン提供 *MCP* サーバー** (ページ 378) を参照してください。

この例は、すべてのメモリ サーバー操作をログに記録し、任意の MCP サーバーからの書き込み操作を検証します。

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "mcp__memory__.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'Memory operation initiated' >> ~/mcp-operations.log"
          }
        ]
      },
      {
        "matcher": "mcp__.*__write.*",
        "hooks": [
          {
            "type": "command",
            "command": "/home/user/scripts/validate-mcp-write.py"
          }
        ]
      }
    ]
  }
}
```

21.2.3 フック ハンドラー フィールド

内側の `hooks` 配列の各オブジェクトはフック ハンドラーです。matcher がマッチしたときに実行されるシェル コマンド、HTTP エンドポイント、MCP ツール、LLM プロンプト、またはエージェントです。5 つのタイプがあります。

- **コマンド フック** (ページ 458)（`type: "command"`）：シェル コマンドを実行します。スクリプトは

イベントの *JSON* **入力** (ページ 466) を stdin で受け取り、終了コードと stdout を通じて結果を通信します。

- *HTTP* **フック** (ページ 460) (`type: "http"`): イベントの JSON 入力を HTTP POST リクエストとして URL に送信します。エンドポイントは、コマンド フックと同じ *JSON* **出力形式** (ページ 474) を使用して、レスポンス本体を通じて結果を通信します。
- *MCP* **ツール フック** (ページ 461) (`type: "mcp_tool"`): 設定済みの *MCP* **サーバー** (ページ 363) 上のツールを呼び出します。ツールのテキスト出力はコマンド フック stdout のように扱われます。
- **プロンプト フック** (ページ 463) (`type: "prompt"`): Claude モデルにプロンプトを送信して、単一ターンの評価を行います。モデルは決定を JSON として返します。**プロンプト ベースのフック** (ページ 552) を参照してください。
- **エージェント フック** (ページ 463) (`type: "agent"`): Read、Grep、Glob などのツールを使用して条件を検証してから決定を返すことができるサブエージェントを生成します。エージェント フックは実験的であり、変更される可能性があります。**エージェント ベースのフック** (ページ 556) を参照してください。

マッチしたすべてのフックは並列で実行されます。同じハンドラーを複数の設定ファイルで定義した場合、実行は 1 回だけです。プラグインまたはスキルが持つ同じハンドラーのコピーは別個のものとして扱われます。

ハンドラーは Claude Code の環境を持つ現在のディレクトリで実行されます。現在のディレクトリが存在しなくなった場合 (たとえば、別のシェルがセッション中に削除した worktree や一時ディレクトリなど)、Claude Code は次のうち最初に存在するディレクトリからコマンドフックを実行します。セッションを開始したディレクトリ、プロジェクトルート、ホームディレクトリ、またはシステムの一時ディレクトリです。Claude Code は、フォールバック先のディレクトリ名を含む警告を**デバッグログ** (ページ 562) に記録します。

`$CLAUDE_CODE_REMOTE` 環境変数はリモート Web 環境で `"true"` に設定され、ローカル CLI では設定されません。Claude Code v2.1.199 以降では、ローカルセッションにアクティブな Remote Control 接続がある間、`$CLAUDE_CODE_BRIDGE_SESSION_ID` が Remote Control のセッション ID に設定されます。

共通フィールド

これらのフィールドはすべてのフック タイプに適用されます。

フィールド	必須	説明
`type`	はい	`"command"`、`"http"`、`"mcp_tool"`、`"prompt"`、または `"agent"`

次のページに続く

表 5 – 前のページからの続き

フィールド	必須	説明
if	いいえ	"Bash(git *)" または "Edit(*.ts)" などの権限ルール構文を使用してこのフックが実行されるタイミングをフィルタリングします。ツール呼び出しがパターンにマッチする場合のみ、フック コマンドが実行されます。Bash パターンがサブコマンド、$()、バッククォートに対してどのように評価されるかについては、後述の *Bash* **マッチング テーブル** (ページ 457) を参照してください。ツール イベントでのみ評価されます。PreToolUse、PostToolUse、PostToolUseFailure、PermissionRequest、PermissionDenied。他のイベントでは、if が設定されたフックは実行されません。**権限ルール** (ページ 1039) と同じ構文を使用します
timeout	いいえ	キャンセルまでの秒数。*async: true* (ページ 558) で実行するコマンドフックには、Claude Code はこれを適用しません。デフォルト: command、http、mcp_tool は 600、prompt は 30、agent は 60。Claude Code は、*UserPromptSubmit* (ページ 487)、*PreModelSwitch* (ページ 542)、*PostModelSwitch* (ページ 547) では command、http、mcp_tool のデフォルトを 30 に、*MessageDisplay* (ページ 491) では 10 に下げます。*SessionEnd* (ページ 548) フックは 1.5 秒の予算を共有します。設定でフックごとにより長い timeout を指定している場合、Claude Code は最大 60 秒までそれに合わせて予算を引き上げます
statusMessage	いいえ	フックの実行中に表示されるカスタム スピナー メッセージ
once	いいえ	true の場合、Claude Code は最初の実行が成功した後にフックを削除します。失敗した実行、終了コード 2 でブロックした実行、またはタイムアウトした実行ではフックがそのまま残るため、次にマッチするイベントで再び実行されます。**スキルのフロントマター** (ページ 465) で宣言されたフックでのみ有効です。設定ファイルとエージェントのフロントマターでは無視されます

if フィールドは正確に 1 つの権限ルールを保持します。ルールを組み合わせるための &&、||、またはリスト構文はありません。複数の条件を適用するには、各条件に対して個別のフック ハンドラーを定義します。

ファイルツールの if 条件では、"Edit(src/**)" のような単一セグメントのディレクトリパターンは、作業ディレクトリ内の src ディレクトリとその配下のファイルにのみマッチします。任意の深さにある src という名前のディレクトリにマッチさせるには、"Edit(**/src/**)" と記述します。v2.1.214 より前は、"Edit(src/**)" は作業ディレクトリ配下の任意の深さにある src という名前のディレクトリにマッチしていました。

Bash パターンの場合、フック コマンドが実行されるかどうかは、パターンの形状と Claude が呼び出している Bash コマンドに依存します。先頭の VAR=value 割り当ては、マッチング前に削除されます。

if パターン	Bash コマンド	フックが実行されるか	理由
Bash(git *)	FOO=bar git push	はい	先頭の割り当ては削除されます。git push がマッチします
Bash(git *)	npm test && git push	はい	各サブコマンドがチェックされます。git push がマッチします
Bash(rm *)	echo $(rm -rf /)	はい	$() とバッククォート内のコマンドがチェックされます。rm -rf / がマッチします
Bash(rm *)	echo $(date)	いいえ	サブコマンドが rm * にマッチしません
Bash(git push *)	echo $(date)	はい	コマンド名以上を指定するパターンは、$()、バッククォート、または $VAR でとにかくフックを実行します

Bash 入力がどのコマンドを実行するかを Claude Code が判断できない場合、パターンに関係なくフックを実行します。if フィルターはベストエフォートであるため、ハードな許可または拒否を強制するには、フックではなく**権限システム** (ページ 1039) を使用してください。

コマンド フック フィールド

共通フィールド (ページ 456) に加えて、コマンド フックはこれらのフィールドを受け入れます。

フィールド	必須	説明
command	はい	実行するシェル コマンド。args を使用する場合、直接生成する実行可能ファイル。*Exec* **フォームとシェル フォーム** (ページ 459) を参照してください
args	いいえ	引数リスト。存在する場合、command は実行可能ファイルとして解決され、args を引数ベクトルとして直接生成されます。シェルは関与しません。*Exec* **フォームとシェル フォーム** (ページ 459) を参照してください
async	いいえ	true の場合、ブロックせずにバックグラウンドで実行されます。**バックグラウンドでフックを実行** (ページ 558) を参照してください
asyncRewake	いいえ	true の場合、バックグラウンドで実行され、終了コード 2 で Claude を起動します。フックの stderr、または stderr が空の場合は stdout が **システムリマインダー**として Claude に表示されるため、Claude は長時間実行されるバックグラウンドの失敗に対応できます

次のページに続く

表 7 – 前のページからの続き

フィールド	必須	説明
shell	いいえ	このフックに使用するシェル。"bash" または "powershell" を受け入れます。デフォルトは "bash"、または Git Bash がインストールされていない場合は Windows で "powershell"。"powershell" を設定すると、Windows 上で PowerShell 経由でコマンドが実行されます。フックは PowerShell を直接生成するため、CLAUDE_CODE_USE_POWERSHELL_TOOL は不要です。args が設定されている場合は無視されます

Exec フォームとシェル フォーム

コマンド フックは args が設定されている場合は exec フォームで実行され、args が省略されている場合はシェル フォームで実行されます。フックが**パス プレースホルダー** (ページ 463) を参照する場合は常に args を設定してください。各要素は引用符なしで 1 つの引数として渡されるためです。パイプや && などのシェル機能が必要な場合、または両方の懸念が適用されない場合は args を省略してください。

Exec フォームは args が存在する場合に実行されます。Claude Code は command を PATH 上の実行可能ファイルとして解決し、args を引数ベクトルとして直接生成します。シェルがないため、各 args 要素は記述されたとおりに正確に 1 つの引数であり、${CLAUDE_PLUGIN_ROOT} などのパス プレースホルダーは command と各 args 要素にプレーン文字列として置換されます。アポストロフィ、$、バッククォートなどの特殊文字は、シェルが解釈しないため、そのまま渡されます。どのプラットフォームでもシェル トークン化は発生しません。

シェル フォームは args が存在しない場合に実行されます。command 文字列はシェルに渡されます。macOS と Linux では sh -c、Windows では Git Bash、または Git Bash がインストールされていない場合は PowerShell。shell フィールドを設定して明示的に選択します。シェルは文字列をトークン化し、変数を展開し、パイプ、&&、リダイレクト、グロブを解釈します。

注釈:

Windows では、exec フォームは command が .exe などの実際の実行可能ファイルに解決されることが必要です。npm、npx、eslint、およびその他のツールが node_modules/.bin にインストールする .cmd と .bat シムは実行可能ファイルではなく、シェルなしで生成することはできません。exec フォームでそれらを実行するには、基になるスクリプトを node で直接呼び出します。例えば "command": "node", "args": ["${CLAUDE_PLUGIN_ROOT}/node_modules/eslint/bin/eslint.js"]。node プラス スクリプト パス パターンは、node.exe が実際のバイナリであるため、すべてのプラットフォームで機能します。.cmd または .bat シムを名前で実行するには、シェル フォームを使用します。

この例は、プラグインにバンドルされた Node スクリプトを実行します。Exec フォームは解決されたスクリプト パスを引用符なしで 1 つの引数として渡します。

```
{
  "type": "command",
  "command": "node",
```

(次のページに続く)

(前のページからの続き)

```
  "args": ["${CLAUDE_PLUGIN_ROOT}/scripts/format.js", "--fix"]
}
```

同等のシェル フォームは、スペースまたは特殊文字を含むパスを処理するために引用符が必要です。

```
{
  "type": "command",
  "command": "node \"${CLAUDE_PLUGIN_ROOT}\"/scripts/format.js --fix"
}
```

両方のフォームは同じ**パス プレースホルダー** (ページ 463) をサポートし、両方とも生成されたプロセスで環境変数 CLAUDE_PROJECT_DIR、CLAUDE_PLUGIN_ROOT、CLAUDE_PLUGIN_DATA としてエクスポートするため、スクリプトは起動方法に関係なく process.env.CLAUDE_PLUGIN_ROOT を読み取ることができます。

プラグイン フックは追加で ${user_config.*} 値を置換します。exec フォームのみ: 値は command と各 args 要素にプレーン文字列として置換されるため、シェルは再解析しません。

command が ${user_config.*} を参照するシェル フォームのプラグイン フックは、実行される代わりにエラーで失敗します。シェル フォーム フックからオプション値を使用するには、$CLAUDE_PLUGIN_OPTION_<KEY> 環境変数（webhook_url オプションの場合は $CLAUDE_PLUGIN_OPTION_WEBHOOK_URL など）を読み取るか、args を設定してフックを exec フォームに切り替えます。v2.1.207 より前では、シェル フォーム プラグイン フック コマンドも ${user_config.*} を置換していました。

注釈:

Exec フォームでは、command は実行可能ファイル名またはパスのみです。args とともに使用される command がパス区切り文字を含まないベア名で、かつ空白を含む場合、生成が失敗するため Claude Code は警告をログに記録します。node script.js という名前の実行可能ファイルは存在しないためです。余分なトークンを args に移動してください。C:\Program Files\nodejs\node.exe などのスペースを含む絶対パスは、単一の有効な実行可能ファイルであり、警告をトリガーしません。

HTTP フック フィールド

共通フィールド (ページ 456) に加えて、HTTP フックはこれらのフィールドを受け入れます。

フィールド	必須	説明
url	はい	POST リクエストを送信する URL
headers	いいえ	キー値ペアとしての追加 HTTP ヘッダー。値は $VAR_NAME または ${VAR_NAME} 構文を使用した環境変数補間をサポートします。allowedEnvVars にリストされている変数のみが解決されます
allowedEnvVars	いいえ	ヘッダー値に補間される可能性のある環境変数名のリスト。リストされていない変数への参照は空の文字列に置き換えられます。環境変数補間が機能するために必須

Claude Code はフックの *JSON* **入力** (ページ 466) を `Content-Type: application/json` の POST リクエスト本体として送信します。レスポンス本体はコマンド フックと同じ *JSON* **出力形式** (ページ 474) を使用します。

エラー処理はコマンド フックと異なります。*HTTP* **レスポンスの処理** (ページ 473) を参照してください。

この例は `PreToolUse` イベントをローカル検証サービスに送信し、`MY_TOKEN` 環境変数からのトークンで認証します。

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "http",
            "url": "http://localhost:8080/hooks/pre-tool-use",
            "timeout": 30,
            "headers": {
              "Authorization": "Bearer $MY_TOKEN"
            },
            "allowedEnvVars": ["MY_TOKEN"]
          }
        ]
      }
    ]
  }
}
```

MCP ツール フック フィールド

共通フィールド (ページ 456) に加えて、MCP ツール フックはこれらのフィールドを受け入れます。

フィールド	必須	説明
`server`	はい	設定された MCP サーバーの名前。**プラグイン バンドル サーバー** (ページ 378) の場合、これはスコープ付き名前 `plugin:<plugin-name>:<server-name>`（例：`plugin:my-plugin:db`）であり、ベア サーバー キーではありません
`tool`	はい	そのサーバー上で呼び出すツールの名前
`input`	いいえ	ツールに渡される引数。文字列値は、フックの *JSON* **入力** (ページ 466) から `${path}` 置換をサポートします（例：`"${tool_input.file_path}"`）

この例は、各 `Write` または `Edit` の後、`my_server` MCP サーバー上の `security_scan` ツールを呼び出し、編集されたファイルのパスを渡します。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "mcp_tool",
            "server": "my_server",
            "tool": "security_scan",
            "input": { "file_path": "${tool_input.file_path}" }
          }
        ]
      }
    ]
  }
}
```

ツールの結果の読み取り方

Claude Code は、**終了コード *0* の解析ルール** (ページ 469) に従い、コマンドフックの stdout と同じ方法でツールのテキストコンテンツを読み取ります。ツールが `isError: true` を返した場合、フックは非ブロッキングエラーを生成し、実行は続行されます。

サーバーがまだ接続中の場合

`PreToolUse` や `Stop` など、フックが結果をブロックまたは変更できるイベントでは、Claude Code は接続中のサーバーを待ってからツールを呼び出します。待機時間は最大で `MCP_TIMEOUT` までで、フック自体の *`timeout`* (ページ 456) の範囲内です。`Notification` や `SessionEnd` などの観察用イベントでは待機しません。

cached **ステータス** (ページ 370) を表示しているサーバーは、フックがそのツールを呼び出したときに接続します。その時点でサーバーが接続されていない場合、フックは非ブロッキングエラーを生成し、実行は続行されます。フックが OAuth フローを開始することはないため、先に */mcp* **からサーバーを認証** (ページ 386) してください。

MCP サーバーが利用可能になる前に発火するイベント

起動時の `SessionStart`（`--continue` や `--resume` の場合を含む）とすべての `Setup` イベントは、セッションの MCP サーバーがフックから利用可能になる前に発火します。Claude Code はツールを呼び出さずにこれらの `mcp_tool` フックをスキップし、**デバッグログ** (ページ 562) には `mcp_tool hooks are not available for the 'SessionStart' hook event (no MCP client context)`、または `Setup` を示す同じメッセージが記録されます。`/clear` やコンテキスト圧縮の後など、セッションの後半で `SessionStart` が再び発火した場合は、その `mcp_tool` フックが実行されます。セッションが起動時に必要とするものについては、代わりに `SessionStart` で `type: "command"` フックを使用してください。

プロンプト フックとエージェント フック フィールド

共通フィールド (ページ 456) に加えて、プロンプト フックとエージェント フックはこれらのフィールドを受け入れます。

フィールド	必須	説明
`prompt`	はい	モデルに送信するプロンプト テキスト。フック入力 JSON のプレースホルダーとして `$ARGUMENTS` を使用します。バックスラッシュでエスケープしてリテラル テキストを含めます。`\$1.00` は `$1.00` としてレンダリングされます
`model`	いいえ	評価に使用するモデル。デフォルトは、Claude Code が**バックグラウンド機能** (ページ 923) に使用するモデル

21.2.4 パスでフック スクリプトを参照

フックが実行されるときの作業ディレクトリに関係なく、プロジェクトまたはプラグイン ルートを基準にしてフック スクリプトを参照するには、これらのプレースホルダーを使用します。

- `${CLAUDE_PROJECT_DIR}`: セッションを開始したプロジェクトルート。Claude Code はこの変数を *stdio MCP* **サーバー** (ページ 366) とプラグイン LSP サーバーの環境にも設定します。
- `${CLAUDE_PLUGIN_ROOT}`: プラグインのインストールディレクトリ。プラグインにバンドルされたスクリプト用です。更新時にこのパスがどう扱われるかについては、プラグインの環境変数を参照してください。
- `${CLAUDE_PLUGIN_DATA}`: プラグインの永続データディレクトリ。プラグインの更新後も保持すべき依存関係と状態用です。

注釈:

worktree の場合は異なります。 セッション中に Claude が worktree に入った場合、Claude Code は `${CLAUDE_PROJECT_DIR}` を元の場所のまま維持し、worktree のパスは別の方法でフックに渡します。

- `${CLAUDE_PROJECT_DIR}` **は変わらない**: セッションを開始したプロジェクトルートを引き続き指すため、`${CLAUDE_PROJECT_DIR}/.claude/hooks/check-style.sh` のようなコマンドは、引き続きメインのチェックアウト内のスクリプトを実行します。
- `cwd` **は Claude に追従する**: フックの**入力** *JSON* (ページ 466) の `cwd` フィールドは、Claude が worktree に入った後はその worktree のルートになり、Claude が `cd` を実行した後は新しいディレクトリになります。Claude がどのディレクトリで作業しているかをフックが知る必要がある場合は、このフィールドを読み取ってください。

パス プレースホルダーを参照するフックには *exec* **フォーム** (ページ 459) を優先してください。シェル フォームでは、各プレースホルダーをダブル クォートで囲みます。

プロジェクト スクリプト

この例は `${CLAUDE_PROJECT_DIR}` を使用して、`Write` または `Edit` ツール呼び出しの後、プロジェクトの `.claude/hooks/` ディレクトリからスタイル チェッカーを実行します。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/check-style.sh",
            "args": []
          }
        ]
      }
    ]
  }
}
```

プラグイン スクリプト

hooks/hooks.json でプラグイン フックを定義し、オプションのトップレベル description フィールドを使用します。プラグインが有効な場合、そのフックはユーザーおよびプロジェクト フックとマージされます。

この例は、プラグインにバンドルされたフォーマット スクリプトを実行します。

```
{
  "description": "Automatic code formatting",
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/format.sh",
            "args": [],
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

プラグイン フックの作成の詳細については、プラグイン コンポーネント リファレンスを参照してください。

21.2.5 スキルとエージェントのフック

設定ファイルとプラグインに加えて、フックは**スキル** (ページ 275) と**サブエージェント** (ページ 317) でフロントマターを使用して直接定義できます。設定形式は設定ベースのフックと同じです。Claude Code がこれらを登録しておく期間は、コンポーネントによって異なります。

- **サブエージェントのフック**: Claude Code は、そのサブエージェントの実行中にのみこれらを実行し、サブエージェントが終了すると削除します。ここでの `Stop` フックは、Claude Code によって `Subagent Stop` に変換されます。これはサブエージェントが完了したときに発火するイベントです。
- **スキルのフック**: Claude Code は、ユーザーまたは Claude がスキルを呼び出したときにこれらを登録し、スキル自体のターンの後のターンも含めて、セッションの残りの期間ずっと実行し続けます。代わりに最初の実行が成功した後で Claude Code にフックを削除させるには、そのフックに *`once: true`* (ページ 456) を設定します。

このスキルは、各 `Bash` コマンドの前にセキュリティ検証スクリプトを実行する `PreToolUse` フックを定義します。

```
---
name: secure-operations
description: Perform operations with security checks
hooks:
  PreToolUse:
    - matcher: "Bash"
      hooks:
        - type: command
          command: "./scripts/security-check.sh"
---
```

サブエージェントは YAML フロントマターで同じ形式を使用します。

プロジェクトスキルのフロントマターのフックは、**設定ファイルのフックと同じワークスペース信頼ルール** (ページ 561) に従います。Claude Code は、ユーザーまたは Claude がスキルを呼び出したときにこれらを登録します。これには、信頼していないフォルダーでの `-p` 実行も含まれます。

プロジェクトサブエージェントのフロントマターのフックは、エージェントファイルの取得元フォルダーについて**ワークスペース信頼ダイアログ** (ページ 1064) を承認した後にのみ実行されます。`-p` セッションは承認したことになりません。**フォルダーを信頼する前に実行されるもの** (ページ 1066) ではこれを設定ファイルのルールと比較しており、サブエージェントのページには**どのスコープが対象外か** (ページ 341) が記載されています。v2.1.218 より前は、これらのフックは信頼していないフォルダーからも実行される可能性がありました。

21.2.6 /hooks メニュー

Claude Code で `/hooks` と入力して、設定されたフックの読み取り専用ブラウザーを開きます。リストでは、各フックにその取得元（ユーザー設定、プロジェクト設定、ローカル設定、プラグイン、現在のセッションなど）を示すラベルが付けられます。

フックを選択すると、そのフックが実行する内容の全文と、定義されている場所（設定ファイルのパスやプラグインの名前など）が表示されます。

フックが設定されていないものも含めてすべてのフック イベントを参照するには、リストの末尾にある `All events` を選択します。

21.2.7 フックを無効化または削除

設定ファイルで定義されたフックを削除するには、そのファイルからエントリを削除します。

すべてのフックを削除せずに一時的に無効化するには、設定ファイルで `"disableAllHooks": true` を設定します。Claude Code は**設定の優先順位** (ページ 719) を適用した後に残る値を読み取るため、プロジェクトの `.claude/settings.json` にある `"disableAllHooks": false` は、ユーザー設定の `true` を上書きします。プロジェクトの設定内容にかかわらず 1 回の実行だけフックをオフにするには、`--settings '{"disableAllHooks": true}'` を渡します。これはプロジェクト設定とローカル設定よりも優先されます。個別のフックを設定に保持したまま無効化する方法はありません。

`disableAllHooks` 設定は管理設定階層を尊重します。管理者が管理ポリシー設定を通じてフックを設定している場合、ユーザー、プロジェクト、またはローカル設定で設定された `disableAllHooks` は、それらの管理フックを無効化できません。管理設定レベルで設定された `disableAllHooks` のみが管理フックを無効化できます。各レベルの影響範囲の詳細については、`disableAllHooks` を参照してください。

設定ファイルのフックへの直接編集は通常、ファイル ウォッチャーによって自動的に取得されます。

21.3 フック入出力

コマンド フックは stdin 経由で JSON データを受け取り、終了コード、stdout、stderr を通じて結果を通信します。HTTP フックは同じ JSON を POST リクエスト本体として受け取り、HTTP レスポンス本体を通じて結果を通信します。このセクションでは、すべてのイベントに共通するフィールドと動作について説明します。**フック イベント** (ページ 480) の各セクションには、その特定の入力スキーマと決定制御オプションが含まれています。

macOS と Linux では、コマンド フックは制御ターミナルのない独自のセッションで実行されます。フック プロセスと子プロセスは `/dev/tty` を開くことも、エスケープ シーケンスを Claude Code インターフェイスに直接送信することもできません。Windows には `/dev/tty` がありません。

任意のプラットフォームでユーザーにメッセージを表示するには、JSON 出力で *`systemMessage`* (ページ 474) を返します。一部のイベントはこれを破棄するか別の場所に配信します。その点は各**イベントのセクション** (ページ 480) に記載されています。デスクトップ通知をトリガーしたり、ウィンドウ タイトルを設定したり、ベルを鳴らしたりするには、代わりに *`terminalSequence`* (ページ 475) を返します。

21.3.1 共通入力フィールド

フック イベントは、各**フック イベント** (ページ 480) セクションで説明されているイベント固有のフィールドに加えて、これらのフィールドを JSON として受け取ります。コマンド フックの場合、この JSON は stdin 経由で到着します。HTTP フックの場合、POST リクエスト本体として到着します。

フィールド	説明
`session_id`	現在のセッション識別子

次のページに続く

表 11 – 前のページからの続き

フィールド	説明
prompt_id	現在処理中のユーザー プロンプトを識別する UUID。*OpenTelemetry* **イベントの** *prompt.id* **属性** (ページ 962) と一致するため、単一のプロンプトのテレメトリでフック出力を相関させることができます。最初のユーザー入力まで存在しません。Claude Code v2.1.196 以降が必要です
transcript_path	会話 JSON へのパス。トランスクリプト ファイルは非同期に書き込まれ、メモリ内の会話に遅れる可能性があるため、フックが発火するときに現在のターンの最新メッセージがまだ含まれていない可能性があります。現在のターンの最終的なアシスタント テキストが必要なフックは、トランスクリプトを読む代わりに *Stop* (ページ 526) と *SubagentStop* (ページ 522) の last_assistant_message を使用する必要があります
cwd	フックが呼び出されるときの現在の作業ディレクトリ
scratchpad_dir	セッションのスクラッチパッド ディレクトリへのパス。Claude はここに一時的な作業ファイルを保持します。セッションにスクラッチパッドがない場合、または一時ディレクトリが利用できない場合は存在しません。Claude Code v2.1.257 以降が必要です
permission_mode	現在の**権限モード** (ページ 1042): "default"、"plan"、"acceptEdits"、"auto"、"dontAsk"、または "bypassPermissions"。**Manual** というラベルが付いたモードは "default" として到着し、"manual" として到着することはないため、"default" と一致するスクリプトは引き続き機能します。すべてのイベントがこのフィールドを受け取るわけではありません。各**フック イベント** (ページ 480) セクションの JSON 例を確認してください
effort	フックの実行時に有効な *effort* **レベル** (ページ 747) を保持する level フィールドを持つオブジェクト: "low"、"medium"、"high"、"xhigh"、または "max"。アクティブなモデルがサポートしていないレベルを設定した場合、level は Claude Code が代わりに実行したレベルを報告します。そのレベルの選び方については *effort* **レベルを調整** (ページ 747) を参照してください。オブジェクトは**ステータスライン** (ページ 789) の effort フィールドと一致します。現在のモデルが effort パラメータをサポートしている場合、PreToolUse、PostToolUse、Stop、Subagent Stop などのツール使用コンテキスト内で発火するイベントに存在します。レベルは、フック コマンドと Bash ツールでも $CLAUDE_EFFORT 環境変数として利用可能です。
hook_event_name	発火したイベントの名前

--agent で実行するか、サブエージェント内で実行する場合、2 つの追加フィールドが含まれます。

フィールド	説明
agent_id	サブエージェントの一意の識別子。フックがサブエージェント呼び出し内で発火する場合にのみ存在します。これを使用して、サブエージェント フック呼び出しをメイン スレッド呼び出しから区別します。

次のページに続く

表 12 – 前のページからの続き

フィールド	説明
agent_type	エージェント名（例えば、"Explore" または "security-reviewer"）。セッションが --agent を使用するか、フックがサブエージェント内で発火する場合に存在します。サブエージェントの場合、サブエージェントのタイプがセッションの --agent 値よりも優先されます。カスタム サブエージェントとプラグイン サブエージェントが報告する値と、プラグイン スコープ名に対する matcher の記述方法については、*SubagentStart* (ページ 521) を参照してください。

SessionStart (ページ 481) フックのみが model フィールドを受け取ることができ、Claude Code が常にそれを含めるとは限りません。*PreModelSwitch* (ページ 542) と *PostModelSwitch* (ページ 547) フックは代わりに from_model と to_model を受け取るため、セッション中に変化するモデルを追跡するには PostModelSwitch フックを使用してください。

$CLAUDE_MODEL 環境変数はありません。シェルで $ANTHROPIC_MODEL を設定した場合、フックはそれを読み取ることができますが、セッション中に /model でモデルを切り替えてもその値は変わりません。

フック プロセスは親環境を継承します。ただし、Claude Code が**起動するすべてのサブプロセスから削除する** (ページ 936) OTEL_* エクスポーター変数と、CLAUDE_CODE_SUBPROCESS_ENV_SCRUB が 1 に設定されている場合に除去される変数は除きます。

例えば、Bash コマンドの PreToolUse フックは stdin で以下を受け取ります。

```
{
  "session_id": "abc123",
  "prompt_id": "550e8400-e29b-41d4-a716-446655440000",
  "transcript_path": "/home/user/.claude/projects/.../transcript.jsonl",
  "cwd": "/home/user/my-project",
  "scratchpad_dir": "/tmp/claude-1000/-home-user-my-project/abc123/scratchpad",
  "permission_mode": "default",
  "hook_event_name": "PreToolUse",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test",
    "description": "Run test suite",
    "timeout": 120000,
    "run_in_background": false
  },
  "tool_use_id": "toolu_01ABC123..."
}
```

tool_name、tool_input、tool_use_id フィールドはイベント固有です。各**フック イベント** (ページ 480) セクションでは、そのイベントの追加フィールドについて説明しています。

21.3.2 終了コード出力

フック コマンドからの終了コードは、Claude Code にアクションが進行すべきか、ブロックされるべきか、無視されるべきかを伝えます。終了コードは単独で作用するわけではありません。Claude Code は 0 だけでなくすべての終了コードで stdout から *JSON* **出力フィールド** (ページ 474) を読み取ります。標準の決定モデルを使用するイベントでは、解析されたオブジェクトがスキーマ検証に合格すると、終了コードとともに効果を持ちます。終了 2 によるブロックは、JSON で上書きできない唯一の結果です。

イベントごとの例外は 2 つの表にまとめられています。**イベントごとの終了コード** *2* **動作** (ページ 472) は各イベントで終了コードが何をするかを示し、**決定制御** (ページ 477) は各イベントがどの決定フィールドを尊重するかを示します。`systemMessage` などのユニバーサル フィールドはほとんどのイベントで機能し、*JSON* **出力** (ページ 474) の表にリストされています。

終了コード 0

終了 0 は成功を意味し、構造化制御のために JSON を出力する場合に想定される終了コードです。

ほとんどのイベントでは、Claude Code は stdout をデバッグ ログに書き込み、トランスクリプトには表示しません。例外は `UserPromptSubmit`、`UserPromptExpansion`、`SessionStart`、`PostModelSwitch` で、Claude Code はプレーン テキストの stdout を Claude が見て行動できるコンテキストとして追加します。

Claude Code が stdout を *JSON* **出力** (ページ 474) として読み取るかプレーン テキストとして読み取るかは、前後の空白を無視したうえで、その開始と終了の文字によって決まります。

- **{ で始まり } で終わる**: Claude Code は JSON として解析します。出力が 2 行以上で、各行が単独で JSON として解析でき、どの行もフィールドを設定する *JSON* **出力** (ページ 474) オブジェクトでない場合、Claude Code は出力全体をプレーン テキストとして扱います。それらの行のいずれかがフィールドを設定している場合、出力全体は解析失敗となります（後述）。
- **{ で始まるが } で終わらない**: Claude Code はプレーン テキストとして扱います。
- **その他の文字で始まる**: JSON 配列や引用符で囲まれた JSON 文字列を含め、Claude Code はプレーン テキストとして扱います。

標準の決定モデルを使用するイベントでは、終了 0 で解析されたオブジェクトがスキーマ検証に失敗した場合は非ブロッキング エラーとなります。アクションは進行し、トランスクリプトには検証メッセージとともに `<hook name> hook error` 通知が表示されます。2 以外のすべての終了コードでも同じことが起こりますが、**終了** *2* **は引き続きブロックします** (ページ 470)。

標準の決定モデルを使用するイベントでは、Claude Code が stdout を JSON として解析しようとして失敗した場合、2 以外のすべての終了コードで非ブロッキング エラーを報告します。トランスクリプトには解析メッセージとともに `<hook name> hook error` 通知が表示されます。プレーン テキストの stdout をコンテキストとして追加するイベントでは、Claude Code はそのテキストを追加しません。v2.1.248 より前は、Claude Code はその stdout をプレーン テキストとして扱っていました。

終了 0 のフックからの stderr はデバッグ ログにのみ送られ、トランスクリプトには表示されず、Claude がそれを見ることはありません。自分で読むには、**デバッグ ログ** (ページ 562) を有効にしてください。`PostToolUse` または `PostToolUseFailure` フックから Claude に警告を表示するには、代わりに終了 2 を使用してください。そうすれば、ツールがすでに実行されていても *Claude* **は** *stderr* **を確認できます** (ページ 472)。

終了コード 2

終了 2 はブロッキング エラーを意味します。**ブロック可能なイベント** (ページ 472) では、JSON を出力するかどうかにかかわらず終了 2 はブロックします。JSON の `permissionDecision` が `"allow"` であっても上書きできません。Claude Code は stdout 上の有効な *JSON* **出力** (ページ 474) を引き続き読み取ります。`Elicitation` と `ElicitationResult` では、終了 2 のフックの `hookSpecificOutput` は無視されます。

ブロッキング メッセージは、JSON がブロッキング決定を行う場合はその理由、それ以外の場合は stderr テキストです。ブロックの効果はイベントによって異なります。`PreToolUse` はツール呼び出しをブロックし、`UserPromptSubmit` はプロンプトを拒否する、などです。**イベントごとの終了コード** *2* **動作** (ページ 472) にはすべてのイベントの効果がリストされており、各イベントのセクションにはメッセージの送信先が記載されています。

JSON **出力** (ページ 474) のスキーマ検証に失敗する JSON を出力しながら終了 2 するフックは、引き続きブロックします。Claude Code は stderr をブロッキング理由として使用し、検証の失敗をデバッグ ログに記録します。v2.1.214 より前は、Claude Code はその組み合わせを非ブロッキング エラーとして扱い、アクションは進行していました。

このスクリプトは終了 2 によって `rm` コマンドをブロックし、それ以外のすべてのコマンドは通常の権限フローに任せます。

```
#!/bin/bash
# Reads JSON input from stdin, checks the command
input=$(cat)
command=$(jq -r '.tool_input.command' <<<"$input")

if [[ "$command" == rm* ]]; then
  echo "Blocked: rm commands are not allowed" >&2
  exit 2  # Blocking error: tool call is prevented
fi

exit 0  # No decision: the normal permission flow applies
```

その他の終了コード

その他の終了コードは、ほとんどのフック イベントでそれ自体ではブロックしません。何が起こるかは stdout によって異なります。

- 標準の決定モデルを使用するイベントで、解析されたオブジェクトがスキーマ検証に合格した場合、Claude Code は終了コードを無視し、JSON のみが結果を決定します。
  - イベントがサポートする各フィールド（`permissionDecision`、`additionalContext`、`updatedInput`、`systemMessage` を含む）が尊重され、フックはエラーとして報告されません。
  - **決定制御** (ページ 477) にはイベントごとの決定フィールドがリストされています。`systemMessage` などのユニバーサル フィールドは *JSON* **出力** (ページ 474) の表に従います。
- 標準の決定モデルを使用するイベントで、解析されたオブジェクトがスキーマ検証に失敗した場合、**終了** *0* **の場合** (ページ 469) と同じ非ブロッキング エラーになります。アクションは進行し、`<hook name> hook error` 通知に検証メッセージが含まれます。

- Claude Code が *JSON* **として解析しようとして** (ページ 469) 失敗した stdout の場合、標準の決定モデルを使用するイベントでは、Claude Code は終了 0 の場合と同じ非ブロッキング エラーを報告します。アクションは進行し、通知に解析メッセージが含まれます。
- Claude Code が**プレーン テキストとして扱う** (ページ 469) stdout、または空の stdout の場合、ほとんどのフック イベントで非ブロッキング エラーとなります。アクションは進行し、トランスクリプトには `<hook name> hook error` 通知と、その後に `Failed with non-blocking status code:` というプレフィックスが付いた stderr の最初の行が表示されます。完全な stderr を取得するには、**デバッグ ログ** (ページ 562) を有効にしてください。

標準の決定モデルに含まれないイベントは、**イベントごとの表** (ページ 472) の独自の行に従います。`WorktreeCreate` は JSON の内容にかかわらず 0 以外の終了で作成に失敗し、`StopFailure` のようにフック出力を完全に破棄するイベントは、すべての終了コードで JSON を無視します。ただし、`terminalSequence` のような副作用フィールドは引き続き発火します。

起動できないフックも同じ非ブロッキングの扱いになります。スクリプト パスが存在しないか実行可能でない場合、シェルは 127 などのコードで終了し、インタープリターのメッセージとともに同じ通知が表示されます（例: `Failed with non-blocking status code: /bin/sh: /path/to/hook.sh: No such file or directory`）。ほとんどのフック イベントでは、アクションは進行します。ポリシー フックを設定するときは、最初の実行時にこの通知に注意してください。`settings.json` でパスを入力ミスすると、ゲートが気付かないうちに無効になります。

警告:

ほとんどのフック イベントでは、終了コード 2 がコードのみでブロックする唯一の終了コードです。stdout に有効な JSON がない場合、1 が従来の Unix 失敗コードであっても、Claude Code は終了コード 1 を非ブロッキング エラーとして扱い、アクションを進行させます。フックがポリシーを実施することを目的としている場合は、`exit 2` を使用してください。worktree イベントは異なります。`WorktreeCreate` からの 0 以外の終了コードは worktree の作成を中止し、`WorktreeRemove` からの 0 以外の終了コードは、その後もディレクトリが存在する場合に worktree の削除を失敗させます。

タイムアウト

`async: true` (ページ 558) で実行するコマンド フックを除き、Claude Code は *`timeout`* (ページ 456) に達した `command`、`http`、または `mcp_tool` フックをキャンセルし、フックの出力を破棄します。そのため、ほとんどのイベントでは、タイムアウトしたフックは決定を下しません。

`PreModelSwitch` (ページ 542) では、タイムアウトでキャンセルされたフックはモデルの切り替えをブロックします。`PreToolUse` では、2 つのフック ファミリーで動作が異なります。

- タイムアウトした `command`、`http`、または `mcp_tool` フックはツール呼び出しをブロックしません。呼び出しは通常の**権限フロー** (ページ 1039) を通じて続行されるため、停止したフックがゲートとして機能することを当てにしないでください。
- タイムアウトを超えた Agent SDK コールバック フックは**ツール呼び出しをブロックします** (ページ 495)。

イベントごとの終了コード 2 動作

終了コード 2 は、フックが「停止、これをしないでください」と通知する方法です。効果はイベントに依存します。一部のイベントはブロック可能なアクション（まだ発生していないツール呼び出しなど）を表し、他のイベントはすでに発生したか防止できないことを表すためです。

フック イベント	ブロック可能？	終了 2 で何が起こるか
`PreToolUse`	はい	ツール呼び出しをブロック
`PermissionRequest`	いいえ	このイベントでは終了コード 2 は尊重されず、権限フローは変更されずに進行します。代わりに *decision* **オブジェクト** (ページ 507) を通じて拒否してください
`UserPromptSubmit`	はい	プロンプトをブロックし、Claude に届かないようにします。**ブロックされたプロンプトが残すもの** (ページ 490) を参照してください
`UserPromptExpansion`	はい	拡張をブロック
`Stop`	はい	Claude が停止するのを防ぎ、会話を続行
`SubagentStop`	はい	サブエージェントが停止するのを防止
`TeammateIdle`	はい	チームメイトがアイドル状態になるのを防止し、作業を続行させる
`TaskCreated`	はい	タスク作成をロールバック
`TaskCompleted`	はい	タスクが完了としてマークされるのを防止
`ConfigChange`	はい	設定変更が有効になるのをブロック（`policy_settings` を除く）
`StopFailure`	いいえ	出力と終了コードは無視（`terminalSequence` を除く）
`PostToolUse`	いいえ	Claude に stderr を表示（ツールはすでに実行）
`PostToolUseFailure`	いいえ	Claude に stderr を表示（ツールはすでに失敗）
`PostToolBatch`	はい	次のモデル呼び出しの前にエージェント型ループを停止
`PermissionDenied`	いいえ	終了コードと stderr は無視（拒否はすでに発生）。JSON `hookSpecificOutput.retry: true` を使用してモデルが再試行できることを伝える。Claude Code は**判定のない拒否** (ページ 517) では `retry: true` を無視します
`Notification`	いいえ	終了コードと stderr は無視
`SubagentStart`	いいえ	ユーザーのみに stderr を表示
`SessionStart`	いいえ	ユーザーのみに stderr を表示
`Setup`	いいえ	終了コードと stderr は無視
`SessionEnd`	いいえ	ユーザーのみに stderr を表示
`CwdChanged`	いいえ	ユーザーのみに stderr を表示
`DirectoryAdded`	いいえ	stderr はデバッグ ログに送られる（ディレクトリはすでに追加済み）
`FileChanged`	いいえ	ユーザーのみに stderr を表示
`PreCompact`	はい	コンテキスト圧縮をブロック

次のページに続く

表 13 – 前のページからの続き

フック イベント	ブロック可能？	終了 2 で何が起こるか
PostCompact	いいえ	ユーザーのみに stderr を表示
PreModelSwitch	はい	モデルの切り替えをブロックし、ユーザーに stderr を表示
PostModelSwitch	いいえ	ユーザーのみに stderr を表示（モデルはすでに切り替え済み）
Elicitation	はい	elicitation を拒否
ElicitationResult	はい	レスポンスをブロック（アクションが decline になる）
WorktreeCreate	はい	0 以外の終了コードで worktree 作成が失敗
WorktreeRemove	はい	0 以外の終了コードは、その後もディレクトリが存在する場合に worktree の削除を失敗させる。ディレクトリがどうなるかについては *WorktreeRemove* (ページ 539) を参照
InstructionsLoaded	いいえ	終了コードは無視
MessageDisplay	いいえ	元のテキストが表示される

SessionStart、SubagentStart、および PostModelSwitch の場合、Claude Code は終了コード 2 の stderr を**非ブロッキング エラー** (ページ 469) と同じ方法で、トランスクリプトに <hook name> hook error 通知としてレンダリングします。Claude はそれを見ず、セッションまたはサブエージェントは進行します。SubagentStart の場合、通知は親会話ではなく、サブエージェント自身のトランスクリプトに表示されます。

21.3.3 HTTP レスポンス処理

HTTP フックは終了コードと stdout の代わりに HTTP ステータス コードとレスポンス本体を使用します。以下の結果はほとんどのイベントに適用されます。WorktreeCreate のように**イベントごとの表** (ページ 472) に独自の失敗時の規約を持つイベントは、失敗した HTTP フックにもその規約を適用します。

- **2xx で空の本体**: 成功、終了コード 0 で出力なしと同等
- **2xx で JSON オブジェクト本体**: コマンド フックと同じ *JSON* **出力** (ページ 474) スキーマを使用して解析。スキーマ検証に失敗した本体は非ブロッキング エラー
- **2xx でその他の本体（プレーン テキストなど）**: 非ブロッキング エラー。2xx 以外のステータスと同じように処理されます。Claude Code はテキストを Claude のコンテキストに追加しません
- **2xx 以外のステータス**: 非ブロッキング エラー、実行は続行
- **接続失敗**: 非ブロッキング エラー、実行は続行
- **タイムアウト**: **タイムアウト** (ページ 471) で説明されているとおり、フックはキャンセルされます

コマンド フックとは異なり、HTTP フックはステータス コードのみでブロッキング エラーを通知できません。ツール呼び出しをブロックまたは権限を拒否するには、適切な決定フィールドを含む JSON 本体を持つ 2xx レスポンスを返します。

21.3.4 JSON 出力

終了コードではブロックするか何もしないかしか選べませんが、JSON 出力はより細かい制御を提供します。終了コード 2 でブロックする代わりに、終了 0 して stdout に JSON オブジェクトを出力します。Claude Code はその JSON から特定のフィールドを読み取り、ブロック、許可、またはユーザーへのエスカレーションのための**決定制御** (ページ 477) を含む動作を制御します。

注釈:

フックごとに 1 つのアプローチを選択してください。終了コードのみでシグナリングするか、終了 0 して構造化制御のために JSON を出力するかのいずれかです。両方を混在させた場合、終了 2 は**ブロッキング効果** (ページ 472) を維持し、Claude Code は引き続き JSON フィールドを読み取ります。ただし、**終了コード** *2* (ページ 470) で説明されている elicitation の例外が 1 つあります。

フックの stdout には JSON オブジェクトのみが含まれている必要があります。シェル プロファイルがスタートアップ時にテキストを出力する場合、JSON 解析に干渉する可能性があります。トラブルシューティング ガイドの**フックの *JSON* が効果を持たない** (ページ 440) を参照してください。

フックの `additionalContext`、`systemMessage`、`initialUserMessage` の文字列、およびプレーン stdout は 10,000 文字に制限されています。

- **スコープ**: 同じイベントに対して複数のフックが実行される場合でも、Claude Code は各文字列を個別に計測します。JSON 出力の場合、各フィールドは個別に計測されます。プレーン stdout は全体として計測されます。
- **制限を超えた場合**: Claude Code は出力をセッション ディレクトリ内のファイルに保存し、ファイルパスと最初の最大 2,000 文字のプレビューに置き換えます。大きな有効な Bash 結果も同じ方法で処理されます（出力制限で説明）。その Bash の上限とは異なり、この上限を引き上げる設定や環境変数はありません。
- **ファイルの読み取り**: Claude Code は Claude にファイルを読むよう求めないため、Claude が常に確認する必要があるものは上限内に収めてください。

JSON オブジェクトは 3 種類のフィールドをサポートしています。

- `continue` **などのユニバーサル フィールド**は以下の表にリストされています。すべてのイベントがこれらを受け入れますが、一部のイベントはこれらを破棄するか、`systemMessage` をトランスクリプト以外の場所に配信します。各イベントのセクションにその旨が記載されています。`terminalSequence` もそれらのイベントで機能しますが、**ターミナル通知を発行** (ページ 475) に記載されている例外があります。
- **トップレベルの** `decision` **と** `reason` は一部のイベントで使用され、ブロックまたはフィードバックを提供します。
- `hookSpecificOutput` はより豊かな制御が必要なイベント用のネストされたオブジェクトです。イベント名に設定された `hookEventName` フィールドが必要です。

フィールド	デフォルト	説明
continue	true	false の場合、フックが実行された後、Claude は完全に処理を停止します。イベント固有の決定フィールドよりも優先されます
stopReason	なし	continue が false のときにユーザーに表示されるメッセージ。会話に残るため、会話が続行された場合は Claude にも表示されます
suppressOutput	false	効果はありません。Claude Code はこのフィールドを受け入れますが、それに基づいて動作しません。成功したフックの stdout はトランスクリプトに表示されることはなく、デバッグ ログに記録されます
systemMessage	なし	ユーザーに表示される警告メッセージ。Agent SDK および *--output-format stream-json* (ページ 695) の出力では、SDKInformationalMessage として届く場合があります
terminalSequence	なし	Claude Code が代わりに発行するターミナル エスケープ シーケンス（デスクトップ通知、ウィンドウ タイトル、ベルなど）。OSC 0/1/2/9/99/777 と BEL に制限されます。値に許可リスト外のものが含まれている場合、フィールドは無視されます。フックでは利用できない /dev/tty への書き込みの代わりにこれを使用してください

Claude を完全に停止するには、次のようにします。

```
{ "continue": false, "stopReason": "Build failed, fix errors before continuing" }
```

PreToolUse および PostToolUse フックの場合、Claude がまだ応答をストリーミングしている間にツール呼び出しが失敗または完了した場合でも、停止が適用されます。

ターミナル通知を発行

フックは制御ターミナルなしで実行されるため、エスケープ シーケンスを /dev/tty に直接書き込むことは失敗します。代わりに、エスケープ シーケンスを terminalSequence フィールドで返し、Claude Code は独自のターミナル書き込みパスを通じてそれを発行します。これはレース フリーで、tmux と GNU screen 内で機能し、/dev/tty がない Windows で機能します。

フィールドは 1 つ以上の許可リストに登録されたエスケープ シーケンスの文字列を受け入れます。

- OSC 0、1、2: ウィンドウとアイコン タイトル
- OSC 9: iTerm2、ConEmu、Windows Terminal、WezTerm 通知（9;4 タスクバー進捗を含む）
- OSC 99: Kitty 通知
- OSC 777: urxvt、Ghostty、Warp 通知
- 裸の BEL

シーケンスは BEL または ST で終了する場合があります。許可リスト外のもの（CSI カーソルと色シーケンス、OSC パレット シーケンス、OSC 8 ハイパーリンク、OSC 52 クリップボード書き込み、OSC 1337 を含む）は拒否され、フィールドは無視されます。

Claude Code はフックの出力を処理するときにシーケンス自体を書き込むため、このフィールドは `Notification` や `StopFailure` など、`systemMessage` と `continue` を破棄するイベントでも機能します。ただし、2 つの制限があります。

- Claude Code は対話セッションでのみ、かつそのインターフェイスが画面に表示されている間のみシーケンスを書き込みます。`-p` フラグを使用した非対話モードおよび Agent SDK では、このフィールドは無視されます。
- `WorktreeCreate` コマンド フックは JSON を返せません。Claude Code はその stdout を worktree パスとして読み取るためです。HTTP `WorktreeCreate` フックは JSON を返すため、このフィールドを含めることができます。

以下の例は `Notification` フックからデスクトップ通知を発火します。エスケープ シーケンスは `printf` 8 進数エスケープで構築されるため、制御バイトはシェル コマンド ラインに表示されず、`jq -n --arg` は JSON 出力を構築するため、通知メッセージの引用符、バックスラッシュ、改行は正しくエスケープされます。

```
#!/bin/bash
# Notification hook: ping the desktop when Claude Code needs attention.
input=$(cat)
title="Claude Code"
body=$(jq -r '.message // "Needs your attention"' <<<"$input")
seq=$(printf '\033]777;notify;%s;%s\007' "$title" "$body")
jq -nc --arg seq "$seq" '{terminalSequence: $seq}'
```

`{ "terminalSequence": "..." }` の形状は、任意のシェルまたは言語から同じです。

Claude 用にコンテキストを追加

`additionalContext` フィールドは、フックから Claude のコンテキストウィンドウに文字列を渡します。Claude Code は文字列を**システムリマインダー**でラップし、フックが発火した時点で会話に挿入します。Claude は次のモデル リクエストでリマインダーを読み取りますが、インターフェイスではチャット メッセージとして表示されません。

`hookSpecificOutput` 内でイベント名と一緒に `additionalContext` を返します。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "This file is generated. Edit src/schema.ts and run `bun␣
↪generate` instead."
  }
}
```

リマインダーが表示される場所はイベントに依存します。

- *SessionStart* (ページ 481) および *SubagentStart* (ページ 521): 会話の開始時、最初のプロンプトの前
- *UserPromptSubmit* (ページ 487) および *UserPromptExpansion* (ページ 490): 送信されたプロンプトの横
- *PreToolUse* (ページ 495)、*PostToolUse* (ページ 509)、*PostToolUseFailure* (ページ 513)、および *PostToolBatch* (ページ 515): ツール結果の横
- *Stop* (ページ 526) および *SubagentStop* (ページ 522): ターンの終了時。会話は続行されるため、Claude はフィードバックに対応できます。*Stop* **決定制御** (ページ 528) を参照してください
- *PostModelSwitch* (ページ 547): 切り替え後の次のリクエストとともに。タイミングについては *PostModelSwitch* **決定制御** (ページ 548) を参照してください

複数のフックが同じイベントに対して `additionalContext` を返す場合、Claude はすべての値を受け取ります。

値が 10,000 文字を超える場合、Claude Code はテキストをセッション ディレクトリ内のファイルに書き込み、代わりにファイル パスと最初の最大 2,000 文字のプレビューを Claude に渡します。Claude はファイルを読むことができますが、Claude Code はそれを読むよう求めません。

Claude が現在の環境の状態または実行されたばかりの操作について知っておくべき情報に `additionalContext` を使用します。

- **環境状態**: 現在のブランチ、デプロイ ターゲット、またはアクティブな機能フラグ
- **条件付きプロジェクト ルール**: 編集されたばかりのファイルに適用されるテスト コマンド、この worktree で読み取り専用のディレクトリ
- **外部データ**: 割り当てられたオープン イシュー、最近の CI 結果、内部サービスから取得されたコンテンツ

変わらない指示については、*CLAUDE.md* (ページ 997) を優先します。スクリプトを実行せずに読み込まれ、静的なプロジェクト規約の標準的な場所です。

テキストを命令型システム指示ではなく、事実的なステートメントとして記述します。「デプロイ ターゲットは本番環境です」または「このリポジトリは `bun test` を使用します」などのフレーズはプロジェクト情報として読み取られます。帯域外システム コマンドとしてフレーム化されたテキストは Claude のプロンプトインジェクション防御をトリガーする可能性があり、その場合 Claude はテキストをコンテキストとして扱う代わりにユーザーに提示します。

Claude Code は注入されたテキストをセッション トランスクリプトに保存します。`PostToolUse` や `UserPromptSubmit` などのセッション途中のイベントの場合、`--continue` または `--resume` で再開すると、Claude Code は過去のターンについてフックを再実行するのではなく保存されたテキストを再生するため、タイムスタンプやコミット SHA などの値は古くなります。`SessionStart` フックは再開時に `source` を `"resume"`（`--fork-session` を追加した場合は `"fork"`）に設定して再度実行されるため、コンテキストをリフレッシュできます。

決定制御

すべてのイベントが JSON を通じたブロッキングまたは動作制御をサポートしているわけではありません。サポートするイベントは、その決定を表現するために異なるフィールド セットを使用します。フックを書く前に、このテーブルをクイック リファレンスとして使用してください。

イベント	決定パターン	キー フィールド
UserPromptSubmit、UserPromptExpansion、PostToolUse、PostToolUseFailure、PostToolBatch、Stop、SubagentStop、ConfigChange、PreCompact	トップレベル `decision`	`decision: "block"`、`reason`。Stop と SubagentStop は**会話を続行する非エラー フィードバック** (ページ 528) のために `hookSpecificOutput.additionalContext` も受け入れます
TeammateIdle、TaskCompleted	終了コードまたは `continue: false`	終了コード 2 は stderr フィードバックとともにアクションをブロックします。JSON `{"continue": false, "stopReason": "..."}` もチームメイトを完全に停止し、`Stop` フックの動作と一致します。*TaskUpdate* **ツールがイベントをトリガーした場合、*TaskCompleted* はこれを無視します** (ページ 525)
TaskCreated	終了コードまたはトップレベル `decision`	終了コード 2 または `decision: "block"` は**タスクをキャンセル** (ページ 524) し、メッセージを Claude に返します。`continue: false` は無視されます
PreToolUse	`hookSpecificOutput`	`permissionDecision`（allow/deny/ask/defer）、`permissionDecisionReason`
PreModelSwitch	`hookSpecificOutput` またはトップレベル `decision`	`permissionDecision`（allow/deny/ask）、`permissionDecisionReason`。`decision: "block"` も**切り替えをキャンセル** (ページ 546) します
PermissionRequest	`hookSpecificOutput`	`decision.behavior`（allow/deny）
PermissionDenied	`hookSpecificOutput`	`retry: true` はモデルが拒否されたツール呼び出しを再試行できることを伝えます。Claude Code は**判定のない拒否** (ページ 517) ではこれを無視します
WorktreeCreate	パス戻り値	コマンド フックは stdout にパスを出力します。HTTP フックは `hookSpecificOutput.worktreePath` を返します。フック失敗またはパス欠落で作成が失敗
WorktreeRemove	終了コード	0 以外の終了コードは、その後もディレクトリが存在する場合に削除を失敗させます。JSON 出力は破棄されます

次のページに続く

表 15 – 前のページからの続き

イベント	決定パターン	キー フィールド
Elicitation	`hookSpecificOutput`	`action`（accept/decline/cancel）、`content`（accept の場合のフォーム フィールド値）
ElicitationResult	`hookSpecificOutput`	`action`（accept/decline/cancel）、`content`（フォーム フィールド値を上書き）
MessageDisplay	`hookSpecificOutput`	`displayContent` は画面に表示されるテキストを置き換えます。表示のみ: トランスクリプトと Claude が見るものは元のままです
SessionStart、SubagentStart、PostModelSwitch	コンテキストのみ	`hookSpecificOutput.additionalContext` は Claude 用にコンテキストを追加します。SessionStart は *initialUserMessage*、*watchPaths*、*sessionTitle*、および *reloadSkills* (ページ 483) も受け入れます。ブロッキングまたは決定制御なし
Setup、Notification、SessionEnd、PostCompact、InstructionsLoaded、StopFailure、CwdChanged、DirectoryAdded、FileChanged	なし	決定制御なし。ログやクリーンアップなどの副作用に使用

いくつかのイベントは、許可またはブロックするだけでなく、コンテンツを書き直すこともできます。

- `PreToolUse`: `hookSpecificOutput` の直下の `updatedInput` は、ツールが実行される前にそのツールの引数を置き換えます。*PreToolUse* **決定制御** (ページ 502) を参照してください
- `PermissionRequest`: `decision` オブジェクト内の `updatedInput`。*PermissionRequest* **決定制御** (ページ 507) を参照してください
- `PostToolUse`: `updatedToolOutput` はツールの結果を置き換えます。*PostToolUse* **決定制御** (ページ 510) を参照してください
- `UserPromptSubmit`: プロンプトを置き換えることはできません。`additionalContext` をそれと一緒に注入するだけです

秘匿化や変換のユースケースの場合、アウトバウンド ツール入力の場合は `PreToolUse` で、インバウンド ツール結果の場合は `PostToolUse` で傍受します。

各パターンの実行例を以下に示します。

トップレベル決定

`decision` の唯一の値は `"block"` です。アクションを進行させるには、JSON から `decision` を省略するか、JSON なしで終了 0 で終了します。

```
{
  "decision": "block",
  "reason": "Test suite must pass before proceeding"
}
```

PreToolUse

より豊かな制御のために hookSpecificOutput を使用します。許可、拒否、またはユーザーへのエスカレーション。ツール入力を実行前に変更したり、Claude 用に追加コンテキストを注入することもできます。オプションの完全なセットについては、*PreToolUse* **決定制御** (ページ 502) を参照してください。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
    "permissionDecision": "deny",
    "permissionDecisionReason": "Database writes are not allowed"
  }
}
```

PermissionRequest

hookSpecificOutput を使用して、ユーザーに代わって権限リクエストを許可または拒否します。許可する場合、ツールの入力を変更したり、権限ルールを適用して、ユーザーが再度プロンプトされないようにすることもできます。オプションの完全なセットについては、*PermissionRequest* **決定制御** (ページ 507) を参照してください。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

Bash コマンド検証、プロンプト フィルタリング、自動承認スクリプトを含む拡張例については、ガイドの**自動化できること** (ページ 413) と Bash コマンド バリデーター リファレンス実装を参照してください。

21.4 フックイベント

各イベントは、フックを実行できる Claude Code のライフサイクル上のポイントに対応しています。以下のセクションはライフサイクルに沿った順序で並んでおり、セッションのセットアップからエージェント型ループを経てセッション終了までを扱います。各セクションでは、イベントが発火するタイミング、サポートする matcher、受け取る JSON 入力、出力を通じて動作を制御する方法を説明します。

21.4.1 SessionStart

Claude Code が新しいセッションを開始するとき、または既存のセッションを再開するときに実行されます。既存の issue やコードベースへの最近の変更といった開発コンテキストの読み込みや、環境変数の設定に便利です。スクリプトを必要としない静的なコンテキストには、代わりに *CLAUDE.md* (ページ 997) を使用してください。

SessionStart はすべてのセッションで実行されるため、これらのフックは高速に保ってください。サポートされるのは `type: "command"` と `type: "mcp_tool"` のフックのみです。`mcp_tool` フックが実行されるタイミングについては、*MCP* **ツールフックのフィールド** (ページ 461) を参照してください。

matcher の値は、セッションがどのように開始されたかに対応します。

Matcher	発火するタイミング
`startup`	新しいセッション
`resume`	`--resume`、`--continue`、または `/resume`
`clear`	`/clear`
`compact`	自動または手動のコンテキスト圧縮
`fork`	既存のセッションからフォークされた新しいセッション。`--resume` または `--continue` と組み合わせた `--fork-session`、`/fork` によるバックグラウンドコピー、`/branch`、またはバックグラウンドに移動した会話が該当します

v2.1.214 より前は、フォークされたセッションはソースとして `"resume"` を報告していました。

対話セッションを開始したとき、起動時に `--continue` または `--resume` で会話を再開したとき、または `/clear` を実行したときは、SessionStart フックがバックグラウンドで実行されます。すぐに入力を始めることができ、再開した会話はフックを待たずに表示されます。ただし、フックのコンテキストが Claude に届くように、Claude の最初の応答はフックの完了を待ちます。

セッション内で `/resume` を使って会話を切り替えた場合は、切り替え自体がフックの完了を待ちます。バックグラウンドのフックがまだ実行中に `/clear` を実行したり別の会話に切り替えたりすると、フックが返す内容はセッションに一切適用されません。

起動時にも同じ待機が適用され、再開したセッションも含まれます。SessionStart フックの実行中に送信したプロンプトは、フックが完了するまで Claude に届きません。

いずれの待機中も、`Esc` を押すとプロンプトを送信せずに入力欄に戻すことができます。フックは実行を続けます。

SessionStart の入力

共通の入力フィールド (ページ 466) に加えて、SessionStart フックは `source` と、オプションで `model`、`agent_type`、`session_title` を受け取ります。

フィールド	説明
source	セッションの開始方法。新しいセッションでは "startup"、再開されたセッションでは "resume"、/clear の後では "clear"、コンテキスト圧縮の後では "compact"、既存のセッションからフォークされた新しいセッションでは "fork"
model	アクティブなモデルの識別子。たとえば /clear の後や、会話の復旧によってセッションが復元された場合など、省略されることがあるため、読み取る前にフィールドの有無を確認してください
agent_type	エージェント名。claude --agent <name> で Claude Code を起動した場合に含まれます
session_title	セッションのカスタムタイトル。--name、/rename、フックの sessionTitle 出力、Agent SDK の renameSession() などで設定されている場合に含まれます。sessionTitle を出力するフックは、既存のカスタムタイトルを上書きしないように、まずこのフィールドを確認できます

名前を付けていないセッションにも生成されたタイトルが付いている場合があります。このタイトルはカスタムタイトルではないため、session_title には含まれません。

source が "resume" または "fork" で、トランスクリプトに Claude の応答が少なくとも 1 つ含まれている場合、SessionStart フックは以下の 4 つのフィールドも受け取ります。フックはこれらを使用して、古い会話を再開するコストを最初のリクエストの前に報告できます。たとえば *systemMessage* (ページ 474) で報告します。これらのフィールドには Claude Code v2.1.251 以降が必要です。

フィールド	説明
seconds_since_last_response	再開されたトランスクリプト内の最後の応答からの経過時間 (実時間の秒数)
context_tokens	再開されたセッションの最初のリクエストがプロンプトとして再送信するトークン数
prompt_cache_likely_expired	最後の応答がセッションのプロンプトキャッシュの有効期間より古い場合、またはその後のコンテキスト圧縮によってキャッシュされた会話が置き換えられた場合に true
estimated_cache_write_usd	セッションのモデルで context_tokens をプロンプトキャッシュに書き込む推定コスト（米ドル）。応答は含みません

次の例は、最後の応答から 90 分後に再開されたセッションの入力を示しています。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "SessionStart",
  "source": "resume",
  "model": "claude-opus-5",
  "seconds_since_last_response": 5400,
```

(次のページに続く)

(前のページからの続き)

```
  "context_tokens": 182340,
  "prompt_cache_likely_expired": true,
  "estimated_cache_write_usd": 1.1396
}
```

SessionStart の判定制御

Claude Code は、**プレーンテキストとして扱う** (ページ 469) stdout を Claude のコンテキストに追加します。すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、以下のイベント固有のフィールドを返すことができます。

フィールド	説明
additionalContext	会話の開始時、最初のプロンプトの前に Claude のコンテキストに追加される文字列。テキストの届け方と記述すべき内容については、*Claude* **にコンテキストを追加する** (ページ 476) を参照してください
initialUserMessage	セッションの最初のユーザーメッセージとして使用される文字列。-p フラグを使った**非対話モード** (ページ 695) で適用され、プロンプトが指定されていなくても最初のターンになります。プロンプトが指定されている場合、それは次のターンとして続きます。既存のターンに付加される additionalContext とは異なり、これはターンそのものを作成します
sessionTitle	セッションのタイトルを設定します。効果は /rename と同じです。起動フォルダ、git ブランチ、worktree 名からセッションに自動的に名前を付けるのに使用します。source が "startup"、"resume"、"fork" の場合に適用され、"clear" と "compact" では無視されます
watchPaths	このセッション中に *FileChanged* (ページ 535) イベントを監視する絶対パスの配列
reloadSkills	ブール値。true の場合、Claude Code は SessionStart フックの完了後に**スキル** (ページ 275) とコマンドのディレクトリを再スキャンするため、フックがインストールしたスキルを同じセッションの最初のプロンプトから利用できます

```
{
  "hookSpecificOutput": {
    "hookEventName": "SessionStart",
    "additionalContext": "Current branch: feat/auth-refactor\nUncommitted changes:␣
↪src/auth.ts, src/login.tsx\nActive issue: #4211 Migrate to OAuth2",
    "sessionTitle": "auth-refactor"
  }
}
```

このイベントではプレーンな stdout がすでに Claude に届くため、コンテキストを読み込むだけのフックは JSON を組み立てずに stdout へ直接出力できます。コンテキストを sessionTitle などの他のフィールドと組み合わせる必要がある場合は、JSON 形式を使用してください。

SessionStart フックがスキルをインストールまたは更新する場合は `reloadSkills` を使用します。スキルの検出は通常 SessionStart フックの完了前に実行されるため、フックが `~/.claude/skills/` や `.claude/skills/` に書き込んだファイルは、これを使わないと次のセッションでしか表示されません。次の例では、共有スキルのリポジトリを同期し、再スキャンを要求します。

```
#!/bin/bash

git -C ~/.claude/skills/team-skills pull --quiet 2>/dev/null || \
  git clone --quiet https://git.example.com/your-org/team-skills.git ~/.claude/skills/
↪team-skills

echo '{"hookSpecificOutput": {"hookEventName": "SessionStart", "reloadSkills": true}}'
```

リポジトリの URL はプレースホルダーです。自分のスキルリポジトリに置き換えてください。プレースホルダーのままでは clone が失敗し、stderr に `fatal:` メッセージが出力されます。終了コード 0 で終了した SessionStart フックの stderr は情報提供のみを目的としているため、`reloadSkills` の要求は引き続き適用されます。

環境変数を永続化する

SessionStart フックは `CLAUDE_ENV_FILE` 環境変数にアクセスできます。この変数は、後続の Bash コマンド用に環境変数を永続化できるファイルパスを提供します。

個々の環境変数を設定するには、`CLAUDE_ENV_FILE` に `export` 文を書き込みます。他のフックが設定した変数を保持するために、追記（`>>`）を使用してください。

```
#!/bin/bash

if [ -n "$CLAUDE_ENV_FILE" ]; then
  echo 'export NODE_ENV=production' >> "$CLAUDE_ENV_FILE"
  echo 'export DEBUG_LOG=true' >> "$CLAUDE_ENV_FILE"
  echo 'export PATH="$PATH:./node_modules/.bin"' >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

セットアップコマンドによるすべての環境の変更を取り込むには、エクスポートされた変数を前後で比較します。

```
#!/bin/bash

ENV_BEFORE=$(export -p | sort)

# Run your setup commands that modify the environment
source ~/.nvm/nvm.sh
nvm use 20

```

(次のページに続く)

(前のページからの続き)

```
if [ -n "$CLAUDE_ENV_FILE" ]; then
  ENV_AFTER=$(export -p | sort)
  comm -13 <(echo "$ENV_BEFORE") <(echo "$ENV_AFTER") >> "$CLAUDE_ENV_FILE"
fi

exit 0
```

注釈:

CLAUDE_ENV_FILE は SessionStart、*Setup* (ページ 485)、*CwdChanged* (ページ 533)、*FileChanged* (ページ 535) フックで利用できます。その他のフックタイプはこの変数にアクセスできません。

21.4.2 Setup

--init-only で Claude Code を起動した場合、または -p フラグを使った**非対話モード** (ページ 695) で --init か --maintenance を付けて起動した場合にのみ発火します。通常の起動時には発火しません。通常のセッション開始とは別に、CI やスクリプトから明示的にトリガーする一度きりの依存関係のインストールや定期的なクリーンアップに使用します。セッションごとの初期化には、代わりに *SessionStart* (ページ 481) を使用してください。

matcher の値は、フックをトリガーした CLI フラグに対応します。

Matcher	発火するタイミング
init	claude --init-only または claude -p --init
maintenance	claude -p --maintenance

claude --init-only を実行すると、Claude Code は Setup フックと startup matcher の SessionStart フックを実行し、会話を開始せずに終了します。

-p で会話を開始または継続する場合は、引数として、または stdin へのパイプでプロンプトも指定する必要があります。SessionStart フックが *initialUserMessage* (ページ 483) を提供する場合や、**延期されたツール呼び出し** (ページ 504) を含むセッションを再開する場合は、プロンプトを省略できます。

成功した場合、--init-only はターミナルに何も出力しません。フックが実行されたことを確認するには、claude --debug-file <path> --init-only で起動し（<path> はログファイルの場所に置き換えます）、ログで Setup と SessionStart のフックのエントリを確認してください。

Setup はすべての起動時に発火するわけではないため、依存関係のインストールを必要とするプラグインは Setup だけに頼ることはできません。実用的なパターンは、初回使用時に依存関係を確認し、見つからなければインストールすることです。たとえば、${CLAUDE_PLUGIN_DATA}/node_modules の有無をテストし、なければ npm install を実行するフックやスキルです。インストールした依存関係の保存場所については、永続データディレクトリを参照してください。マーケットプレイスを通じてプラグインを配布する場合は、このパターンが不要なこともあります。Claude Code はプラグインをキャッシュする際に、対象となる Node.js パッケージの依存関係を自動的にインストールします。

Setup の入力

共通の入力フィールド (ページ 466) に加えて、Setup フックは "init" または "maintenance" のいずれかが設定された trigger フィールドを受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "Setup",
  "trigger": "init"
}
```

Setup の判定制御

Setup フックはブロックできず、どの終了コードでも実行は継続されます。どの終了コードであっても、Claude Code は Setup フックの *JSON* **出力フィールド** (ページ 474)（systemMessage、continue、hookSpecificOutput.additionalContext など）を破棄します。-p を使用する場合、Setup フックの stdout、stderr、終了コードは、--output-format stream-json --verbose で起動したときに限り、*hook_response* **イベント** (ページ 702) として実行の出力に表示されます。

Setup フックは CLAUDE_ENV_FILE にアクセスできます。このファイルに書き込まれた変数は、*SessionStart* **フック** (ページ 484) と同様に、セッションの後続の Bash コマンドに引き継がれます。Setup で実行されるのは type: "command" フックのみです。Setup 上の type: "mcp_tool" フックは、*MCP* **ツールフックのフィールド** (ページ 461) で説明されているとおり、常にスキップされます。

21.4.3 InstructionsLoaded

CLAUDE.md または .claude/rules/*.md ファイルがコンテキストに読み込まれたときに発火します。このイベントは、即時に読み込まれるファイルについてはセッション開始時に発火し、ファイルが遅延読み込みされたときにも再度発火します。遅延読み込みの例としては、ネストされた CLAUDE.md を含むサブディレクトリに Claude がアクセスしたときや、paths: フロントマターを持つ条件付きルールが一致したときがあります。このフックはブロックや判定制御をサポートしていません。可観測性を目的として非同期で実行されます。

Claude が **Project instructions** 設定を通じて *AGENTS.md* **を直接読み込む** (ページ 1008) 場合、このイベントは発火しません。CLAUDE.md が AGENTS.md をインポートする場合は、他のインポートされたファイルと同様に load_reason が include に設定されて発火し、CLAUDE.md が AGENTS.md へのシンボリックリンクである場合は、通常の CLAUDE.md の読み込みとして発火します。

matcher は load_reason に対して照合されます。たとえば、セッション開始時に読み込まれたファイルに対してのみ発火させるには "matcher": "session_start" を、遅延読み込みに対してのみ発火させるには "matcher": "path_glob_match|nested_traversal" を使用します。

InstructionsLoaded の入力

共通の入力フィールド (ページ 466) に加えて、InstructionsLoaded フックは以下のフィールドを受け取ります。

フィールド	説明
file_path	読み込まれた指示ファイルの絶対パス
memory_type	ファイルのスコープ："User"、"Project"、"Local"、または "Managed"
load_reason	ファイルが読み込まれた理由："session_start"、"nested_traversal"、"path_glob_match"、"include"、または "compact"。"compact" の値は、コンテキスト圧縮イベントの後に指示ファイルが再読み込みされたときに発火します
globs	ファイルの paths: フロントマターにあるパスの glob パターン（存在する場合）。path_glob_match の読み込みの場合にのみ含まれます
trigger_file_path	遅延読み込みの場合に、この読み込みをトリガーしたアクセス先のファイルのパス
parent_file_path	include の読み込みの場合に、このファイルをインクルードした親の指示ファイルのパス

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project",
  "hook_event_name": "InstructionsLoaded",
  "file_path": "/Users/my-project/CLAUDE.md",
  "memory_type": "Project",
  "load_reason": "session_start"
}
```

InstructionsLoaded の判定制御

InstructionsLoaded フックには判定制御がありません。指示の読み込みをブロックしたり変更したりすることはできません。Claude Code は、systemMessage や continue などの *JSON* **出力フィールド** (ページ 474) を破棄します。このイベントは、監査ログ、コンプライアンスの追跡、可観測性に使用してください。

21.4.4 UserPromptSubmit

プロンプトが送信されたとき、Claude がそれを処理する前に実行されます。これにより、プロンプトや会話に基づいて追加のコンテキストを加えたり、プロンプトを検証したり、特定の種類のプロンプトをブロックしたりできます。

UserPromptSubmit フックは、ユーザーが入力したプロンプトだけで発火するわけではありません。Claude Code は以下の場合にもこれらのフックを実行します。

- スケジュールタスクの発火（/loop の反復を含む）

- **バックグラウンドのサブエージェント** (ページ 344) が、それを開始したセッションに結果を報告したとき
- 別のセッションが送信したメッセージがメインの会話に届いたとき

UserPromptSubmit フックのデフォルトのタイムアウトは、command、http、mcp_tool タイプで 30 秒です。これは、他のほとんどのイベントでのこれらのタイプのデフォルトである 600 秒より短くなっています。このフックはすべてのプロンプトの前に実行され、完了するまでモデルの処理をブロックするため、フックが停止するとセッションも停止します。フックにより長い時間が必要な場合は、フックエントリの timeout フィールドを設定してください。

async: true (ページ 558) で実行するコマンドフックを除き、タイムアウトに達した UserPromptSubmit のコマンド、HTTP、または MCP ツールのフックはキャンセルされ、その出力は additionalContext も含めて破棄されます。プロンプトはそのコンテキストなしで Claude に届きます。トランスクリプトには、フック名、発生したタイムアウト、出力が破棄されたことを示す通知が表示されます。

UserPromptSubmit 上の Agent SDK コールバックフックがタイムアウトに達すると、フック名とタイムアウトを示すメッセージとともにプロンプトがブロックされます。このイベントのコールバックは、フェイルオープンしてはならないポリシーゲートとして機能している可能性があるためです。セッションは継続します。v2.1.208 より前は、このイベントでコールバックがタイムアウトすると、実行エラーでターンが終了していました。

UserPromptSubmit の入力

共通の入力フィールド (ページ 466) に加えて、UserPromptSubmit フックは送信されたテキストを含む prompt フィールドを受け取ります。[Pasted text #N] プレースホルダーに折りたたまれた貼り付けコンテンツは、その場で展開された状態で届きます。Claude Code が**貼り付けたテキストを *Claude* 向けにマークする** (ページ 838) セッションでは、展開されたコンテンツは <pasted_content id="…"> の行と </pasted_content id="…"> の行の間に置かれるため、フックがプロンプトを解析する場合はこれらの行を考慮してください。

UserPromptSubmit フックは、セッションにカスタムタイトルがある場合に session_title も受け取ります。意味は *SessionStart* **の** *session_title* **フィールド** (ページ 481) と同じです。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "UserPromptSubmit",
  "prompt": "Write a function to calculate the factorial of a number"
}
```

UserPromptSubmit の判定制御

`UserPromptSubmit` フックは、送信されたプロンプトを処理するかどうかを制御し、コンテキストを追加できます。すべての *JSON* **出力フィールド** (ページ 474) を使用できます。

終了コード 0 で会話にコンテキストを追加する方法は 2 つあります。

- **プレーンテキストの stdout**：Claude Code は、**プレーンテキストとして扱う** (ページ 469) stdout を Claude のコンテキストに追加します
- `additionalContext` **を含む JSON**：より細かく制御するには、以下の JSON 形式を使用します。`additionalContext` フィールドがコンテキストとして追加されます

どちらの経路でも、トランスクリプトに表示されるエントリは作成されません。プレーンな stdout と `additionalContext` の値は、それぞれフック名で始まるシステムリマインダーとして挿入され、Claude は両方を読みます。配信を確認するには、**デバッグログ** (ページ 562) を確認してください。

プロンプトをブロックするには、`decision` を `"block"` に設定した JSON オブジェクトを返します。

フィールド	説明
`decision`	`"block"` は、プロンプトが Claude に届く前に停止します。プロンプトの続行を許可するには省略します
`reason`	`decision` が `"block"` の場合にユーザーに表示されます。コンテキストには追加されません
`additionalContext`	送信されたプロンプトとともに Claude のコンテキストに追加される文字列。*Claude* **にコンテキストを追加する** (ページ 476) を参照してください
`sessionTitle`	セッションのタイトルを設定します。プロンプトの内容に基づいてセッションに自動的に名前を付けるのに使用します
`suppressOriginalPrompt`	フックがプロンプトをブロックするときに `true` の場合、ブロックメッセージからプロンプトのテキストを除外します。**ブロックされたプロンプトが残すもの** (ページ 490) を参照してください

終了コード 2 で終了してブロックするフックは、`reason` と同じ経路をたどります。ブロックメッセージは stderr のテキストをユーザーに表示し、それはコンテキストには追加されません。

```
{
  "decision": "block",
  "reason": "Explanation for decision",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptSubmit",
    "additionalContext": "My additional context here",
    "sessionTitle": "My session title",
    "suppressOriginalPrompt": true
  }
}
```

ブロックされたプロンプトが残すもの

ブロックされたプロンプトは Claude に届きませんが、そのテキストがすべての場所から削除されるわけではありません。デフォルトでは、ユーザーに表示されるブロックメッセージの末尾に `Original prompt:` と送信されたテキストが続き、Claude Code はそのメッセージをディスク上のセッションのトランスクリプトファイルに書き込みます。メッセージからテキストを除外するには、`hookSpecificOutput` 内に `"suppressOriginalPrompt": true` を含む JSON を出力します。これは、フックが `decision: "block"` でブロックする場合でも、終了コード 2 で終了してブロックする場合でも機能します。JSON を出力しない終了コード 2 のフックでは、ブロックメッセージに常にプロンプトのテキストが含まれます。

`suppressOriginalPrompt` が変更するのはブロックメッセージのみです。送信されたテキストは、セッションのトランスクリプトやプロンプト履歴などのローカルファイルに引き続き現れる可能性があるため、ブロックするフックは機密情報をディスクに残さないための手段にはなりません。これらのファイルを制限または削除するには、プレーンテキストでの保存とローカルデータの消去を参照してください。

21.4.5 UserPromptExpansion

ユーザーが入力したコマンドが、Claude に届く前にプロンプトへ展開されるときに実行されます。特定のコマンドの直接呼び出しをブロックしたり、特定のスキルにコンテキストを挿入したり、ユーザーが呼び出したコマンドをログに記録したりするのに使用します。たとえば、`deploy` に一致するフックは承認ファイルが存在しない限り `/deploy` をブロックでき、レビュースキルに一致するフックはチームのレビューチェックリストを `additionalContext` として追加できます。

このイベントは、`PreToolUse` がカバーしない経路を扱います。`Skill` ツールに一致する `PreToolUse` フックは Claude がツールを呼び出したときにのみ発火しますが、`/skillname` を直接入力すると `PreToolUse` を経由しません。`UserPromptExpansion` はその直接の経路で発火します。

`command_name` に対して照合します。すべてのプロンプト型コマンドで発火させるには、matcher を空のままにします。

UserPromptExpansion の入力

共通の入力フィールド (ページ 466) に加えて、UserPromptExpansion フックは `expansion_type`、`command_name`、`command_args`、`command_source`、および元の `prompt` 文字列を受け取ります。`expansion_type` フィールドは、スキルとカスタムコマンドでは `slash_command`、MCP サーバーのプロンプトでは `mcp_prompt` になります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../00893aaf.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "UserPromptExpansion",
  "expansion_type": "slash_command",
  "command_name": "example-skill",
  "command_args": "arg1 arg2",
  "command_source": "plugin",
```

(次のページに続く)

(前のページからの続き)

```
  "prompt": "/example-skill arg1 arg2"
}
```

UserPromptExpansion の判定制御

UserPromptExpansion フックは、展開をブロックしたり、コンテキストを追加したりできます。すべての *JSON* **出力フィールド** (ページ 474) を利用できます。

フィールド	説明
decision	"block" は、コマンドの展開を防ぎます。続行を許可するには省略します
reason	decision が "block" の場合にユーザーに表示されます
additionalContext	展開されたプロンプトとともに Claude のコンテキストに追加される文字列。*Claude* **にコンテキストを追加する** (ページ 476) を参照してください

終了コード 2 で終了してブロックするフックは、reason と同じ経路をたどります。ブロックメッセージは stderr のテキストをユーザーに表示します。

```
{
  "decision": "block",
  "reason": "This slash command is not available",
  "hookSpecificOutput": {
    "hookEventName": "UserPromptExpansion",
    "additionalContext": "Additional context for this expansion"
  }
}
```

21.4.6 MessageDisplay

アシスタントのメッセージが画面にストリーミングされている間に実行されます。Claude Code はメッセージを段階的に表示します。新たに完成した行のバッチがレンダリングできる状態になるたびに、フックがそれらの行を使って 1 回実行され、Claude Code はその位置にフックの置換テキストをレンダリングします。長いメッセージでは複数回呼び出され、短いメッセージでは 1 回だけのこともあります。

MessageDisplay は次の用途に使用します。

- 最小限の表示のために markdown を取り除く
- Agent SDK アプリケーションがユーザーに表示するテキストを変換する
- Claude の応答から API キーや内部ホスト名を伏せる

Claude Code はフックが返るまで各バッチを保留するため、フックは高速に保ってください。フックが失敗するかタイムアウトした場合、Claude Code は元のテキストを表示します。このイベントのデフォルトのタ

イムアウトは 10 秒です。フックにより長い時間が必要な場合は、フックエントリの `timeout` フィールドを設定してください。

MessageDisplay は表示専用です。置換テキストは画面にレンダリングされる内容のみを変更します。トランスクリプトと Claude が参照する内容は元のテキストのままなので、Claude が置換テキストを目にすることはなく、詳細モードでは元のテキストが表示されます。フックが受け取るのはアシスタントのメッセージテキストのみなので、ツールの結果やユーザーが入力したテキストは変更されずにレンダリングされます。

MessageDisplay は matcher をサポートしておらず、テキストをストリーミングするすべてのアシスタントメッセージで発火します。ツール呼び出しのみの応答など、テキストを含まないメッセージではトリガーされません。

Agent SDK のクエリや `claude -p` を含む非対話の実行では、MessageDisplay は行のバッチごとではなく、アシスタントメッセージごとに 1 回実行されます。この 1 回の呼び出しはメッセージの完了後に届き、メッセージのテキスト全体を含みます。`index` は `0`、`final` は `true` で、`delta` にメッセージ全体が入ります。各メッセージの `delta` テキストを収集するフックは、どちらのモードでも同じ合計テキストを受け取ります。

MessageDisplay の入力

共通の入力フィールド (ページ 466) に加えて、MessageDisplay フックはターンとメッセージの識別子、メッセージ内でのこの呼び出しの位置、および `delta` 内の新しいテキストを受け取ります。バッチの境界はテキストのストリーミング方法に依存するため、行が特定の方法でグループ化されることを期待するのではなく、`index` と `final` を使ってメッセージの進行状況を追跡してください。

フィールド	説明
`turn_id`	現在のターンの UUID
`message_id`	表示中のアシスタントメッセージの UUID。同じメッセージのすべてのバッチで一定です。これは API の `msg_…` ID ではないため、トランスクリプトのメッセージ ID と対応付けることはできません
`index`	メッセージ内でのこのバッチの 0 始まりのインデックス
`final`	メッセージの最後のバッチで `true`。各メッセージにはちょうど 1 つの最終バッチがあります
`delta`	前のバッチ以降に新たに完成した行 (末尾の改行を含む)。常に行全体ですが、最終バッチは行の途中で終わる場合があります。対話的な実行では、メッセージが改行で終わる場合は最終バッチの delta が空になるため、空でない delta ではなく `final` をメッセージ終了のシグナルとして扱ってください。Agent SDK と `claude -p` の実行では、1 回の呼び出しにメッセージ全体が含まれます


```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project",
  "hook_event_name": "MessageDisplay",
  "turn_id": "0c9e6a2f-7d41-4f4e-9a15-3f4f7c2b8d10",
```


(次のページに続く)

(前のページからの続き)

```
  "message_id": "5b2a9c8e-1f63-4d8a-b7c4-9e0d2a6f1c3b",
  "index": 0,
  "final": false,
  "delta": "Here is the plan:\n"
}
```

MessageDisplay の出力

すべてのフックで利用できる *JSON 出力フィールド* (ページ 474) に加えて、MessageDisplay フックは `displayContent` を返して、画面上の delta を置き換えることができます。

フィールド	説明
`displayContent`	delta の代わりに表示されるテキスト。元のテキストを表示するには省略します

MessageDisplay フックには判定制御がありません。メッセージをブロックしたり、トランスクリプトに保存される内容や Claude に送信される内容を変更したりすることはできません。Claude Code は JSON 出力の `displayContent` に従って動作し、`systemMessage` と `continue` は破棄します。

次の例では、プレーンテキストで表示するために Claude の応答から markdown の書式を取り除きます。スクリプトは stdin から各バッチを読み取り、`delta` から太字のマーカーとインラインコードのバッククォートを削除し、結果を `displayContent` として返します。

macOS/Linux

設定ファイルで、このイベントのコマンドフックを登録します。

```
{
  "hooks": {
    "MessageDisplay": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/plain-display.sh",
            "args": []
          }
        ]
      }
    ]
  }
}
```

このスクリプトをプロジェクトの `.claude/hooks/plain-display.sh` に保存し、`chmod +x` で実行可能にします。

```
#!/bin/bash
jq '{hookSpecificOutput: {hookEventName: "MessageDisplay", displayContent: (.
↪delta | gsub("\\*\\*"; "") | gsub("`"; ""))}}'
```

Windows (PowerShell)

PowerShell 経由でスクリプトを実行するコマンドフックを登録します。

```
{
  "hooks": {
    "MessageDisplay": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "powershell.exe",
            "args": [
              "-NoProfile",
              "-ExecutionPolicy",
              "Bypass",
              "-File",
              "${CLAUDE_PROJECT_DIR}/.claude/hooks/plain-display.ps1"
            ]
          }
        ]
      }
    ]
  }
}
```

`-NoProfile` フラグは PowerShell プロファイルの読み込みをスキップしてフックを高速に起動させ、`-ExecutionPolicy Bypass` は PowerShell がローカルのスクリプトファイルを実行できるようにします。

このスクリプトをプロジェクトの `.claude/hooks/plain-display.ps1` に保存します。

```
$batch = [Console]::In.ReadToEnd() | ConvertFrom-Json
$text = $batch.delta -replace '\*\*', '' -replace '``', ''
@{
  hookSpecificOutput = @{
    hookEventName = "MessageDisplay"
    displayContent = $text
  }
} | ConvertTo-Json
```

markdown を含まないバッチは変更されずにそのまま通過します。たとえば `jq` がないためにスクリプトが失敗した場合、Claude Code は元のテキストを表示し、その失敗はセッション内ではなく**デバッグ出力** (ページ 562) にのみ記録されます。

21.4.7 PreToolUse

Claude がツールのパラメーターを作成した後、ツール呼び出しを処理する前に実行されます。`End Conversation` を除く任意のツール名に一致します。対象は、`Bash`、`PowerShell`、`Edit`、`Write`、`Read`、`Glob`、`Grep`、`Agent`、`Workflow`、`WebFetch`、`WebSearch`、`AskUserQuestion`、`ExitPlanMode` などの組み込みツールと、任意の *MCP* **ツール名** (ページ 454) です。

書き込んだものが何であれ、特定のファイルがディスク上で変更されたときにフックを実行するには、ファイル編集ツールを名前で照合する代わりに *FileChanged* (ページ 535) を使用してください。PreToolUse とは異なり、Claude Code は FileChanged フックを変更の後に実行し、判定制御もないため、書き込みをブロックすることはできません。

警告:

PreToolUse は、Claude がツールを呼び出したときにのみ実行されます。**プロンプト内で @ を使って参照した** (ページ 111) ファイルは、ツール呼び出しなしで追加されます。Claude Code はプロンプトを組み立てる際にその内容を挿入するため、`Read` に一致するフックを含め、PreToolUse フックは発火しません。特定のパスを `@` 参照からブロックするには、代わりに *Read* **の拒否ルール** (ページ 1051) を使用してください。
PreToolUse は `EndConversation` でも発火しません。

PreToolUse **の判定制御** (ページ 502) を使用して、ツール呼び出しを許可、拒否、確認、または延期します。

`PreToolUse` 上の Agent SDK コールバックフックがタイムアウトを超えると、ツール呼び出しがブロックされ、Claude はタイムアウトを示すエラー結果を受け取ります。他のフックが返した明示的な拒否は引き続き優先されます。

PreToolUse の入力

共通の入力フィールド (ページ 466) に加えて、PreToolUse フックは `tool_name`、`tool_input`、`tool_use_id` を受け取ります。

MCP **ツール** (ページ 454) の場合、入力には `mcp_server` も含まれます。これは、サーバーの `name` と、サーバーの定義がどこから来たかを示す `source` を持つオブジェクトです。`source` の値には、`plugin`、`sdk`、および `user` や `project` などの設定スコープが含まれます。Agent SDK リファレンスの `McpServerProvenance` にすべての値が記載されており、認識できない値の扱い方も説明されています。信頼の判断は、`name` や `mcp__<server>__` というツール名のプレフィックスではなく、`source` に基づいて行ってください。`mcp_server` フィールドには Claude Code v2.1.274 以降が必要です。

ファイルツールの `Write`、`Edit`、`Read` では、`tool_input.file_path` は常に絶対パスです。

- Claude Code はフックの実行前に `~` と相対パスを展開するため、パスに対して照合するフックを、`~` や同じパスの相対表記で回避することはできません
- Windows では、フックが `$PWD` が `/c/project` のように見える Git Bash で実行される場合でも、パスはバックスラッシュ区切りで届きます
- `/src/` のチェックのようにスラッシュで記述した比較はバックスラッシュのパスに決して一致せず、ツール呼び出しはフックがブロックするものがなかったかのように続行されます

- 比較の前に区切り文字を正規化してください。Bash では `FILE_PATH="${FILE_PATH//\\//}"`、Python では `file_path.replace("\\", "/")` を使用します。その後、パスは絶対パスなので、`^` で固定するのではなく `/src/` のようなパスセグメントで照合してください

Windows での `Write` 呼び出しでは、次のように届きます。

```
{
  "hook_event_name": "PreToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "C:\\project\\src\\index.ts",
    "content": "..."
  },
  ...
}
```

`tool_input` のフィールドはツールによって異なります。

Bash

シェルコマンドを実行します。

フィールド	型	例	説明
`command`	string	`"npm test"`	実行するシェルコマンド
`description`	string	`"Run test suite"`	コマンドの動作についての説明（省略可）
`timeout`	number	`120000`	タイムアウト（ミリ秒、省略可）。最大値を超える値は拒否されず、最大値に切り下げられます
`run_in_background`	boolean	`false`	コマンドをバックグラウンドで実行するかどうか

Bash コマンドが Git リポジトリ内のファイルを変更すると、Claude Code は変更内容を記録できます。`bashEditDiffEnabled` 設定で記録がオンになっている場合は、すべての権限モードで変更を記録します。どのファイルでこの設定を行えるかは、その設定の項目に記載されています。それ以外の場合は、auto モードと `bypassPermissions` モードで、かつ Claude Code が Claude に Bash 経由でファイルを編集するよう指示した場合にのみ記録します。記録をオフにするには、`bashEditDiffEnabled` を `false` に設定します。バックグラウンドのコマンドと読み取り専用のコマンドには差分は付きません。

その後、*PostToolUse* **フック** (ページ 509) は `tool_response.bashEditDiff` で変更されたファイルを受け取ります。この一覧は、コマンドの実行中にリポジトリ配下で変更されたものを対象とします。Git が無視するファイルとサブモジュール内のファイルは含まれません。Claude Code v2.1.269 以降が必要です。

注釈:

この一覧はベストエフォートであり、パブリックベータ版です。Claude Code は変更を見落としたり、同時に別のプロセスが変更したファイルを含めたり、サイズの上限で打ち切ったりすることがあります。フィールドの形式は変更される可能性があります。この一覧はレビュー対象を見つけるために使用し、ポリシーの強制には使用しないでください。

`changedFiles` と `files` はコマンドが変更したものを一覧にし、残りのフィールドはその一覧がどれだけ完全で信頼できるかを示します。

フィールド	型	例	説明
`changedFiles`	array	`["/path/to/src/app.ts"]`	コマンドが変更したファイルの絶対パス (最大 200 件)。`files` に差分がある場合、または `moreFiles` が 0 より大きい場合に常に含まれます
`files`	array	`[{"filePath": "/path/to/src/app.ts", "hunks": [...]}]`	表示用の、最大 5 つの変更ファイルの差分。コマンドが追加または削除したファイルでは `created` または `deleted` が `true` になります
`moreFiles`	number	`2`	`files` に差分が含まれていない変更ファイルの数
`unavailable`	boolean	`true`	差分が不完全な場合、または取得できなかった場合に設定されます
`skipped`	boolean	`true`	`git checkout` や `git stash` など、作業ツリーを移動させる Git コマンドの場合に設定され、Claude Code は差分を取得しません
`shared`	boolean	`true`	サブエージェントのものなど、別の Bash ツール呼び出しが同じリポジトリで同時に実行された場合に設定されます。一覧の変更の一部はそのコマンドによるものである可能性があります

PowerShell

PowerShell コマンドを実行します。プラットフォームごとの利用可否については、PowerShell ツールを参照してください。

フィールドは Bash ツールと同じで、コマンド文字列は `command` に入ります。

フィールド	型	例	説明
`command`	string	`"Get-ChildItem -Recurse"`	実行する PowerShell コマンド

次のページに続く

表 28 – 前のページからの続き

フィールド	型	例	説明
description	string	"List files recursively"	コマンドの動作についての説明（省略可）
timeout	number	120000	タイムアウト（ミリ秒、省略可）
run_in_background	boolean	false	コマンドをバックグラウンドで実行するかどうか

シェルコマンドを検査するフックでは、両方のツールをカバーするように Bash|PowerShell で照合してください。

- Windows では、PowerShell ツールが有効になっている環境であればどこでも、Claude は PowerShell をプライマリシェルとして扱い、シェルコマンドをそれ経由で実行します。
- Git Bash のない Windows では、このツールが自動的に有効になり、Claude Code は Bash ツールをまったく登録しません。
- Bash のみに一致するフックは、その環境では決して発火しません。

Write

ファイルを作成または上書きします。

フィールド	型	例	説明
file_path	string	"/path/to/file.txt"	書き込むファイルの絶対パス
content	string	"file content"	ファイルに書き込む内容

Edit

既存のファイル内の文字列を置換します。

フィールド	型	例	説明
file_path	string	"/path/to/file.txt"	編集するファイルの絶対パス
old_string	string	"original text"	検索して置換するテキスト
new_string	string	"replacement text"	置換後のテキスト
replace_all	boolean	false	すべての出現箇所を置換するかどうか

Read

ファイルの内容を読み取ります。

フィールド	型	例	説明
file_path	string	"/path/to/file.txt"	読み取るファイルの絶対パス

次のページに続く

表 31 – 前のページからの続き

フィールド	型	例	説明
offset	number	10	読み取りを開始する行番号（省略可）
limit	number	50	読み取る行数（省略可）

Glob

glob パターンに一致するファイルを検索します。

フィールド	型	例	説明
pattern	string	"**/*.ts"	ファイルを照合する glob パターン
path	string	"/path/to/dir"	検索するディレクトリ（省略可）。デフォルトは現在の作業ディレクトリです

Grep

正規表現でファイルの内容を検索します。

フィールド	型	例	説明
pattern	string	"TODO.*fix"	検索する正規表現パターン
path	string	"/path/to/dir"	検索するファイルまたはディレクトリ（省略可）
glob	string	"*.ts"	ファイルを絞り込む glob パターン（省略可）
output_mode	string	"content"	"content"、"files_with_matches"、または "count"。デフォルトは "files_with_matches" です
-i	boolean	true	大文字と小文字を区別しない検索
multiline	boolean	false	複数行のマッチングを有効にする

WebFetch

Web コンテンツを取得して処理します。

フィールド	型	例	説明
url	string	"https://example.com/api"	コンテンツを取得する URL
prompt	string	"Extract the API endpoints"	取得したコンテンツに対して実行するプロンプト

WebSearch

Web を検索します。

フィールド	型	例	説明
query	string	"react hooks best practices"	検索クエリ
allowed_domains	array	["docs.example.com"]	省略可：これらのドメインの結果のみを含める
blocked_domains	array	["spam.example.com"]	省略可：これらのドメインの結果を除外する

Agent

サブエージェント (ページ 317) を起動します。

フィールド	型	例	説明
prompt	string	"Find all API endpoints"	エージェントが実行するタスク
description	string	"Find API endpoints"	タスクの短い説明
subagent_type	string	"Explore"	使用する特化型エージェントの種類
model	string	"sonnet"	デフォルトを上書きするモデルエイリアス（省略可）

フォアグラウンドの Agent 呼び出しが完了すると、*PostToolUse* **フック** (ページ 509) は tool_response でサブエージェントの結果と実行のテレメトリを受け取ります。実行を調べるにはこれらのフィールドを読んでください。totalTokens と usage は最後のリクエストのみを対象とするため、サブエージェント全体のトークンとコストの集計には、query_source が "subagent" で絞り込んだ**トークンとコストのカウンター** (ページ 961) を使用してください。

フィールド	型	例	説明
status	string	"completed"	フォアグラウンドのサブエージェントでは "completed"、バックグラウンドのサブエージェントでは "async_launched"。サブエージェントはデフォルトでバックグラウンドで実行されるため、run_in_background を省略した Agent 呼び出しも "async_launched" になります
agentId	string	"a4d2c8f1e0b3a297"	サブエージェントの実行の識別子
content	array	[{"type": "text", "text": "Found 12 endpoints..."}]	サブエージェントの最終的なテキストブロック。レポートが SubagentHandback を経由するサブエージェントの場合は、代わりにその引き渡しについての短い注記

次のページに続く

表 37 – 前のページからの続き

フィールド	型	例	説明
resolvedModel	string	"claude-sonnet-4-5"	サブエージェントが開始時に使用したモデル。要求されたモデルと異なる場合があります
modelsUsed	array	["claude-sonnet-4-5", "claude-haiku-4-5"]	使用されたモデルの順序（連続する重複はまとめられます）。実行中にモデルが切り替えられた場合にのみ設定されます。Claude Code v2.1.212 以降が必要です
totalTokens	number	12450	サブエージェントの最後の API リクエストのトークン数（入力、出力、キャッシュのトークンの合計）。実行全体の合計ではありません
totalDurationMs	number	48211	サブエージェントの実行にかかった実時間
totalToolUseCount	number	7	サブエージェントが行ったツール呼び出しの数
usage	object	{"input_tokens": 8320, ...}	最後の API リクエストの種類別トークン内訳：input_tokens、output_tokens、cache_creation_input_tokens、cache_read_input_tokens

Claude Code v2.1.271 以降では、auto モードで Claude Code が提供する SubagentHandback ツールを使って実行されるサブエージェントは、レポートをテキストとして返すのではなく、そのツールを通じて届けます。その場合、completed 結果の content フィールドには、レポート自体ではなく、その引き渡しについての短い注記が含まれます。レポートを読むには、SubagentHandback に一致する PreToolUse または PostToolUse フックを設定し、tool_input.message を読み取ってください。

バックグラウンドのサブエージェントの場合、ツールはタスクがバックグラウンドに移った時点で返るため、tool_response には使用量のフィールドが含まれません。バックグラウンドでの起動はすぐに返り、Claude Code が実行中にバックグラウンドへ移したフォアグラウンドのタスクはその移行の時点で返ります。status: "async_launched"、agentId、description、prompt、outputFile、resolvedModel を持ちます。

completed の応答では、resolvedModel はサブエージェントが開始時に使用したモデルを示します。これは、availableModels や他の上書きが適用される場合など、tool_input の model の値と異なることがあります。async_launched の応答では、resolvedModel はエージェントがバックグラウンドに移った時点で使用していたモデルを示すため、バックグラウンドに移る前に行われた切り替えがそこに反映されます。modelsUsed と、バックグラウンド移行時の resolvedModel の動作には Claude Code v2.1.212 以降が必要です。

AskUserQuestion

ユーザーに 1〜4 個の多肢選択式の質問をします。

フィールド	型	例	説明
questions	array	[{"question": "Which framework?", "header": "Framework", "options": [{"label": "React"}], "multiSelect": false}]	提示する質問。それぞれ question 文字列、短い header、options 配列、省略可能な multiSelect フラグを持ちます
answers	object	{"Which framework?": "React"}	省略可。質問のテキストを選択されたオプションのラベルに対応付けます。複数選択の回答は、ラベルをカンマで連結します。Claude はこのフィールドを設定しません。プログラムで回答するには updatedInput 経由で指定してください

ExitPlanMode

Claude が plan モードを終了する前に、計画を提示してユーザーに承認を求めます。Claude はツールを呼び出す前に計画をディスク上のファイルに書き込むため、モデルからの tool_input そのものは通常空です。Claude Code は、入力をフックに渡す前に計画の内容とファイルパスを挿入します。

フィールド	型	例	説明
plan	string	"## Refactor auth\n1. Extract..."	Markdown 形式の計画の内容。ディスク上の計画ファイルから挿入されます
planFilePath	string	"/Users/.../plans/refactor-auth.md"	計画ファイルのパス。挿入されます
allowedPrompts	array	[{"tool": "Bash", "prompt": "run tests"}]	非推奨。Claude Code はこのフィールドを受け付けますが、無視します。v2.1.205 より前は、計画を実行するために Claude が要求したプロンプトベースの権限を保持していました

PostToolUse では、tool_response は承認された計画を保持する plan と filePath のフィールドに加えて、内部のステータスフラグを持つオブジェクトです。計画の内容は、ディスクからファイルを読み直すのではなく、tool_response.plan から読み取ってください。

PreToolUse の判定制御

PreToolUse フックは、ツール呼び出しを続行するかどうかを制御できます。トップレベルの decision フィールドを使用する他のフックとは異なり、PreToolUse は hookSpecificOutput オブジェクト内で判定を返します。これにより、より細かな制御が可能になります。4 つの結果（許可、拒否、確認、延期）に加えて、実行前にツールの入力を変更できます。

フィールド	説明
permissionDecision	"allow" は権限プロンプトをスキップします。ただし、どのモードでも自動承認されないアクションと、*updatedInput* **との組み合わせ** (ページ 504) が必要な AskUserQuestion と ExitPlanMode は除きます。"deny" はツール呼び出しを防ぎます。"ask" はユーザーに確認を求めます。"defer" は、ツールを後で再開できるように正常に終了します。フックが何を返しても、**拒否ルールと確認ルール** (ページ 1041) は引き続き評価されます
permissionDecisionReason	"ask" の場合、権限プロンプトでユーザーに表示されます。誰もそのプロンプトに応答できない -p の実行で Claude Code が**呼び出しを拒否する** (ページ 705) 場合は、代わりに Claude がツール結果でその理由を読みます。"deny" の場合、Claude に表示されます。"allow" と "defer" の場合、**デバッグログ** (ページ 562) にのみ書き込まれます
updatedInput	実行前にツールの入力パラメーターを変更します。入力オブジェクト全体を置き換えるため、変更したフィールドとともに変更していないフィールドも含めてください。Claude Code は、権限ルールと Bash コマンドの自動バックグラウンド化の対象かどうかを、Claude が送信した入力ではなく、フックが返した入力に対して評価します。自動承認するには "allow" と、変更した入力をユーザーに表示するには "ask" と組み合わせます。"defer" の場合は無視されます
additionalContext	ツールの結果とともに Claude のコンテキストに追加される文字列。permissionDecision が "defer" の場合は無視されます。*Claude* **にコンテキストを追加する** (ページ 476) を参照してください

複数の PreToolUse フックが異なる判定を返した場合、優先順位は deny > defer > ask > allow です。

終了コード 2 で終了してブロックするフックは、"deny" と同じ経路をたどります。Claude は stderr のメッセージを拒否の理由として受け取ります。

フックが "ask" を返すと、ユーザーに表示される権限プロンプトには、フックの出どころを示すラベルが含まれます。任意の設定ファイルまたはエージェントのフロントマターからのフックでは [settings]、プラグインのフックでは [plugin:<name>]、スキルのフロントマターからのフックでは [skill] です。これにより、どの設定ソースが確認を求めているかをユーザーが理解しやすくなります。

フックの "ask" は、auto モードでも権限プロンプトを強制します。分類器は引き続きツール呼び出しを拒否できますが、呼び出しを黙って承認することはできません。v2.1.211 より前は、分類器は**サンドボックス** (ページ 1069) 外で実行される Bash コマンドを、フックが要求したプロンプトを表示せずに承認できました。その場合も分類器はそのコマンドに独自の安全ルールを適用しており、フックの "deny" は常に尊重されていました。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreToolUse",
```

(次のページに続く)

(前のページからの続き)

```
    "permissionDecision": "allow",
    "permissionDecisionReason": "My reason here",
    "updatedInput": {
      "field_to_modify": "new value"
    },
    "additionalContext": "Current environment: production. Proceed with caution."
  }
}
```

-p フラグを使った**非対話モード** (ページ 695) では、Claude Code は、Agent SDK の canUseTool コールバックなど、プロンプトを受け取る**権限ホスト** (ページ 705) が実行にある場合にのみ、AskUserQuestion と ExitPlanMode を提供します。これらのツールにはユーザーの操作が必要です。permissionDecision: "allow" を updatedInput とともに返すと、その要件を満たせます。フックは stdin からツールの入力を読み取り、独自の UI で回答を収集し、それを updatedInput で返すことで、ツールはプロンプトなしで実行されます。これらのツールでは、"allow" だけを返しても十分ではありません。AskUserQuestion の場合は、元の questions 配列をそのまま返し、各質問のテキストを選択された回答に対応付ける *answers* (ページ 502) オブジェクトを追加してください。

サーバーが *_meta["anthropic/requiresUserInteraction"]* (ページ 404) でマークした MCP ツールはさらに厳格です。フックは updatedInput の有無にかかわらず、"allow" でその承認プロンプトをスキップすることはできません。ツールが必要とする操作をフックが収集したことを Claude Code が確認できないためです。

注釈:

PreToolUse は以前はトップレベルの decision と reason フィールドを使用していましたが、このイベントではこれらは非推奨です。代わりに hookSpecificOutput.permissionDecision と hookSpecificOutput.permissionDecisionReason を使用してください。非推奨の値 "approve" と "block" は、それぞれ "allow" と "deny" に対応します。PostToolUse や Stop などの他のイベントでは、現在の形式としてトップレベルの decision と reason を引き続き使用します。

ツール呼び出しを後で実行するために延期する

"defer" は、Agent SDK アプリや Claude Code 上に構築したカスタム UI など、claude -p をサブプロセスとして実行し、その JSON 出力を読み取るインテグレーション向けです。これにより、呼び出し元のプロセスは Claude をツール呼び出しの時点で一時停止し、独自のインターフェースで入力を収集して、中断した場所から再開できます。Claude Code がこの値を尊重するのは、-p フラグを使った**非対話モード** (ページ 695) のみです。対話セッションでは警告をログに記録し、フックの結果を無視します。

典型的なケースは AskUserQuestion ツールです。Claude はユーザーに何かを尋ねたいのに、回答するためのターミナルがありません。-p の実行では、--permission-prompt-tool で渡す MCP ツールなどの**権限ホスト** (ページ 705) がある場合にのみ AskUserQuestion が提供されるため、権限ホストを指定して実行を開始してください。往復の流れは次のとおりです。

1. Claude が AskUserQuestion を呼び出します。PreToolUse フックが発火します。
2. フックが permissionDecision: "defer" を返します。ツールは実行されません。プロセスは stop_reason: "tool_deferred" で終了し、保留中のツール呼び出しはトランスクリプトに保存されます。

3. 呼び出し元のプロセスは SDK の結果から deferred_tool_use を読み取り、独自の UI で質問を表示して回答を待ちます。
4. 呼び出し元のプロセスは、同じ権限ホストを指定して claude -p --resume <session-id> を実行します。同じツール呼び出しで再び PreToolUse が発火します。
5. フックは updatedInput に回答を入れて permissionDecision: "allow" を返します。ツールが実行され、Claude は処理を続けます。

deferred_tool_use フィールドには、ツールの id、name、input が含まれます。input は、Claude がツール呼び出しのために生成したパラメーターで、実行前に取得されたものです。

```
{
  "type": "result",
  "subtype": "success",
  "stop_reason": "tool_deferred",
  "session_id": "abc123",
  "deferred_tool_use": {
    "id": "toolu_01abc",
    "name": "AskUserQuestion",
    "input": { "questions": [{ "question": "Which framework?", "header": "Framework",
↪"options": [{"label": "React"}, {"label": "Vue"}], "multiSelect": false }] }
  }
}
```

タイムアウトや再試行の上限はありません。セッションは再開するまでディスク上に残りますが、cleanupPeriodDays の保持期間による削除の対象となります。この削除は、保持期間の削除ルールに従い、デフォルトでは 30 日後にセッションファイルを削除します。再開時に回答の準備ができていない場合、フックは再び "defer" を返すことができ、プロセスは同じ方法で終了します。呼び出し元のプロセスは、最終的にフックから "allow" または "deny" を返すことで、いつループを抜けるかを制御します。

"defer" は、Claude がそのターンで単一のツール呼び出しを行う場合にのみ機能します。Claude が複数のツール呼び出しを一度に行う場合、"defer" は警告とともに無視され、ツールは通常の権限フローで処理されます。この制約は、再開時に再実行できるツールが 1 つだけだからです。バッチ内の 1 つの呼び出しだけを延期すると、他の呼び出しが未解決のまま残ってしまいます。

再開時に延期されたツールが利用できなくなっている場合、プロセスはフックが発火する前に stop_reason: "tool_deferred_unavailable" と is_error: true で終了します。これは、ツールを提供していた MCP サーバーが再開されたセッションで接続されていない場合に発生します。deferred_tool_use ペイロードは引き続き含まれるため、どのツールが失われたかを特定できます。

注釈:

延期されたセッションを plan モードで再開するには、Claude Code が承認のために計画を提示できるよう、`--resume` とともに *`--permission-prompt-tool`* (ページ 51) を渡してください。特定の他の起動フラグを渡すと、再開された実行は plan モードに戻りません。-p で plan モードで再開するを参照してください。Claude Code v2.1.246 以降が必要です。
-p で再開する場合、Claude Code は他の保存された権限モードを復元しません。新しい `claude -p` の実行が開始する権限モードで実行を開始するため、延期されたセッションで `--permission-mode` や `--dangerously-skip-permissions` を使用していた場合は、再度渡してください。-p なしで `claude --resume <session-id>` を使って再開する場合、Claude Code は保存された権限モードを復元します。例外は再開時の権限モードに記載されています。

21.4.8 PermissionRequest

Claude Code がツールを使用するための権限をユーザーに求めようとしているときに実行されます。**非対話モード** (ページ 695) のバックグラウンドのサブエージェントなど、プロンプトを表示できないセッションでも、Claude Code はこれらのフックを実行し、どのフックも判定を返さない場合はツール呼び出しを拒否します。`--permission-prompt-tool` または Agent SDK の canUseTool コールバックに到達する呼び出しの場合、フックはホストと並行して実行され、先に判定したほうが適用されます。 ***PermissionRequest* の判定制御** (ページ 507) を使用して、ユーザーに代わって許可または拒否します。

Claude がツールを使用するための権限を求めた瞬間にシグナルが必要な場合に、このイベントを使用します。Claude Code が `permission_prompt` タイプの *Notification* (ページ 518) フックを実行するのは、プロンプトが約 6 秒間待機した後です。

Claude Code は、サンドボックス化されたコマンドの**ネットワークリクエスト** (ページ 1087) に対しては PermissionRequest フックを実行しません。そのプロンプトのシグナルを得るには、`permission_prompt` 通知タイプを使用してください。

ツール名に対して照合し、値は PreToolUse と同じです。

PermissionRequest の入力

PermissionRequest フックは、PreToolUse フックと同様に `tool_name` と `tool_input` フィールドを受け取りますが、`tool_use_id` は含まれません。MCP ツールの場合は、*`mcp_server`* (ページ 495) オブジェクトも受け取ります。省略可能な `permission_suggestions` 配列には、許可ルールの追加や権限モードの変更など、このリクエストに対して Claude Code が提案する**権限の更新** (ページ 508) が含まれます。

各権限ダイアログは独自のオプションを構築するため、`permission_suggestions` 配列は表示されるオプションの正確な一覧ではありません。ファイル編集用のダイアログなど、一部のダイアログはこの配列をまったく読み取らず、リクエスト自体からオプションを導き出します。配列を読み取るダイアログでも、提案が配列に残っているオプションを表示しないことがあります。たとえば、`allowManagedPermissionRulesOnly` がルールを保存するオプションを非表示にする場合です。また、Yes, and switch to auto mode のように、提案エントリを持たないオプションを提示することもあります。このオプションは、権限の更新を介さずに権限モードを直接変更します。

PreToolUse フックは、権限が必要かどうかにかかわらず、すべてのツール呼び出しの前に実行されます。PermissionRequest フックは、Claude Code が権限をユーザーに求めようとしているとき、またはプロンプ

トを表示できない呼び出しを本来なら自動拒否するときにのみ実行されます。どちらのイベントも End Conversation では発火しません。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PermissionRequest",
  "tool_name": "Bash",
  "tool_input": {
    "command": "rm -rf node_modules",
    "description": "Remove node_modules directory"
  },
  "permission_suggestions": [
    {
      "type": "addRules",
      "rules": [{ "toolName": "Bash", "ruleContent": "rm -rf node_modules" }],
      "behavior": "allow",
      "destination": "localSettings"
    }
  ]
}
```

PermissionRequest の判定制御

PermissionRequest フックは、権限リクエストを許可または拒否できます。すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、フックスクリプトは以下のイベント固有のフィールドを持つ decision オブジェクトを返すことができます。

フィールド	説明
behavior	"allow" は権限を付与し、"deny" は拒否します。**拒否ルールと確認ルール** (ページ 1041) は引き続き評価されるため、"allow" を返すフックが一致する拒否ルールを上書きすることはありません
updatedInput	"allow" の場合のみ：実行前にツールの入力パラメーターを変更します。入力オブジェクト全体を置き換えるため、変更したフィールドとともに変更していないフィールドも含めてください。変更された入力は、拒否ルールと確認ルールに対して再評価されます
updatedPermissions	"allow" の場合のみ：適用する**権限の更新エントリ** (ページ 508) の配列。許可ルールの追加やセッションの権限モードの変更などです
message	"deny" の場合のみ：権限が拒否された理由を Claude に伝えます
interrupt	"deny" の場合のみ：true の場合、Claude を停止します

decision オブジェクトなしで終了コード 2 で終了するフックは権限フローを変更せず、その stderr は破棄

されます。リクエストを許可または拒否できるのは decision オブジェクトのみです。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionRequest",
    "decision": {
      "behavior": "allow",
      "updatedInput": {
        "command": "npm run lint"
      }
    }
  }
}
```

権限の更新エントリ

updatedPermissions 出力フィールドと *permission_suggestions* **入力フィールド** (ページ 506) は、どちらも同じエントリオブジェクトの配列を使用します。各エントリには、他のフィールドを決定する type と、変更の書き込み先を制御する destination があります。

type	フィールド	効果
addRules	rules、behavior、destination	権限ルールを追加します。rules は {toolName, ruleContent?} オブジェクトの配列です。ツール全体に一致させるには ruleContent を省略します。behavior は "allow"、"deny"、または "ask" です
replaceRules	rules、behavior、destination	destination にある指定の behavior のすべてのルールを、指定した rules で置き換えます
removeRules	rules、behavior、destination	指定の behavior の一致するルールを削除します
setMode	mode、destination	権限モードを変更します。有効なモードは default、auto、acceptEdits、dontAsk、bypassPermissions、plan、および default のエイリアスとしての manual です。manual エイリアスには Claude Code v2.1.200 以降が必要です
addDirectories	directories、destination	作業ディレクトリを追加します。directories はパス文字列の配列です
removeDirectories	directories、destination	作業ディレクトリを削除します

注釈:

bypassPermissions を指定した setMode は、バイパスモードがすでに利用可能な状態でセッションを起動した場合にのみ有効になります。バイパスモードを利用可能にするには、--dangerously-skip-permissions、--permission-mode bypassPermissions、--allow-dangerously-skip-permissions のいずれかを使用するか、ユーザー設定、--settings、または管理設定で permissions.defaultMode: "bypassPermissions" を指定します。それ以外の場合、この更新は何も行いません。また、*permissions.disableBypassPermissionsMode* (ページ 1063) によってこのモードが無効化されている場合や、セッションが *restricted* **モード** (ページ 51) で開始された場合も、この更新は何も行いません。
bypassPermissions は、destination に関係なく defaultMode として永続化されることはありません。

各エントリの destination フィールドは、変更をメモリ内にとどめるか、設定ファイルに永続化するかを決定します。

destination	書き込み先
session	メモリ内のみ。セッション終了時に破棄されます
localSettings	.claude/settings.local.json
projectSettings	.claude/settings.json
userSettings	~/.claude/settings.json

フックは、受け取った permission_suggestions のいずれかを、そのまま自身の updatedPermissions 出力として返すことができます。

21.4.9 PostToolUse

ツールが正常に完了した直後に実行されます。

ツール名でマッチします。値は PreToolUse と同じです。

ツール名が適切なフィルターにならない場合は、より広くマッチさせます。

- いずれかのツールが正常に完了した後にフックを実行するには、matcher を省略するか "*" に設定します。フック側で何が変更されたかを自ら調べることができます。たとえば git status --porcelain を実行すると、git diff では見落とされる未追跡ファイルも一覧表示されます。失敗したツール呼び出しについては、同じフックを *PostToolUseFailure* (ページ 513) の下に追加します。
- 書き込んだのが何であれ、特定のファイルがディスク上で変更されたときにフックを実行するには、*FileChanged* (ページ 535) を使用します。Bash コマンドや Claude Code 外部のプロセスが同じファイルを書き換えた場合、Claude Code は Edit|Write にマッチする PostToolUse フックを実行しません。

PostToolUse の入力

PostToolUse フックは、ツールがすでに正常に実行された後に発火します。入力には、ツールに送られた引数である tool_input と、ツールが返した結果である tool_response の両方が含まれます。両者の正確なスキーマはツールによって異なります。ファイルツールの tool_input のパスは *PreToolUse* (ページ 495) と同じ形式で渡されます。つまり、常に絶対パスで、プラットフォーム固有の区切り文字が使われるため、Windows ではバックスラッシュになります。MCP ツールの場合、入力には *mcp_server* (ページ 495) オブジェクトも含まれます。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PostToolUse",
  "tool_name": "Write",
  "tool_input": {
    "file_path": "/path/to/file.txt",
    "content": "file content"
  },
  "tool_response": {
    "filePath": "/path/to/file.txt",
    "type": "create"
  },
  "tool_use_id": "toolu_01ABC123...",
  "duration_ms": 12
}
```

フィールド	説明
duration_ms	省略可能。ツールの実行時間 (ミリ秒)。権限プロンプトと PreToolUse フックに費やされた時間は含みません

PostToolUse の判定制御

PostToolUse フックは、ツール実行後に Claude へフィードバックを提供できます。すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、フックスクリプトは次のイベント固有フィールドを返すことができます。

フィールド	説明
decision	"block" を指定すると、ツール結果の横に reason が追加されます。Claude には元の出力も引き続き表示されます。出力を置き換えるには updatedToolOutput を使用します
reason	decision が "block" のときに Claude に示される説明

次のページに続く

表 45 – 前のページからの続き

フィールド	説明
additionalContext	ツール結果とともに Claude のコンテキストに追加される文字列。*Claude* **にコンテキストを追加する** (ページ 476) を参照してください
classifierContext	この呼び出しの結果について、Claude ではなく auto モードの分類器に向けた短い注記。*auto* **モードの分類器向けに結果に注記を付ける** (ページ 512) を参照してください。Claude Code v2.1.236 以降が必要です
updatedToolOutput	Claude に送られる前に、ツールの出力を指定した値で置き換えます。値はツールの出力の形式と一致している必要があります
updatedMCPToolOutput	*MCP* **ツール** (ページ 454) に限り出力を置き換えます。すべてのツールで機能する updatedToolOutput の使用を推奨します

次の例は、Bash 呼び出しの出力を置き換えます。置き換える値は Bash ツールの出力の形式に一致しています。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "additionalContext": "Additional information for Claude",
    "updatedToolOutput": {
      "stdout": "[redacted]",
      "stderr": "",
      "interrupted": false,
      "isImage": false
    }
  }
}
```

警告:

updatedToolOutput が変更するのは Claude に見える内容だけです。フックが発火する時点でツールはすでに実行されているため、書き込まれたファイル、実行されたコマンド、送信されたネットワークリクエストはすでに反映されています。OpenTelemetry のツールスパンや分析イベントなどのテレメトリも、フックが実行される前の元の出力を記録します。ツール呼び出しを実行前に阻止または変更するには、代わりに *PreToolUse* (ページ 495) フックを使用してください。
置き換える値はツールの出力の形式と一致している必要があります。組み込みツールはプレーンな文字列ではなく構造化されたオブジェクトを返します。たとえば Bash は、stdout、stderr、interrupted、isImage フィールドを持つオブジェクトを返します。組み込みツールの場合、ツールの出力スキーマに一致しない値は無視され、元の出力が使用されます。MCP ツールの出力はスキーマ検証なしでそのまま渡されます。Claude が必要とするエラーの詳細を取り除くと、Claude が誤った前提のまま作業を進める可能性があります。

auto モードの分類器向けに結果に注記を付ける

`classifierContext` を返すと、ツール呼び出しの結果に関する短い注記を、Claude ではなく auto モードの分類器に送ることができます。分類器はツール結果そのものを受け取ることはないため、後続のアクションを審査する前に、呼び出しが何を返したかについて分類器に伝えるには、このフィールドを使うのがサポートされた方法です。このフィールドには Claude Code v2.1.236 以降が必要です。

次の例は、クエリの出力がどこから得られたかを分類器に伝えます。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUse",
    "classifierContext": "This query ran against the staging database, not production.
↪"
  }
}
```

分類器が注記をどの程度重視するかは、フックをどこで設定したかによって異なります。

- **Claude Code で設定されたフック**: 設定ファイル、プラグイン、スキル、エージェントのフロントマターから読み込まれたフックの場合、分類器は注記を未検証の、アプリケーションから提供されたコンテキストとして扱います。注記がユーザーの意図を確立することはなく、ユーザーが何かを承認または要求したと注記が主張する場合、分類器はその主張を会話内のユーザー自身のメッセージと照合します
- **インプロセスの Agent SDK コールバック**: Claude Code を組み込んだアプリケーションがフックを TypeScript SDK コールバックとして登録し、ライブセッション中に注記を返す場合、分類器は注記で伝えられたユーザーの発言をユーザーの意図として考慮することがあります。そのような発言は、ユーザーが送信したメッセージであれば分類器が受け入れる同意要件を満たすことはありますが、ユーザー自身のメッセージでも解除できないブロックを解除することはありません。セッションが再開された後は、Claude Code は復元された注記を未検証のコンテキストとして扱います。両方のグループのフックが同じ呼び出しに注記を付けた場合、分類器は結合された注記を未検証として扱います

Claude Code は注記を渡す際に次の制限を適用します。

- **長さ**: Claude Code は 1 回のツール呼び出しに対する注記を 2,000 文字までに制限し、残りを切り捨てます。この上限は、その呼び出しに応答するすべてのフックで共有されます
- **同期的な応答のみ**: **バックグラウンドで実行される** (ページ 558) フックの応答では、Claude Code はこのフィールドを無視します。その応答は Claude Code がツール結果を記録した後に届くためです
- **分類器が記録しない呼び出し**: 分類器のトランスクリプトには、ファイルの読み取りや検索などの読み取り専用の参照は含まれません。Claude Code は、そのような呼び出しに付けられた注記を破棄します
- **書き換えとの相互作用**: `updatedToolOutput` で置き換える出力について注記が説明している場合は、同じフックの応答で両方のフィールドを返してください。その書き換えが拒否された場合や、別のフックの書き換えで置き換えられた場合、Claude Code は注記を破棄します。書き換えなしで返した注記は、別のフックが出力を書き換えた場合でも Claude Code によって渡されます

警告:

分類器は `classifierContext` に入れた内容を、セッションをホストしているアプリケーションからの情報として読み取ります。そのため、信頼できないツール出力やサードパーティのテキストをコピーして入れないでください。注記は、その出所に関する事実やそれについてのユーザーの発言など、この 1 回の呼び出しについての短い主張にとどめてください。無関係なメッセージやイベントのストリームを渡すためにこのフィールドを使用しないでください。

21.4.10 PostToolUseFailure

実行を開始したツールが失敗したとき、つまりツールがエラーをスローしたか、MCP ツールがエラー結果を返したときに実行されます。失敗のログ記録、アラートの送信、Claude への修正フィードバックの提供に使用します。

ツール名でマッチします。値は PreToolUse と同じです。

注釈:

このイベントは、実行前に拒否されたツール呼び出しでは発火しません。該当するのは、不明なツール名、スキーマ検証やツール固有の検証に失敗した入力、権限の拒否です。検証による拒否は `tool_use_error` 結果として返され、フックが実行される前に発生するため、`PreToolUse` も `PostToolUseFailure` も発火しません。権限の拒否では `PreToolUse` は発火しますが、このイベントは発火しません。*PermissionDenied* (ページ 516) を参照してください。

PostToolUseFailure の入力

PostToolUseFailure フックは、PostToolUse と同じ `tool_name` と `tool_input` フィールドに加えて、エラー情報をトップレベルのフィールドとして受け取ります。MCP ツールの場合は、*mcp_server* (ページ 495) オブジェクトも受け取ります。たとえば、失敗した `npm test` コマンドでは次のような入力が渡されます。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PostToolUseFailure",
  "tool_name": "Bash",
  "tool_input": {
    "command": "npm test",
    "description": "Run test suite"
  },
  "tool_use_id": "toolu_01ABC123...",
  "error": "Exit code 1\nError: Cannot find module 'express'",
  "is_interrupt": false,
  "duration_ms": 4187
}
```

フィールド	説明
error	何が問題だったかを説明する文字列。形式は失敗したツールによって異なります
is_interrupt	省略可能なブール値。ツールが報告したエラーとしてではなく、中断として Claude Code に失敗が伝わった場合に true になります。実行中のツールをキャンセルしてもこのフックは発火せず、代わりにツール結果に中断メッセージが含まれます
duration_ms	省略可能。ツールの実行時間 (ミリ秒)。権限プロンプトと PreToolUse フックに費やされた時間は含みません

error 文字列は通常、失敗したツールの結果として Claude が受け取るテキストと同じです。形式はツールと失敗の種類によって異なります。フックの判定には tool_name、is_interrupt、および先頭行の Exit code N を使用し、文字列の残りの部分は安定した形式ではなく表示用テキストとして扱ってください。

- Bash と PowerShell の場合、実行されて終了したコマンドでは、先頭行が Exit code N となり、その後にコマンドが生成した出力が stdout と stderr の混在した 1 つのブロックとして続きます
- Claude Code がシェルプロセス自体を起動できなかった場合、ペイロードには終了コードの行がない失敗メッセージだけが含まれることもあります
- Claude Code は長い文字列を ... [N characters truncated] ... マーカーを挟んで中間部分を切り詰めることがあり、Command timed out after 2m 0s のような独自の行を挿入することもあります

PostToolUseFailure の判定制御

PostToolUseFailure フックは、ツールの失敗後に Claude へコンテキストを提供できます。すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、フックスクリプトは次のイベント固有フィールドを返すことができます。

フィールド	説明
additionalContext	エラーとともに Claude のコンテキストに追加される文字列。*Claude* **にコンテキストを追加する** (ページ 476) を参照してください

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolUseFailure",
    "additionalContext": "Additional information about the failure for Claude"
  }
}
```

21.4.11 PostToolBatch

バッチ内のすべてのツール呼び出しが解決された後、Claude Code が次のリクエストをモデルに送信する前に 1 回実行されます。PostToolUse はツールごとに 1 回発火するため、Claude が並列でツールを呼び出すと同時に発火します。PostToolBatch はバッチ全体に対して正確に 1 回だけ発火するため、単一のツールではなく実行されたツールの組み合わせに依存するコンテキストを注入するのに適しています。このイベントには matcher はありません。

PostToolBatch の入力

共通入力フィールド (ページ 466) に加えて、PostToolBatch フックは、バッチ内のすべてのツール呼び出しを記述する配列 tool_calls を受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "PostToolBatch",
  "tool_calls": [
    {
      "tool_name": "Read",
      "tool_input": {"file_path": "/.../ledger/accounts.py"},
      "tool_use_id": "toolu_01...",
      "tool_response": "1\tfrom __future__ import annotations\n2\t..."
    },
    {
      "tool_name": "Read",
      "tool_input": {"file_path": "/.../ledger/transactions.py"},
      "tool_use_id": "toolu_02...",
      "tool_response": "1\tfrom __future__ import annotations\n2\t..."
    }
  ]
}
```

tool_response には、対応する tool_result ブロックでモデルが受け取るのと同じ内容が含まれます。値は、ツールが出力したとおりのシリアライズされた文字列またはコンテンツブロックの配列です。Read の場合、これは生のファイル内容ではなく、行番号が先頭に付いたテキストを意味します。応答は大きくなる場合があるため、必要なフィールドだけを解析してください。

注釈:

tool_response の形式は PostToolUse のものとは異なります。PostToolUse はツールの構造化された Output オブジェクト（Write の場合は {filePath: "...", type: "create"} など）を渡しますが、PostToolBatch はモデルに見えるシリアライズされた tool_result の内容を渡します。

PostToolBatch の判定制御

PostToolBatch フックは、Claude 向けのコンテキストを注入できます。すべてのフックで利用できる *JSON 出力フィールド* (ページ 474) に加えて、フックスクリプトは次のイベント固有フィールドを返すことができます。

フィールド	説明
additionalContext	次のモデル呼び出しの前に 1 回注入されるコンテキスト文字列。渡され方の詳細、入れるべき内容、再開されたセッションで過去の値がどう扱われるかについては、*Claude にコンテキストを追加する* (ページ 476) を参照してください

```
{
  "hookSpecificOutput": {
    "hookEventName": "PostToolBatch",
    "additionalContext": "These files are part of the ledger module. Run pytest␣
↪before marking the task complete."
  }
}
```

decision: "block" または continue: false を返すと、次のモデル呼び出しの前にエージェント型ループが停止します。ブロックメッセージは、JSON の reason または stopReason、あるいは終了コード 2 の場合は stderr から取得されます。このメッセージはトランスクリプトに警告として表示され、会話にも残るため、会話が続行されると Claude はそれを確認できます。

21.4.12 PermissionDenied

auto モードがツール呼び出しを拒否したときに実行されます。これには、auto モードとは別の安全性チェックが分類器自身のリクエストを拒否した場合や、分類器の応答を解析できなかった場合など、分類器の判定なしで拒否された場合も含まれます。このフックは auto モードでのみ発火します。ユーザーが権限ダイアログを手動で拒否した場合、PreToolUse フックが呼び出しをブロックした場合、deny ルールがマッチした場合には実行されません。拒否のログ記録、設定の調整、またはツール呼び出しを再試行してよいことをモデルに伝えるために使用します。

ツール名でマッチします。値は PreToolUse と同じです。

PermissionDenied の入力

共通入力フィールド (ページ 466) に加えて、PermissionDenied フックは tool_name、tool_input、tool_use_id、reason を受け取ります。MCP ツールの場合は、*mcp_server* (ページ 495) オブジェクトも受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
```

(次のページに続く)

(前のページからの続き)

```
  "cwd": "/Users/...",
  "permission_mode": "auto",
  "hook_event_name": "PermissionDenied",
  "tool_name": "Bash",
  "tool_input": {
    "command": "rm -rf /tmp/build",
    "description": "Clean build directory"
  },
  "tool_use_id": "toolu_01ABC123...",
  "reason": "[Irreversible Local Destruction]"
}
```

フィールド	説明
reason	拒否の理由。分類器の判定による場合、ほとんどのセッションでは、[Data Exfiltration] のように角括弧でマッチしたルールの名前が示されます。その他の形式については 拒否を確認するを参照してください。**判定なしの拒否** (ページ 517) の場合は、Auto mode could not evaluate this action and is blocking it for safety で始まります。分類器モデルが利用できなかったことによる拒否の場合は、固定テキスト Classifier unavailable になります

PermissionDenied の判定制御

PermissionDenied フックは、拒否されたツール呼び出しを再試行してよいことをモデルに伝えることができます。hookSpecificOutput.retry を true に設定した JSON オブジェクトを返します。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PermissionDenied",
    "retry": true
  }
}
```

retry が true の場合、Claude Code は、ツール呼び出しを再試行してよいことをモデルに伝えるメッセージを会話に追加します。Claude Code 自体が拒否を取り消すことはありません。フックが JSON を返さない場合、または retry: false を返した場合、拒否はそのまま維持され、モデルは元の拒否メッセージを受け取ります。

分類器がアクションについて判定を出さなかった場合、つまり分類器の応答を解析できなかった場合や、auto モードとは別の安全性チェックが分類器自身のリクエストを拒否した場合、Claude Code は retry: true を無視します。そのような拒否については、後で再試行するか先に進むかを、Claude Code がすでに拒否メッセージでモデルに伝えています。

21.4.13 Notification

Claude Code が通知を送信するときに実行されます。通知の種類でマッチします。すべての通知の種類でフックを実行するには、matcher を省略します。

デスクトップ通知をオフにしていても、これらのフックイベントは受け取ります。`preferredNotifChannel` 設定（`notifications_disabled` を含む）が変更するのはユーザーへの通知方法だけで、フックが実行されるかどうかは変わりません。

Matcher	発火するタイミング
`permission_prompt`	Claude がツールの使用またはサンドボックス化されたコマンドの**ネットワークリクエスト** (ページ 1087) について承認を必要としており、プロンプトが約 6 秒間待機している
`idle_prompt`	Claude が約 60 秒前に応答を終え、それ以降ユーザーが入力していない
`auth_success`	認証が完了した
`elicitation_dialog`	MCP サーバーが elicitation フォームを開き、ユーザーが約 6 秒間入力していない
`elicitation_url_dialog`	MCP サーバーがブラウザーの URL を開くようユーザーに求め、ユーザーが約 6 秒間入力していない
`elicitation_complete`	MCP サーバーが *URL* **モードの** *elicitation* (ページ 550) の完了を報告した
`elicitation_response`	MCP の elicitation 応答がサーバーに返送された
`agent_needs_input`	ターミナルでエージェントビューが開いている間に、バックグラウンドセッションがユーザーの入力を待ち始めた。また、ターミナルセッションがエージェントチームのチームメイトのターミナル設定に関する質問や、分類器リクエストの料金に関する auto モードの通知を表示し、ユーザーが約 6 秒間入力していない場合にも発火します
`agent_completed`	バックグラウンドセッションが終了または失敗した。ターミナルでエージェントビューが開いている間のみ発火します
`quota_auto_resume_fired`	claude.ai の使用制限によって一時停止したタスクを Claude Code が続行した。続行はリセット時点、または待機中に Claude Code で行った操作（使用クレジットの追加、プランのアップグレード、モデルの切り替えなど）によって使用量が再び利用可能になった場合はそれより早く行われます。ただし**モデル設定の例外** (ページ 100) があります
`quota_auto_resume_stale`	コンピューターが約 30 分を超えてスリープしている間に claude.ai の使用制限がリセットされた。Claude Code は続行せず、ユーザーが `Enter` を押すのを待ちます。スリープがそれより短い場合は続行し、代わりに `quota_auto_resume_fired` を発火します

次のページに続く

表 50 – 前のページからの続き

Matcher	発火するタイミング
quota_auto_resume_disabled	Claude Code がタスクを続行せずに claude.ai の使用制限の待機を終了した。原因は、autoContinueAtUsageLimit がオフにされた、Claude Code が自ら開始した待機中にリセットが 24 時間以上先に移動した、続行したタスクが繰り返し制限に達した、または続行がモデルに届く前にブロックされた、のいずれかです。ユーザーが Esc または Ctrl+C を押した場合や、**Don't continue automatically** を選択した場合は発火しません

quota_auto_resume_fired、quota_auto_resume_stale、quota_auto_resume_disabled の種類には Claude Code v2.1.234 以降が必要です。

ターミナルセッションでは、サンドボックス化されたコマンドのネットワークリクエストに対する permission_prompt には Claude Code v2.1.246 以降が必要です。

チームメイトのターミナル設定に関する質問に対する agent_needs_input には Claude Code v2.1.248 以降が必要です。

注釈:

permission_prompt、idle_prompt、elicitation_dialog、elicitation_url_dialog の種類はデスクトップ通知とタイミングを共有しているため、ターミナルセッションでは、ユーザーがターミナルから離れているとみなされる場合にのみ発生します。

- permission_prompt は、ユーザーが約 6 秒間入力していない時点で発生します。タイマーは権限プロンプトが表示されたときに開始し、キー入力のたびに延期されます。Claude がツールの使用権限を求めたときにすぐにフックを実行するには、代わりに *PermissionRequest* (ページ 506) を使用してください。
- idle_prompt は、Claude が応答を終えてから約 60 秒後に発生します。ただし、それ以降ユーザーが入力しておらず、バックグラウンドの**サブエージェント** (ページ 317) などのバックグラウンドエージェントが実行中でない場合に限ります。claude.ai の使用制限のリセットを待っている間、Claude Code は idle_prompt を送信しません。待機が自然に終了すると、代わりに quota_auto_resume_* の種類のいずれかが発火します。
- elicitation_dialog（elicitation フォームの場合）または elicitation_url_dialog（ブラウザー URL のリクエストの場合）は、ユーザーが約 6 秒間入力していない時点で発生します。どちらも permission_prompt と同じ 6 秒の待機条件を共有しており、タイマーはダイアログが表示されたときに開始し、キー入力のたびに延期されます。

別のダイアログが画面に表示されている間に届いた権限リクエストや elicitation にも、同じ 6 秒の待機条件が適用され、リクエストが届いた時点から計測されます。そのため、リクエストが開いているダイアログの後ろでまだ待機している間に通知が届くことがあります。

Claude Code が権限リクエストを Agent SDK の canUseTool コールバックに送信するセッションでは、permission_prompt のタイミングが異なります。Claude Desktop と VS Code 拡張機能は、この方法で Claude Code をホストしています。

- permission_prompt は、Claude が権限を求めてから約 6 秒後に発生します。入力中でも Claude Code は延期しません。
- それより早くユーザーまたは *PermissionRequest* (ページ 506) フックが応答した場合、Claude Code は permission_prompt を実行しません。
- これらのセッションで permission_prompt をオフにするには、CLAUDE_CODE_DISABLE_PERMISSION_PROMPT_NOTIFY_HOOKS を 1 に設定します。

v2.1.233 より前は、これらのセッションで permission_prompt は発火しませんでした。

通知の種類に応じて異なるハンドラーを実行するには、個別の matcher を使用します。次の設定では、Claude が権限の承認を必要とするときに権限専用のアラートスクリプトを、Claude がアイドル状態になったときに別の通知を実行します。

```
{
  "hooks": {
    "Notification": [
      {
        "matcher": "permission_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/permission-alert.sh"
          }
        ]
      },
      {
        "matcher": "idle_prompt",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/idle-notification.sh"
          }
        ]
      }
    ]
  }
}
```

Notification の入力

共通入力フィールド (ページ 466) に加えて、Notification フックは、通知テキストを含む message、省略可能な title、どの種類が発火したかを示す notification_type を受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
```

(次のページに続く)

(前のページからの続き)

```
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "Notification",
  "message": "Claude needs your permission",
  "title": "Permission needed",
  "notification_type": "permission_prompt"
}
```

Notification フックは通知をブロックしたり変更したりすることはできません。Claude Code はその `systemMessage` と `continue` フィールドを破棄しますが、*terminalSequence* (ページ 475) は引き続き出力します。デスクトップ通知の例はこれを利用しています。Notification フックは、通知を外部サービスに転送するといった副作用を目的としています。

21.4.14 SubagentStart

Claude が Agent ツールでサブエージェントを生成したとき、Claude が**サブエージェントを再開** (ページ 352) したとき、およびインプロセスのエージェントチームのチームメイトが新しいメッセージを処理するたびに実行されます。エージェントタイプ名でフィルタリングする matcher をサポートしています。組み込みエージェントの場合、これは `general-purpose`、`Explore`、`Plan` のようなエージェント名です。**カスタムサブエージェント** (ページ 317) の場合、これはファイル名ではなく、エージェントのフロントマターの `name` フィールドです。

プラグインで提供されるサブエージェントの場合、エージェントタイプは、フロントマターの名前そのものではなく、`my-plugin:reviewer` のようなプラグインスコープの識別子になります。コロンが含まれるとプラグインスコープの名前は正規表現として扱われるため、完全一致させるには matcher を `^` と `$` で固定します: `^my-plugin:reviewer$`。

SubagentStart の入力

共通入力フィールド (ページ 466) に加えて、SubagentStart フックは、サブエージェントの一意の識別子を含む `agent_id` と、matcher がフィルタリングに使うエージェント名を含む `agent_type` を受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "SubagentStart",
  "agent_id": "agent-abc123",
  "agent_type": "Explore"
}
```

SubagentStart フックはサブエージェントの作成をブロックすることはできませんが、サブエージェントにコンテキストを注入することはできます。すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、次のフィールドを返すことができます。

フィールド	説明
additionalContext	会話の開始時、最初のプロンプトの前にサブエージェントのコンテキストに追加される文字列。*Claude* **にコンテキストを追加する** (ページ 476) を参照してください

```
{
  "hookSpecificOutput": {
    "hookEventName": "SubagentStart",
    "additionalContext": "Follow security guidelines for this task"
  }
}
```

同じサブエージェントに対してフックが再度実行された場合、Claude Code は、サブエージェントのコンテキストに以前の実行で注入したコピーがまだ残っていない場合にのみ、返されたコンテキストを注入します。起動時に注入されたコピーはそのまま残るため、サブエージェントのプロンプトキャッシュは損なわれません。**自動圧縮** (ページ 353) によってそのコピーが破棄された後は、Claude Code は次の実行のコンテキストを再び注入します。

21.4.15 SubagentStop

Claude Code のサブエージェントが応答を終えたときに実行されます。エージェントタイプでマッチします。値は SubagentStart と同じです。

SubagentStop の入力

共通入力フィールド (ページ 466) に加えて、SubagentStop フックは `stop_hook_active`、`agent_id`、`agent_type`、`agent_transcript_path`、`last_assistant_message` を受け取ります。`agent_type` フィールドは matcher のフィルタリングに使用される値です。`transcript_path` はメインセッションのトランスクリプトで、`agent_transcript_path` はネストされた `subagents/` フォルダーに保存されるサブエージェント自身のトランスクリプトです。`last_assistant_message` フィールドにはサブエージェントの最終応答のテキスト内容が含まれるため、フックはトランスクリプトファイルを解析せずにそれを参照できます。

すべての SubagentStop イベントが、Claude が生成したサブエージェントから来るわけではありません。Claude Code は、**プロンプトの提案** (ページ 92) や */btw* **のサイドクエスチョン** (ページ 98) など、自身の一部の機能のために内部エージェントも実行しており、それらが終了したときにも SubagentStop が発火します。これらのイベントでは、`agent_type` は、*--agent* (ページ 51) や agent 設定で設定されたものなど、セッション自体が実行されているエージェント名になり、セッションがエージェントなしで実行されている場合は空文字列になります。

エージェントタイプを指定する `matcher` は、空の `agent_type` にはマッチしません。matcher が省略されているか、"" または "*" であるか、空文字列にマッチする正規表現であるフックは、空の `agent_type` のイベントでも実行されます。

Claude Code v2.1.271 以降では、`SubagentHandback` ツールを使って実行されるサブエージェントは、停止する前にそのツールを通じてレポートを渡します。その場合、`last_assistant_message` フィールドにはサブエージェントの締めくくりのテキスト（ある場合）が入り、渡されたレポートは含まれません。レ

ポートはその呼び出しの message 入力であり、SubagentHandback にマッチする PreToolUse または PostToolUse フックは、それを tool_input.message として受け取ります。

SubagentStop フックは、*Stop* **の入力** (ページ 526) で説明している background_tasks と session_crons の配列も受け取ります。どちらの配列も、サブエージェントではなく親セッションを対象としています。

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../abc123.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "SubagentStop",
  "stop_hook_active": false,
  "agent_id": "def456",
  "agent_type": "Explore",
  "agent_transcript_path": "~/.claude/projects/.../abc123/subagents/agent-def456.jsonl
↪",
  "last_assistant_message": "Analysis complete. Found 3 potential issues...",
  "background_tasks": [],
  "session_crons": []
}
```

SubagentStop フックは *Stop* **フック** (ページ 528) と同じ判定制御形式を使用します。これには、サブエージェントを実行し続けるエラー以外のフィードバックのために、hookEventName を "SubagentStop" に設定した hookSpecificOutput.additionalContext も含まれます。reason とともに decision: "block" を返すと、サブエージェントは実行を続け、reason が次の指示としてサブエージェントに渡されます。終了コード 2 でブロックするフックも、同じ方法で stderr のメッセージを渡します。サブエージェントが戻った後に親セッションにコンテキストを注入するには、代わりに Agent ツールに対する *PostToolUse* (ページ 509) フックを使用してください。

21.4.16 TaskCreated

TaskCreate ツールでタスクが作成されるときに実行されます。命名規則を強制したり、タスクの説明を必須にしたり、特定のタスクの作成を阻止したりするために使用します。Task ツールがないセッションでは、このイベントは発火しません。

TaskCreated フックは matcher をサポートしておらず、発生するたびに発火します。

TaskCreated の入力

共通入力フィールド (ページ 466) に加えて、TaskCreated フックは task_id、task_subject、および省略可能な task_description、teammate_name、team_name を受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
```

(次のページに続く)

(前のページからの続き)

```
  "cwd": "/Users/...",
  "hook_event_name": "TaskCreated",
  "task_id": "task-001",
  "task_subject": "Implement user authentication",
  "task_description": "Add login and signup endpoints",
  "teammate_name": "implementer",
  "team_name": "session-a1b2c3d4"
}
```

フィールド	説明
task_id	作成されるタスクの識別子
task_subject	タスクのタイトル
task_description	タスクの詳細な説明。存在しない場合があります
teammate_name	タスクを作成するチームメイトの名前。存在しない場合があります
team_name	非推奨。セッションから導出されたチーム名。今後のリリースで削除されます

TaskCreated の判定制御

TaskCreated フックは、2 つの方法で作成をブロックできます。いずれの場合も、Claude Code はタスクを削除し、メッセージをツールのエラーとして Claude に返します。このイベントからの `continue: false` は Claude Code によって無視され、Claude は作業を続けます。

- **終了コード 2**: Claude Code は stderr のテキストをメッセージとして返します。
- **JSON** `{"decision": "block", "reason": "..."}`: Claude Code は `reason` をメッセージとして返します。

次の例は、件名が必要な形式に従っていないタスクをブロックします。

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

if [[ ! "$TASK_SUBJECT" =~ ^\[TICKET-[0-9]+\] ]]; then
  echo "Task subject must start with a ticket number, e.g. '[TICKET-123] Add feature'
↪" >&2
  exit 2
fi

exit 0
```

21.4.17 TaskCompleted

タスクが完了としてマークされるときに実行されます。これは 2 つの状況で発火します。いずれかのエージェントが TaskUpdate ツールを通じてタスクを明示的に完了としてマークした場合と、エージェントチームのチームメイトが進行中のタスクを抱えたままターンを終えた場合です。タスクを閉じる前に、テストや lint チェックの合格などの完了基準を強制するために使用します。

TaskCompleted フックは matcher をサポートしておらず、発生するたびに発火します。

TaskCompleted の入力

共通入力フィールド (ページ 466) に加えて、TaskCompleted フックは `task_id`、`task_subject`、および省略可能な `task_description`、`teammate_name`、`team_name` を受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TaskCompleted",
  "task_id": "task-001",
  "task_subject": "Implement user authentication",
  "task_description": "Add login and signup endpoints",
  "teammate_name": "implementer",
  "team_name": "session-a1b2c3d4"
}
```

フィールド	説明
`task_id`	完了するタスクの識別子
`task_subject`	タスクのタイトル
`task_description`	タスクの詳細な説明。存在しない場合があります
`teammate_name`	タスクを完了するチームメイトの名前。存在しない場合があります
`team_name`	非推奨。セッションから導出されたチーム名。今後のリリースで削除されます

TaskCompleted の判定制御

TaskCompleted フックは、タスクの完了を制御する 2 つの方法をサポートしています。

- **終了コード 2**: タスクは完了としてマークされず、stderr のメッセージがフィードバックとしてモデルに返されます。
- **JSON** `{"continue": false, "stopReason": "..."}`: チームメイトがターンを終えたことでイベントがトリガーされた場合、`Stop` フックの動作と同様にチームメイトを完全に停止します。`stopReason` はユーザーに表示されます。`TaskUpdate` ツールによってイベントがトリガーされた場合、Claude Code は `continue: false` を無視します。その場合でも終了コード 2 は完了をブロックし

ます。

次の例はテストを実行し、失敗した場合はタスクの完了をブロックします。

```
#!/bin/bash
INPUT=$(cat)
TASK_SUBJECT=$(echo "$INPUT" | jq -r '.task_subject')

# Run the test suite
if ! npm test 2>&1; then
  echo "Tests not passing. Fix failing tests before completing: $TASK_SUBJECT" >&2
  exit 2
fi

exit 0
```

21.4.18 Stop

メインの Claude Code エージェントが応答を終えたときに実行されます。ユーザーによる中断で停止した場合は実行されません。API エラーの場合は、代わりに *StopFailure* (ページ 529) が発火します。

Tip:

`/goal` コマンドは、セッションスコープのプロンプトベースの Stop フックの組み込みショートカットです。フック設定を書かずに、ある条件に向けて Claude に作業を続けさせたい場合に使用します。

Stop の入力

共通入力フィールド (ページ 466) に加えて、Stop フックは `stop_hook_active`、`last_assistant_message`、`background_tasks`、`session_crons` を受け取ります。`stop_hook_active` フィールドは、Claude Code が Stop フックの結果としてすでに続行している場合に `true` になります。解決しない条件でブロックし続けることがないよう、この値を確認するか、トランスクリプトを処理してください。Claude Code は連続 8 回の続行上限を適用します。Stop フックがターンを 8 回連続で続行させると、Claude Code は次のブロックを上書きしてターンを終了します。上限を引き上げるには、`CLAUDE_CODE_STOP_HOOK_BLOCK_CAP` を設定します。

`last_assistant_message` フィールドには Claude の最終応答のテキスト内容が含まれるため、フックはトランスクリプトファイルを解析せずにそれを参照できます。読み上げや通知のフックなど、完了したばかりのターンに対して動作するフックでは、`transcript_path` を読み取るのではなくこのフィールドを使用してください。すべてのバージョンで、Stop の時点でトランスクリプトファイルに最終メッセージが含まれているとは限りません。

`background_tasks` と `session_crons` の配列により、フックは「セッションが完了した」状態と「セッションが一時停止し、バックグラウンドの作業によって再び起こされるのを待っている」状態を区別できます。どちらの配列も、タスクレジストリにアクセスできる場合に存在し、進行中またはスケジュール済みのものがない場合は空になります。

background_tasks の各エントリは進行中のタスク 1 つを表し、次のフィールドを使用します。

フィールド	説明
id	タスクの識別子
type	shell、subagent、monitor、workflow、teammate、cloud session、MCP task など、わかりやすいタスクタイプのラベル。各ラベルは、タスクを作成した Claude Code の機能を示します。認識されないタイプの場合は、生の判別値が使われます
status	現在のタスクの状態
description	自由形式の説明。上限は 1000 文字で、切り詰められた場合は文字列内に … [+N chars] マーカーが付きます
command	シェルのコマンドライン。上限は 1000 文字です。shell タスクにのみ存在します
agent_type	サブエージェントのタイプ名。subagent タスクにのみ存在します
server	MCP サーバー名。monitor と MCP task タスクにのみ存在します
tool	MCP ツール名。monitor と MCP task タスクにのみ存在します
name	ワークフロー名。workflow タスクにのみ存在します

session_crons の各エントリは、CronCreate、ScheduleWakeup、/loop から作成された、セッションスコープのスケジュール済みウェイクアップ 1 つを表します。

フィールド	説明
id	cron タスクの識別子
schedule	cron 式。例: 0 9 * * 1-5
recurring	スケジュールが単一の発火時刻を表す 1 回限りのウェイクアップの場合は false、マッチするたびに再発火するタスクの場合は true
prompt	cron の発火時に送信されるプロンプト。上限は 1000 文字で、同じ … [+N chars] マーカーが付きます

次の例は、進行中の shell タスク 1 つと繰り返しの cron 1 つを含む Stop の入力を示しています。

```
{
  "session_id": "abc123",
  "transcript_path": "~/.claude/projects/.../00893aaf-19fa-41d2-8238-13269b9b3ca0.
↪jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "Stop",
  "stop_hook_active": true,
  "last_assistant_message": "I've completed the refactoring. Here's a summary...",
  "background_tasks": [
    {
      "id": "task-001",
      "type": "shell",
```

(次のページに続く)

(前のページからの続き)

```
      "status": "running",
      "description": "tail logs",
      "command": "tail -f /var/log/syslog"
    }
  ],
  "session_crons": [
    {
      "id": "cron-001",
      "schedule": "0 9 * * 1-5",
      "recurring": true,
      "prompt": "check the build"
    }
  ]
}
```

Stop の判定制御

Stop と SubagentStop フックは、Claude が続行するかどうかを制御できます。すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、フックスクリプトは次のイベント固有フィールドを返すことができます。

フィールド	説明
decision	"block" を指定すると Claude の停止を阻止します。Claude の停止を許可するには省略します
reason	decision が "block" の場合は必須です。Claude に続行すべき理由を伝えます
hookSpecificOutput.additionalContext	Claude へのエラー以外のフィードバック。Claude がそれに基づいて行動できるよう会話は続行しますが、decision: "block" とは異なり、トランスクリプトにはフックエラーではなくフックのフィードバックとして表示されます

終了コード 2 でブロックするフックは、reason と同じように扱われます。Claude は続行すべき理由の説明として stderr のメッセージを受け取ります。

```
{
  "decision": "block",
  "reason": "Must be provided when Claude is blocked from stopping"
}
```

フックが設計どおりに動作しており、「終了する前にテストスイートを実行してください」のようなガイダンスを Claude に与えている場合は、additionalContext を使用します。これは decision: "block" と同じループ保護、つまり stop_hook_active 入力と連続 8 回の続行上限を通じて会話を続行させますが、トランスクリプトでは Stop hook feedback というラベルが付き、フックエラーの通知は表示されません。

```
{
  "hookSpecificOutput": {
    "hookEventName": "Stop",
    "additionalContext": "Please run the test suite before finishing"
  }
}
```

21.4.19 StopFailure

API エラーによってターンが終了した場合に、*Stop* (ページ 526) の代わりに実行されます。Claude Code は、*terminalSequence* (ページ 475) を除き、フックの出力と終了コードを無視します。レート制限、認証の問題、その他の API エラーのために Claude が応答を完了できない場合に、失敗のログ記録、アラートの送信、または回復アクションの実行に使用します。

StopFailure の入力

共通入力フィールド (ページ 466) に加えて、StopFailure フックは `error`、省略可能な `error_details`、省略可能な `last_assistant_message` を受け取ります。`error` フィールドはエラーの種類を示し、matcher のフィルタリングに使用されます。

フィールド	説明
`error`	エラーの種類: `rate_limit`、`overloaded`、`authentication_failed`、`oauth_org_not_allowed`、`account_on_hold`、`billing_error`、`invalid_request`、`model_not_found`、`server_error`、`max_output_tokens`、`cloud_credential_error`、または `unknown`
`error_details`	エラーに関する追加の詳細（利用可能な場合）
`last_assistant_message`	会話に表示されるレンダリング済みのエラーテキスト。このフィールドに Claude の会話出力が入る `Stop` や `SubagentStop` とは異なり、`StopFailure` では `"API Error: Rate limit reached"` のような API エラー文字列そのものが入ります

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "StopFailure",
  "error": "rate_limit",
  "error_details": "429 Too Many Requests",
  "last_assistant_message": "API Error: Rate limit reached"
}
```

StopFailure フックには判定制御がありません。通知とログ記録の目的でのみ実行されます。

21.4.20 TeammateIdle

エージェントチームのチームメイトがターンを終えてアイドル状態になろうとしているときに実行されます。lint チェックの合格を必須にしたり、出力ファイルの存在を確認したりするなど、チームメイトが作業を停止する前に品質ゲートを強制するために使用します。

TeammateIdle フックは matcher をサポートしておらず、発生するたびに発火します。

TeammateIdle の入力

共通入力フィールド (ページ 466) に加えて、TeammateIdle フックは `teammate_name` と `team_name` を受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "permission_mode": "default",
  "hook_event_name": "TeammateIdle",
  "teammate_name": "researcher",
  "team_name": "session-a1b2c3d4"
}
```

フィールド	説明
`teammate_name`	アイドル状態になろうとしているチームメイトの名前
`team_name`	非推奨。セッションから導出されたチーム名。今後のリリースで削除されます

TeammateIdle の判定制御

TeammateIdle フックは、チームメイトの動作を制御する 2 つの方法をサポートしています。

- **終了コード 2**: チームメイトは stderr のメッセージをフィードバックとして受け取り、アイドル状態にならずに作業を続けます。
- **JSON** `{"continue": false, "stopReason": "..."}`: Stop フックの動作と同様に、チームメイトを完全に停止します。`stopReason` はユーザーに表示されます。

次の例は、チームメイトがアイドル状態になるのを許可する前に、ビルドアーティファクトが存在することを確認します。

```
#!/bin/bash

if [ ! -f "./dist/output.js" ]; then
  echo "Build artifact missing. Run the build before stopping." >&2
  exit 2
```

(次のページに続く)

(前のページからの続き)

```
fi

exit 0
```

21.4.21 ConfigChange

セッション中に設定ファイルが変更されたときに実行されます。設定変更の監査、セキュリティポリシーの強制、または設定ファイルへの不正な変更のブロックに使用します。

Claude Code は、設定ファイル、管理ポリシーファイル、またはスキルファイルが変更されたときに ConfigChange フックを実行します。管理ポリシーについては、`managed-settings.json` または `managed-settings.d/` 内のファイルが変更された場合にのみ実行します。サーバー管理設定や、macOS の管理された環境設定、Windows レジストリのポリシーへの変更は、フックを実行せずに適用します。`wslInheritsWindowsSettings` を使用している WSL では、Windows 側の管理設定ファイルの変更も、ポリシーのポーリング時にフックを実行せずに適用します。

matcher は設定のソースでフィルタリングします。

Matcher	発火するタイミング
`user_settings`	`~/.claude/settings.json` が変更された
`project_settings`	`.claude/settings.json` が変更された
`local_settings`	`.claude/settings.local.json` が変更された
`policy_settings`	`managed-settings.json` または `managed-settings.d/` 内のファイルが変更された
`skills`	`.claude/skills/` 内のスキルファイルが変更された

次の例は、セキュリティ監査のためにすべての設定変更をログに記録します。

```
{
  "hooks": {
    "ConfigChange": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/audit-config-change.sh",
            "args": []
          }
        ]
      }
    ]
  }
}
```

ConfigChange の入力

共通入力フィールド (ページ 466) に加えて、ConfigChange フックは source と、省略可能な file_path を受け取ります。source フィールドはどの種類の設定が変更されたかを示し、file_path は変更された特定のファイルのパスを示します。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "ConfigChange",
  "source": "project_settings",
  "file_path": "/Users/.../my-project/.claude/settings.json"
}
```

ConfigChange の判定制御

ConfigChange フックは、設定変更が反映されるのをブロックできます。変更を阻止するには、終了コード 2 または JSON の decision を使用します。ブロックされた場合、新しい設定は実行中のセッションに適用されません。

フィールド	説明
decision	"block" を指定すると、設定変更が適用されるのを阻止します。変更を許可するには省略します
reason	受け付けられますが、表示されることはありません

```
{
  "decision": "block",
  "reason": "Configuration changes to project settings require admin approval"
}
```

policy_settings の変更はブロックできません。マシン上の管理設定ファイルが変更されたときには policy_settings ソースに対してもフックが発火するため、それらの編集をログに記録するのには使用できますが、ブロックの判定はすべて無視されます。これにより、エンタープライズで管理される設定が常に反映されることが保証されます。サーバー管理設定が届いたときや更新されたときには、Claude Code は ConfigChange フックを実行しません。

Claude Code は ConfigChange フックの JSON 出力のうちブロックの判定に従い、systemMessage と continue は破棄します。ブロックされた変更については、reason でブロックした場合も終了コード 2 の stderr でブロックした場合も、ユーザーにも Claude にもメッセージは表示されません。Claude Code はデバッグログに 1 行書き込むだけです。

21.4.22 CwdChanged

メインの会話内のシェルコマンドが作業ディレクトリを変更したとき、たとえば Claude が cd コマンドを実行したときに実行されます。ディレクトリの変更に反応して、環境変数の再読み込み、プロジェクト固有のツールチェーンの有効化、セットアップスクリプトの自動実行などを行うために使用します。ディレクトリごとの環境を管理する direnv のようなツールでは、*FileChanged* (ページ 535) と組み合わせて使用します。

CwdChanged フックは *CLAUDE_ENV_FILE* (ページ 484) にアクセスできます。そのファイルに書き込まれた変数は、次の CwdChanged イベントで Claude Code によってクリアされるまで、後続の Bash コマンドに引き継がれます。

CwdChanged は matcher をサポートしておらず、発生するたびに発火します。

CwdChanged の入力

共通入力フィールド (ページ 466) に加えて、CwdChanged フックは `old_cwd` と `new_cwd` を受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project/src",
  "hook_event_name": "CwdChanged",
  "old_cwd": "/Users/my-project",
  "new_cwd": "/Users/my-project/src"
}
```

CwdChanged の出力

すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、CwdChanged フックは `watchPaths` を返して、*FileChanged* (ページ 535) が監視するファイルパスを動的に設定できます。

フィールド	説明
`watchPaths`	絶対パスの配列。現在の動的な監視リストを置き換えます。`matcher` 設定のパスは常に監視されます。空の配列を返すと動的リストがクリアされます。これは新しいディレクトリに入るときの典型的な使い方です

CwdChanged フックには判定制御がありません。ディレクトリの変更をブロックすることはできません。

Claude Code は JSON 出力から `watchPaths` と `systemMessage` を読み取り、`continue` を破棄します。対話型セッションでは、`systemMessage` を短いターミナル通知として表示します。このメッセージは SDK のメッセージストリームには届きません。

21.4.23 DirectoryAdded

セッションの途中で /add-dir コマンドを使って作業ディレクトリを追加した後、または SDK クライアントが register_repo_root 制御リクエストで作業ディレクトリを追加した後に実行されます。新たに追加されたリポジトリの準備、たとえば依存関係のインストールに使用します。

Claude Code は次の場合にはこのイベントを発火しません。

- --add-dir 起動フラグでディレクトリを渡した場合。これらのディレクトリは *SessionStart* (ページ 481) でカバーされます
- /permissions の Workspace タブでディレクトリを追加した場合
- すでに作業ディレクトリであるディレクトリ、またはその内部にあるディレクトリを追加した場合

Claude Code はサンドボックスと権限の状態を更新した後に DirectoryAdded を発火するため、フックが実行される時点で、サンドボックス化されたツールにはすでに新しいディレクトリが見えています。フックコマンド自体はサンドボックス化されずに実行されます。

Claude Code はフックを待ちません。追加はすぐに完了し、フックはデフォルトの 600 秒のタイムアウトでバックグラウンドで実行されます。

matcher は、ディレクトリがどのように追加されたかでフィルタリングします。

Matcher	発火するタイミング
slash_command	/add-dir でディレクトリを追加した
register_repo_root	SDK クライアントが register_repo_root 制御リクエストでディレクトリを追加した

DirectoryAdded の入力

共通入力フィールド (ページ 466) に加えて、DirectoryAdded フックは directory と source を受け取ります。

フィールド	説明
directory	追加されたディレクトリの絶対パス
source	ディレクトリの追加方法。/add-dir の場合は "slash_command"、SDK 制御リクエストの場合は "register_repo_root"

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project",
  "hook_event_name": "DirectoryAdded",
  "directory": "/Users/my-other-repo",
  "source": "slash_command"
}
```

DirectoryAdded フックには判定制御がありません。フックが実行される時点で追加はすでに完了しているため、追加をブロックすることはできません。Claude Code は JSON 出力から `continue` フィールドを破棄し、残りをソースごとに異なる方法で表示します。

- `slash_command`: Claude Code はフックの `systemMessage` をユーザーに表示するのではなく、次の会話ターンでコンテキストとして Claude に渡します。失敗したフックの数がトランスクリプトに表示されます。失敗時の出力全体はデバッグログに記録されます
- `register_repo_root`: Claude Code は `systemMessage` の出力と失敗時の出力をデバッグログにのみ書き込みます

21.4.24 FileChanged

監視対象のファイルがディスク上で変更されたときに実行されます。Claude Code はツール呼び出しを調べるのではなく、ファイルシステムウォッチャーで変更を検出するため、何がファイルを変更したかに関係なくフックを実行します。対象となるのは、`Edit` や `Write` のツール呼び出し、Claude が `Bash` で実行するスクリプト、あるいは Claude Code の完全に外部のプロセスです。一般的な用途は、プロジェクトの設定ファイルが変更されたときに環境変数を再読み込みすることです。

このイベントの `matcher` には 2 つの役割があります。

- **監視リストを構築する**: 値は `|` で分割され、各セグメントが作業ディレクトリ内のリテラルなファイル名として登録されます。そのため、`".envrc|.env"` はちょうどその 2 つのファイルを監視します。ここでは正規表現パターンは役に立ちません。`^\.env` のような値は、文字どおり `^\.env` という名前のファイルを監視することになります。
- **実行するフックをフィルタリングする**: 監視対象のファイルが変更されると、同じ値が、変更されたファイルのベース名に対して標準の *matcher* **ルール** (ページ 451) を使い、どのフックグループを実行するかをフィルタリングします。

次の例は、Bash コマンドや外部スクリプトによるファイルの書き換えを含め、あらゆる変更の後に `data.csv` の改行コードを正規化します。

```
{
  "hooks": {
    "FileChanged": [
      {
        "matcher": "data.csv",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/normalize-line-endings.sh"
          }
        ]
      }
    ]
  }
}
```

このフックは、stdin の *JSON* **入力** (ページ 536) の `file_path` フィールドから、変更されたファイルの絶対パスを読み取ります。`grep` によるガードは、`perl` が削除するのと同じもの、つまり行末の CR を検査するため、正規化後の実行ではファイルに触れずに終了します。これより緩いガードだと無限ループになります。`perl -i` は何も置換しなくてもファイルを書き換え、Claude Code は書き換えのたびにフックを再実行するためです。このスクリプトを `/path/to/normalize-line-endings.sh` に保存し、実行可能にしてください。

```
#!/bin/bash
FILE=$(jq -r .file_path)
if grep -q $'\r$' "$FILE"; then
  perl -pi -e 's/\r$//' "$FILE"
fi
```

フックが機能することを確認するには、Bash コマンドで `data.csv` に CRLF の行を追加するよう Claude に依頼します。Claude Code がフックを実行し、ファイルの改行コードは LF になります。

事前に名前を指定できないファイルを監視するには、フックから *watchPaths* (ページ 537) を返して監視リストを動的に更新します。Claude Code は、何かが監視するファイルを指定した場合にのみウォッチャーを起動するため、少なくとも 1 つのファイルを指定する matcher を持つ FileChanged グループか、`watchPaths` を返す *SessionStart* (ページ 483) または *CwdChanged* (ページ 533) フックでリストの初期値を設定してください。監視対象のファイルが変更されたとき、matcher は引き続きどのフックグループを実行するかをフィルタリングするため、動的なパスを処理するグループでは matcher を省略してください。省略した matcher はすべての監視対象ファイルにマッチし、監視リストには何も追加しません。`"*"` の matcher もすべてのファイルにマッチしますが、Claude Code はそれを他の値と同様に、`*` という名前のリテラルなファイルとして監視リストに登録します。

FileChanged フックは *CLAUDE_ENV_FILE* (ページ 484) にアクセスできます。そのファイルに書き込まれた変数は、次の *CwdChanged* (ページ 533) イベントで Claude Code によってクリアされるまで、後続の Bash コマンドに引き継がれます。

FileChanged の入力

共通入力フィールド (ページ 466) に加えて、FileChanged フックは `file_path` と `event` を受け取ります。

フィールド	説明
`file_path`	変更されたファイルの絶対パス
`event`	発生した内容: 変更されたファイルの場合は `"change"`、作成されたファイルの場合は `"add"`、削除されたファイルの場合は `"unlink"`

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../transcript.jsonl",
  "cwd": "/Users/my-project",
  "hook_event_name": "FileChanged",
  "file_path": "/Users/my-project/.envrc",
  "event": "change"
```

(次のページに続く)

(前のページからの続き)

```
}
```

FileChanged の出力

すべてのフックで利用できる *JSON* **出力フィールド** (ページ 474) に加えて、FileChanged フックは `watchPaths` を返して、監視するファイルパスを動的に更新できます。

フィールド	説明
`watchPaths`	絶対パスの配列。現在の動的な監視リストを置き換えます。`matcher` 設定のパスは常に監視されます。変更されたファイルに基づいて、フックスクリプトが監視すべき追加のファイルを見つけた場合に使用します

FileChanged フックには判定制御がありません。ファイルの変更が発生するのをブロックすることはできません。

Claude Code は JSON 出力から `watchPaths` と `systemMessage` を読み取り、`continue` を破棄します。対話型セッションでは、`systemMessage` を短いターミナル通知として表示します。このメッセージは SDK のメッセージストリームには届きません。

21.4.25 WorktreeCreate

worktree が作成されるときに実行されます。対象は、`claude --worktree` から作成される場合、*isolation: "worktree"* **を使用するサブエージェント** (ページ 321) から作成される場合、Claude Code が独自の worktree に分離するバックグラウンドセッションのために作成される場合です。デフォルトでは、Claude Code は `git worktree` を使って分離された作業コピーを作成します。WorktreeCreate フックを設定するとこのデフォルトの Git の動作が置き換えられ、SVN、Perforce、Mercurial などの別のバージョン管理システムを使用できるようになります。

フックはデフォルトの動作を完全に置き換えるため、`.worktreeinclude` は処理されません。`.env` のようなローカルの設定ファイルを新しい worktree にコピーする必要がある場合は、フックスクリプト内で行ってください。

フックは、作成された worktree ディレクトリのパスを返す必要があります。Claude Code はこのパスを、分離されたセッションの作業ディレクトリとして使用します。各フックタイプがパスを返す方法については、*WorktreeCreate* **の出力** (ページ 538) を参照してください。

Claude Code はフックの成否と返されたパスに従い、`systemMessage` と `continue` を破棄します。

次の例は、SVN の作業コピーを作成し、Claude Code が使用するパスを出力します。リポジトリの URL は独自のものに置き換えてください。

```
{
  "hooks": {
    "WorktreeCreate": [
      {
        "hooks": [
```

(次のページに続く)

(前のページからの続き)

```
          {
            "type": "command",
            "command": "bash -c 'NAME=$(jq -r .name); DIR=\"$HOME/.claude/worktrees/
↪$NAME\"; svn checkout https://svn.example.com/repo/trunk \"$DIR\" >&2 && echo \"
↪$DIR\"'"
          }
        ]
      }
    ]
  }
}
```

このフックは、stdin の JSON 入力から worktree の name を読み取り、新しいディレクトリに新規コピーをチェックアウトし、そのディレクトリのパスを出力します。最後の行の echo が、Claude Code が worktree のパスとして読み取るものです。パスに干渉しないよう、その他の出力はすべて stderr にリダイレクトしてください。

WorktreeCreate の入力

共通入力フィールド (ページ 466) に加えて、WorktreeCreate フックは name フィールドを受け取ります。これは新しい worktree のスラッグ識別子で、ユーザーが指定したものか自動生成されたもの（例: bold-oak-a3f2）です。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeCreate",
  "name": "feature-auth"
}
```

WorktreeCreate の出力

WorktreeCreate フックは、標準の許可/ブロックの判定モデルを使用しません。代わりに、フックの成功または失敗によって結果が決まります。フックは作成した worktree ディレクトリのパスを返す必要があります。

- **コマンドフック**（type: "command"）: パスを stdout の最後の空でない行として出力します。Claude Code はその行を読み取る前に ANSI エスケープコードを除去するため、echo より前に出力されたシェルの起動バナーは無視されます。それ以外のフックの出力はすべて stderr にリダイレクトしてください。
- **HTTP フック**（type: "http"）: レスポンス本文で { "hookSpecificOutput": { "hookEventName": "WorktreeCreate", "worktreePath": "/absolute/path" } } を返します。

フックが失敗した場合やパスを出力しなかった場合、worktree の作成はエラーで失敗します。

Claude Code は相対パスをフックが実行されたディレクトリを基準に解決し、その中の `.` や `..` セグメントを畳み込みます。結果のパスが Claude Code の移動できるディレクトリでない場合、セッションはそのパスを示すエラーを出力し、終了コード 1 で終了します。

Claude Code は、`.` や `..` セグメントを含む絶対パスと、リポジトリルート以下のシンボリックリンクを経由するパスを拒否します。リポジトリにコミットされたシンボリックリンクによって worktree がリポジトリ外へリダイレクトされる可能性があるためです。エラーには拒否されたコンポーネントが示されます。正規化され、リポジトリ内のシンボリックリンクを経由しないパスを返してください。v2.1.216 より前は、worktree の作成はこのチェックを行わずにフックのパスに従っていました。

21.4.26 WorktreeRemove

worktree が削除されるときに実行されます。これは *WorktreeCreate* (ページ 537) に対応するクリーンアップ用のイベントです。このイベントは次の場合に発生します。

- `--worktree` セッションを終了し、削除を選択したとき
- `isolation: "worktree"` を指定したサブエージェントが終了したとき
- フックが worktree を作成したバックグラウンドセッションを削除したとき

Git ベースの worktree の場合、Claude Code は `git worktree remove` で自動的にクリーンアップを行います。WorktreeCreate フックを設定した場合は、WorktreeRemove フックと組み合わせて、作成した worktree のクリーンアップを制御してください。

- **WorktreeRemove フックがない場合**: `--worktree` セッションを終了して削除を選択すると、Claude Code は WorktreeCreate フックが返したパスに対して `git worktree remove --force` にフォールバックするため、Git が認識している worktree は削除されます。Git が認識していない worktree（たとえば Git 以外のバージョン管理システムでフックが作成したもの）はディスクに残ります。バックグラウンドセッションの削除がフックで作成された worktree をどう扱うかについては、エージェントビューの削除ルールを参照してください。
- **フックが 0 で終了した場合**: worktree は削除済みとして扱われます。Claude Code はフックからそれ以外に何も読み取らないため、フックがディレクトリを確実に削除するようにしてください。
- **フックが 0 以外で終了した場合**: その後も `worktree_path` のディレクトリが存在していれば削除は失敗し、Git へのフォールバックなしで worktree はディスクに残ります。0 以外で終了する前にディレクトリを削除したフックは、削除済みとして扱われます。失敗の報告方法については、*WorktreeRemove* **の入力** (ページ 540) を参照してください。

Claude Code は WorktreeCreate フックが返したパスしか把握していないため、フックで作成された worktree に属するブランチを削除することはありません。WorktreeCreate フックでブランチを作成する場合は、WorktreeRemove フックでそのブランチを削除してください。

Claude Code は、`systemMessage` や `continue` など、WorktreeRemove フックの *JSON* **出力フィールド** (ページ 474) を破棄します。

バックグラウンドセッションの削除では、Claude Code はフックを実行する前に保存されている worktree パスを検証し、シンボリックリンクであるパス、またはリポジトリルート以下でシンボリックリンクを経由するパスを拒否します。まだファイルを含む worktree に対しては、エージェントビューで削除を確認した場合にのみフックが実行されます。そのような worktree の場合、`claude rm` はセッションと worktree を

残します。v2.1.216 より前は、フックはこれらのチェックなしで保存されたパスに対して実行されていました。

Claude Code は、WorktreeCreate が返したパスをフック入力の worktree_path として渡します。次の例では、そのパスを読み取ってディレクトリを削除します。

```
{
  "hooks": {
    "WorktreeRemove": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "bash -c 'jq -r .worktree_path | xargs rm -rf'"
          }
        ]
      }
    ]
  }
}
```

WorktreeRemove の入力

共通の入力フィールド (ページ 466) に加えて、WorktreeRemove フックは worktree_path フィールドを受け取ります。これは削除される worktree の絶対パスです。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "WorktreeRemove",
  "worktree_path": "/Users/.../my-project/.claude/worktrees/feature-auth"
}
```

WorktreeRemove フックの終了コードによって結果が決まります。フックが 0 以外で終了し、その後も worktree_path のディレクトリが存在している場合、削除は失敗します。

- worktree はディスクに残り、フックのコマンドと stderr は**デバッグログ** (ページ 562) に記録されます。
- バックグラウンドセッションを削除していた場合、セッションも残ります。エージェントビューの拒否メッセージには、exited 1 のようなフックの終了状況、stderr の冒頭部分、およびセッションを再度削除した場合にディレクトリがそれでも削除されるかどうかが表示されます。

21.4.27 PreCompact

Claude Code がコンテキスト圧縮を実行する直前に実行されます。

matcher の値は、圧縮が手動でトリガーされたか自動でトリガーされたかを示します。

Matcher	発生するタイミング
`manual`	`/compact`
`auto`	会話が**自動圧縮ウィンドウ** (ページ 754) に達したときの自動圧縮

圧縮をブロックするには、終了コード 2 で終了します。手動の `/compact` の場合、stderr のメッセージがユーザーに表示されます。`"decision": "block"` を含む JSON を返してブロックすることもできます。

自動圧縮をブロックした場合の影響は、発生したタイミングによって異なります。コンテキスト上限に達する前に予防的に圧縮がトリガーされた場合、Claude Code は圧縮をスキップし、会話は圧縮されないまま続行されます。API がすでに返したコンテキスト上限エラーから回復するために圧縮がトリガーされた場合は、元のエラーが表面化し、現在のリクエストは失敗します。

Claude Code は PreCompact フックの `systemMessage` と `continue` フィールドを破棄します。

PreCompact の入力

共通の入力フィールド (ページ 466) に加えて、PreCompact フックは `trigger` と `custom_instructions` を受け取ります。`manual` の場合、`custom_instructions` にはユーザーが `/compact` に渡した内容が含まれ、何も渡さなかった場合は `null` になります。`auto` の場合、`custom_instructions` は `null` です。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "PreCompact",
  "trigger": "manual",
  "custom_instructions": null
}
```

21.4.28 PostCompact

Claude Code がコンテキスト圧縮を完了した後に実行されます。このイベントを使用すると、圧縮後の新しい状態に対応できます。たとえば、生成された要約をログに記録したり、外部の状態を更新したりできます。Claude Code は PostCompact フックの `systemMessage` と `continue` フィールドを破棄します。

`PreCompact` と同じ matcher の値が適用されます。

Matcher	発生するタイミング
manual	/compact の後
auto	会話が**自動圧縮ウィンドウ** (ページ 754) に達したときの自動圧縮の後

PostCompact の入力

共通の入力フィールド (ページ 466) に加えて、PostCompact フックは trigger と compact_summary を受け取ります。compact_summary フィールドには、圧縮によって生成された会話の要約が含まれます。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "PostCompact",
  "trigger": "manual",
  "compact_summary": "Summary of the compacted conversation..."
}
```

PostCompact フックには判定の制御はありません。圧縮の結果に影響を与えることはできませんが、後続のタスクを実行できます。

21.4.29 PreModelSwitch

ユーザーまたはクライアントが要求したモデルの切り替えを Claude Code が適用する前に実行されます。切り替えをブロックしたり、確認を求めたり、切り替えにかかるコストを事前に表示したりするために使用します。

PreModelSwitch には Claude Code v2.1.251 以降が必要です。Claude Code は次のリクエストに対してこのフックを実行します。

- /model <name> と /model ピッカー
- Option+P または Alt+P のモデルピッカー
- /config の Model 設定
- セッションのモデルが変わる場合の fast mode のオン
- Agent SDK ホストまたは Remote Control からの set_model リクエスト、または apply_flag_settings リクエストでのモデル変更

Claude Code は、**自動モデルフォールバック** (ページ 744) やセッション再開時のモデルの復元など、Claude Code 自身が行う切り替えに対しては PreModelSwitch フックを実行しません。これらの変更は *PostModelSwitch* (ページ 547) にのみ届きます。

Claude Code は、[1m] サフィックスを無視して、切り替え先モデルの正規名と matcher を比較します。opus のようなエイリアス、日付付きのモデル ID、Amazon Bedrock のモデル ID のようなプロバイダー固有の ID は、いずれも解決先の 1 つの正規名にマッチするため、claude-opus-5 は Opus 5 のあらゆる表記をカバーします。

切り替え先の正規名を特定できない場合（たとえば *LLM* **ゲートウェイ** (ページ 841) だけが認識するカスタムモデル ID の場合）、Claude Code は matcher に関係なくすべての PreModelSwitch フックを実行します。そのため、ブロックするフックは matcher だけに頼らず、入力の `to_model` を確認する必要があります。

matcher は、完全な名前、`claude-opus-4-6|claude-opus-5` のような `|` 区切りのリスト、または `.*opus.*` のような正規表現で記述します。次の例では、完全名の matcher を使用し、さらにフック入力の `to_model` も確認することで、Opus 4.6 への切り替えを終了コード 2 で拒否し、それ以外の切り替え先は許可します。

macOS/Linux

コマンドは `jq` で `to_model` を確認します。

```
{
  "hooks": {
    "PreModelSwitch": [
      {
        "matcher": "claude-opus-4-6",
        "hooks": [
          {
            "type": "command",
            "command": "jq -e '.to_model | test(\"opus-4-6\")' > /dev/null && {
 echo 'Opus 4.6 is retired for this project. Use a newer model.' >&2; exit 2; };␣
↪exit 0"
          }
        ]
      }
    ]
  }
}
```

Windows (PowerShell)

PowerShell でスクリプトを実行するコマンドフックを登録します。

```
{
  "hooks": {
    "PreModelSwitch": [
      {
        "matcher": "claude-opus-4-6",
        "hooks": [
          {
            "type": "command",
            "command": "powershell.exe",
            "args": [
              "-NoProfile",
              "-ExecutionPolicy",
              "Bypass",
              "-File",
              "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-opus-46.ps1"
```

(次のページに続く)

(前のページからの続き)

```
            ]
          }
        ]
      }
    ]
  }
}
```

次のスクリプトをプロジェクトの `.claude/hooks/block-opus-46.ps1` に保存します。

```
$hookInput = [Console]::In.ReadToEnd() | ConvertFrom-Json
if ($hookInput.to_model -match 'opus-4-6') {
  [Console]::Error.WriteLine('Opus 4.6 is retired for this project. Use a newer␣
↪model.')
  exit 2
}
exit 0
```

フックが動作することを確認するには、別のモデルを実行しているセッションから `/model claude-opus-4-6` を実行します。Claude Code は現在のモデルを維持し、PreModelSwitch フックが切り替えをブロックしたことを、指定したメッセージを理由として報告します。

PreModelSwitch の入力

共通の入力フィールド (ページ 466) に加えて、PreModelSwitch フックは次の表のフィールドを受け取ります。最後の 5 つは会話を新しいモデルに再送信するコストを表すため、フックは切り替えの前にその金額を表示できます。

フィールド	型	説明
`from_model`	string	切り替え元のモデル ID
`to_model`	string	切り替え先のモデル ID。matcher はこのモデルの正規名と比較されます
`requested_model`	string または `null`	リクエストで指定されたモデル: `opus` のようなエイリアス、完全なモデル ID、またはデフォルトモデルが要求された場合は `null`
`source`	string	リクエストの送信元: `/model <name>`、`/config` の Model 設定、または fast mode のオンの場合は `"command"`、モデルピッカーの場合は `"picker"`、Agent SDK ホストまたは Remote Control からの `set_model` リクエスト、または `apply_flag_settings` リクエストでのモデル変更の場合は `"sdk"`

次のページに続く

表 68 – 前のページからの続き

フィールド	型	説明
context_tokens	number	次のリクエストがプロンプトとして再送信するトークン数: メイン会話の最後の応答の入力、キャッシュ読み取り、キャッシュ作成、出力トークンの合計。最初の応答の前は 0
prompt_cache_warm	boolean	現在のモデルのプロンプトキャッシュがまだウォームである可能性が高いかどうか。つまり、切り替えによってそれが失われるかどうか
cache_ttl	string	Claude Code がこのセッションで要求するプロンプトキャッシュの有効期間: "5m" または "1h"
estimated_cache_write_usd	number	to_model 上で cache_ttl の料金で context_tokens をプロンプトキャッシュに書き込む推定コスト（米ドル）。次の応答は含みません。サーバーがコンテキスト全体を再キャッシュする必要がない場合もあるため、推定値として扱ってください
pricing	string	Claude Code が estimated_cache_write_usd の料金をどのように算出したか: 組織独自の料金が設定されている場合はその料金による "configured"、定価による "catalog"、または to_model の価格が不明で Claude Code がデフォルトの料金を想定した場合は "default"

次の例は、Sonnet 5 を実行しているセッションでの /model opus の入力を示しています。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "PreModelSwitch",
  "from_model": "claude-sonnet-5",
  "to_model": "claude-opus-5",
  "requested_model": "opus",
  "source": "command",
  "context_tokens": 182340,
  "prompt_cache_warm": true,
  "cache_ttl": "5m",
  "estimated_cache_write_usd": 1.1396,
  "pricing": "catalog"
}
```

PreModelSwitch の判定の制御

PreModelSwitch フックは、切り替えをキャンセルしたり、ユーザーに確認を求めたり、続行させたりできます。終了コード 2 またはトップレベルの decision: "block" で切り替えがキャンセルされます。

より細かく制御するには、*PreToolUse* (ページ 502) と同様に、hookSpecificOutput オブジェクト内で permissionDecision と permissionDecisionReason を返します。PreModelSwitch は "allow"、"deny"、"ask" を受け付けます。"defer"、updatedInput、additionalContext は受け付けません。次の表で両方のフィールドを説明します。

フィールド	説明
permissionDecision	"allow" は続行し、プロンプトキャッシュがウォームな間に Claude Code が表示する確認をスキップします。"deny" は切り替えをキャンセルします。"ask" はユーザーに確認を求めます
permissionDecisionReason	"deny" の場合、切り替えがブロックされた理由としてユーザーに表示されるか、set_model リクエストに対するエラーとして返されます。"ask" の場合、確認プロンプトに表示されます。"allow" の場合は無視されます

"ask" のプロンプトを表示できるのは、対話セッションでの /model だけです。-p フラグを使用した非対話モード、/config、set_model リクエストなど、その他のすべてのサーフェスでは、Claude Code は "ask" を拒否として扱います。

次の例では、context_tokens のトークン数を示してユーザーに確認を求めます。

```
{
  "hookSpecificOutput": {
    "hookEventName": "PreModelSwitch",
    "permissionDecision": "ask",
    "permissionDecisionReason": "Switching now re-sends about 180k tokens to the new␣
↪model. Continue?"
  }
}
```

複数の PreModelSwitch フックが異なる判定を返した場合、優先順位は deny > ask > allow です。

Claude Code は判定に関係なく、フックが返した systemMessage をユーザーに表示します。そのため、コストを報告するフックは {"systemMessage": "..."} を返して 0 で終了できます。

タイムアウトまでに応答しない PreModelSwitch フックは、切り替えをブロックします。これに対して *PreToolUse* (ページ 471) では、タイムアウトしたコマンドフックはツール呼び出しを続行させます。このイベントのデフォルトのタイムアウトは 30 秒です。PreModelSwitch は command、http、mcp_tool フックのみを実行するため、prompt と agent のデフォルトは適用されません。

0 と 2 以外のコードで終了し、JSON の判定を出力しないフックはブロックしません。**その他の終了コード** (ページ 470) で説明されているように、Claude Code はその stderr を表示して切り替えを適用します。

21.4.30 PostModelSwitch

セッションのモデルが変更された後に実行されます。すべての CLAUDE.md を編集することなく、Claude にモデル固有のガイダンスを与えるために使用します。たとえば、特定のモデルに適用される組織全体の指示などです。

PostModelSwitch には Claude Code v2.1.251 以降が必要です。モデルはすでに変更されているため、ブロックすることはできません。Claude Code は次のいずれかの変更の後に PostModelSwitch フックを実行します。

- ユーザーまたはクライアントが要求した切り替え
- セッションのモデルを変更する**自動モデルフォールバック** (ページ 744)
- *opusplan* (ページ 742) のような設定による plan モードへの移行または plan モードからの離脱
- セッション再開時に Claude Code がモデルを復元したとき

フォールバックモデルチェーン (ページ 743) のモデルがターンを処理する場合、その置き換えは 1 ターンだけでセッションのモデルは変わらないため、Claude Code は PostModelSwitch フックを実行しません。

matcher は *PreModelSwitch* (ページ 542) と同じルールに従います。Claude Code は、セッションの切り替え先モデルの正規名と matcher を比較します。

次の例では、セッションのモデルがいずれかの Opus モデルに変わるたびにガイダンスを追加します。

```
{
  "hooks": {
    "PostModelSwitch": [
      {
        "matcher": ".*opus.*",
        "hooks": [
          {
            "type": "command",
            "command": "echo 'On Opus, delegate implementation work to subagents and␣
↪keep this conversation for planning and review.'"
          }
        ]
      }
    ]
  }
}
```

フックが動作することを確認するには、別のモデルを実行しているセッションから Opus モデルに切り替え（たとえば Sonnet のセッションから `/model opus` を実行し）、現在のモデルについてどのようなガイダンスがあるかを Claude に尋ねます。

PostModelSwitch の入力

PostModelSwitch フックは *PreModelSwitch* (ページ 544) と同じフィールドを受け取ります。ただし、`hook_event_name` は `"PostModelSwitch"` に設定され、`source` には 2 つの値が追加されます。自動フォールバックなど Claude Code 自身が行った変更を表す `"auto"` と、セッション再開時に復元されたモデルを表す `"resume"` です。

`source` が `"auto"` の場合、`requested_model` は `null` です。`source` が `"resume"` の場合は、Claude Code が復元した保存済みのモデル設定です。

PostModelSwitch の判定の制御

Claude Code は、終了コード 0 の場合のフックの**プレーンテキストの** *stdout* (ページ 469)、または JSON 出力の `additionalContext` を受け取り、切り替え後の次のリクエストで Claude に渡します。すべてのフックで使用できる *JSON* **出力フィールド** (ページ 474) に加えて、次のフィールドを返すことができます。

フィールド	説明
`additionalContext`	次のリクエストで Claude のコンテキストに追加される文字列。*Claude* **にコンテキストを追加する** (ページ 476) を参照してください

次のプロンプトを送信してから 5 秒以内にフックが完了しない場合、Claude Code はその出力なしでリクエストを送信し、代わりにその次のリクエストに出力を添付します。次のリクエストまでにモデルが複数回変更された場合、Claude Code は最後の切り替え先モデルの出力のみを渡します。

21.4.31 SessionEnd

Claude Code セッションが終了するときに実行されます。クリーンアップタスク、セッション統計のログ記録、セッション状態の保存に便利です。終了理由でフィルタリングするための matcher をサポートしています。

フック入力の `reason` フィールドは、セッションが終了した理由を示します。

理由	説明
`clear`	`/clear` コマンドでセッションがクリアされた
`resume`	対話的な `/resume` でセッションが切り替えられた
`logout`	ユーザーがログアウトした
`prompt_input_exit`	プロンプト入力が表示されている間にユーザーが終了した
`other`	その他の終了理由
`bypass_permissions_disabled`	v2.1.234 で削除されました。Claude Code はこの値を送信しません。`SessionEnd` の matcher から削除してください

SessionEnd の入力

共通の入力フィールド (ページ 466) に加えて、SessionEnd フックはセッションが終了した理由を示す reason フィールドを受け取ります。すべての値については、上記の**理由の表** (ページ 548) を参照してください。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "SessionEnd",
  "reason": "other"
}
```

SessionEnd フックには判定の制御はありません。セッションの終了をブロックすることはできませんが、クリーンアップタスクを実行できます。Claude Code は、systemMessage などの *JSON* **出力フィールド** (ページ 474) を破棄します。

SessionEnd フックのデフォルトのタイムアウトは 1.5 秒です。これは、終了するとき、/clear を実行するとき、または対話的な /resume でセッションを切り替えるときに適用されます。フックにより多くの時間を与えるには、次の 2 つの方法があります。

- **フックごとの** timeout: そのフックの設定で timeout を指定します。全体の制限時間は、設定ファイル内で最も大きいフックごとの timeout に合わせて、最大 60 秒まで自動的に引き上げられます。この方法で制限時間を引き上げても、独自の timeout を持たないフックはデフォルトのままです。プラグインが提供するフックに設定されたタイムアウトは、制限時間を引き上げません。
- CLAUDE_CODE_SESSIONEND_HOOKS_TIMEOUT_MS: この環境変数をミリ秒単位で設定すると、制限時間を明示的に上書きできます。設定した値は、独自の timeout を持たない各フックのタイムアウトにもなります。

次の例では、制限時間を 5 秒に設定します。

```
CLAUDE_CODE_SESSIONEND_HOOKS_TIMEOUT_MS=5000 claude
```

v2.1.268 より前は、CLAUDE_CODE_SESSIONEND_HOOKS_TIMEOUT_MS は全体の制限時間のみを引き上げ、独自の timeout を持たないフックは引き続き 1.5 秒後にキャンセルされていました。

21.4.32 Elicitation

MCP サーバーがタスクの途中でユーザー入力を要求したときに実行されます。デフォルトでは、Claude Code はユーザーが応答するための対話ダイアログを表示します。フックはこのリクエストをインターセプトしてプログラムで応答し、ダイアログを完全にスキップできます。

matcher フィールドは MCP サーバー名と照合されます。

Elicitation の入力

共通の入力フィールド (ページ 466) に加えて、Elicitation フックは mcp_server_name、message、およびオプションの mode、url、elicitation_id、requested_schema フィールドを受け取ります。

最も一般的なケースであるフォームモードの elicitation の場合:

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "Elicitation",
  "mcp_server_name": "my-mcp-server",
  "message": "Please provide your credentials",
  "mode": "form",
  "requested_schema": {
    "type": "object",
    "properties": {
      "username": { "type": "string", "title": "Username" }
    }
  }
}
```

ブラウザベースの認証に使用される URL モードの elicitation の場合:

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "Elicitation",
  "mcp_server_name": "my-mcp-server",
  "message": "Please authenticate",
  "mode": "url",
  "url": "https://auth.example.com/login"
}
```

Elicitation の出力

ダイアログを表示せずにプログラムで応答するには、hookSpecificOutput を含む JSON オブジェクトを返します。

```
{
  "hookSpecificOutput": {
    "hookEventName": "Elicitation",
    "action": "accept",
```

(次のページに続く)

(前のページからの続き)

```
    "content": {
      "username": "alice"
    }
  }
}
```

フィールド	値	説明
action	accept、decline、cancel	リクエストを承諾、拒否、またはキャンセルするかどうか
content	object	送信するフォームフィールドの値。action が accept の場合にのみ使用されます

終了コード 2 は elicitation を拒否します。Claude Code は stderr のメッセージをどこにも表示しません。

Claude Code は Elicitation フックの JSON 出力の hookSpecificOutput に基づいて動作し、systemMessage と continue は破棄します。

21.4.33 ElicitationResult

ユーザーが MCP の elicitation に応答した後に実行されます。フックは、応答が MCP サーバーに返送される前に、その応答を監視、変更、またはブロックできます。

matcher フィールドは MCP サーバー名と照合されます。

ElicitationResult の入力

共通の入力フィールド (ページ 466) に加えて、ElicitationResult フックは mcp_server_name、action、およびオプションの mode、elicitation_id、content フィールドを受け取ります。

```
{
  "session_id": "abc123",
  "transcript_path": "/Users/.../.claude/projects/.../00893aaf-19fa-41d2-8238-
↪13269b9b3ca0.jsonl",
  "cwd": "/Users/...",
  "hook_event_name": "ElicitationResult",
  "mcp_server_name": "my-mcp-server",
  "action": "accept",
  "content": { "username": "alice" },
  "mode": "form",
  "elicitation_id": "elicit-123"
}
```

ElicitationResult の出力

ユーザーの応答を上書きするには、hookSpecificOutput を含む JSON オブジェクトを返します。

```
{
  "hookSpecificOutput": {
    "hookEventName": "ElicitationResult",
    "action": "decline",
    "content": {}
  }
}
```

フィールド	値	説明
action	accept、decline、cancel	ユーザーのアクションを上書きします
content	object	フォームフィールドの値を上書きします。action が accept の場合にのみ意味を持ちます

終了コード 2 は応答をブロックし、実際のアクションを decline に変更します。Claude Code は stderr のメッセージをどこにも表示しません。

Claude Code は ElicitationResult フックの JSON 出力の hookSpecificOutput に基づいて動作し、systemMessage と continue は破棄します。

21.5 プロンプト ベースのフック

コマンド、HTTP、MCP ツール フックに加えて、Claude Code はプロンプト ベースのフック（type: "prompt"）をサポートしており、LLM を使用してアクションを許可またはブロックするかどうかを評価し、エージェント フック（type: "agent"）はツール アクセスを持つ agentic ベリファイアーを生成します。すべてのイベントがすべてのフック タイプをサポートしているわけではありません。

5 つのフック タイプ（command、http、mcp_tool、prompt、agent）すべてをサポートするイベント：

- PermissionDenied
- PostToolBatch
- PostToolUse
- PostToolUseFailure
- PreToolUse
- Stop
- SubagentStop
- TaskCompleted
- TaskCreated
- TeammateIdle

- UserPromptExpansion
- UserPromptSubmit

PermissionRequest は command、http、mcp_tool、prompt フックをサポートしますが、agent フックはサポートしません。このイベントにエージェント フックを設定した場合、Claude Code はそれをスキップし、権限フローは変更されずに進行します。フックから許可または拒否するには、コマンド フックまたは HTTP フックから**決定オブジェクト** (ページ 507) を返します。

command、http、mcp_tool フックをサポートするが、prompt または agent をサポートしないイベント：

- ConfigChange
- CwdChanged
- DirectoryAdded
- Elicitation
- ElicitationResult
- FileChanged
- InstructionsLoaded
- MessageDisplay
- Notification
- PostCompact
- PostModelSwitch
- PreCompact
- PreModelSwitch
- SessionEnd
- StopFailure
- SubagentStart
- WorktreeCreate
- WorktreeRemove

SessionStart と Setup は command と mcp_tool フックをサポートしており、それらの mcp_tool フックがいつ実行されるかについては *MCP* **ツール フックのフィールド** (ページ 461) で説明しています。これらは http、prompt、agent フックをサポートしていません。

21.5.1 プロンプト ベースのフックの仕組み

プロンプト ベースのフックは Bash コマンドを実行する代わりに：

1. フック入力とプロンプトを Claude モデル（デフォルトでは Claude Code が**バックグラウンド機能** (ページ 923) に使用するモデル）に送信

2. LLM は決定を含む構造化 JSON で応答
3. Claude Code は決定を自動的に処理

21.5.2 プロンプト フック設定

type を "prompt" に設定し、command の代わりに prompt 文字列を提供します。$ARGUMENTS プレースホルダーを使用して、フックの JSON 入力データをプロンプト テキストに注入します。

この Stop フックは、Claude が終了する前にすべてのタスクが完了しているかどうかを評価するよう LLM に求めます：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "Evaluate if Claude should stop: $ARGUMENTS. Check if all tasks␣
↪are complete."
          }
        ]
      }
    ]
  }
}
```

フィールド	必須	説明
type	はい	"prompt" である必要があります
prompt	はい	LLM に送信するプロンプト テキスト。フック入力 JSON のプレースホルダーとして $ARGUMENTS を使用します。$ARGUMENTS が存在しない場合、入力 JSON がプロンプトに追加されます
model	いいえ	評価に使用するモデル。デフォルトは Claude Code が**バックグラウンド機能** (ページ 923) に使用するモデル
timeout	いいえ	タイムアウト（秒単位）。デフォルト：30
continueOnBlock	いいえ	適用されるイベントでは、true にすると ok: false の理由を Claude にフィードバックし、ターンを終了する代わりに続行します。デフォルト：false。イベント ごとの動作については、**レスポンス スキーマ** (ページ 555) を参照してください

21.5.3 レスポンス スキーマ

LLM は以下を含む JSON で応答する必要があります：

```
{
  "ok": true | false,
  "reason": "Explanation for the decision",
  "impossible": true | false
}
```

フィールド	説明
ok	true で許可します。false の場合は、以下のイベント ごとの動作を参照してください
reason	ok が false のときに必須
impossible	省略可能。条件が決して満たされないとモデルが判断した場合に、ok: false とともに返します。Stop と SubagentStop では、Claude Code は理由をフィードバックする代わりにターンを終了させます。エージェント フックとその他のイベントはこれを無視します

ok: false で何が起こるかはイベントによって異なります：

- Stop と SubagentStop：理由は Claude の次の指示としてフィードバックされ、ターンが続行されます。ただし、応答で impossible: true も設定されている場合は、Claude Code が停止を許可し、ターンが終了します
- PreToolUse：ツール呼び出しが拒否されます。デフォルトではターンが終了し、拒否理由は警告行としてチャットに表示されます。代わりに理由をツール エラーとして Claude に返し、Claude が調整して続行できるようにするには、continueOnBlock: true を設定します。これはコマンド フックの permissionDecision: "deny" と同等です。v2.1.210 より前は、拒否理由はツール エラーとして Claude に返され、ターンは続行されていました
- PostToolUse：デフォルトではターンが終了し、理由は警告行としてチャットに表示されます。continueOnBlock: true を設定して、理由を Claude にフィードバックし、ターンを続行する代わりに使用します
- PostToolBatch、UserPromptSubmit、UserPromptExpansion：ターンが終了し、理由は警告行として表示されます。これらのイベントは continue に関係なく decision: "block" でターンを終了します
- PostToolUseFailure、TaskCreated：理由は continueOnBlock に関係なくツール エラーとして Claude に返され、ターンが続行されます
- TaskCompleted：ターン中にタスクが完了としてマークされたために発火した場合、理由は continueOnBlock に関係なくツール エラーとして Claude に返され、ターンが続行されます。チームメイトが停止したために発火した場合は、TeammateIdle と同様に動作し、デフォルトでチームメイトを停止します
- TeammateIdle：デフォルトではチームメイトが停止し、理由は警告行として表示されます。continueOnBlock: true を設定して、理由をチームメイトにフィードバックし、代わりに作業を続行させ

ます

- `PermissionRequest`：`ok: false` は効果がありません。フックから承認を拒否するには、**コマンド フック** (ページ 458) を使用して `hookSpecificOutput.decision.behavior: "deny"` を返します
- `PermissionDenied`：`ok: false` は効果がありません。拒否は既に発生しているためです。このイベントが読み取る唯一の出力は `hookSpecificOutput.retry` です。プロンプト フックとエージェント フックはこれを設定できません。これらはこのイベントで実行されますが、その出力は破棄されます。`retry` を返すには、**コマンド フック** (ページ 458) を使用してください

任意のイベントでより細かい制御が必要な場合は、**決定制御** (ページ 477) で説明されているイベント ごとのフィールドを使用して、**コマンド フック** (ページ 458) を使用してください。

21.5.4 停止する前に複数の条件をチェック

この `Stop` フックは詳細なプロンプトを使用して、Claude が停止することを許可する前に 3 つの条件をチェックします。`SubagentStop` フックは同じ形式を使用して、**サブエージェント** (ページ 317) が停止すべきかどうかを評価します。条件がまだ満たされていないためにモデルが `"ok": false` を返した場合、Claude は提供された理由を次の指示として受け取り、作業を続行します：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "prompt",
            "prompt": "You are evaluating whether Claude should stop working.␣
↪Context: $ARGUMENTS\n\nAnalyze the conversation and determine if:\n1. All user-
↪requested tasks are complete\n2. Any errors need to be addressed\n3. Follow-up work␣
↪is needed\n\nRespond with JSON: {\"ok\": true} to allow stopping, or {\"ok\": false,
 \"reason\": \"your explanation\"} to continue working.",
            "timeout": 30
          }
        ]
      }
    ]
  }
}
```

21.6 エージェント ベースのフック

警告:

エージェント フックは実験的です。動作と設定は将来のリリースで変更される可能性があります。本番ワークフローの場合は、**コマンド フック** (ページ 458) を優先してください。

エージェント ベースのフック（type: "agent"）はプロンプト ベースのフックのようですが、マルチターン ツール アクセスを備えています。単一の LLM 呼び出しの代わりに、エージェント フックはサブエージェントを生成し、ファイルを読み取り、コードを検索し、コードベースを検査して条件を検証できます。エージェント フックは、PermissionRequest を除き、**プロンプト ベースのフック** (ページ 552) と同じイベントをサポートしています。

21.6.1 エージェント フックの仕組み

エージェント フックが発火するとき：

1. Claude Code はプロンプトとフックの JSON 入力を持つサブエージェントを生成します
2. サブエージェントは Read、Grep、Glob などのツールを使用して調査できます
3. 最大 50 ターン後、サブエージェントは構造化 { "ok": true/false } 決定を返します
4. Claude Code は ok が true の場合にアクションを許可します。ok が false の場合、Claude Code は**レスポンス スキーマ** (ページ 555) に記載されているとおり、そのイベントで continueOnBlock: true を指定したプロンプト フックと同じ方法でブロックを処理します

エージェント フックは、フック入力データのみを評価するのではなく、実際のファイルを検査したりテスト出力を検査したりする必要がある場合に便利です。

21.6.2 エージェント フック設定

type を "agent" に設定し、prompt 文字列を提供します。フック入力 JSON のプレースホルダーとして $ARGUMENTS を使用します。設定フィールドは**プロンプト フック** (ページ 554) と同じですが、エージェント フックはデフォルト タイムアウトが 60 秒と長く、continueOnBlock フィールドがありません。

レスポンス スキーマは、許可する場合は { "ok": true }、ブロックする場合は { "ok": false, "reason": "..." } です。ok: false の場合、Claude Code はエージェント フックを、同じイベントにおける *continueOnBlock: true* **を指定したプロンプト フック** (ページ 555) と同じ方法で処理します。エージェント フックには continueOnBlock フィールドがなく、プロンプト フックの impossible フィールドもサポートしていません。

この Stop フックは、Claude が終了することを許可する前にすべてのユニット テストが合格することを検証します：

```
{
  "hooks": {
    "Stop": [
      {
        "hooks": [
          {
            "type": "agent",
            "prompt": "Verify that all unit tests pass. Run the test suite and check␣
↪the results. $ARGUMENTS",
            "timeout": 120
          }
```

(次のページに続く)

(前のページからの続き)

```
        ]
      }
    ]
  }
}
```

21.7 バックグラウンドでフックを実行

デフォルトでは、フックは完了するまで Claude の実行をブロックします。デプロイ、テスト スイート、外部 API 呼び出しなどの長時間実行タスクの場合、`"async": true` を設定してフックをバックグラウンドで実行し、Claude が作業を続行できるようにします。非同期フックはブロックまたは Claude の動作を制御できません。`decision`、`permissionDecision`、`continue` などのレスポンス フィールドは、制御しようとしたアクションがすでに完了しているため、効果がありません。

21.7.1 非同期フックを設定

コマンド フックの設定に `"async": true` を追加して、Claude をブロックせずにバックグラウンドで実行します。このフィールドは `type: "command"` フックでのみ利用可能です。

このフックは、すべての `Write` ツール呼び出しの後にテスト スクリプトを実行します。`run-tests.sh` の実行中も、Claude はすぐに作業を続行します。スクリプトが完了すると、その出力は次の会話ターンで配信されます。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
            "command": "/path/to/run-tests.sh",
            "async": true
          }
        ]
      }
    ]
  }
}
```

非同期フックがバックグラウンドで実行され始めると、Claude Code はそのフックに `timeout` を適用しません。`asyncRewake` で実行するフックには、Claude Code は引き続き `timeout` を適用します。

Claude Code が非同期フックの結果を配信するのは、セッションの実行中のみです。

- `-p` フラグを使用した**非対話モード** (ページ 695) では、Claude Code は終了処理時にまだ実行中の非同期フックをすべて強制終了し、結果 `cancelled` で確定します

- フックの処理を `claude -p` セッションより長く存続させる必要がある場合は、フックから完全にデタッチされたプロセスを起動します

21.7.2 非同期フックの実行方法

非同期フックが発火すると、Claude Code はフック プロセスを開始し、完了を待たずにすぐに続行します。フックは同期フックと同じ JSON 入力を stdin 経由で受け取ります。

バックグラウンド プロセスが終了した後、Claude Code はフックの JSON レスポンスに含まれる `additionalContext` フィールドと `systemMessage` フィールドを次の会話ターンで Claude に配信します。同期フックの `systemMessage` とは異なり、どちらのフィールドもユーザーには表示されません。

Claude Code は JSON レスポンスを同期フックと同じ**出力スキーマ** (ページ 474) に対して検証し、`systemMessage` が文字列でないなど、値の型が間違っているフィールドをドロップします。これは配信する代わりに行われます。`--debug` で実行すると、ドロップされた各フィールドに名前を付けた警告が表示されます。v2.1.202 より前では、非同期フックからの不正な形式の JSON 出力はセッションをクラッシュさせる可能性があり、セッションが再開されるたびにクラッシュが再発生していました。

非同期フック完了通知はデフォルトで抑制されます。これらを表示するには、`Ctrl+O` で詳細モードを有効にするか、`--verbose` で Claude Code を開始します。

21.7.3 ファイル変更後にテストを実行

このフックは Claude がファイルを書き込むたびにバックグラウンドでテスト スイートを開始し、テストが完了したら結果を Claude に報告します。このスクリプトをプロジェクトの `.claude/hooks/run-tests-async.sh` に保存し、`chmod +x` で実行可能にします。

```
#!/bin/bash
# run-tests-async.sh

# stdin からフック入力を読み取る
INPUT=$(cat)
FILE_PATH=$(echo "$INPUT" | jq -r '.tool_input.file_path // empty')

# ソース ファイルのみテストを実行
if [[ "$FILE_PATH" != *.ts && "$FILE_PATH" != *.js ]]; then
  exit 0
fi

# テストを実行し、additionalContext 経由で結果を Claude に報告
RESULT=$(npm test 2>&1)
EXIT_CODE=$?

if [ $EXIT_CODE -eq 0 ]; then
  MSG="Tests passed after editing $FILE_PATH"
else
  MSG="Tests failed after editing $FILE_PATH: $RESULT"
```

(次のページに続く)

(前のページからの続き)

```
fi
jq -nc --arg msg "$MSG" '{hookSpecificOutput: {hookEventName: "PostToolUse",␣
↪additionalContext: $msg}}'
```

次に、プロジェクト ルートの .claude/settings.json にこの設定を追加します。async: true フラグにより、Claude はテストの実行中に作業を続行できます。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/run-tests-async.sh",
            "args": [],
            "async": true
          }
        ]
      }
    ]
  }
}
```

21.7.4 制限事項

非同期フックは同期フックと比べていくつかの制約があります。

- フック出力は次の会話ターンで配信されます。セッションがアイドル状態の場合、レスポンスは次のユーザー操作まで待機します。例外: asyncRewake フックが終了コード 2 で終了すると、セッションがアイドル状態でも Claude を直ちに起動します。
- 各実行は個別のバックグラウンド プロセスを作成します。同じ非同期フックの複数の発火全体で重複排除はありません。

21.8 セキュリティに関する考慮事項

21.8.1 免責事項

警告:

コマンド フックはユーザー アカウントの完全な権限でシェル コマンドを実行します。ユーザー アカウントがアクセスできるファイルを変更、削除、またはアクセスできます。フック コマンドを設定に追加する前に、すべてのフック コマンドを確認してテストしてください。

21.8.2 ワークスペースの信頼

Claude Code は、設定ファイルのフックを実行する前にワークスペースの信頼を確認します。何が信頼済みとみなされるかは、セッションの種類によって異なります。

- **インタラクティブ セッション**: ユーザーがそのフォルダー、またはそのフォルダーにまで信頼が及ぶ親ディレクトリについて**ワークスペースの信頼ダイアログ** (ページ 1064) を承認するまで、Claude Code はユーザー自身の `~/.claude/settings.json` を含むすべての設定ファイルのフックを保留します
- **-p または SDK セッション**: Claude Code はダイアログを表示せず、フォルダーを信頼済みとして扱います。そのため、リポジトリの `.claude/settings.json` にコミットされたフックは、一度も信頼したことのないフォルダーでも実行されます

自分が作成していないリポジトリに対して `claude -p` をスクリプトで実行する前に、そのリポジトリの `.claude/` 設定ファイルを確認するか、*`--bare`* (ページ 696) で開始するか、`--settings '{"disableAllHooks": true}'` を使用して**その実行ではフックをオフにして** (ページ 466) ください。プロジェクト サブエージェントのフロントマター フックには、設定ファイルのフックよりも厳しいルールが適用されます。**フォルダーを信頼する前に実行されるもの** (ページ 1066) では、リポジトリのコンテンツの種類ごとにセッションの種類別の動作を示しています。

21.8.3 セキュリティ ベストプラクティス

フックを書くときは、これらのプラクティスに留意してください。

- **入力を検証およびサニタイズ**: 入力データを盲目的に信頼しないでください
- **常にシェル変数を引用**: `$VAR` ではなく `"$VAR"` を使用
- **パス トラバーサルをブロック**: ファイル パスで `..` をチェック
- **絶対パスを使用**: スクリプトの完全なパスを指定します。exec 形式では、`${CLAUDE_PROJECT_DIR}` を使用し、パスは引用符で囲む必要がありません。シェル形式では、ダブル クォートで囲みます
- **機密ファイルをスキップ**: `.env`、`.git/`、キーなどを避ける

21.9 Windows PowerShell ツール

Windows では、コマンド フックで `"shell": "powershell"` を設定することで、個別のフックを PowerShell で実行できます。Claude Code は `pwsh.exe` (PowerShell 7 以降の実行可能ファイル) を自動検出し、Windows PowerShell 5.1 の `powershell.exe` にフォールバックします。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "hooks": [
          {
            "type": "command",
```

(次のページに続く)

(前のページからの続き)

```
            "shell": "powershell",
            "command": "Write-Host 'File written'"
          }
        ]
      }
    ]
  }
}
```

PowerShell シェル形式のコマンドからプロジェクト ルートを参照するには、${CLAUDE_PROJECT_DIR} または $env:CLAUDE_PROJECT_DIR を記述します。Claude Code は settings.json、プラグイン、またはスキルで定義されているかどうかに関係なく、PowerShell シェル形式のコマンド内の ${CLAUDE_PROJECT_DIR}、${CLAUDE_PLUGIN_ROOT}、および ${CLAUDE_PLUGIN_DATA} プレースホルダーを PowerShell の ${env:NAME} 形式に書き換えます。PowerShell は解析後にエクスポートされた環境から値を解決するため、プレースホルダーはダブルクォート文字列内では機能しますが、PowerShell が変数を展開しないシングルクォート文字列内では機能しません。

PowerShell フックで裸の $CLAUDE_PROJECT_DIR スペルを記述しないでください。PowerShell はそれを未定義のローカル変数として解析し、$null に解決します。これにより、スクリプト パスがプロジェクト ルート プレフィックスなしで残されます。Claude Code はその形式を書き換えません。代わりに、**デバッグ ログ** (ページ 562) に警告を記録します。

以下の例は、$env: 形式でプロジェクト スクリプトを実行する settings.json フックを示しています。

```
{
  "type": "command",
  "shell": "powershell",
  "command": "& \"$env:CLAUDE_PROJECT_DIR\\.claude\\hooks\\check.ps1\""
}
```

21.10 フックをデバッグ

フック実行の詳細はデバッグ ログ ファイルに書き込まれます。claude --debug-file <path> で既知の場所にログを書き込むか、claude --debug を実行してログを ~/.claude/debug/<session-id>.txt で読み取ります。--debug フラグはターミナルに出力しません。

たとえば、Write に対する PostToolUse フックで、コマンドが hook-ran を出力する場合、次のようなエントリが生成されます。

```
2026-07-19T02:03:24.382Z [DEBUG] Hook output does not start with {, treating as plain␣
↪text
2026-07-19T02:03:24.382Z [DEBUG] "Hook PostToolUse:Write (PostToolUse) success:\nhook-
↪ran"
```

より詳細なフック マッチング詳細については、CLAUDE_CODE_DEBUG_LOG_LEVEL=verbose を設定して、フック matcher 数とクエリ マッチングなどの追加ログ行を確認します。

フックが発火しない、Stop フックが実行をブロックし続ける、または設定エラーなどの一般的な問題のトラブルシューティングについては、ガイドの**制限事項とトラブルシューティング** (ページ 437) を参照してください。`/context`、`/doctor`、および設定の優先順位をカバーするより広範な診断チュートリアルについては、設定をデバッグを参照してください。

第22章　プラグインを作成する

スキル、エージェント、フック、MCP サーバーで Claude Code を拡張するカスタムプラグインを作成します。

プラグインを使用すると、Claude Code をカスタム機能で拡張でき、プロジェクトとチーム全体で共有できます。このガイドでは、スキル、エージェント、フック、MCP サーバーを使用して独自のプラグインを作成する方法について説明します。

既存のプラグインをインストールしたいですか？**プラグインを検出してインストールする** (ページ 677) を参照してください。完全な技術仕様については、**プラグインリファレンス** (ページ 579) を参照してください。

22.1 プラグインとスタンドアロン設定を使い分ける

Claude Code では、カスタムスキル、エージェント、フックを追加する 2 つの方法をサポートしています。

アプローチ	スキル名	最適な用途
スタンドアロン（`.claude/` ディレクトリ）	`/hello`	個人的なワークフロー、プロジェクト固有のカスタマイズ、クイック実験
プラグイン（スキル、エージェント、フック、または `.claude-plugin/plugin.json` マニフェストを含む自己完結型ディレクトリ）	`/plugin-name:hello`	チームメンバーとの共有、コミュニティへの配布、バージョン管理されたリリース、プロジェクト全体で再利用可能

Tip:

`.claude/` でスタンドアロン設定を使用してクイック反復を行い、共有する準備ができたら**既存の設定をプラグインに変換** (ページ 575) してください。

22.2 クイックスタート

このクイックスタートでは、カスタムスキルを使用してプラグインを作成する手順を説明します。マニフェスト（プラグインを定義する設定ファイル）を作成し、スキルを追加して、`--plugin-dir` フラグを使用してローカルでテストします。

22.2.1 前提条件

- Claude Code **インストール済みで認証済み** (ページ 11)

22.2.2 最初のプラグインを作成する

1. **プラグインディレクトリを作成する**

   すべてのプラグインは、スキル、エージェント、またはフックを含む独自のディレクトリに存在し、オプションで `.claude-plugin/plugin.json` マニフェストと一緒に配置されます。このクイックスタートではテストステップで `--plugin-dir` を使用して Claude Code をディレクトリに指すため、場所は重要ではありません。スクラッチフォルダやプロジェクトディレクトリなど、便利な場所に作成してください。

   ```
   mkdir my-first-plugin
   ```

   残りのステップは親ディレクトリから実行され、`my-first-plugin/...` のようなパスを相対的に参照します。

2. **プラグインマニフェストを作成する**

   `.claude-plugin/plugin.json` のマニフェストファイルは、プラグインの ID（名前、説明、バージョン）を定義します。Claude Code はこのメタデータを使用して、プラグインマネージャーにプラグインを表示します。

   プラグインフォルダ内に `.claude-plugin` ディレクトリを作成します。

   ```
   mkdir my-first-plugin/.claude-plugin
   ```

   次に、このコンテンツで `my-first-plugin/.claude-plugin/plugin.json` を作成します。

   リスト 1: my-first-plugin/.claude-plugin/plugin.json

   ```
   {
     "name": "my-first-plugin",
     "description": "A greeting plugin to learn the basics",
     "version": "1.0.0",
     "author": {
       "name": "Your Name"
     }
   }
   ```

フィールド	目的
`name`	一意の識別子とスキル名前空間。スキルにはこれが接頭辞として付きます（例：`/my-first-plugin:hello`）。
`description`	プラグインマネージャーでプラグインを参照またはインストールするときに表示されます。

次のページに続く

表 2 – 前のページからの続き

フィールド	目的
version	オプション。設定されている場合、ユーザーはこのフィールドをバンプしたときにのみ更新を受け取ります。*command* **ソース** (ページ 647) を除きます。**プラグインをその場で読み込む** (ページ 605) 場合も除きます。**バージョン管理** (ページ 624) を参照してください。省略された場合、バージョンは**バージョン管理** (ページ 624) の次のソースから取得されます。
author	オプション。属性に役立ちます。

homepage、repository、license などの追加フィールドについては、**完全なマニフェストスキーマ** (ページ 592) を参照してください。

3. **スキルを追加する**

スキルは skills/ ディレクトリに存在します。各スキルは SKILL.md ファイルを含むフォルダです。フォルダ名がスキル名になり、プラグインの名前空間が接頭辞として付きます（my-first-plugin という名前のプラグイン内の hello/ は /my-first-plugin:hello を作成します）。

プラグインフォルダ内にスキルディレクトリを作成します。

```
mkdir -p my-first-plugin/skills/hello
```

次に、このコンテンツで my-first-plugin/skills/hello/SKILL.md を作成します。

リスト 2: my-first-plugin/skills/hello/SKILL.md

```
---
description: Greet the user with a friendly message
disable-model-invocation: true
---

Greet the user warmly and ask how you can help them today.
```

4. **プラグインをテストする**

--plugin-dir フラグを使用して Claude Code を実行し、プラグインを読み込みます。

```
claude --plugin-dir ./my-first-plugin
```

Claude Code が起動したら、新しいスキルを試してください。

```
/my-first-plugin:hello
```

Claude がグリーティングで応答します。/help を実行して、**カスタムコマンド**タブを開き、プラグイン名前空間の下にリストされたスキルを確認してください。

注釈:

名前空間を使う理由は？ プラグインスキルは常に名前空間が付きます（/my-first-plugin:hello など）。複数のプラグインが同じ名前のスキルを持つ場合の競合を防ぐためです。
名前空間プレフィックスを変更するには、plugin.json の name フィールドを更新してください。

5. **スキル引数を追加する**

   $ARGUMENTS プレースホルダーを使用してユーザー入力をキャプチャすることで、スキルを動的にします。

   SKILL.md ファイルを更新します。

リスト 3: my-first-plugin/skills/hello/SKILL.md

```
---
description: Greet the user with a personalized message
---

# Hello Skill

Greet the user named "$ARGUMENTS" warmly and ask how you can help them today.␣
↪Make the greeting personal and encouraging.
```

   /reload-plugins を実行して変更を反映させ、スキルを名前で試してください。

```
/my-first-plugin:hello Alex
```

   Claude があなたを名前で挨拶します。スキルに引数を渡す方法の詳細については、**スキル** (ページ 299) を参照してください。

Tip:

--plugin-dir フラグは開発とテストに役立ちます。プラグインを他のユーザーと共有する準備ができたら、**プラグインマーケットプレイスを作成して配布する** (ページ 627) を参照してください。

22.3 スキルディレクトリでプラグインを開発する

毎回起動時に --plugin-dir を渡す代わりに、スキルディレクトリにプラグインを保持して、Claude Code に自動的に読み込ませることができます。claude plugin init がスキャフォルドします。

```
claude plugin init my-tool
```

これにより、.claude-plugin/plugin.json マニフェストとスターター SKILL.md を含む ~/.claude/skills/my-tool/ が作成されます。次のセッションでは、マーケットプレイスやインストール手順なしで my-tool@skills-dir として読み込まれます。

自動読み込みルール、個人スコープ対プロジェクトスコープ、ワークスペース信頼要件、および更新または削除方法については、**スキルディレクトリプラグイン** (ページ 589) を参照してください。

22.4 プラグイン構造の概要

スキルを使用してプラグインを作成しましたが、プラグインにはさらに多くの機能を含めることができます。カスタムエージェント、フック、MCP サーバー、LSP サーバー、バックグラウンドモニターです。

警告:

よくある間違い：`commands/`、`agents/`、`skills/`、`hooks/` を `.claude-plugin/` ディレクトリ内に配置しないでください。`plugin.json` のみが `.claude-plugin/` 内に入ります。他のすべてのディレクトリはプラグインルートレベルにある必要があります。
プラグインルートは個別プラグイン自体のディレクトリです。例えば、**クイックスタート** (ページ 565) の `my-first-plugin/` のようなものです。`~/.claude/` ではありません。例えば、Claude Code は `~/.claude/.mcp.json` に配置された `.mcp.json` を読み込みません。

ディレクトリ	場所	目的
`.claude-plugin/`	プラグインルート	`plugin.json` マニフェストを含みます（コンポーネントがデフォルトの場所を使用する場合はオプション）
`skills/`	プラグインルート	`<name>/SKILL.md` ディレクトリとしてのスキル
`commands/`	プラグインルート	フラットな Markdown ファイルとしてのスキル。新しいプラグインには `skills/` を使用してください
`agents/`	プラグインルート	カスタムエージェント定義
`hooks/`	プラグインルート	`hooks.json` のイベントハンドラー
`.mcp.json`	プラグインルート	MCP サーバー設定
`.lsp.json`	プラグインルート	コード インテリジェンス用の LSP サーバー設定
`monitors/`	プラグインルート	`monitors.json` のバックグラウンドモニター設定
`bin/`	プラグインルート	プラグインが有効になっている間に Bash ツールの `PATH` に追加される実行可能ファイル。*Claude.ai* **組織設定を通じて配布するプラグインにはこのディレクトリを含めることはできません** (ページ 655)
`settings.json`	プラグインルート	プラグインが有効になったときに適用されるデフォルト**設定** (ページ 711)

正確に 1 つのスキルを含むプラグインは、`skills/` ディレクトリを作成する代わりに、`SKILL.md` をプラグインルートに直接配置できます。Claude Code はそれを単一のスキルとして読み込み、フロントマター `name` フィールドを呼び出し名として使用します。複数のスキルに成長する可能性があるプラグインには、`skills/` レイアウトを使用してください。

22.5 より複雑なプラグインを開発する

基本的なプラグインに慣れたら、より高度な拡張機能を作成できます。

22.5.1 プラグインに Skills を追加する

プラグインは *Agent Skills* (ページ 275) を含めることで、Claude の機能を拡張できます。Skills はモデルが呼び出すもので、Claude はタスクのコンテキストに基づいて自動的に使用します。

プラグインのルートに `skills/` ディレクトリを追加し、`SKILL.md` ファイルを含む Skill フォルダを配置します。

```
my-plugin/
├──── .claude-plugin/
│     └──── plugin.json
└──── skills/
      └──── code-review/
            └──── SKILL.md
```

各 `SKILL.md` には YAML フロントマターと説明が含まれます。Claude がいつ Skill を使用するかを知るために `description` を含めます。

```
---
description: Reviews code for best practices and potential issues. Use when reviewing␣
↪code, checking PRs, or analyzing code quality.
---

When reviewing code, check for:
1. Code organization and structure
2. Error handling
3. Security concerns
4. Test coverage
```

プラグインをインストール後、インストール概要を確認します。`Run /reload-plugins to activate.` と表示される場合は、**プラグインの変更を再起動なしで適用する** (ページ 689) を参照して、現在のセッションで Skills を読み込みます。段階的な情報開示とツール制限を含む完全な Skill 作成ガイダンスについては、*Agent Skills* (ページ 275) を参照してください。

22.5.2 プラグインに LSP サーバーを追加する

Tip:

TypeScript、Python、Rust などの一般的な言語については、公式マーケットプレイスから事前構築された LSP プラグインをインストールしてください。カスタム LSP プラグインは、まだカバーされていない言語のサポートが必要な場合にのみ作成してください。

LSP（Language Server Protocol）プラグインは Claude にリアルタイムのコード インテリジェンスを提供します。公式 LSP プラグインがない言語をサポートする必要がある場合は、プラグインに `.lsp.json` ファイルを追加することで、独自のプラグインを作成できます。

リスト 4: .lsp.json

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

プラグインをインストールするユーザーは、言語サーバーのバイナリをマシンにインストールしておく必要があります。

サーバーが起動することを確認するには、プラグインを有効にして Claude Code を起動し、`/plugin` Errors タブを確認します。起動に失敗した言語サーバーはそこに表示されます。例えば、バイナリがインストールされていない場合は `Executable not found in $PATH` と表示されます。無効な設定を持つエントリはスキップされます。理由を確認するには `claude --debug` を実行してください。

完全な LSP 設定オプションについては、*LSP servers* (ページ 584) を参照してください。

22.5.3 プラグインにバックグラウンド モニターを追加する

バックグラウンド モニターを使用すると、プラグインはログ、ファイル、または外部ステータスをバックグラウンドで監視し、イベントが到着したときに Claude に通知できます。Claude Code はプラグインがアクティブな場合、各モニターを自動的に起動するため、Claude にウォッチを開始するよう指示する必要はありません。

プラグインのルートに `monitors/monitors.json` ファイルを追加し、モニター エントリの配列を含めます。

リスト 5: monitors/monitors.json

```
[
  {
    "name": "error-log",
    "command": "tail -F ./logs/error.log",
    "description": "Application error log"
  }
]
```

`command` からの各 stdout 行は、セッション中に Claude への通知として配信されます。`when` トリガーと変数置換を含む完全なスキーマについては、*Monitors* (ページ 587) を参照してください。

22.5.4 プラグインでデフォルト設定を配布する

プラグインはプラグインのルートに `settings.json` ファイルを含めて、プラグインが有効になったときにデフォルト設定を適用できます。現在、`agent` と `subagentStatusLine` キーのみがサポートされています。

`agent` を設定すると、プラグインの *custom agents* (ページ 317) の 1 つがメイン スレッドとしてアクティブになり、そのシステム プロンプト、ツール制限、およびモデルが適用されます。これにより、プラグインは有効になったときに Claude Code のデフォルトの動作を変更できます。

リスト 6: settings.json

```
{
  "agent": "security-reviewer"
}
```

この例は、プラグインの `agents/` ディレクトリで定義された `security-reviewer` エージェントをアクティブにします。`settings.json` の設定は、`plugin.json` で宣言された `settings` よりも優先されます。不明なキーは無視されます。

22.5.5 複雑なプラグインを整理する

多くのコンポーネントを持つプラグインの場合、機能別にディレクトリ構造を整理します。完全なディレクトリ レイアウトと整理パターンについては、*Plugin directory structure* (ページ 608) を参照してください。

22.5.6 プラグインをローカルでテストする

`--plugin-dir` フラグを使用して、開発中にプラグインをテストします。これにより、インストールを必要とせずにプラグインを直接読み込みます。

```
claude --plugin-dir ./my-plugin
```

このフラグはプラグイン ディレクトリの `.zip` アーカイブも受け入れます。

```
claude --plugin-dir ./my-plugin.zip
```

`--plugin-dir` プラグインがインストール済みのマーケットプレイス プラグインと同じ名前を持つ場合、そのセッションではローカル コピーが優先されます。これにより、最初にアンインストールしなくても、既にインストール済みのプラグインへの変更をテストできます。例外は、管理設定によって強制的に有効にされたまたは強制的に無効にされたプラグインです。`--plugin-dir` はそれらをオーバーライドできません。

プラグインに変更を加えると、`/reload-plugins` を実行して、再起動せずに更新を取得します。これにより、プラグイン、Skills、エージェント、hooks、プラグイン MCP サーバー、およびプラグイン LSP サーバーが再読み込みされます。インタラクティブ ターミナルのないセッションでは、プラグイン MCP サーバーの変更は **次のセッションまで待機** (ページ 689) します。プラグイン コンポーネントをテストします。

- `/plugin-name:skill-name` で Skills を試す
- エージェントが `/context` の Custom Agents に表示されるか、またはスコープ付き名で @-mention できるかを確認する

- `PostToolUse` hook の場合は Claude にファイルを編集するよう求めるなど、各 hook が一致するイベントをトリガーし、その効果を確認する。Claude Code は、一致した hooks、終了コード、および出力を *debug log* (ページ 562) に記録します。

Tip:

複数のプラグインを一度に読み込むには、フラグを複数回指定します。

```
claude --plugin-dir ./plugin-one --plugin-dir ./plugin-two
```

プラグインとそれが依存するプラグインをテストするには、プラグインとその依存関係をローカルでテストする を参照してください。

フラグを追加できないセッションでプラグインを読み込むには、`CLAUDE_CODE_PLUGIN_DIRS` 環境変数にそれらの絶対パスをリストします。Claude Code は各パスを `--plugin-dir` パスとして読み込みます。これらのプラグインは、`--plugin-dir` で渡したものに加えて読み込まれます。プロジェクトとローカル設定はこの変数を設定できません。`CLAUDE_CODE_PLUGIN_DIRS` には Claude Code v2.1.280 以降が必要です。

`--plugin-dir` でプラグインを試すことで、それが機能することがわかります。Claude が実際にどのくらいの頻度でそれに到達し、正しい結果を得るかを確認するには、`claude plugin eval` を使用してテストプロンプトのセットに対して実行します。各プロンプトはプラグインが読み込まれた状態と読み込まれていない状態で複数回実行されるため、プラグインが何を貢献しているかを確認し、プラグインを変更したときまたは新しいモデルがリリースされたときの回帰を検出できます。

複数のプラグインを 1 つの場所から読み込むには、それらを保持するフォルダを渡します（例：`--plugin-dir ./plugins`）。フォルダからプラグインを読み込むには Claude Code v2.1.265 以降が必要です。Claude Code はフォルダのトップ レベルを読み取り、どのプラグインを読み込むかを決定し、インタラクティブ セッションではフォルダの後の変更も監視します。

- **読み込まれるもの**: フォルダにマニフェストまたはプラグイン コンポーネントがトップ レベルにない場合、Claude Code はそれをプラグインのフォルダとして扱います。`.claude-plugin/plugin.json` マニフェストを持つ各直下のサブフォルダは、別のプラグインとして読み込まれます。Claude Code はフォルダ内の他のすべてをスキップします。マニフェストのないプラグインを含め、エラーを報告せずにスキップします。
- **インタラクティブ セッション中の変更**: 追加したサブフォルダは、マニフェストが配置されると新しいプラグインとして読み込まれ、サブフォルダを削除するとそのプラグインがアンロードされます。Claude Code は各変更についてセッションに行を出力します。変更を会話の途中で適用すると プロンプト キャッシュが無効になる 場合、Claude Code はそれを保持し、行は `/reload-plugins` を実行して適用するよう指示します。

既に `.zip` アーカイブとしてパッケージ化され、CI ビルド アーティファクトなどの URL でホストされているプラグインをテストするには、代わりに `--plugin-url` を使用します。Claude Code は起動時にアーカイブをフェッチし、そのセッションのみ読み込みます。Claude Code がアーカイブをフェッチできない場合、またはアーカイブが無効な場合、プラグインなしで起動し、`/plugin` マネージャーの **Errors** タブで確認できるプラグイン読み込みエラーを記録します。同じ **信頼に関する考慮事項** (ページ 692) が、任意のプラグイン ソースに適用されます。このフラグは、制御または信頼するアーカイブのみを指します。

複数のプラグインを読み込むには、各 URL に対してフラグを繰り返します。

```
claude --plugin-url https://example.com/my-plugin.zip --plugin-url https://example.
↪com/other.zip
```

または、スペース区切りの URL を 1 つの引用符付き引数として渡します。

```
claude --plugin-url "https://example.com/my-plugin.zip https://example.com/other.zip"
```

22.5.7 プラグインの問題をデバッグする

プラグインが期待どおりに機能していない場合：

1. **構造を確認する**: ディレクトリが `.claude-plugin/` 内ではなく、プラグイン ルートにあることを確認します。
2. **コンポーネントを個別にテストする**: 各 Skill、エージェント、および hook を個別に確認します。
3. **検証とデバッグ ツールを使用する**: CLI コマンドとトラブルシューティング技術については、*Debugging and development tools* (ページ 622) を参照してください。

22.5.8 プラグインを共有する

プラグインを共有する準備ができたら：

1. **ドキュメントを追加する**: インストールと使用方法の説明を含む `README.md` を含めます。
2. **バージョン管理戦略を選択する**: 明示的な `version` を設定するか、*version management* (ページ 624) で説明されているフォールバックに依存するかを決定します。
3. **マーケットプレイスを作成または使用する**: *plugin marketplaces* (ページ 627) を通じて配布してインストールします。
4. **他の人でテストする**: より広い配布の前に、チーム メンバーにプラグインをテストしてもらいます。

プラグインがマーケットプレイスに登録されたら、他のユーザーは *Discover and install plugins* (ページ 677) の説明を使用してインストールできます。プラグインをチーム内に保つには、*private repository* (ページ 653) でマーケットプレイスをホストします。

22.5.9 プラグインをコミュニティ マーケットプレイスに送信する

Anthropic は Claude Code プラグイン用に 2 つの公開マーケットプレイスを管理しています。

- `claude-plugins-official`: Anthropic によって管理されるキュレーションされたプラグイン セット。Claude Code は初めて対話的に Claude Code を起動するときに自動的に登録します。初回の対話的な起動の前に Claude Code を非対話的に実行した場合、または *marketplace policy* (ページ 658) が以前の試行をブロックした場合は、`claude plugin marketplace add anthropics/claude-plugins-official` で自分で登録します。
- `claude-community`: レビュー後にサードパーティの送信が行われる公開コミュニティ マーケットプレイス。ユーザーは `/plugin marketplace add anthropics/claude-plugins-community` で追加し、`@claude-community` としてインストールします。

コミュニティ マーケットプレイスのレビューのためにプラグインを送信するには、アプリ内フォームの 1 つを使用します。

- **claude.ai**: claude.ai/admin-settings/directory/submissions/plugins/new

- **Console**: platform.claude.com/plugins/submit

claude.ai フォームには Team または Enterprise 組織とディレクトリ管理アクセスが必要です。組織の所有者はデフォルトでこのアクセス権を持っています。Team または Enterprise 組織に属していない個別の作成者は、代わりに Console フォームを使用できます。

送信する前に、`claude plugin validate ./your-plugin` をローカルで実行します。`./your-plugin` をプラグイン ディレクトリへのパスに置き換えます。レビュー パイプラインはすべての送信に対して同じチェックを実行し、自動化されたセーフティ スクリーニングも実行します。検証が成功すると、Claude Code は ✓ `Validation passed` を出力するか、警告がある場合は ✓ `Validation passed with warnings` を出力します。警告は検証を失敗させません。警告をエラーとして扱うには `--strict` を追加します。

承認されたプラグインは `anthropics/claude-plugins-community` カタログの特定のコミット SHA にピン留めされ、CI はリポジトリに新しいコミットをプッシュするときに自動的にピンをバンプします。公開カタログは毎晩レビュー パイプラインから同期されるため、承認と `marketplace.json` にプラグインが表示されるまでの間に遅延が生じる可能性があります。プラグインがインストール可能かどうかを確認するには、community catalog でその名前を検索します。

公式マーケットプレイス `claude-plugins-official` は別途キュレーションされています。Anthropic は、どのプラグインを含めるかを裁量で決定します。申請プロセスはなく、送信フォームは公式マーケットプレイスにプラグインを追加しません。

Anthropic がプラグインを公式マーケットプレイスにリストしている場合、CLI は Claude Code ユーザーにインストールを促すことができます。CLI からプラグインを推奨する を参照してください。

22.6 既存の設定をプラグインに変換する

`.claude/` ディレクトリにスキルまたはフックが既にある場合は、それらをプラグインに変換して、より簡単に共有および配布できます。

22.6.1 移行手順

1. **プラグイン構造を作成する**

   プロジェクトルートに新しいプラグインディレクトリを作成します。既存の `.claude/` フォルダの隣に配置することで、次のステップの相対 `cp` パスが解決されます。

   ```
   mkdir -p my-plugin/.claude-plugin
   ```

   `my-plugin/.claude-plugin/plugin.json` にマニフェストファイルを作成します。

   リスト 7: my-plugin/.claude-plugin/plugin.json

   ```
   {
     "name": "my-plugin",
     "description": "Migrated from standalone configuration",
     "version": "1.0.0"
   }
   ```

2. **既存のファイルをコピーする**

   既存の各設定ディレクトリをプラグインルートにコピーします。3 つすべてがない場合もあります。ディレクトリが存在しない場合、cp は `No such file or directory` を出力してコピーしないため、そのコマンドをスキップするか、エラーを無視してください。

```
cp -r .claude/commands my-plugin/

cp -r .claude/agents my-plugin/

cp -r .claude/skills my-plugin/
```

   プラグインには、`.claude/` の下にあったディレクトリのコピーが含まれるようになりました。`ls my-plugin` を実行して確認します。コピーした各ディレクトリが表示されるはずです。

3. **フックを移行する**

   設定にフックがある場合は、フックディレクトリを作成します。

```
mkdir my-plugin/hooks
```

   `my-plugin/hooks/hooks.json` をフック設定で作成します。`.claude/settings.json` または `settings.local.json` から `hooks` オブジェクトをコピーします。形式は同じです。コマンドはフック入力を stdin で JSON として受け取るため、`jq` を使用してファイルパスを抽出します。

リスト 8: my-plugin/hooks/hooks.json

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [{ "type": "command", "command": "jq -r '.tool_input.file_path'␣
↪| xargs npm run lint:fix" }]
      }
    ]
  }
}
```

4. **移行したプラグインをテストする**

   プラグインを読み込んで、すべてが機能することを確認します。

```
claude --plugin-dir ./my-plugin
```

   各コンポーネントをテストします。コマンドを実行し、`/context` にエージェントが表示されることを確認し、フックが一致するイベントをトリガーして、その効果を確認します。Claude Code は、どのフックが一致し、どのように終了したかを **デバッグログ** (ページ 562) に記録します。

22.6.2 移行時の変更点

スタンドアロン（.claude/）	プラグイン
1 つのプロジェクトでのみ利用可能	マーケットプレイス経由で共有可能
.claude/commands/ 内のファイル	plugin-name/commands/ 内のファイル
settings.json のフック	hooks/hooks.json のフック
共有するには手動でコピーする必要がある	/plugin install でインストール

注釈:

移行後、重複を避けるために .claude/ から元のファイルを削除してください。プロジェクトおよびユーザーの .claude/agents/ 定義は、同じ名前のプラグインエージェントをオーバーライドするため、元のファイルを削除した後にのみプラグインバージョンが有効になります。プラグインスキルは /plugin-name:skill-name として名前空間化されるため、元の /skill-name とプラグインコピーの両方が利用可能なままになり、一方が他方をオーバーライドするのではなく両方が共存します。

22.7 次のステップ

Claude Code のプラグインシステムを理解したので、異なる目標のための推奨パスを以下に示します。

22.7.1 プラグインユーザー向け

- **プラグインを検出してインストールする** (ページ 677)：マーケットプレイスを参照してプラグインをインストール
- **チームマーケットプレイスを設定する** (ページ 691)：チーム用のリポジトリレベルプラグインを設定

22.7.2 プラグイン開発者向け

- evals でプラグインをテストする：プラグインが何を変更するかを測定し、CI でゲートする
- **マーケットプレイスを作成して配布する** (ページ 627)：プラグインをパッケージ化して共有
- **プラグインリファレンス** (ページ 579)：完全な技術仕様
- 特定のプラグインコンポーネントをさらに詳しく調べる：
  - *Skills* (ページ 275)：スキル開発の詳細
  - *Subagents* (ページ 317)：エージェント設定と機能
  - *Hooks* (ページ 443)：イベント処理と自動化
  - *MCP* (ページ 363)：外部ツール統合

第23章　プラグインリファレンス

Claude Code プラグインシステムの完全な技術リファレンス。スキーマ、CLI コマンド、コンポーネント仕様を含みます。

Tip:

プラグインをインストールしたいですか？「**プラグインの検出とインストール** (ページ 677)」を参照してください。プラグインの作成については、「**プラグイン** (ページ 565)」を参照してください。プラグインの配布については、「**プラグインマーケットプレイス** (ページ 627)」を参照してください。

プラグインは、Claude Code をカスタム機能で拡張する自己完結型のコンポーネントディレクトリです。プラグインコンポーネントには、skills、agents、hooks、MCP servers、LSP servers、および monitors が含まれます。

23.1 プラグインコンポーネントリファレンス

23.1.1 Skills

プラグインは Claude Code に skills を追加し、`/name` ショートカットを作成します。これらは、ユーザーまたは Claude が呼び出すことができます。

場所: プラグインルートの `skills/` または `commands/` ディレクトリ、またはプラグインルートの単一の `SKILL.md` ファイル

ファイル形式: Skills はディレクトリで `SKILL.md` を含みます。commands はシンプルな markdown ファイルです

Skill の構造:

```
skills/
├── pdf-processor/
│   ├── SKILL.md
│   ├── reference.md (optional)
│   └── scripts/ (optional)
└── code-reviewer/
    └── SKILL.md
```

Skills と commands は、プラグインがインストールされると自動的に検出されます。

プラグインに `skills/` ディレクトリがなく、`skills` マニフェストフィールドもない場合、プラグインルートの `SKILL.md` は単一の skill として読み込まれます。frontmatter の `name` フィールドを設定して、skill の呼び出し名を制御します。これがない場合、Claude Code はインストールディレクトリ名にフォールバック

します。**キャッシュにコピーされたプラグイン** (ページ 605) の場合、その名前は更新のたびに変わるバージョン文字列です。複数の skill を含むプラグインの場合は、上記の `skills/` ディレクトリレイアウトを使用します。

プラグイン skills と commands では、`disable-model-invocation` などのブール値 frontmatter フィールドが、`true` と `false` に加えて、任意の大文字小文字で `yes`、`no`、`on`、`off`、`1`、`0` を受け入れます。v2.1.218 より前では、Claude Code は `true` と `false` のみを認識していました。

詳細については、*Skills* (ページ 275) を参照してください。

23.1.2 Agents

プラグインは、Claude が必要に応じて自動的に呼び出すことができる特定のタスク用の特化したサブエージェントを提供できます。

場所: プラグインルートの `agents/` ディレクトリ

ファイル形式: エージェント機能を説明する markdown ファイル

エージェント構造:

```
---
name: agent-name
description: このエージェントが専門とする内容と Claude がそれを呼び出すべき時期
model: sonnet
effort: medium
maxTurns: 20
disallowedTools: Write, Edit
---

エージェントの役割、専門知識、および動作を説明する詳細なシステムプロンプト。
```

プラグインエージェント frontmatter

プラグインエージェントファイルは、**サブエージェントファイルと同じ *frontmatter* フィールド** (ページ 326) を使用しますが、Claude Code はプラグインから来たエージェントの場合、そのうちのいくつかのみを尊重します。

- **サポート対象**: `name`、`description`、`model`、`effort`、`maxTurns`、`tools`、`disallowedTools`、`skills`、`memory`、`background`、`omitClaudeMd`、`isolation`、`color`、および `experimental`。唯一の有効な `isolation` 値は `"worktree"` です。
- **セキュリティ上の理由からサポート対象外**: `hooks`、`mcpServers`、および `permissionMode`。Claude Code はプラグインからエージェントを読み込む場合、これらを無視します。これらを使用するには、エージェントファイルを `.claude/agents/` または `~/.claude/agents/` にコピーします。
- **サポート対象外**: `initialPrompt`。

プラグインエージェントファイルを `agents/` のサブフォルダーに配置できます。Claude Code は **それらを再帰的に読み込み** (ページ 321)、プラグイン名、各サブフォルダー名、およびファイル名をコロンで結合して、エージェントのスコープ付き名を形成します。たとえば、`my-plugin` という名前のプラグイン内の

`agents/review/security.md` は `my-plugin:review:security` として読み込まれます。2 つの設定がその名前を変更します。

- Frontmatter `name`: ファイル名のみを置き換えるため、`agents/review/security.md` の `name: audit` は `my-plugin:review:audit` として読み込まれます
- マニフェスト *agents* (ページ 596) フィールド: そこにリストされているファイルはサブフォルダー名なしで読み込まれるため、`"agents": "./custom/review/security.md"` は `my-plugin:security` として読み込まれます

Claude Code は、frontmatter に `name` がない場合またはパースに失敗した場合でも、プラグインエージェントを読み込みます。

- `name` がない場合: Claude Code はファイル名に基づいてエージェントに名前を付けるため、`my-plugin` という名前のプラグイン内の `agents/reviewer.md` は `my-plugin:reviewer` として読み込まれます
- Frontmatter がパースに失敗した場合: Claude Code はファイル名に基づいてエージェントに名前を付け、説明として `Agent from my-plugin plugin` を使用し、ファイル内のすべてのフィールドを無視します

対照的に、Claude Code は、frontmatter に `name` がない場合またはパースに失敗した場合、プロジェクト、ユーザー、または管理エージェントファイルをスキップします。

プラグインのデフォルト `agents/` ディレクトリ内で frontmatter がパースに失敗したファイルを見つけるには、`claude plugin validate` を実行します。渡すパスは、プラグインがマニフェストを持つかどうかによって異なり、両方の例では `./my-plugin` をプラグインディレクトリとして使用します。

- マニフェスト付きプラグイン: `claude plugin validate ./my-plugin`
- マニフェストなしプラグイン: `claude plugin validate ./my-plugin/agents`。Claude Code v2.1.233 以降が必要です。

エージェントは、プラグインが有効になると、*@-mention typeahead* (ページ 343) に `my-plugin:code-reviewer` などのスコープ付き名で表示されます。

詳細については、*Subagents* (ページ 317) を参照してください。

23.1.3 Hooks

プラグインは、Claude Code イベントに自動的に応答するイベントハンドラーを提供できます。

場所: プラグインルートの `hooks/hooks.json`、または plugin.json 内のインライン

形式: イベントマッチャーとアクションを含む JSON 設定

`hooks/hooks.json` は、エディターのオートコンプリートと検証用に JSON Schema URL を指定する最上位の `$schema` キーを含むことができます。Claude Code は読み込み時にこのキーを無視します。

Hook 設定:

```
{
  "hooks": {
    "PostToolUse": [
```

(次のページに続く)

(前のページからの続き)

```
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "\"${CLAUDE_PLUGIN_ROOT}\"/scripts/format-code.sh"
          }
        ]
      }
    ]
  }
}
```

プラグイン hooks は、**ユーザー定義** *hooks* (ページ 443) と同じライフサイクルイベントに応答します。

イベント	発火するタイミング
`SessionStart`	セッションが開始または再開されたとき
`Setup`	`--init-only` で Claude Code を起動するとき、または `-p` モードで `--init` または `--maintenance` を使用するとき。CI またはスクリプトでの 1 回限りの準備用
`UserPromptSubmit`	プロンプトを送信するとき、Claude が処理する前
`UserPromptExpansion`	ユーザーが入力したコマンドがプロンプトに展開されるとき、Claude に到達する前。展開をブロックできます
`PreToolUse`	ツール呼び出しが実行される前。ブロックできます
`PermissionRequest`	ツール呼び出しが権限決定を必要とするとき
`PermissionDenied`	オートモードがツール呼び出しを拒否するとき、分類器の判定がない拒否を含みます。JSON `hookSpecificOutput.retry: true` を使用して、モデルが拒否されたツール呼び出しを再試行できることを伝えます。Claude Code は分類器が判定を出さなかった場合、`retry` を無視します
`PostToolUse`	ツール呼び出しが成功した後
`PostToolUseFailure`	ツール呼び出しが失敗した後
`PostToolBatch`	並列ツール呼び出しの完全なバッチが解決した後、次のモデル呼び出しの前
`Notification`	Claude Code が通知を送信するとき
`MessageDisplay`	アシスタントメッセージテキストが表示されている間
`SubagentStart`	サブエージェントがスポーンされるとき
`SubagentStop`	サブエージェントが終了するとき
`TaskCreated`	`TaskCreate` 経由でタスクが作成されるとき
`TaskCompleted`	タスクが完了としてマークされるとき
`Stop`	Claude が応答を終了するとき
`StopFailure`	API エラーが原因でターンが終了するとき
`TeammateIdle`	エージェントチーム のチームメイトがアイドル状態になろうとするとき

次のページに続く

表 1 – 前のページからの続き

イベント	発火するタイミング
InstructionsLoaded	CLAUDE.md または .claude/rules/*.md ファイルがコンテキストに読み込まれるとき。セッション開始時およびセッション中にファイルが遅延読み込みされるときに発火します
ConfigChange	セッション中に設定ファイルが変更されるとき
CwdChanged	作業ディレクトリが変更されるとき、例えば Claude が cd コマンドを実行するとき。direnv などのツールを使用したリアクティブな環境管理に便利です
DirectoryAdded	/add-dir または SDK register_repo_root コントロールリクエスト経由でセッション中盤に作業ディレクトリが追加されるとき
FileChanged	監視対象ファイルがディスク上で変更されるとき。matcher フィールドは監視するファイル名を指定します
WorktreeCreate	--worktree、isolation: "worktree"、またはバックグラウンドセッション経由で worktree が作成されるとき。デフォルトの git 動作を置き換えます
WorktreeRemove	セッション終了時、サブエージェント終了時、またはバックグラウンドセッションを削除するときに worktree が削除されるとき
PreCompact	コンテキスト圧縮の前
PostCompact	コンテキスト圧縮が完了した後
PreModelSwitch	Claude Code があなたまたはクライアントがリクエストしたモデルスイッチを適用する前。スイッチをブロックできます
PostModelSwitch	セッションのモデルが変更された後、Claude Code が独自に行う変更（セッションを再開するときのモデル復元など）を含みます
Elicitation	MCP サーバーがツール呼び出し中にユーザー入力をリクエストするとき
ElicitationResult	ユーザーが MCP エリシテーションに応答した後、レスポンスがサーバーに送り返される前
SessionEnd	セッションが終了するとき

Hook タイプ:

- command: シェルコマンドまたはスクリプトを実行
- http: イベント JSON を URL への POST リクエストとして送信
- mcp_tool: 設定された *MCP* **サーバー** (ページ 363) 上のツールを呼び出す
- prompt: LLM でプロンプトを評価（コンテキスト用に $ARGUMENTS プレースホルダーを使用）
- agent: 複雑な検証タスク用にツール付きの agentic verifier を実行

プラグイン自身の **バンドルされた** *MCP* **サーバー** (ページ 584) をターゲットとする hooks は、スコープ付き名を使用する必要があります。ツールマッチャーと if フィールドはスコープ付きツール名 mcp__plugin_<plugin-name>_<server-name>__<tool> を取り、mcp_tool hook の server フィールドは plugin:<plugin-name>:<server-name> を取ります。ベアサーバーキーに対して記述されたマッチャーは発火しません。*MCP* **ツールをマッチ** (ページ 454) および **プラグイン提供** *MCP* **サーバー** (ページ 378) を参照してください。

23.1.4 MCP servers

プラグインは Model Context Protocol（MCP）サーバーをバンドルして、Claude Code を外部ツールおよびサービスに接続できます。

場所: プラグインルートの `.mcp.json`、または plugin.json 内のインライン

形式: 標準 MCP サーバー設定

MCP サーバー設定:

```
{
  "mcpServers": {
    "plugin-database": {
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
      "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"],
      "env": {
        "DB_PATH": "${CLAUDE_PLUGIN_ROOT}/data"
      }
    },
    "plugin-api-client": {
      "command": "npx",
      "args": ["@company/mcp-server", "--plugin-mode"]
    }
  }
}
```

統合動作:

- プラグイン MCP サーバーはプラグインが有効になると自動的に起動します
- サーバーは Claude のツールキット内の標準 MCP ツールとして表示されます
- プラグインサーバーはユーザー MCP サーバーとは独立して設定できます
- セッション中に */reload-plugins* (ページ 689) を実行する場合、Claude Code は設定が変わらないサーバーのライブ接続を保持します

23.1.5 LSP servers

Tip:

LSP プラグインを使用したいですか？公式マーケットプレイスからインストールしてください。`/plugin` Discover タブで「lsp」を検索してください。このセクションでは、公式マーケットプレイスでカバーされていない言語用の LSP プラグインを作成する方法を説明しています。

プラグインは Language Server Protocol（LSP）サーバーを提供して、コードベースで作業する際に Claude に **リアルタイムコード インテリジェンス** (ページ 678) を提供できます。

場所: プラグインルートの `.lsp.json`、または `plugin.json` 内のインライン

形式: 言語サーバー名をその設定にマップする JSON 設定

`.lsp.json` **ファイル形式**:

```
{
  "go": {
    "command": "gopls",
    "args": ["serve"],
    "extensionToLanguage": {
      ".go": "go"
    }
  }
}
```

`plugin.json` **内のインライン**:

```
{
  "name": "my-plugin",
  "lspServers": {
    "go": {
      "command": "gopls",
      "args": ["serve"],
      "extensionToLanguage": {
        ".go": "go"
      }
    }
  }
}
```

必須フィールド:

フィールド	説明
`command`	実行する LSP バイナリ（PATH に含まれている必要があります）
`extensionToLanguage`	ファイル拡張子を言語識別子にマップします

オプションフィールド:

フィールド	説明
`args`	LSP サーバーのコマンドライン引数
`transport`	通信トランスポート: `stdio`（デフォルト）または `socket`。Claude Code は `socket` を受け入れますが、すべてのサーバーを stdio 経由で実行するため、stdout プロトコルルールがすべてのサーバーに適用されます
`env`	サーバー起動時に設定する環境変数

次のページに続く

表 3 – 前のページからの続き

フィールド	説明
initializationOptions	初期化中にサーバーに渡されるオプション
settings	workspace/didChangeConfiguration 経由で渡される設定
workspaceFolder	サーバーのワークスペースフォルダーパス
startupTimeout	サーバー起動を待つ最大時間（ミリ秒）
shutdownTimeout	グレースフルシャットダウンを待つ最大時間（ミリ秒）。タイムアウトが経過すると、Claude Code はサーバープロセスを終了します。設定されていない場合、タイムアウトは適用されません
restartOnCrash	クラッシュ後にサーバーを再起動するかどうか。デフォルトは true。クラッシュしたサーバーを再起動する代わりに停止したままにするには false に設定します
maxRestarts	諦める前の最大再起動試行回数
diagnostics	編集後に診断を Claude のコンテキストにプッシュするかどうか（デフォルト true）。コード ナビゲーションは保持しながら自動診断注入を抑制するには false に設定します

restartOnCrash と shutdownTimeout には Claude Code v2.1.205 以降が必要です。v2.1.205 より前では、設定スキーマは両方のオプションを受け入れていましたが、どちらかを設定すると Claude Code はその LSP サーバーを起動時に完全にスキップしていました。理由は claude --debug 出力でのみ表示されます。

同じ拡張子の複数サーバー: 複数の有効な LSP サーバーが extensionToLanguage で同じファイル拡張子を宣言する場合、サーバーが 1 つのプラグインから来ているか異なるプラグインから来ているかに関わらず、最初に登録されたサーバーがその拡張子のファイルを処理し、他のサーバーは起動しません。/plugin インターフェイスは、アクティブなサーバーを持つプラグインに名前を付ける警告を表示します。

初期化に失敗したサーバー: Claude Code は、command または extensionToLanguage が見つからないなど、設定が無効なサーバーをスキップし、他の設定されたサーバーは起動します。claude --debug を実行して、サーバーがスキップされた理由を確認します。

スキップされたサーバーはそのファイル拡張子を要求しないため、同じ拡張子を宣言する別の有効なサーバー（同じプラグインまたは異なるプラグインから）がそれらのファイルを処理します。

ログ出力を stdout ではなく stderr に送信: Claude Code はサーバーの stdout をプロトコルメッセージとしてのみ読み取り、メッセージヘッダーは最大 64 KiB、メッセージボディは最大 32 MiB を受け入れます。Claude Code は、どちらかの制限を超えるか、非プロトコル出力を stdout に書き込むサーバーを切断し、その切断を restartOnCrash と maxRestarts のクラッシュとしてカウントします。--debug で実行する場合、Claude Code は原因に名前を付けるエラーをデバッグログに書き込みます。

警告:

言語サーバーバイナリを別途インストールする必要があります。 LSP プラグインは Claude Code が言語サーバーに接続する方法を設定しますが、サーバー自体は含まれていません。/plugin Errors タブに Executable not found in $PATH が表示される場合は、言語に必要なバイナリをインストールしてください。

利用可能な LSP プラグイン:

プラグイン	言語サーバー	インストールコマンド
`pyright-lsp`	Pyright（Python）	`pip install pyright` または `npm install -g pyright`
`typescript-lsp`	TypeScript Language Server	`npm install -g typescript-language-server typescript`
`rust-analyzer-lsp`	rust-analyzer	rust-analyzer インストール参照

言語サーバーをインストールしてから、マーケットプレイスからプラグインをインストールします。

23.1.6 Monitors

プラグインは、プラグインがアクティブな場合に Claude Code が自動的に起動するバックグラウンドモニターを宣言できます。各モニターはセッションの期間中シェルコマンドを実行し、すべての stdout 行を Claude に通知として配信するため、Claude は自分自身でウォッチを開始するよう求められることなく、ログエントリ、ステータス変更、またはポーリングイベントに反応できます。

プラグインモニターは Monitor ツール と同じメカニズムを使用し、その可用性制約を共有します。これらはインタラクティブ CLI セッションでのみ実行され、*hooks* (ページ 581) と同じ信頼レベルでサンドボックス化されずに実行され、Monitor ツールが利用できないホストではスキップされます。

場所: プラグインルートの `monitors/monitors.json`、または plugin.json 内のインライン

形式: モニターエントリの JSON 配列

次の `monitors/monitors.json` はデプロイメントステータスエンドポイントとローカルエラーログを監視します。

```
[
  {
    "name": "deploy-status",
    "command": "\"${CLAUDE_PLUGIN_ROOT}\"/scripts/poll-deploy.sh",
    "description": "Deployment status changes"
  },
  {
    "name": "error-log",
    "command": "tail -F ./logs/error.log",
    "description": "Application error log",
    "when": "on-skill-invoke:debug"
  }
]
```

モニターをインラインで宣言するには、`plugin.json` の `experimental.monitors` を同じ配列に設定します。デフォルト以外のパスから読み込むには、`experimental.monitors` を `"./config/monitors.json"` などの相対パス文字列に設定します。モニターは **実験的コンポーネント** (ページ 597) です。

必須フィールド:

フィールド	説明
name	プラグイン内で一意の識別子。プラグインが再読み込みされるか skill が再度呼び出されるときに重複プロセスを防ぎます
command	セッション作業ディレクトリで永続的なバックグラウンドプロセスとして実行されるシェルコマンド
description	監視対象の簡潔な説明。タスクパネルと通知サマリーに表示されます

オプションフィールド:

フィールド	説明
when	モニターが開始するタイミングを制御します。"always" はセッション開始時とプラグイン再読み込み時に開始し、デフォルトです。"on-skill-invoke:<skill-name>" はこのプラグイン内の名前付き skill が最初にディスパッチされるときに開始します

command 値は **パス置換** (ページ 602) ${CLAUDE_PLUGIN_ROOT}、${CLAUDE_PLUGIN_DATA}、および ${CLAUDE_PROJECT_DIR} をサポートしており、環境からの任意の ${ENV_VAR} もサポートしています。スクリプトがプラグイン自身のディレクトリから実行される必要がある場合は、コマンドの前に cd "${CLAUDE_PLUGIN_ROOT}" && を付けます。

モニター command は *${user_config.*}* (ページ 597) 値を参照できません。コマンドはシェルを通じて実行されるため、Claude Code は値を置換する代わりに エラー でモニターを拒否します。モニタープロセスは CLAUDE_PLUGIN_OPTION_<KEY> 環境変数を受け取らないため、モニタースクリプトが所有する設定ファイルから値を読み取ります。

セッション中にプラグインを無効にする場合、Claude Code は既に実行中のモニターを停止しません。セッションが終了するときに停止します。

23.1.7 Themes

プラグインは、/theme に組み込みプリセットおよびユーザーのローカルテーマと一緒に表示されるカラーテーマを配布できます。テーマは themes/ 内の JSON ファイルで、base プリセットとカラートークンのスパース overrides マップを持ちます。テーマは **実験的コンポーネント** (ページ 597) です。

```
{
  "name": "Dracula",
  "base": "dark",
  "overrides": {
    "claude": "#bd93f9",
    "error": "#ff5555",
    "success": "#50fa7b"
  }
}
```

ユーザーがプラグインテーマを選択すると、Claude Code は custom:<plugin-name>:<slug> をその設定に保存します。プラグインテーマは読み取り専用です。ユーザーが /theme でそれに対して Ctrl+E を押

すと、Claude Code はそれを ~/.claude/themes/ にコピーして、編集できるようにします。

23.2 プラグインのインストールスコープ

プラグインをインストールする際に、プラグインが利用可能な場所と他のユーザーが使用できるかどうかを決定する**スコープ**を選択します。

スコープ	設定ファイル	ユースケース
user	~/.claude/settings.json	すべてのプロジェクト全体で利用可能な個人用プラグイン（デフォルト）
project	.claude/settings.json	バージョン管理を通じて共有されるチームプラグイン
local	.claude/settings.local.json	プロジェクト固有のプラグイン。Claude Code が設定を保存する際に gitignore される
managed	Managed settings	管理されたプラグイン（読み取り専用、更新のみ）

プラグインは、他の Claude Code 設定と同じスコープシステムを使用します。インストール手順とスコープフラグについては、**プラグインのインストール** (ページ 685) を参照してください。スコープの完全な説明については、**設定スコープ** (ページ 711) を参照してください。

23.3 スキルディレクトリプラグイン

スキルディレクトリの下にあるフォルダで .claude-plugin/plugin.json マニフェストを含むフォルダは、次のセッションで <name>@skills-dir という名前のプラグインとして読み込まれます。マーケットプレイスもインストール手順もありません。*plugin init* (ページ 610) でスキャフォルドできます。コピーされたマーケットプレイスインストールとは異なり、プラグインはプラグインキャッシュにコピーされるのではなく、その場で検出されます。

スキルディレクトリツリーは 3 つの異なるものをサポートしています。

内容	説明
マニフェストなしの <skills-dir>/foo/SKILL.md	foo という名前の通常の **スキル** (ページ 275)
<skills-dir>/foo/.claude-plugin/plugin.json	プラグイン foo@skills-dir。独自のスキル、エージェント、hooks などをバンドルできます
<plugin>/skills/bar/SKILL.md	プラグイン内にパッケージされたスキル bar

23.3.1 プラグインの読み込み元を選択する

スキルディレクトリ	スコープ	読み込み
~/.claude/skills/	個人	すべてのプロジェクトで読み込まれます。この場所はあなた自身のものだからです
<cwd>/.claude/skills/	プロジェクト	そのフォルダのワークスペース **信頼ダイアログ** (ページ 1066) を受け入れた後のみ

プロジェクトスコープのプラグインはリポジトリにチェックインされ、それをクローンしたすべての協力者に到達します。そのコンテンツはあなたではなくリポジトリから来ているため、.claude/settings.json のプロジェクト許可ルールを管理するのと同じ信頼ゲートの後にのみ読み込まれます。親フォルダを信頼したり -p で実行したりするだけでは不十分で、コードを実行するコンポーネントはさらに制限されます。

- 宣言する MCP サーバーはプロジェクト .mcp.json と同じ **サーバーごとの承認** (ページ 363) を通過します
- LSP サーバーはワークスペースを信頼した後にのみ開始します
- **バックグラウンドモニター** (ページ 587) は読み込まれません

個人スコープのプラグインにはこれらの制限はありません。

警告:

プロジェクトスコープの @skills-dir プラグインはセッションの **プライマリワーキングディレクトリ** (ページ 1060) の .claude/skills/ からのみ読み込まれます。通常のスキルとコマンドのように **リポジトリルートまで遡りません** (ページ 281)。そのため、サブディレクトリから起動するとリポジトリルートにあるプラグインが見つかりません。リポジトリルートから起動するか、*v2.1.246* **以降で** */cd* **でセッションをそこに移動** (ページ 1061) してください。

23.3.2 スキルディレクトリプラグインを編集、リロード、無効化する

スキルの SKILL.md に加えた変更は現在のセッションで即座に有効になります。プラグインの他のコンポーネント（hooks/、.mcp.json、agents/、output-styles/ など）への変更は有効になりません。/reload-plugins を実行するか Claude Code を再起動してそれらを反映させてください。**ライブ変更検出** (ページ 286) を参照してください。

スキルディレクトリプラグインの読み込みを停止するには、そのフォルダを削除するか、名前で無効化します。マーケットプレイスからインストールされていないため、uninstall ステップはありません。

```
claude plugin disable my-tool@skills-dir
```

23.4 claude.ai から同期されたプラグイン

Claude Code は claude.ai アカウント用に有効化されたプラグインを読み込みます。これには、組織がメンバー向けに有効化するプラグインと、マーケットプレイスからインストールするプラグインが含まれます。各プラグインを ~/.claude/plugins/synced/ にダウンロードし、<name>@synced として読み込みます。マーケットプレイスはなく、インストール記録もありません。同期されたプラグインは、インストールしたマーケットプレイスプラグインと同じ信頼レベルで実行されます。スキル、エージェント、フック、MCP サーバー、LSP サーバーはすべて読み込まれます。

Claude Code がこれらのプラグインを同期する場所はセッションによって異なります。

- Cowork とクラウドセッションでは、Claude Code はセッション開始時にセッション独自の環境にダウンロードします。v2.1.239 より前では、Claude Code はこれらのプラグインを `<name>@inline` として読み込んでいました。これは `--plugin-dir` プラグインが使用する ID です。
- claude.ai アカウントでサインインするターミナルセッションでは、Claude Code は起動時にアカウントを 1 回チェックし、新しいプラグインと更新されたプラグインをダウンロードし、ユーザーまたは組織がオフにしたプラグインを削除します。すべてバックグラウンドで実行されます。ターミナルセッションでの同期には Claude Code v2.1.273 以降が必要です。

起動チェックはバックグラウンドで実行されるため、セッション開始後に完了することがあります。インタラクティブセッションで同期されたプラグインを追加、更新、または削除する場合、Claude Code は `Plugins changed. Run /reload-plugins to activate.` と表示します。*/reload-plugins* (ページ 689) を実行してそのセッションで変更を読み込むか、次回 Claude Code を起動するまで待つことができます。セッション実行中に claude.ai でプラグインを有効化した場合、Claude Code は次回起動時にダウンロードします。

ターミナルセッションでのプラグイン同期は、*claude.ai* **から同期されたスキル** (ページ 283) と同じサインイン条件下で実行されます。また、Claude Code がアカウントのプラグインにアクセスできるようにするサインインが必要です。

Claude Code の以前のバージョンからのサインインは、Claude Code がバックグラウンドでそのサインインを更新する次回（数時間以内）、または `/login` を再度実行した場合はすぐに、プラグインアクセスを取得します。その後、Claude Code を起動する次回にプラグイン同期が開始されます。

`claude plugin list` は同期されたプラグインを `Synced from claude.ai` という見出しの下に表示し、`/plugin` **Installed** タブはソースとして `synced` を使用してリストアップします。`claude plugin list` が出力する `<name>@synced` ID で同期されたプラグインを管理します。

- **1 つをオフにする**: `claude plugin disable <name>@synced` を実行するか、`/plugin` **Installed** タブから無効化します。Claude Code はこの選択をユーザーレベルの `enabledPlugins` に `"<name>@synced": false` として保存します。プラグインを再度オンにするには、`claude plugin enable <name>@synced` を実行します。
- **すべての場所から除外する**: *claude.ai* **アカウント用にプラグインをオフにします** (ページ 151)。すべての環境で 1 つのプロジェクトから除外するには、そのプロジェクトのコミットされた `.claude/settings.json` の `enabledPlugins` の下に `"<name>@synced": false` を設定します。
- **claude.ai でプラグイン自体を管理する**: `claude plugin install`、`update`、`uninstall` は同期されたプラグインには適用されません。Claude Code はプラグインの更新を次の同期時にダウンロードします。削除するには、claude.ai アカウント用にプラグインをオフにし、Claude Code は次の同期時に削除します。
- **マシンでの同期を停止する**: ユーザー設定で `syncClaudeAiPlugins` を `false` に設定します。Claude Code はダウンロードを停止し、次回起動時に既に同期したプラグインを `~/.claude/plugins/.trash/` に移動し、それ以上読み込みません。組織はマネージド設定で同じキーを設定するか、claude.ai でスキルをオフにすることができます。これはプラグインの同期も停止します。

組織が claude.ai で必須とマークしたプラグインをオフにすることはできません。Claude Code は以前無効化した場合でも読み込み、`claude plugin disable` は `Plugin "<name>@synced" is required by your organization and can't be disabled here. Contact your admin to change it.` で拒否します。`claude plugin list` では、これらのプラグインは `required by your org` とマークされています。

他のソースからの有効化されたプラグインが同期されたプラグインの名前と一致する場合、Claude Code はそのプラグインを読み込み、同期されたコピーが読み込まれていないと報告します。他のソースにはマーケットプレイスインストール、**スキルディレクトリプラグイン** (ページ 589)、`--plugin-dir` プラグイン、Claude Code に組み込まれたプラグインが含まれます。claude.ai のコピーを代わりに使用するには、独自のコピーを無効化します。v2.1.239 より前では、Claude Code は同じ名前のマーケットプレイスインストールの代わりに同期されたコピーを読み込んでいました。

23.5 プラグインマニフェストスキーマ

`.claude-plugin/plugin.json` ファイルはプラグインのメタデータと設定を定義します。

マニフェストはオプションです。省略した場合、Claude Code は**デフォルトの場所** (ページ 609) のコンポーネントを自動検出し、ディレクトリ名からプラグイン名を導出します。メタデータまたはカスタムコンポーネントパスを提供する必要がある場合は、マニフェストを使用してください。

23.5.1 完全なスキーマ

```
{
  "name": "plugin-name",
  "displayName": "Plugin Name",
  "version": "1.2.0",
  "description": "Brief plugin description",
  "author": {
    "name": "Author Name",
    "email": "author@example.com",
    "url": "https://github.com/author"
  },
  "homepage": "https://docs.example.com/plugin",
  "repository": "https://github.com/author/plugin",
  "license": "MIT",
  "keywords": ["keyword1", "keyword2"],
  "metadata": { "catalogId": "cat-123", "tier": "pro" },
  "skills": "./custom/skills/",
  "commands": ["./custom/commands/special.md"],
  "agents": ["./custom/agents/reviewer.md"],
  "hooks": "./config/hooks.json",
  "mcpServers": "./mcp-config.json",
  "outputStyles": "./styles/",
  "lspServers": "./.lsp.json",
  "experimental": {
    "themes": "./themes/",
    "monitors": "./monitors.json",
    "evals": "quality/evals"
  },
  "dependencies": [
    "helper-lib",
```

(次のページに続く)

(前のページからの続き)

```
    { "name": "secrets-vault", "version": "~2.1.0" }
  ]
}
```

23.5.2 必須フィールド

マニフェストを含める場合、name は唯一の必須フィールドです。

フィールド	型	説明	例
name	string	ケバブケースの一意の識別子。スペース、制御文字、双方向フォーマット文字を含みません。**マーケットプレイスエントリ** (ページ 633) がプラグインを別の名前でリストする場合、マーケットプレイスエントリ名が enabledPlugins キーと /plugin で使用されます	"deployment-tools"

この名前はコンポーネントの名前空間化に使用されます。たとえば、UI では、名前が plugin-dev のプラグインのエージェント agent-creator は plugin-dev:agent-creator として表示されます。

23.5.3 認識されないフィールド

Claude Code は認識しないトップレベルフィールドを無視します。別のエコシステムからのメタデータを plugin.json に保持でき、プラグインは引き続き読み込まれます。これにより、VS Code または Cursor 拡張マニフェスト、npm package.json、または MCPB/DXT バンドルマニフェストとして機能する 1 つのマニフェストを保守することが実用的になります。

claude plugin validate は認識されないフィールドを警告として報告し、エラーではありません。フィールドが認識されたフィールドから 1 文字または 2 文字異なる場合、警告は意図された名前を示唆します。認識されないフィールド警告のみを持つプラグインは検証に合格し、実行時に読み込まれます。

Claude Code が値の型が間違っている認識されたフィールドを処理する方法は、フィールドによって異なります。

- **ほとんどのフィールド**: プラグインは読み込みに失敗します。たとえば、文字列の代わりに配列である keywords 値は読み込みエラーであり、claude plugin validate はそれをエラーとして報告します。
- experimental と metadata: Claude Code は非オブジェクト値を無視し、claude plugin validate は警告を報告します。

--strict を渡して、警告をエラーとして扱います。CI で使用して、公開前に別のツールのマニフェストから残された綴り間違いのフィールド名またはフィールドをキャッチします。プラグインは実行時に読み込まれますが。

```
claude plugin validate ./my-plugin --strict
```

23.5.4 メタデータフィールド

フィールド	型	説明	例
`$schema`	string	エディタのオートコンプリートと検証用の JSON Schema URL。Claude Code は読み込み時にこのフィールドを無視します。	`"https://json.schemastore.org/claude-code-plugin-manifest.json"`
`displayName`	string	`/plugin` ピッカーおよび他の UI サーフェスに表示される人間が読める名前。マーケットプレイスにインストールされたプラグインの場合、**マーケットプレイスエントリ** (ページ 634) の `displayName` はこの値より優先されます。どちらの場所にも表示名が設定されていない場合、ユーザーは `name` を見ます。`name` とは異なり、スペースと任意の大文字小文字を含むことができます。名前空間化またはルックアップには使用されません。	`"Deployment Tools"`
`version`	string	オプション。セマンティックバージョン。これを設定するとプラグインをそのバージョン文字列にピン留めするため、ユーザーはバージョンをバンプしたときにのみ更新を受け取ります。*command* **ソース** (ページ 647) またはプラグイン**読み込み中** (ページ 605) を除きます。**バージョン管理** (ページ 624) を参照してください。マーケットプレイスエントリにも設定されている場合、`plugin.json` が優先されます。省略した場合、バージョンは**バージョン管理** (ページ 624) の次のソースから取得されます。	`"2.1.0"`
`description`	string	プラグインの目的の簡潔な説明	`"Deployment automation tools"`
`author`	object	著者情報	`{"name": "Dev Team", "email": "dev@company.com"}`
`homepage`	string	ドキュメント URL	`"https://docs.example.com"`

次のページに続く

表 11 – 前のページからの続き

フィールド	型	説明	例
repository	string	ソースコード URL	"https://github.com/user/plugin"
license	string	ライセンス識別子	"MIT"、"Apache-2.0"
keywords	array	検出タグ	["deployment", "ci-cd"]
metadata	object	権利付与またはカタログフィールドなど、独自のデータ用の自由形式オブジェクト。Claude Code はこれを読まないため、値はプラグインの動作に影響しません。Claude Code は非オブジェクト値を無視し、claude plugin validate は警告として報告します。v2.1.222 より前では、Claude Code はキーを**認識されないフィールド** (ページ 593) として扱いました。	{"catalogId": "cat-123"}
defaultEnabled	boolean	ユーザーが設定を設定していない場合、プラグインが有効な状態で開始するかどうか。デフォルトは true です。**デフォルト有効化** (ページ 595) を参照してください。	false

23.5.5 デフォルト有効化

plugin.json で defaultEnabled: false を設定して、無効な状態でインストールされるプラグインを配布します。ユーザーは claude plugin enable <plugin> または /plugin インターフェースでオンにします。外部サービスに接続するものなど、ユーザーがオプトインすべきコストまたはスコープを追加するプラグインに使用します。

defaultEnabled は、他に何もプラグインの状態を決定していない場合のフォールバックです。ユーザーの設定と依存関係の要件が優先されます。

- **ユーザーの設定**: 任意の設定スコープで enabledPlugins のプラグインのエントリ。一度書き込まれると、プラグイン更新と再インストール全体で永続化されるため、後のリリースで defaultEnabled を変更しても既存ユーザーは反転しません。
- **依存関係の要件**: プラグインがアクティブな別のプラグインによって必要とされる場合、Claude Code はインストール時または有効化時に true を書き込みます。これにより明示的な設定が与えられるため、独自のデフォルトはもはや適用されません。依存関係を持つプラグインを有効または無効にするを参照してください。

同じフィールドはプラグインのマーケットプレイスエントリに表示でき、plugin.json の値より優先されます。**オプションプラグインフィールド** (ページ 634) を参照してください。

23.5.6 コンポーネントパスフィールド

フィールド	型	説明	例
skills	string|array	<name>/SKILL.md を含むカスタムスキルディレクトリ。デフォルト skills/ スキャンに追加します。マーケットプレイスルート例外については**パス動作ルール** (ページ 601) を参照してください	"./custom/skills/"
commands	string|array	カスタムフラット .md スキルファイルまたはディレクトリ（デフォルト commands/ を置き換え）	"./custom/cmd.md" または ["./cmd1.md"]
agents	string|array	カスタムエージェントファイル（デフォルト agents/ を置き換え）	"./custom/agents/reviewer.md"
workflows	string|array	カスタムワークフロースクリプトファイルまたはディレクトリ（デフォルト workflows/ を置き換え）	"./custom/workflows/"
hooks	string|array|object	フック設定パスまたはインライン設定	"./my-extra-hooks.json"
mcpServers	string|array|object	MCP 設定パスまたはインライン設定	"./my-extra-mcp-config.json"
outputStyles	string|array	カスタム出力スタイルファイル/ディレクトリ（デフォルト output-styles/ を置き換え）	"./styles/"
lspServers	string|array|object	コード知能（定義へのジャンプ、参照の検索など）用の言語サーバープロトコル設定	"./.lsp.json"
experimental.themes	string|array	カラーテーマファイル/ディレクトリ（デフォルト themes/ を置き換え）。**テーマ** (ページ 588) を参照してください	"./themes/"
experimental.monitors	string|array	プラグインがアクティブな場合に自動的に開始するバックグラウンド Monitor 設定。**モニター** (ページ 587) を参照してください	"./monitors.json"

次のページに続く

表 12 – 前のページからの続き

フィールド	型	説明	例
experimental.evals	string|array	デフォルト evals/ ではない場合、プラグインの eval ケースを保持するプラグインルート下のディレクトリ。claude plugin eval --eval-dir はこれをオーバーライドします	"quality/evals"
userConfig	object	ユーザーが有効化時にプロンプトされる設定可能な値。**ユーザー設定** (ページ 597) を参照してください	
channels	array	メッセージ注入用のチャネル宣言 (Telegram、Slack、Discord スタイル)。**チャネル** (ページ 600) を参照してください	
dependencies	array	このプラグインが必要とする他のプラグイン。オプションで semver バージョン制約付き。プラグイン依存関係バージョンを制約するを参照してください	[{ "name": "secrets-vault", "version": "~2.1.0" }]

23.5.7 実験的コンポーネント

experimental キー、themes および monitors の下のコンポーネントは、安定化中にリリース間でマニフェストスキーマが変更される可能性があります。それらを宣言する場所は別の移行です。トップレベルはまだ機能し、claude plugin validate は警告を出し、将来のリリースは experimental.* を必要とします。

23.5.8 ユーザー設定

userConfig フィールドは、プラグインが有効化されたときに Claude Code がユーザーにプロンプトする値を宣言します。ユーザーに settings.json を手動で編集させる代わりにこれを使用してください。

```
{
  "userConfig": {
    "api_endpoint": {
      "type": "string",
      "title": "API endpoint",
      "description": "Your team's API endpoint"
    },
    "api_token": {
      "type": "string",
      "title": "API token",
      "description": "API authentication token",
      "sensitive": true
```

(次のページに続く)

(前のページからの続き)

```
    }
  }
}
```

キーは有効な識別子である必要があります。各オプションはこれらのフィールドをサポートします。

フィールド	必須	説明
`type`	はい	`string`、`number`、`boolean`、`directory`、または `file` のいずれか
`title`	はい	設定ダイアログに表示されるラベル
`description`	はい	フィールドの下に表示されるヘルプテキスト
`sensitive`	いいえ	`true` の場合、入力をマスクし、値を `settings.json` の代わりにセキュアストレージに保存します
`required`	いいえ	`true` の場合、フィールドが空の場合は検証が失敗します
`default`	いいえ	ユーザーが何も提供しない場合に使用される値
`options`	いいえ	`string` 型の場合、フィールドが受け入れる値。`/config` にピッカーとして表示されます。**フィールドを固定オプションに制限する** (ページ 599) を参照してください。Claude Code v2.1.271 以降が必要です
`multiple`	いいえ	`string` 型の場合、文字列の配列を許可します
`min` / `max`	いいえ	`number` 型の境界

`sensitive` フィールドと `multiple` リストを除き、有効な各プラグインの各フィールドは `/config` パネルの行としても表示されます。行には Claude Code v2.1.269 以降が必要です。

各値は MCP および LSP サーバー設定とフックコマンドで `${user_config.KEY}` として置換可能です。機密でない値はスキルおよびエージェントコンテンツでも置換できます。すべての値は、`<KEY>` がオプションキーを大文字にしたフックプロセスに `CLAUDE_PLUGIN_OPTION_<KEY>` 環境変数としてエクスポートされます。

シェルで実行されるフィールドは `${user_config.*}` を拒否します。設定された値をシェルコマンドに置換すると、シェルはその値に含まれるものを実行できるため、コンポーネントはエラーで失敗します。拒否された各フィールドには、値を渡す別の方法があります。

拒否されたフィールド	値を渡す方法
シェル形式フックコマンド	*exec* **形式** (ページ 459) を `args` で使用するか、フックの環境から `CLAUDE_PLUGIN_OPTION_<KEY>` を読み取ります
Monitor (ページ 587) コマンド	スクリプトの設定ファイルから値を読み取ります
MCP *`headersHelper`* (ページ 392)	スクリプトの設定ファイルから値を読み取ります

v2.1.207 より前では、これらのフィールドは `${user_config.KEY}` 値を置換しました。これに依存していたプラグインを更新してください。

機密でない値は、ユーザー `settings.json` の `pluginConfigs` キーの下に `pluginConfigs[<plugin-id>].options` として保存されます。

macOS では、Claude Code は macOS キーチェーンに機密値を保存し、キーチェーンが書き込みを拒否した場合は ~/.claude/.credentials.json にフォールバックします。サポートされているキーチェーンのないプラットフォームでは、~/.claude/.credentials.json に保存されます。キーチェーンストレージは OAuth トークンと共有され、約 2 KB の合計制限があるため、機密値を小さく保ちます。

Claude Code は 3 つの設定ソースからのみすべての pluginConfigs 値を読み取ります。

- **ユーザー設定**: ~/.claude/settings.json。有効化時プロンプトが書き込むファイル
- --settings: CLI フラグまたは SDK インライン設定
- **管理設定**: **組織制御ポリシー** (ページ 1063)

複数のソースが同じキーを設定する場合、管理設定が優先され、次に --settings、次にユーザー設定が優先されます。このリストから削除できる唯一のソースはユーザー設定です。*--setting-sources* (ページ 51) を user なしで渡し、Claude Code はそれらをスキップします。管理設定と --settings は渡すものが何であれ保持されます。SDK の settingSources オプションは同じリストを設定します。

プロジェクトの .claude/settings.json または .claude/settings.local.json のエントリは無視されます。両方のファイルはワークスペースに存在するため、クローンされたリポジトリはそこに値を提供でき、それらの値はプラグインフックコマンド、MCP サーバー設定、LSP コマンド、およびモニターコマンドに流れます。v2.1.207 より前では、これらのエントリが読み取られました。制限は pluginConfigs に固有です。enabledPlugins はまだプロジェクトおよびローカル設定を尊重します。

フィールドを固定オプションに制限する

userConfig フィールドに options を設定して、ユーザーが固定リストからその値を選択するようにします。

tone フィールドを 3 つのオプションに制限するには、options にそれらをリストし、default をそのうちの 1 つに設定します。

```
{
  "userConfig": {
    "tone": {
      "type": "string",
      "title": "Tone",
      "description": "Voice for generated replies",
      "options": ["neutral", "warm", "formal"],
      "default": "neutral"
    }
  }
}
```

任意のフィールドで options を宣言する場合、Claude Code v2.1.271 より前のバージョンのユーザーはプラグインを読み込むことができません。

フィールドに options を設定する場合、これらのルールに従います。

- type を string に設定します
- multiple または sensitive を true に設定しません

- `default` をオプションの 1 つに設定します
- `default` を設定しない場合は、`required` を `true` に設定します
- 少なくとも 1 つのオプションをリストし、各 1 〜 64 文字の長さです
- オプションをスペースで開始または終了しません
- 制御文字、非表示文字、テキスト方向を変更する文字、またはオプション内の通常のスペース以外のスペースを使用しません
- 異なる大文字小文字でも同じオプションを 2 回リストしません

これらのルールのいずれかを破った場合、プラグインは読み込みに失敗します。`claude plugin validate` を実行して、どのフィールドがどのルールを破るかを確認してください。

23.5.9 チャネル

`channels` フィールドを使用すると、プラグインは会話にコンテンツを注入する 1 つ以上のメッセージチャネルを宣言できます。各チャネルはプラグインが提供する MCP サーバーにバインドされます。

```
{
  "channels": [
    {
      "server": "telegram",
      "userConfig": {
        "bot_token": {
          "type": "string",
          "title": "Bot token",
          "description": "Telegram bot token",
          "sensitive": true
        },
        "owner_id": {
          "type": "string",
          "title": "Owner ID",
          "description": "Your Telegram user ID"
        }
      }
    }
  ]
}
```

`server` フィールドは必須で、プラグインの `mcpServers` のキーと一致する必要があります。オプションのチャネルごとの `userConfig` はトップレベルフィールドと同じスキーマを使用し、プラグインがプラグイン有効化時にボットトークンまたはオーナー ID をプロンプトできるようにします。

23.5.10 パス動作ルール

カスタムパスがプラグインのデフォルトディレクトリを置き換えるか拡張するかは、フィールドによって異なります。

- **デフォルトを置き換え**: commands、agents、workflows、outputStyles、experimental.themes、experimental.monitors。たとえば、マニフェストが commands を指定する場合、デフォルト commands/ ディレクトリはスキャンされません。デフォルトを保持してさらに追加するには、明示的にリストします。"commands": ["./commands/", "./extras/"]
- **デフォルトに追加**: skills。デフォルト skills/ ディレクトリは常にスキャンされ、skills にリストされているディレクトリはそれと一緒に読み込まれます。例外: **ソースがマーケットプレイスルートに解決される** (ページ 650) マーケットプレイスエントリの場合、特定のサブディレクトリを宣言するとデフォルト skills/ スキャンが置き換わります
- **独自のマージルール**: **フック** (ページ 581)、*MCP* **サーバー** (ページ 584)、および *LSP* **サーバー** (ページ 584)。各セクションで複数のソースがどのように結合されるかを参照してください

プラグインにデフォルトフォルダと一致するマニフェストキーの両方がある場合、Claude Code は claude plugin list と /plugin 詳細ビューで無視されたフォルダについて警告します。プラグインはマニフェストパスを使用して引き続き読み込まれます。マニフェストキーがデフォルトフォルダを指す場合、Claude Code は警告しません。たとえば "commands": ["./commands/deploy.md"] は、そのパスがフォルダを明示的に名前付けするためです。

すべてのパスフィールドについて。

- すべてのパスはプラグインルートに相対的で ./ で始まる必要があります。ただし、skills フィールドは . も受け入れます
  - "." と "./" の両方はプラグインルート自体を示します
  - v2.1.221 より前では、"." はマニフェスト検証に失敗し、プラグインは読み込まれなかったため、"./" を使用して以前のバージョンをサポートします
- カスタムパスのコンポーネントは、エージェントファイルを除き、同じ命名および名前空間化ルールを使用します。エージェント名がどのように機能するかについては**エージェント** (ページ 580) を参照してください
- 複数のパスは配列として指定できます
- スキルパスは SKILL.md を直接含むディレクトリを指すことができます。たとえば、プラグインルートの場合は "skills": ["."]
  - Claude Code は SKILL.md のフロントマター name フィールドからスキルの呼び出し名を取得するため、インストールディレクトリの名前が何であれ、名前は安定したままです
  - フロントマターで name が設定されていない場合、Claude Code はディレクトリベース名にフォールバックします

ルートに SKILL.md があり、skills/ サブディレクトリがなく、skills マニフェストフィールドがないプラグインは、単一スキルプラグインとして自動的に読み込まれます。このレイアウトの場合、plugin.json で "skills": ["./"] を設定する必要はありません。

パスの例:

```
{
  "commands": [
    "./specialized/deploy.md",
    "./utilities/batch-process.md"
  ],
  "agents": [
    "./custom-agents/reviewer.md",
    "./custom-agents/tester.md"
  ]
}
```

23.5.11 環境変数

Claude Code はパスを参照するための 3 つの変数を提供します。

変数	解決先	用途
`${CLAUDE_PLUGIN_ROOT}`	プラグインのインストールディレクトリへの絶対パス	プラグインにバンドルされたスクリプト、バイナリ、および設定ファイル
`${CLAUDE_PLUGIN_DATA}`	プラグイン更新を超えて存続する**永続ディレクトリ** (ページ 603)。最初の参照時に作成されます	`node_modules` または Python 仮想環境などのインストール済み依存関係、生成されたコード、およびキャッシュ
`${CLAUDE_PROJECT_DIR}`	プロジェクトルート	プロジェクトローカルスクリプトおよび設定ファイル

3 つすべてが環境変数としてフックプロセスおよび MCP と LSP サーバーサブプロセスにエクスポートされます。これらはメインセッションまたはサブエージェントで Bash ツールを通じて Claude が実行するコマンドの環境には存在しません。プラグインコンテンツで、プレースホルダーを書き込み、Claude Code はコンテンツを読み込むときにパスをインラインで置換します。どのフィールドがそれらをインラインで置換するかは、プラグインコンポーネントによって異なります。

プラグインコンポーネント	プレースホルダーが解決されるフィールド
スキルおよびエージェントコンテンツ	プレースホルダーが表示される任意の場所
フックおよびモニターコマンド	プレースホルダーが表示される任意の場所
MCP `stdio` サーバー	`command`、`args`、`env`
MCP `http`、`sse`、`ws` サーバー	`url`、`headers`、`headersHelper`
LSP サーバー	`command`、`args`、`env`、`workspaceFolder`

フックコマンドで、*exec* **形式** (ページ 459) を `args` で使用して、各パスが引用符なしで 1 つの引数として渡されるようにします。シェル形式フックおよびモニターコマンドで、`"${CLAUDE_PROJECT_DIR}/scripts/server.sh"` のように変数をダブルクォートで囲みます。このシェル形式フックはプラグインにバンドルされたスクリプトを実行します。

```
{
  "hooks": {
    "PostToolUse": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "\"${CLAUDE_PLUGIN_ROOT}\"/scripts/process.sh"
          }
        ]
      }
    ]
  }
}
```

コピーされたプラグインの場合、`${CLAUDE_PLUGIN_ROOT}` はプラグインが更新されるときに変更されます。前のバージョンのディレクトリは更新後の猶予期間ディスク上に残りますが、それを一時的なものとして扱い、そこに状態を書き込まないでください。ローカルディレクトリマーケットプレイスから読み込まれたプラグインの場合、変数は安定したソースディレクトリを指します。どのプラグインがコピーされるか、およびクリーンアップセマンティクスについては、**プラグインキャッシング** (ページ 605) を参照してください。

コピーされたプラグインがセッション中に更新される場合、フックコマンド、モニター、MCP サーバー、および LSP サーバーは前のバージョンのパスを使用し続けます。`/reload-plugins` を実行して、フック、MCP サーバー、および LSP サーバーを新しいパスに切り替えます。モニターはセッション再開が必要です。インタラクティブターミナルのないセッションでは、リロードはプラグイン MCP サーバーを次のセッションまで古いパスに残します。

command ソースを持つプラグインの場合、Claude Code は**プラグイン自体を再読み込みできます** (ページ 649)。

MCP サーバーは `roots/list` リクエストを呼び出して、実行時にセッションの作業ディレクトリを読み取ることもできます。*roots/list* **が返すもの、および** *Claude Code* **がサーバーに変更を通知するとき** (ページ 366) を参照してください。

永続データディレクトリ

`${CLAUDE_PLUGIN_DATA}` ディレクトリは `~/.claude/plugins/data/{id}/` に解決されます。ここで `{id}` はプラグイン識別子で、`a-z`、`A-Z`、`0-9`、`_`、および `-` の外の文字は `-` に置き換えられます。`formatter@my-marketplace` としてインストールされたプラグインの場合、ディレクトリは `~/.claude/plugins/data/formatter-my-marketplace/` です。

一般的な用途は、言語依存関係を 1 回インストールし、セッションとプラグイン更新全体で再利用することです。Python 依存関係、Yarn または pnpm でロックされた依存関係、およびライフサイクルスクリプトを実行する必要があるパッケージに使用します。マーケットプレイスにインストールされたプラグインの場合、それをまったく必要としない場合があります。Claude Code はキャッシュ時に適格な *Node.js* **パッケージ依存関係** (ページ 605) を自動的にインストールします。

データディレクトリは単一のプラグインバージョンより長く存続するため、ディレクトリ存在チェックだけでは、更新がプラグインの依存関係マニフェストを変更したときを検出できません。推奨パターンはバ

ンドルされたマニフェストをデータディレクトリのコピーと比較し、異なる場合は再インストールします。

この SessionStart フックは最初の実行時に node_modules をインストールし、プラグイン更新に変更された package.json が含まれるたびに再度インストールします。

```
{
  "hooks": {
    "SessionStart": [
      {
        "hooks": [
          {
            "type": "command",
            "command": "diff -q \"${CLAUDE_PLUGIN_ROOT}/package.json\" \"${CLAUDE_
↪PLUGIN_DATA}/package.json\" >/dev/null 2>&1 || (cd \"${CLAUDE_PLUGIN_DATA}\" && cp \
↪"${CLAUDE_PLUGIN_ROOT}/package.json\" . && npm install) || rm -f \"${CLAUDE_PLUGIN_
↪DATA}/package.json\""
          }
        ]
      }
    ]
  }
}
```

diff は保存されたコピーが見つからないか、バンドルされたコピーと異なる場合にゼロ以外で終了し、最初の実行と依存関係変更更新の両方をカバーします。npm install が失敗した場合、末尾の rm はコピーされたマニフェストを削除して、次のセッションが再試行されるようにします。

${CLAUDE_PLUGIN_ROOT} にバンドルされたスクリプトは、永続化された node_modules に対して実行できます。

```
{
  "mcpServers": {
    "routines": {
      "command": "node",
      "args": ["${CLAUDE_PLUGIN_ROOT}/server.js"],
      "env": {
        "NODE_PATH": "${CLAUDE_PLUGIN_DATA}/node_modules"
      }
    }
  }
}
```

データディレクトリは、最後のスコープからプラグインをアンインストールするときに自動的に削除されます。/plugin インターフェースはディレクトリサイズを表示し、削除前にプロンプトします。CLI はデフォルトで削除します。*--keep-data* (ページ 613) を渡して保持します。

23.6 プラグインのキャッシングとファイル解決

プラグインは以下の 3 つの方法のいずれかで指定されます。

- `claude --plugin-dir` または `claude --plugin-url` を通じて、セッションの期間中。
- マーケットプレイスを通じて、今後のセッション用にインストール。
- claude.ai アカウントを通じて、**同期** (ページ 590) されて `~/.claude/plugins/synced/` に。

セキュリティと検証の目的で、Claude Code はマーケットプレイス プラグインをユーザーのローカル **プラグインキャッシュ** （`~/.claude/plugins/cache`）にコピーします。ただし、プラグインがインプレイスでロードされる場合を除きます。**リンクモードの *command* ソース** (ページ 648) はキャッシュエントリ内のリンクを通じてインプレイスでロードされます。**ローカルディレクトリから追加されたマーケットプレイスの相対パスソース** (ページ 638) はマーケットプレイスフォルダからインプレイスでロードされます。

ローカルディレクトリマーケットプレイスからインプレイスでロードされたプラグインの場合、ソースディレクトリへの編集は次のセッション開始時または `/reload-plugins` で有効になります。バージョンバンプは不要です。プラグインのフックプロセスと MCP および LSP サーバーは、ソースディレクトリを指す `CLAUDE_PLUGIN_ROOT` を受け取ります。Claude Code はプラグインの ***Node.js* パッケージ依存関係** (ページ 605) をソースディレクトリにインストールしません。それらを自分でインストールするか、**永続データディレクトリ** (ページ 603) へのフックからインストールしてください。

コピーされたプラグインの場合、インストールされた各バージョンはキャッシュ内の個別のディレクトリであり、マーケットプレイスとプラグインでグループ化され、解決されたバージョンに対して名前が付けられ、プラグインのファイルと ***Node.js* パッケージ依存関係** (ページ 605) の独自のコピーを持ちます。リリースタグから解決された依存関係は、コミット SHA サフィックス付きのディレクトリ名を取得します。

プラグインを更新またはアンインストールすると、Claude Code は前のバージョンディレクトリを孤立したものとしてマークし、約 14 日後のバックグラウンドスイープで削除します。猶予期間により、既に古いバージョンをロードした同時実行中の Claude Code セッションがエラーなく実行を続けることができます。Claude Code はスイープを実行するのは、少なくとも 1 つのプラグインがインストールされている場合のみです。最後のプラグインをアンインストールした後、孤立したディレクトリはディスク上に残り、プラグインを再度インストールするまで保持されます。

Claude Code は、プラグインまたはマーケットプレイスフォルダをキャッシュから削除するのは、ディレクトリまたはシンボリックリンクが含まれなくなった場合のみです。開発チェックアウトをキャッシュにシンボリックリンクとしてプラグインのバージョンエントリにリンクする場合、Claude Code はリンクを孤立したものとしてマークすることはなく、削除することもなく、それを保持するフォルダも削除しません。Claude Code はリンクされたチェックアウト内にバージョン追跡ファイルを書き込むこともありません。

Claude の Glob および Grep ツールは検索中に孤立したバージョンディレクトリをスキップするため、ファイル結果には古いプラグインコードが含まれません。

23.6.1 Node.js パッケージ依存関係

Claude Code がプラグインをキャッシュにコピーするとき、プラグインの Node.js パッケージ依存関係もそこにインストールするため、プラグインのフックと MCP サーバーはそれらをロードできます。このセクションでは、プラグインが独自の `package.json` で宣言する npm および Bun パッケージについて説明します。他のプラグインに依存するプラグインについては、プラグイン依存関係バージョンを参照してください。

Claude Code は、コピーされたバージョンディレクトリを作成するたびに、その内部でインストールを実行します。プラグインをインストールするとき、Claude Code がプラグインを新しいバージョンに更新するとき、および有効なプラグインがまだキャッシュされていない場合のセッション開始時（新しいマシンなど）です。インストールは、プラグインのルートディレクトリに `package.json` とサポートされているロックファイルの両方が含まれている場合にのみ実行されます。

ロックファイル	コマンド
`bun.lock` または `bun.lockb`	`bun install --frozen-lockfile --ignore-scripts`
`npm-shrinkwrap.json` または `package-lock.json`	`npm ci --ignore-scripts`

プラグインにこれらのロックファイルが複数含まれている場合、Claude Code は最初のマッチを使用し、順序をチェックします。`bun.lock`、`bun.lockb`、`npm-shrinkwrap.json`、`package-lock.json`。

Claude Code は 2 つのケースでインストールをスキップします。それぞれ独自の修正があります。

- プラグインが `yarn.lock` または `pnpm-lock.yaml` のみを配布している場合は、npm ロックファイルに置き換えてください。
- `bunfig.toml` が bun ロックファイルの横にある場合は、`bunfig.toml` を削除するか、bun ロックファイルを npm ロックファイルに置き換えてください。

最も広いリーチのために npm ロックファイルを配布してください。Claude Code はマッチされたロックファイルのパッケージマネージャーをユーザーの PATH から実行し、ロックファイルが見つからない場合は他のロックファイルにフォールバックしません。npm ソースを通じて配布されるプラグインの場合は、`npm-shrinkwrap.json` を使用してください。npm は公開されたパッケージから `package-lock.json` を除外します。

Claude Code はこの依存関係インストールを制約して、プラグインまたはそのパッケージからのコードがインストール中に実行されず、実行時間が制限されます。

- **凍結された解決:** Bun と npm はロックファイルがピンしたものを正確にインストールし、`package.json` とロックファイルが一致しない場合は再解決するのではなく失敗します。
- **ライフサイクルスクリプトなし:** `--ignore-scripts` は `preinstall`、`install`、および `postinstall` スクリプトが実行されないようにするため、これらのスクリプトでネイティブモジュールをビルドする依存関係はダウンロードされますが、このインストール中にはコンパイルされません。
- **60 秒のタイムアウト:** Claude Code は実行時間が長いインストールを停止し、失敗として扱います。

Claude Code は npm ソースプラグインをこの依存関係インストールの前にフェッチし、このフェッチ中にパッケージ独自のインストールスクリプトは実行されません。*npm* **パッケージ** (ページ 641) を参照してください。

失敗またはスキップされたインストールはプラグインをブロックすることはありません。インストールが失敗した場合、または Claude Code が yarn または pnpm ロックファイルをスキップした場合、または `bunfig.toml` が横にある場合、理由は **デバッグ出力** (ページ 622) の警告として記録されます。`package.json` とロックファイルがないプラグインはログエントリなしでスキップされます。タイムアウトしたインストールは、キャッシュされたコピーに部分的な `node_modules` ツリーを残すことができます。

自動インストールをオフにすることはできません。設定または環境変数はそれを無効にしません。制限されたネットワークでは、**ネットワークアクセス要件** (ページ 904) を参照して、許可するホストを確認してください。

自動インストールが提供できない依存関係（ライフサイクルスクリプトをビルドする必要があるパッケージ、Python 依存関係、または Yarn または pnpm でロックされたプラグインなど）については、**永続データディレクトリ** (ページ 603) へのフックからインストールしてください。

23.6.2 パストラバーサルの制限

Claude Code はプラグインが独自のディレクトリ外のファイルを参照することを許可しません。プラグインルートの外に解決されるコンポーネントパスを拒否します。パスが `plugin.json` で宣言されているか、**マーケットプレイスエントリ** (ページ 633) で宣言されているかに関わらず。これは、`../shared-utils` などのように書かれたプラグインの外を指すパス、および *1* **つのマーケットプレイス内のリンク** (ページ 607) 以外のプラグインの外につながるシンボリックリンクをカバーします。

macOS と Linux では、Claude Code はコンポーネントパスにバックスラッシュが含まれている場合も拒否します。バックスラッシュパスで宣言されたコンポーネントは、Windows でのみロードされます。`./commands/deploy.md` などのようにフォワードスラッシュを使用してコンポーネントパスを記述してください。

Claude Code がパスを拒否すると、`path escapes plugin directory` エラーを報告し、そのコンポーネントなしでプラグインをロードします。

Claude Code はプラグインをインストールするときにプラグインディレクトリ外のファイルをキャッシュにコピーしないため、コピーされたプラグイン内のスクリプトがプラグインルート上のパスを読み取る場合、それらのファイルも見つかりません。

23.6.3 シンボリックリンクを使用してマーケットプレイス内でファイルを共有する

プラグインが同じマーケットプレイスの他の部分とファイルを共有する必要がある場合は、プラグインディレクトリ内にシンボリックリンクを作成できます。プラグインがキャッシュにコピーされるときにシンボリックリンクがどのように処理されるかは、そのターゲットがどこに解決されるかによって異なります。

- **プラグイン独自のディレクトリ内:** シンボリックリンクはキャッシュ内の相対シンボリックリンクとして保持されるため、実行時にコピーされたターゲットへの解決を続けます。
- **同じマーケットプレイス内の他の場所:** シンボリックリンクは逆参照されます。ターゲットのコンテンツはキャッシュにコピーされます。これにより、メタプラグインの `skills/` ディレクトリがマーケットプレイス内の他のプラグインで定義されたスキルにリンクできます。
- **マーケットプレイス外:** シンボリックリンクはセキュリティのためスキップされます。これにより、プラグインがシステムパスなどの任意のホストファイルをキャッシュに取り込むことを防ぎます。

`--plugin-dir` でインストールされたプラグイン、ローカルパスから、または **コピーモードの** *command* **ソース** (ページ 648) から、プラグイン独自のディレクトリ内で解決されるシンボリックリンクのみが保持されます。その他はすべてスキップされます。

次のコマンドは、マーケットプレイスプラグイン内から、兄弟プラグインで定義された共有スキルへのリンクを作成します。Windows では、昇格されたコマンドプロンプトから `mklink /D` を使用するか、開発者モードを有効にしてください。

```
ln -s ../../shared-plugin/skills/foo ./skills/foo
```

23.7 プラグインディレクトリ構造

23.7.1 標準プラグインレイアウト

完全なプラグインは以下の構造に従います：

```
enterprise-plugin/
├─── .claude-plugin/           # メタデータディレクトリ（オプション）
│    └─── plugin.json             # プラグインマニフェスト
├─── skills/                   # Skills
│    ├─── code-reviewer/
│    │    └─── SKILL.md
│    └─── pdf-processor/
│         ├─── SKILL.md
│         └─── scripts/
├─── commands/                 # Skills をフラット .md ファイルとして
│    ├─── status.md
│    └─── logs.md
├─── agents/                   # Subagent 定義
│    ├─── security-reviewer.md
│    ├─── performance-tester.md
│    ├─── compliance-checker.md
│    └─── review/               # ここのエージェントは enterprise-plugin:review:<name>
 として読み込まれます
│         └─── accessibility.md
├─── workflows/                # ワークフロースクリプト
│    └─── release-audit.js
├─── output-styles/            # 出力スタイル定義
│    └─── terse.md
├─── themes/                   # カラーテーマ定義
│    └─── dracula.json
├─── monitors/                 # バックグラウンドモニター設定
│    └─── monitors.json
├─── hooks/                    # Hook 設定
│    ├─── hooks.json           # メイン hook 設定
│    └─── security-hooks.json  # 追加 hooks
├─── bin/                      # プラグイン実行ファイルが PATH に追加される
│    └─── my-tool              # Bash tool で裸のコマンドとして呼び出し可能
├─── settings.json            # プラグインのデフォルト設定
├─── .mcp.json                # MCP サーバー定義
├─── .lsp.json                # LSP サーバー設定
├─── scripts/                 # Hook とユーティリティスクリプト
│    ├─── security-scan.sh
```

(次のページに続く)

(前のページからの続き)

```
│   ├── format-code.py
│   └── deploy.js
├── LICENSE                    # ライセンスファイル
└── CHANGELOG.md               # バージョン履歴
```

警告:

`.claude-plugin/` ディレクトリには `plugin.json` ファイルが含まれます。その他すべてのディレクトリ（commands/、agents/、skills/、workflows/、output-styles/、themes/、monitors/、hooks/）は `.claude-plugin/` 内ではなく、プラグインルートに配置する必要があります。

プラグインルートの `CLAUDE.md` ファイルはプロジェクトコンテキストとして読み込まれません。プラグインは CLAUDE.md ではなく、skills、agents、hooks を通じてコンテキストを提供します。Claude のコンテキストに読み込まれる命令を配布するには、*skill* (ページ 579) に配置してください。

23.7.2 ファイルロケーション参照

コンポーネント	デフォルトロケーション	目的
マニフェスト	`.claude-plugin/plugin.json`	プラグインメタデータと設定（オプション）
Skills	`skills/`	<name>/SKILL.md 構造の Skills
コマンド	`commands/`	フラット Markdown ファイルとしての Skills。新しいプラグインには `skills/` を使用してください
Agents	`agents/`	Subagent Markdown ファイル。サブフォルダは**エージェント名** (ページ 580) の一部です
ワークフロー	`workflows/`	ワークフロー スクリプトファイル
出力スタイル	`output-styles/`	出力スタイル定義
テーマ	`themes/`	カラーテーマ定義
Hooks	`hooks/hooks.json`	Hook 設定
MCP サーバー	`.mcp.json`	MCP サーバー定義
LSP サーバー	`.lsp.json`	言語サーバー設定
モニター	`monitors/monitors.json`	バックグラウンドモニター設定
実行ファイル	`bin/`	Bash tool の `PATH` に追加され、プラグインが有効な間は裸のコマンドとして呼び出し可能な実行ファイル。*claude.ai* **組織設定を通じて配布するプラグイン** (ページ 655) にはこのディレクトリを含めることはできません
設定	`settings.json`	プラグインが有効になったときに適用されるデフォルト設定。*`agent`* (ページ 317) と *`subagentStatusLine`* (ページ 815) キーのみがサポートされています

23.8 CLI コマンドリファレンス

Claude Code は、非対話的なプラグイン管理用の CLI コマンドを提供します。スクリプトとオートメーションに便利です。

23.8.1 plugin init

~/.claude/skills/<name>/ に新しいプラグインをスキャフォルドします。次の Claude Code セッションで、<name>@skills-dir として自動的に読み込まれ、/plugin と claude plugin list に表示されます。インストール手順は不要です。

スキルディレクトリプラグイン (ページ 589) のスコープと信頼要件を参照してください。

```
claude plugin init <name> [options]
```

コマンドは以下の引数を取ります:

- <name>: プラグイン名。スキル名前空間と ~/.claude/skills/ の下のディレクトリ名になるため、スペースやパス区切り文字を含めることはできません。

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
--description <text>	マニフェストの説明	
--author <name>	作成者名	git config user.name
--author-email <email>	作成者メール	git config user.email
--with <components..>	コンポーネントフォルダもスキャフォルドします。有効な値: skills、agents、hooks、mcp、lsp、output-style、channel	
-f, --force	ターゲットの既存 .claude-plugin/ を上書きします	
-h, --help	コマンドのヘルプを表示	

claude plugin new はこのコマンドのエイリアスです。

各 --with 値は、そのコンポーネント用のスターターファイルを追加し、編集可能な状態にします:

コンポーネント	スキャフォルドされるもの
skills	デフォルトのスキルと並んで、追加の名前空間付き <name>:example スキル
agents	agents/ サブエージェント定義
hooks	サンプルイベントハンドラを含む hooks/hooks.json
mcp	HTTP と stdio サーバーの例を含む .mcp.json
lsp	.lsp.json 言語サーバーの例

次のページに続く

表 20 – 前のページからの続き

コンポーネント	スキャフォルドされるもの
output-style	プラグインが有効な間に自動的に適用される output-styles/<name>.md
channel	MCP ベースのチャネル: stdio サーバー（server.ts）、その .mcp.json、および package.json

スキャフォルドされたプラグインは、マーケットプレイスではなく @skills-dir ソースを使用します。管理者は strictKnownMarketplaces でこのソースをブロックするか、**管理設定** (ページ 658) の blockedMarketplaces に {"source": "skills-dir"} を追加することでブロックできます。ブロックされている場合、plugin init は書き込み前に失敗します。

これらの例は一般的な呼び出しを示しています:

```
# 最小限のプラグインをスキャフォルド
claude plugin init my-helper

# スキルとフックフォルダを含めてスキャフォルド
claude plugin init my-helper --with skills hooks

# 既存のスキャフォルドを上書き
claude plugin init my-helper --force
```

23.8.2 plugin install

利用可能なマーケットプレイスからプラグインをインストールします。

```
claude plugin install <plugin> [options]
```

コマンドは以下の引数を取ります:

- <plugin>: プラグイン名、または特定のマーケットプレイス用の plugin-name@marketplace-name

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
-s, --scope <scope>	インストールスコープ: user、project、または local	user
--config <key=value>	プラグインのマニフェストで宣言された *userConfig* (ページ 597) オプションを設定します。複数のオプションを設定するにはフラグを繰り返します	

次のページに続く

表 21 – 前のページからの続き

オプション	説明	デフォルト
`-y, --yes`	確認プロンプトなしで、プラグインのマーケットプレイスが宣言するコマンドを受け入れます: *command* **ソース** (ページ 647) を持つプラグインを生成するコマンド、またはアーカイブダウンロードを認証する *headersHelper* (ページ 643)。`headersHelper` を受け入れるには Claude Code v2.1.238 以降が必要です。Claude Code はまずコマンドを出力します。stdin または stdout が TTY でない場合は必須です。Claude Code セッション内では効果がないため、独自のターミナルからコマンドを実行してください	
`--accept-command <sha256>`	前の*--json* **実行** (ページ 612) が `shownCommand` で報告した `sha256` を持つマーケットプレイス宣言コマンドを受け入れます。`-y` の代わりに使用します。受け入れは、正確にそのコマンド、プラグイン、およびマーケットプレイスカタログに対してカウントされます。コマンドが表示されてから変更された場合（実行自体のマーケットプレイス更新を含む）、Claude Code はダイジェストを受け入れず、コマンドを再度表示します。`-y` と組み合わせることはできません。Claude Code セッション内では効果がないため、独自のターミナルからコマンドを実行してください。Claude Code v2.1.271 以降が必須です	
`--json`	結果を stdout の最後の行に 1 つの JSON オブジェクトとして出力します。スクリプトで使用するための人間が読める形式の代わりに。*JSON* **結果形式** (ページ 612) を参照してください。Claude Code v2.1.268 以降が必須です	
`-h, --help`	コマンドのヘルプを表示	

スコープは、インストールされたプラグインが追加される設定ファイルを決定します。たとえば、`--scope project` は .claude/settings.json の `enabledPlugins` に書き込み、プロジェクトリポジトリをクローンした全員がプラグインを利用できるようにします。

`--json` を使用すると、stdout の最後の行は 1 つの JSON オブジェクトです。マーケットプレイスが宣言するコマンドが前に出力される可能性があるため、その行のみを解析してください。3 つのフィールドは常に存在します:

- `command`: 実行されたサブコマンド（`install` など）
- `outcome`: `ok` または `failed`
- `message`: 結果の人間が読める説明

`pluginId`、`scope`、`failureCode` などの他のフィールドは、適用される場合にのみ表示されます。`plugin uninstall`、`plugin update`、`plugin enable`、および `plugin disable` の `--json` オプションは、そのサブコマンド独自のフィールドを持つ同じオブジェクトを出力します。`--scope` が無効な場合などの使用エラーは、結果行を出力せず、終了コード 1 で理由を stderr に出力します。

実行がマーケットプレイス宣言コマンドを表示し、それを実行しない場合、`failed` 結果は、表示されたコ

マンド、それが属するプラグイン、およびコマンドの sha256 を含むフィールドを持つ shownCommand オブジェクトも含みます。正確にそのコマンドを受け入れるには、その sha256 を --accept-command として再実行します。Claude Code v2.1.271 以降が必須です。

shownCommand.acceptCommandMatched が false の場合、渡したダイジェストは現在表示されているコマンドと一致しません。そのコマンドを人に見せてから、その sha256 を渡してください。

これらの例は一般的な呼び出しを示しています:

```
# ユーザースコープにインストール（デフォルト）
claude plugin install formatter@my-marketplace

# プロジェクトスコープにインストール（チームと共有）
claude plugin install formatter@my-marketplace --scope project

# ローカルスコープにインストール（チームと共有しない）
claude plugin install formatter@my-marketplace --scope local
```

23.8.3 plugin uninstall

インストール済みプラグインを削除します。

```
claude plugin uninstall <plugin> [options]
```

コマンドは以下の引数を取ります:

- <plugin>: プラグイン名、または plugin-name@marketplace-name

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
-s, --scope <scope>	スコープからアンインストール: user、project、または local	user
--keep-data	プラグインの**永続データディレクトリ** (ページ 603) を保持します	
--prune	他のプラグインが必要としない自動インストール依存関係も削除します。*plugin prune* (ページ 614) を参照	
-y, --yes	--prune 確認プロンプトをスキップします。stdin または stdout が TTY でない場合は必須	
--json	結果を stdout の最後の行に 1 つの JSON オブジェクトとして出力します。*plugin install --json* (ページ 612) と同じ形式で。--prune と組み合わせることはできません。Claude Code v2.1.268 以降が必須です	
-h, --help	コマンドのヘルプを表示	

claude plugin remove と claude plugin rm はこのコマンドのエイリアスです。

デフォルトでは、最後に残ったスコープからアンインストールすると、プラグインの ${CLAUDE_PLUGIN_

DATA} ディレクトリも削除されます。新しいバージョンをテストした後に再インストールする場合など、保持するには --keep-data を使用します。

注釈:

異なるマーケットプレイスからインストールされたプラグインが同じ名前を共有する場合、plugin-name@marketplace-name 形式は指定されたマーケットプレイスからのプラグインのみをアンインストールします。v2.1.212 より前は、修飾形式は異なるマーケットプレイスから同じ名前のプラグインにマッチしてアンインストールする可能性がありました。

23.8.4 plugin prune

インストール済みプラグインによって不要になった自動インストール依存関係を削除します。Claude Code が別のプラグインの dependencies フィールドを満たすために取得した依存関係は削除されます。直接インストールしたプラグインは決して削除されません。

```
claude plugin prune [options]
```

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
-s, --scope <scope>	スコープでプルーン: user、project、または local	user
--dry-run	削除せずに削除されるものをリストします	
-y, --yes	確認プロンプトをスキップします。stdin または stdout が TTY でない場合は必須	
-h, --help	コマンドのヘルプを表示	

claude plugin autoremove はこのコマンドのエイリアスです。

コマンドは孤立した依存関係をリストし、削除前に確認を求めます。プラグインを削除し、その依存関係をワンステップでクリーンアップするには、claude plugin uninstall <plugin> --prune を実行します。

23.8.5 plugin enable

無効なプラグインを有効にします。ターゲットがマーケットプレイスからインストールされ、依存関係を宣言している場合、Claude Code は同じスコープで推移的にそれらを有効にします。コマンドは依存関係を持つプラグインを有効または無効にするがリストする条件下で失敗します。

```
claude plugin enable <plugin> [options]
```

コマンドは以下の引数を取ります:

- <plugin>: プラグイン名、plugin-name@marketplace-name、または *claude.ai* **から同期されたプラグイン** (ページ 590) 用の plugin-name@synced

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
`-s, --scope <scope>`	有効にするスコープ: `user`、`project`、または `local`。省略した場合、Claude Code はプラグインがインストールされているスコープを検出します	自動検出
`--json`	結果を stdout の最後の行に 1 つの JSON オブジェクトとして出力します。*`plugin install --json`* (ページ 612) と同じ形式で。Claude Code v2.1.268 以降が必須です	
`-h, --help`	コマンドのヘルプを表示	

23.8.6 plugin disable

プラグインをアンインストールせずに無効にします。

ターゲットがマーケットプレイスからインストールされている場合、別の有効なプラグインがそれに依存している場合、コマンドは失敗します。エラーメッセージには、最初にすべての依存プラグインを無効にするチェーンコマンドが含まれます。

組織が必須とする**同期プラグイン** (ページ 590) の場合、コマンドは失敗し、何も保存しません。

```
claude plugin disable [plugin] [options]
```

コマンドは以下の引数を取ります:

- `[plugin]`: プラグイン名、`plugin-name@marketplace-name`、または *claude.ai* **から同期されたプラグイン** (ページ 590) 用の `plugin-name@synced`。`--all` を使用する場合はオプション

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
`-a, --all`	すべての有効なプラグインを無効にします。`--scope` と組み合わせることはできません	
`-s, --scope <scope>`	無効にするスコープ: `user`、`project`、または `local`。省略した場合、Claude Code はプラグインがインストールされているスコープを検出します	自動検出
`--json`	結果を stdout の最後の行に 1 つの JSON オブジェクトとして出力します。*`plugin install --json`* (ページ 612) と同じ形式で。Claude Code v2.1.268 以降が必須です	
`-h, --help`	コマンドのヘルプを表示	

23.8.7 plugin update

プラグインを最新バージョンに更新します。

```
claude plugin update <plugin> [options]
```

コマンドは以下の引数を取ります:

- <plugin>: プラグイン名、または plugin-name@marketplace-name

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
-s, --scope <scope>	更新するスコープ: user、project、local、または managed	user
-y, --yes	確認プロンプトなしで、プラグインのマーケットプレイスが宣言するコマンドを受け入れます: *command* **ソース** (ページ 647) を持つプラグインを生成するコマンド、またはアーカイブダウンロードを認証する *headersHelper* (ページ 643)。headersHelper を受け入れるには Claude Code v2.1.238 以降が必要です。Claude Code はまずコマンドを出力します。stdin または stdout が TTY でない場合は必須です。Claude Code セッション内では効果がないため、独自のターミナルからコマンドを実行してください	
--accept-command <sha256>	前の*--json* **実行** (ページ 612) が shownCommand で報告した sha256 を持つマーケットプレイス宣言コマンドを受け入れます。-y の代わりに使用します。受け入れは、正確にそのコマンド、プラグイン、およびマーケットプレイスカタログに対してカウントされます。コマンドが表示されてから変更された場合（実行自体のマーケットプレイス更新を含む）、Claude Code はダイジェストを受け入れず、コマンドを再度表示します。-y と組み合わせることはできません。Claude Code セッション内では効果がないため、独自のターミナルからコマンドを実行してください。Claude Code v2.1.271 以降が必須です	
--json	結果を stdout の最後の行に 1 つの JSON オブジェクトとして出力します。*plugin install --json* (ページ 612) と同じ形式で。Claude Code v2.1.268 以降が必須です	
-h, --help	コマンドのヘルプを表示	

注釈:

Claude Code は、インストール済みプラグインに対して修飾されていないプラグイン名を解決します。異なるマーケットプレイスからインストールされたプラグインが名前を共有する場合、Claude Code は更新を拒否し、代わりに実行する修飾 plugin-name@marketplace-name コマンドをリストします。v2.1.246 より前は、Claude Code は修飾形式のみを受け入れ、修飾されていない名前を見つからないものとして拒否していました。

23.8.8 plugin list

インストール済みプラグインをバージョン、ソースマーケットプレイス、および有効状態と共にリストします。

```
claude plugin list [options]
```

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
`--json`	JSON として出力します。読み込み問題またはオーサリング警告を含むプラグイン行は `errors` または `notes` 文字列配列を含みます。Claude Code v2.1.268 以降では、並列 `errorDetails` および `noteDetails` 配列は各エントリの診断 `type` と、プラグイン、マーケットプレイス、サーバー、またはファイルなど、それが参照する名前を提供します	
`--available`	マーケットプレイスから利用可能なプラグインを含めます。`--json` が必須	
`-h, --help`	コマンドのヘルプを表示	

対話的セッション内では、`/plugin list` は同様のリストをインラインで出力しますが、マーケットプレイスからインストールされたプラグインのみをカバーします:

- スキルディレクトリから読み込まれたプラグインは `/plugin` インターフェイスと `claude plugin list` に表示されますが、インラインの `/plugin list` 出力には表示されません。
- *claude.ai* **から同期されたプラグイン** (ページ 590) は Claude Code v2.1.239 以降で `claude plugin list` に表示され、`/plugin` インターフェイスに表示されますが、インラインの `/plugin list` 出力には表示されません。
- `--plugin-dir` または `--plugin-url` でセッション用に読み込まれたプラグインは `/plugin` インターフェイスに表示され、`claude --plugin-dir <dir> plugin list` のように同じフラグがサブコマンドの前にある場合にのみ `claude plugin list` に表示されます。フラグ名のみがそれらの場所を指定するため、修飾されていない `claude plugin list` は同期されたプラグインとスキルディレクトリプラグインとは異なり、Claude Code がスキャンする固定ディレクトリを持たないため、それらを見つけることができません。

対話的形式は、`--enabled` または `--disabled` を受け入れてそのスタイルのプラグインのみを表示し、`ls` を `list` の短縮形として受け入れます。

23.8.9 plugin details

プラグインのコンポーネント在庫と予想トークンコストを表示します。出力は、プラグインが提供するすべてのコンポーネントをスキル、エージェント、フック、MCP サーバー、および LSP サーバーとしてグループ化し、各セッションに追加するトークン数の推定値を含めてリストします。スキルグループには `skills/` と `commands/` エントリの両方が含まれます。

```
claude plugin details <name>
```

コマンドは以下の引数を取ります:

- <name>: プラグイン名、または plugin-name@marketplace-name

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
-h, --help	コマンドのヘルプを表示	

出力は各コンポーネントの 2 つのコスト数値を表示します:

- **常時オン:** スキル説明、エージェント説明、コマンド名など、プラグインのリストテキストによってすべてのセッションに追加されるトークン。コンポーネントが発火するかどうかに関係なく。
- **呼び出し時:** コンポーネントが発火するときにコンポーネントがコストするトークン。プラグイン全体ではなくコンポーネントごとに表示されます。典型的なセッションはコンポーネントのサブセットのみを呼び出すため。

この例は、2 つのスキルを持つプラグインの出力がどのように見えるかを示しています:

```
dependency-guard 1.2.0
  Dependency analysis for Claude Code sessions
  Source: dependency-guard@example-marketplace

Component inventory
  Skills (2)  scan-dependencies, review-changes
  Agents (0)
  Hooks (1)  SessionStart  (harness-only — no model context cost)
  MCP servers (0)
  LSP servers (0)

Projected token cost
  Always-on:   ~180 tok   added to every session

Per-component (rounded)
  component            always-on  on-invoke
  scan-dependencies        ~100      ~2400
  review-changes            ~80      ~1800

  On-invoke cost is paid each time a skill or agent fires.
  Token counts are estimates and may differ from actual usage.
```

常時オンの合計は、アクティブなモデルの count_tokens API を介して計算されます。コンポーネントごとの数値はその合計から比例的にスケーリングされます。API に到達できない場合、コマンドは文字ベースの推定値にフォールバックします。

23.8.10 plugin validate

公開前にプラグインまたはマーケットプレイスの構文とスキーマエラーをチェックします。

検証が成功すると終了コード 0、失敗すると 1、検証実行自体が失敗した場合（渡したパスが読み取り不可能な場合など）は 2 で終了します。

```
claude plugin validate <path> [options]
```

コマンドは以下の引数を取ります:

- <path>: プラグインディレクトリまたはマーケットプレイスディレクトリへのパス。プラグイン実行がカバーするファイルについては、**マニフェストなしでプラグインまたはディレクトリを検証** (ページ 671) を参照してください。

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
`--strict`	警告をエラーとして扱い、それらで終了コード 1 で終了します。CI で使用して、**認識されないフィールド** (ページ 593) など、ランタイムが許容する問題をキャッチします	
`--json`	検証レポートを同じ終了コードを持つ 1 つの JSON オブジェクトとして出力します。Claude Code v2.1.259 以降が必須	
`-h, --help`	コマンドのヘルプを表示	

`--json` を使用すると、Claude Code はレポートを stdout に 1 つの JSON オブジェクトとして書き込み、これらのトップレベルフィールドを持ちます:

- `success`: 終了コードが与える同じ判定
- `strict`: 実行が警告をエラーとして扱ったかどうか
- `target`: Claude Code が検証した解決されたパス
- `manifest`: マニフェスト自体の結果、または**マニフェストなしの実行** (ページ 671) の場合は `null`
- `contents`: ファイルごとの結果。各結果は `file` を指定し、`errors`、`warnings`、および `notes` 配列を含みます

終了コード 2 では、コマンドは stdout に何も書き込みません。エラーメッセージは stderr に送られます。

対話的セッション内では、`/plugin validate <path>` は同じチェックをインラインで実行します。

23.8.11 plugin eval

プラグインの eval ケースを実行し、スコア付き結果をレポートします。Claude Code v2.1.269 以降が必須です。各ケースはプロンプトとグレーダーです。Claude Code はターゲットプラグインのみが読み込まれた分離されたセッションで複数回実行し、デフォルトではプラグインなしでも実行するため、レポートは差を示します。ケース形式、グレーダー、結果、および CI 使用については、プラグインを eval でテストするを参照してください。

```
claude plugin eval [target] [options]
```

オプションの target は、プラグインディレクトリ、単一の prompt.md または case.yaml ファイル、name または name@marketplace としてインストールされたプラグイン、または name@skills-dir であり、デフォルトは現在のディレクトリです。--tag、--allow-tools、および --json の前に配置します。

このテーブルは、ほとんどの実行が使用するオプションをリストします。claude plugin eval --help を実行して、--case、--tag、--output-dir、--report、--allow-real-servers、--keep-temp、および --verbose を含む完全なセットを確認してください。

オプション	説明	デフォルト
--runs <n>	アーム当たりケース当たりの実行	各ケースの runs、それ以外は 3
-j, --concurrency <n>	一度に実行するエージェントセッション、1 から 8。レート制限を共有します	1
--model <model>	テスト対象のエージェント用モデル	各ケースの model、それ以外は ANTHROPIC_MODEL が設定されている場合はそれ、それ以外は Claude Code のデフォルト
--judge-model <model>	llm および baseline グレーダー用モデル	小さく高速なモデル
--ablation <mode>	none または with-without。プラグインなしベースラインと比較するを参照	プラグインが解決される場合は with-without、それ以外は none
--threshold <0..1>	いずれかのケースがこれ以下でスコアされた場合は終了コード 1	1.0
--max-cost-usd <usd>	支出がこれに達したら次の実行前に停止し、終了コード 2 を返し、部分的な結果をレポート	上限なし
--allow-tools <tools...>	Bash、Write、Edit、または "mcp__plugin_<plugin>_<server>__*" など、読み取り専用セット以外のツールを付与します。ツールを付与するを参照	
--scaffold	各ケースの scaffold_script を実行	オフ
--trust-plugin	最初の実行信頼プロンプトをスキップします。CI 用。実行がアクセスできるものを参照	オフ
--mocks <mode>	record または off。MCP サーバーをモックを参照	record
--eval-dir <dir>	ケースを保持するプラグイン下のディレクトリ	マニフェストの experimental.evals、それ以外は evals
--json [path]	結果ドキュメントを stdout に出力するか、.json パスに書き込み	
--no-publish	HTML レポートをローカルに保持	
-h, --help	コマンドのヘルプを表示	

コマンドは、すべてのケースがしきい値を満たす場合は終了コード 0、失敗したケース、読み込みエラー、または信頼されていないプラグインディレクトリの場合は 1、部分的な実行の場合は 2、中断された場合は 130、終了された場合は 143 で終了します。CI で eval を実行するを参照してください。

23.8.12 plugin eval init

現在のディレクトリのプラグイン用の eval スイートを作成します。Claude Code v2.1.269 以降が必須です。ターミナルでは、これはプラグインを読み取り、ケースとグレーダーを提案し、それらをパイロットし、ファイルを書き込むオーサリングインタビューを開始します。`--bare` を使用するか、ターミナルなしで、代わりに空白の単一ケーステンプレートを書き込みます。対話的な Claude Code セッション内から実行すると、そのセッションが従うべきインタビュー指示を出力します。最初の eval スイートを作成するを参照してください。

```
claude plugin eval init [name] [options]
```

オプションの `name` はケース名です: インタビューは 1 つを必要としませんが、`--bare` とターミナルなしテンプレートパスはそれを必要とします。これらのオプションを受け入れます:

オプション	説明	デフォルト
`--bare`	インタビューを実行する代わりに、`<name>` 用の空白の `prompt.md` と `graders/criteria.md` を書き込み	
`-i, --interactive`	インタビューを必須にします。テンプレートを書き込む代わりにターミナルなしで失敗	
`--eval-dir <dir>`	ケースを書き込む現在のディレクトリ下のディレクトリ	マニフェストの `experimental.evals`、それ以外は `evals`
`-h, --help`	コマンドのヘルプを表示	

23.8.13 plugin tag

プラグインのリリース git タグを作成します。デフォルトではコマンドは現在のディレクトリのプラグインにタグを付けます。別の場所のプラグインにタグを付けるにはパスを渡します。プラグインリリースにタグを付けるを参照してください。

```
claude plugin tag [path] [options]
```

コマンドは以下の引数を取ります:

- `[path]`: プラグインディレクトリへのパス。デフォルトは現在のディレクトリです。

コマンドは以下のオプションを受け入れます:

オプション	説明	デフォルト
`--push`	タグを作成した後、リモートにプッシュします	
`--dry-run`	タグを作成せずにタグ付けされるものを出力します	
`-f, --force`	ワーキングツリーがダーティであるか、タグが既に存在する場合でもタグを作成します	
`-m, --message <msg>`	タグアノテーションメッセージ。バージョンのプレースホルダーとして `%s` を使用します	
`--remote <name>`	`--push` でプッシュするリモート	`origin`
`-h, --help`	コマンドのヘルプを表示	

23.9 デバッグと開発ツール

23.9.1 デバッグコマンド

`claude --debug` を使用してプラグインの読み込み詳細を確認します：

これにより以下が表示されます：

- どのプラグインが読み込まれているか
- プラグインマニフェストのエラー
- Skill、agent、hook の登録
- MCP サーバーの初期化

23.9.2 よくある問題

問題	原因	解決策
プラグインが読み込まれない	無効な `plugin.json`	`claude plugin validate ./my-plugin` または `/plugin validate ./my-plugin` を実行します。ここで `./my-plugin` はプラグインディレクトリです。`plugin.json`、`hooks/hooks.json`、およびプラグインのデフォルトディレクトリ内の skill、agent、command のフロントマターの構文とスキーマエラーをチェックします。実行内容については **プラグインまたはマニフェストなしのディレクトリを検証する** (ページ 671) を参照してください
Skill が表示されない	ディレクトリ構造が間違っている	`skills/` または `commands/` がプラグインルートにあることを確認します。`.claude-plugin/` 内にはありません
Hook が発火しない	スクリプトが実行可能でない	`chmod +x script.sh` を実行します

次のページに続く

表 33 – 前のページからの続き

問題	原因	解決策
MCP サーバーが失敗する	`${CLAUDE_PLUGIN_ROOT}` が見つからない	すべてのプラグインパスに変数を使用します
パスエラー	絶対パスが使用されている	パスを相対パスにします。`./` で始まります。**パス動作ルール** (ページ 601) を参照してください。これは `skills` フィールドの `"."` 例外をカバーしています
LSP `Executable not found in $PATH`	言語サーバーがインストールされていない	バイナリをインストールします（例：`npm install -g typescript-language-server typescript`）

23.9.3 エラーメッセージの例

マニフェスト検証エラー：

- `Invalid JSON syntax: Unexpected token } in JSON at position 142`：コンマの欠落、余分なコンマ、またはクォートされていない文字列がないか確認してください
- `Plugin <name> has an invalid manifest file at .claude-plugin/plugin.json. Validation errors: name: Invalid input: expected string, received undefined`：必須フィールドが見つかりません
- `Plugin <name> has a corrupt manifest file at .claude-plugin/plugin.json. JSON parse error: ...`：JSON 構文エラー。v2.1.246 より前では、Claude Code は UTF-8 で保存され、バイト順マーク（BOM）が先頭にある `plugin.json` に対してもこのエラーを生成していました。JSON が有効な場合でも同様です。

プラグイン読み込みエラー：

- `Warning: No commands found in plugin my-plugin custom directory: ./cmds. Expected .md files or SKILL.md in subdirectories.`：コマンドパスは存在しますが、有効なコマンドファイルが含まれていません
- `Plugin directory not found at path: ./plugins/my-plugin. Check that the marketplace entry has the correct path.`：marketplace.json の `source` パスが存在しないディレクトリを指しています
- `Plugin my-plugin has conflicting manifests: both plugin.json and marketplace entry specify components.`：重複するコンポーネント定義を削除するか、marketplace エントリから `strict: false` を削除します

23.9.4 Hook のトラブルシューティング

Hook スクリプトが実行されない：

1. スクリプトが実行可能であることを確認します：`chmod +x ./scripts/your-script.sh`
2. shebang 行を確認します：最初の行は `#!/bin/bash` または `#!/usr/bin/env bash` である必要があります

3. パスが ${CLAUDE_PLUGIN_ROOT} を使用していることを確認します："command": "\"${CLAUDE_PLUGIN_ROOT}\"/scripts/your-script.sh"
4. スクリプトを手動でテストします：./scripts/your-script.sh

Hook が予期されたイベントでトリガーされない：

1. イベント名が正しいことを確認します（大文字と小文字を区別）：postToolUse ではなく PostToolUse
2. マッチャーパターンがツールと一致することを確認します：ファイル操作の場合は "matcher": "Write|Edit"
3. hook タイプが有効であることを確認します：command、http、mcp_tool、prompt、または agent

23.9.5 MCP サーバーのトラブルシューティング

サーバーが起動しない：

1. コマンドが存在し、実行可能であることを確認します
2. すべてのパスが ${CLAUDE_PLUGIN_ROOT} 変数を使用していることを確認します
3. MCP サーバーログを確認します：claude --debug は初期化エラーを表示します
4. Claude Code の外部でサーバーを手動でテストします

サーバーツールが表示されない：

1. サーバーが .mcp.json または plugin.json で正しく設定されていることを確認します
2. サーバーが MCP プロトコルを正しく実装していることを確認します
3. デバッグ出力で接続タイムアウトを確認します

23.9.6 ディレクトリ構造の間違い

症状：プラグインは読み込まれますが、コンポーネント（skill、agent、hook）が見つかりません。

正しい構造：コンポーネントはプラグインルートにある必要があります。.claude-plugin/ 内にはありません。plugin.json のみが .claude-plugin/ に属します。

デバッグチェックリスト：

1. claude --debug を実行し、「loading plugin」メッセージを探します
2. 各コンポーネントディレクトリがデバッグ出力に表示されていることを確認します
3. ファイルのアクセス許可がプラグインファイルの読み取りを許可していることを確認します

23.10 配布とバージョン管理リファレンス

23.10.1 バージョン管理

Claude Code はプラグインのバージョンをキャッシュキーとして使用し、アップデートが利用可能かどうかを判断します。/plugin update を実行するか自動アップデートが実行されると、Claude Code は現在

のバージョンを計算し、既にインストールされているものと一致する場合はアップデートをスキップします。**ローカルディレクトリマーケットプレイスから所定の場所に読み込まれた** (ページ 605) プラグインは、バージョン文字列が何を示していても、セッション開始時に現在のソースファイルを読み込みます。

command 以外のすべてのソースタイプについて、Claude Code は以下の最初に設定されたものからバージョンを解決します。

1. プラグインの `plugin.json` の `version` フィールド
2. `marketplace.json` のプラグインのマーケットプレイスエントリの `version` フィールド
3. git ホストマーケットプレイス内の `github`、`url`、`git-subdir`、および相対パスソースのプラグインの git コミット SHA
4. *archive* **ソース** (ページ 642) の SHA-256 ダイジェスト。マーケットプレイスエントリの `sha256` ピン、またはピンを設定しない場合はダウンロードされたファイルのダイジェスト。Claude Code はこれを最初の 12 文字に短縮します
5. `npm` ソースまたは git リポジトリ内にないローカルディレクトリの場合は `unknown`。Claude Code は、`~/.claude` のような git 管理されたインストールパスを囲むリポジトリからバージョンを取得しません

command **ソース** (ページ 647) の場合、Claude Code は常にコマンドが生成したものからバージョンを導出します。単独の 12 文字のコンテンツハッシュ、または 1 つが設定されている場合は `plugin.json` バージョンに `<version>-<hash>` として追加されます。Claude Code はコマンドソースのマーケットプレイスエントリの `version` フィールドを無視します。ハッシュされた出力が変更されるコマンドは、作成されたバージョン文字列が同じままでも、新しいバージョンを生成します。**リンクモード** (ページ 648) では、ハッシュはファイルコンテンツではなく、印刷されたディレクトリの実際のパスとそのトップレベルエントリをカバーします。

これらのソースタイプについて、プラグインをバージョン管理する 3 つの方法があります。

アプローチ	方法	アップデート動作	最適な用途
明示的なバージョン	`plugin.json` で `"version": "2.1.0"` を設定	ユーザーはこのフィールドをバンプした場合のみアップデートを取得します。バンプせずに新しいコミットをプッシュしても効果がなく、`/plugin update` は「既に最新バージョンです」と報告します。**所定の場所に読み込まれたプラグイン** (ページ 605) の場合、新しいコンテンツは読み込まれます。	安定したリリースサイクルを持つ公開プラグイン

次のページに続く

表 34 – 前のページからの続き

アプローチ	方法	アップデート動作	最適な用途
コミット SHA バージョン	plugin.json とマーケットプレイスエントリの両方から version を省略	ユーザーはソースの解決されたコミットが変更されるたびにアップデートを取得します	アクティブに開発中の内部またはチームプラグイン
ダイジェストバージョン	*archive* **ソース** (ページ 642) を使用し、plugin.json とマーケットプレイスエントリの両方から version を省略	sha256 ピンを使用する場合、ユーザーはピンを変更するとアップデートを取得します。ピンがない場合、ユーザーはホストされている zip ファイルのバイトが変更されるたびにアップデートを取得します	静的サーバーまたはアーティファクトリポジトリに zip ファイルとして公開されるプラグイン

明示的なバージョンを使用する場合は、セマンティックバージョニング（MAJOR.MINOR.PATCH）に従ってください。破壊的な変更の場合は MAJOR をバンプし、新機能の場合は MINOR をバンプし、バグ修正の場合は PATCH をバンプします。CHANGELOG.md で変更を文書化します。

23.11 関連項目

- **プラグイン** (ページ 565) - チュートリアルと実践的な使用法
- **プラグインマーケットプレイス** (ページ 627) - マーケットプレイスの作成と管理
- *Skills* (ページ 275) - Skill 開発の詳細
- *Subagents* (ページ 317) - エージェント設定と機能
- *Hooks* (ページ 443) - イベント処理と自動化
- *MCP* (ページ 363) - 外部ツール統合
- **設定** (ページ 711) - プラグインの設定オプション

第24章　プラグインマーケットプレイスの作成と配布

Claude Code 拡張機能を配布するためのプラグインマーケットプレイスを構築およびホストします。

プラグインマーケットプレイスは、他のユーザーにプラグインを配布できるカタログです。マーケットプレイスは、一元化された検出、バージョン追跡、自動更新、および複数のソースタイプ（Git リポジトリ、ローカルパスなど）のサポートを提供します。このガイドでは、チームやコミュニティとプラグインを共有するための独自のマーケットプレイスを作成する方法を説明します。

既存のマーケットプレイスからプラグインをインストールしたいですか？**既成プラグインの検出とインストール** (ページ 677) を参照してください。

24.1 概要

マーケットプレイスの作成と配布には、以下が含まれます。

1. **プラグインの作成**：skills、agents、hooks、MCP サーバー、または LSP サーバーを使用して 1 つ以上のプラグインを構築します。このガイドでは、配布するプラグインが既にあることを前提としています。プラグインの作成方法の詳細については、**プラグインの作成** (ページ 565) を参照してください。
2. **マーケットプレイスファイルの作成**：プラグインとその場所を一覧表示する `marketplace.json` を定義します。**マーケットプレイスファイルの作成** (ページ 630) を参照してください。
3. **マーケットプレイスのホスト**：GitHub、GitLab、または別の Git ホストにプッシュします。**マーケットプレイスのホストと配布** (ページ 652) を参照してください。
4. **ユーザーと共有**：ユーザーが `/plugin marketplace add` でマーケットプレイスを追加し、個別のプラグインをインストールします。**プラグインの検出とインストール** (ページ 677) を参照してください。

マーケットプレイスがライブになったら、リポジトリに変更をプッシュして更新できます。ユーザーは `/plugin marketplace update` でローカルコピーを更新します。

24.2 チュートリアル：ローカルマーケットプレイスの作成

この例では、1 つのプラグイン（コードレビュー用の `quality-review` skill）を含むマーケットプレイスを作成します。ディレクトリ構造を作成し、skill を追加し、プラグインマニフェストとマーケットプレイスカタログを作成してから、インストールしてテストします。

1. **ディレクトリ構造の作成**

```
mkdir -p my-marketplace/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/.claude-plugin
mkdir -p my-marketplace/plugins/quality-review-plugin/skills/quality-review
```

2. **skill の作成**

   `quality-review` skill が何をするかを定義する `SKILL.md` ファイルを作成します。

   リスト 1: my-marketplace/plugins/quality-review-plugin/skills/quality-review/SKILL.md

```
---
description: Review code for bugs, security, and performance
---

Review the code I've selected or the recent changes for:
- Potential bugs or edge cases
- Security concerns
- Performance issues
- Readability improvements

Be concise and actionable.
```

3. **プラグインマニフェストの作成**

   プラグインを説明する `plugin.json` ファイルを作成します。マニフェストは `.claude-plugin/` ディレクトリに配置されます。

   リスト 2: my-marketplace/plugins/quality-review-plugin/.claude-plugin/plugin.json

```
{
  "name": "quality-review-plugin",
  "description": "Adds a quality-review skill for quick code reviews",
  "version": "1.0.0",
  "author": {
    "name": "Your Name"
  }
}
```

注釈:

`version` を設定すると、ユーザーはこのフィールドを変更した場合にのみ更新を受け取ります。そのため、リリースのたびにバージョンを上げてください。*command* **ソース** (ページ 647) を持つプラグインはこのフィールドでピン留めされません。**ローカルディレクトリから追加されたマーケットプレイスから** (ページ 605) その場で読み込まれるプラグインもそうです。`version` を省略した場合、バージョンは**バージョン管理** (ページ 624) の次のソースから取得されます。

4. **マーケットプレイスファイルの作成**

   プラグインを一覧表示するマーケットプレイスカタログを作成します。

リスト 3: my-marketplace/.claude-plugin/marketplace.json

```
{
  "name": "my-plugins",
  "owner": {
    "name": "Your Name"
  },
  "plugins": [
    {
      "name": "quality-review-plugin",
      "source": "./plugins/quality-review-plugin",
      "description": "Adds a quality-review skill for quick code reviews"
    }
  ]
}
```

5. **追加とインストール**

   `my-marketplace` を含むディレクトリから Claude Code を起動し、以下のコマンドを実行します。install コマンドはプラグイン詳細ビューを開き、インストールスコープを選択してインストールを確認します。インストール概要を確認します。`Run /reload-plugins to activate.` と報告される場合は、**プラグイン変更の再起動なしでの適用** (ページ 689) を参照してください。

```
/plugin marketplace add ./my-marketplace
/plugin install quality-review-plugin@my-plugins
```

6. **試してみる**

   エディタでコードを選択し、新しい skill を実行します。プラグイン skill はプラグイン名でネームスペース化されます。

```
/quality-review-plugin:quality-review
```

プラグインが実行できることの詳細（hooks、agents、MCP サーバー、LSP サーバーを含む）については、**プラグイン** (ページ 565) を参照してください。

注釈:

プラグインのインストール方法：ユーザーがプラグインをインストールすると、Claude Code はプラグインディレクトリをキャッシュロケーションにコピーします。ただし、プラグインがその場で読み込まれる場合は除きます。*link mode* (ページ 648) の *command* **ソース** (ページ 647) はその場で読み込まれ、**ローカルディレクトリから追加されたマーケットプレイスの相対パスソース** (ページ 638) もそうです。コピーされたプラグインは、`../shared-utils` のようなパスを使用してプラグインディレクトリの外部のファイルを参照できません。これらのファイルはコピーされないためです。
プラグイン間でファイルを共有する必要がある場合は、symlinks を使用します。詳細については、**プラグインキャッシングとファイル解決** (ページ 605) を参照してください。

24.3 マーケットプレイスファイルの作成

リポジトリルートに `.claude-plugin/marketplace.json` を作成します。このファイルは、マーケットプレイスの名前、所有者情報、およびソースを含むプラグインのリストを定義します。

各プラグインエントリには、最低限 `name` と `source`（Claude Code がどこから取得するかを指定）が必要です。利用可能なすべてのフィールドについては、以下の**完全なスキーマ** (ページ 631) を参照してください。

```
{
  "name": "company-tools",
  "owner": {
    "name": "DevTools Team",
    "email": "devtools@example.com"
  },
  "plugins": [
    {
      "name": "code-formatter",
      "source": "./plugins/formatter",
      "description": "Automatic code formatting on save",
      "version": "2.1.0",
      "author": {
        "name": "DevTools Team"
      }
    },
    {
      "name": "deployment-tools",
      "source": {
        "source": "github",
        "repo": "company/deploy-plugin"
      },
      "description": "Deployment automation tools"
    }
  ]
```

(次のページに続く)

(前のページからの続き)

```
}
```

24.4 マーケットプレイススキーマ

24.4.1 必須フィールド

フィールド	タイプ	説明	例
`name`	string	ケバブケースのマーケットプレイス識別子。スペース、制御文字、双方向フォーマット文字は含まれません。これは公開向けです。ユーザーはプラグインをインストールするときに表示されます（例：`/plugin install my-tool@your-marketplace`）。各ユーザーは、マーケットプレイス名ごとに 1 つのマーケットプレイスのみを登録できます。同じ名前の 2 番目のマーケットプレイスを追加すると、Claude Code は最初のマーケットプレイスを置き換えます。1 つのマーケットプレイス名の下に複数のプラグインを公開するには、すべてを**単一の** *marketplace.json* (ページ 630) にリストします。	`"acme-tools"`
`owner`	object	マーケットプレイスメンテナー情報。**所有者フィールド** (ページ 632) を参照してください	
`plugins`	array	利用可能なプラグインのリスト	**プラグインエントリ** (ページ 633) を参照してください

注釈:

予約名：以下のマーケットプレイス名は Anthropic の公式使用のために予約されており、サードパーティのマーケットプレイスでは使用できません：`claude-code-marketplace`、`claude-code-plugins`、`claude-plugins-official`、`claude-plugins-community`、`claude-community`、`anthropic-marketplace`、`anthropic-plugins`、`agent-skills`、`anthropic-agent-skills`、`knowledge-work-plugins`、`life-sciences`、`claude-for-legal`、`claude-for-financial-services`、`financial-services-plugins`、`first-party-plugins`、`claude-tag-plugins`、`healthcare`。公式マーケットプレイスになりすましている名前（`official-claude-plugins` や `anthropic-plugins-v2` など）もブロックされています。これらの名前を予約することで、サードパーティのマーケットプレイスが Anthropic 公開ソースとして自らを提示することを防ぎます。

Claude Code は、マーケットプレイスを追加するときだけでなく、マーケットプレイスをロードするたびに予約名を再チェックします。これらの名前の 1 つの下に登録されていたマーケットプレイスが、その名前が予約されるようになると、ロードが停止し、信頼できないソースから登録されていることを報告します。そのマーケットプレイスを削除し、公式 Anthropic ソースから再度追加してください。新しく予約された名前の影響を受けるサードパーティのマーケットプレイスは、別の名前の下で再度追加するとすぐにロードされます。v2.1.205 より前では、`first-party-plugins` と `healthcare` は予約されておらず、予約名の下に既に登録されているマーケットプレイスはロードされ続けていました。v2.1.265 より前では、`claude-tag-plugins` は予約されていませんでした。

マーケットプレイスに `npm`、`pip`、`uv`、`cargo`、`github`、または `gh` という名前を付けることもできません。大文字小文字は問いません。このチェックには Claude Code v2.1.275 以降が必要です。

24.4.2 所有者フィールド

フィールド	タイプ	必須	説明
`name`	string	はい	メンテナーまたはチームの名前
`email`	string	いいえ	メンテナーの連絡先メール
`url`	string	いいえ	ウェブサイト、GitHub プロフィール、または組織の URL

24.4.3 オプションフィールド

フィールド	タイプ	説明
`$schema`	string	エディターのオートコンプリートと検証用の JSON Schema URL。Claude Code はロード時にこのフィールドを無視します。
`description`	string	マーケットプレイスの簡潔な説明
`version`	string	マーケットプレイスマニフェストバージョン

次のページに続く

表 3 – 前のページからの続き

フィールド	タイプ	説明
metadata.pluginRoot	string	Claude Code が裸のプラグインソース名を解決するディレクトリ。**相対パス** (ページ 638) を参照してください。Claude Code v2.1.239 以降が必要です。
allowCrossMarketplaceDependenciesOn	array	このマーケットプレイス内のプラグインが依存する可能性のある他のマーケットプレイス。ここにリストされていないマーケットプレイスからの依存関係はインストール時にブロックされます。別のマーケットプレイスからプラグインに依存するを参照してください。
renames	object	プラグインの以前の name から現在の名前へのマッピング、またはプラグインが削除された場合は null。マーケットプレイス内のエントリの名前を変更または削除するときに、既存ユーザーが自動的に移行できるようにします。**プラグインの名前変更または削除** (ページ 664) を参照してください。Claude Code v2.1.193 以降が必要です。

description と version は後方互換性のため metadata の下でも受け入れられます。

24.5 プラグインエントリ

plugins 配列内の各プラグインエントリは、プラグインとその場所を説明します。**プラグインマニフェストスキーマ** (ページ 592) のフィールド（description、version、author、commands、hooks など）を含めることができます。さらに、これらのマーケットプレイス固有のフィールド：source、category、tags、strict、relevance、headers、および headersHelper があります。

24.5.1 必須フィールド

フィールド	タイプ	説明
name	string	ケバブケースのプラグイン識別子。スペース、制御文字、双方向フォーマット文字は含まれません。これは公開向けです。ユーザーはインストール時に表示されます（例：/plugin install my-plugin@marketplace）。
source	string|object	プラグインを取得する場所 (以下の**プラグインソース** (ページ 636) を参照)

24.5.2 オプションプラグインフィールド

標準メタデータフィールド：

フィールド	タイプ	説明
`displayName`	string	UI サーフェスに表示される人間が読める名前。エントリもプラグインの `plugin.json` も設定しない場合、ユーザーはプラグインの `name` を表示されます。スペースと任意の大文字小文字を含めることができます。名前空間指定またはルックアップには使用されません。
`description`	string	プラグインの簡潔な説明
`version`	string	プラグインバージョン。設定されている場合（ここまたは `plugin.json` で）、プラグインはこの文字列にピン留めされ、ユーザーは変更時にのみ更新を受け取ります。**コマンドソース** (ページ 647) を持つプラグインは、どちらのフィールドでもピン留めされません。**マーケットプレイスから所定の場所に読み込まれた** (ページ 605) プラグインもそうではありません。どちらにも設定されていない場合、バージョンは**バージョン管理** (ページ 624) の次のソースから取得されます。
`author`	object	プラグイン作成者情報（`name` は必須、`email` と `url` はオプション）
`homepage`	string	プラグインホームページまたはドキュメント URL
`repository`	string	ソースコードリポジトリ URL
`license`	string	SPDX ライセンス識別子（例：MIT、Apache-2.0）
`keywords`	array	プラグイン検出と分類用のタグ
`metadata`	object	エンタイトルメントやカタログデータなど、独自のフィールド用のフリーフォームオブジェクト。Claude Code はこれを読みません。v2.1.222 より前では、`claude plugin validate` はキーを認識されないフィールドとして報告していました。
`category`	string	整理用のプラグインカテゴリ
`tags`	array	検索可能性用のタグ
`strict`	boolean	`plugin.json` がコンポーネント定義の権限であるかどうかを制御します（デフォルト：true）。以下の**厳密モード** (ページ 652) を参照してください。
`relevance`	object	Claude Code がこのプラグインをユーザーに提案するタイミングを示すシグナル。管理者が管理設定でホワイトリストに登録したマーケットプレイスに対してのみ有効になります。組織向けプラグインの推奨を参照してください。

次のページに続く

表 5 – 前のページからの続き

フィールド	タイプ	説明
`defaultEnabled`	boolean	プラグインがインストール後に有効になるかどうか（デフォルト：true）。ユーザーがオプトインするまでプラグインを無効にしてインストールする場合は `false` に設定します。プラグインの `plugin.json` 内の同じフィールドより優先されます。**デフォルト有効化** (ページ 595) を参照してください。

エントリとプラグイン自体の `plugin.json` の両方が、表示フィールド `displayName`、`description`、`author`、`homepage`、`repository`、`license`、および `keywords` を設定できます。プラグインリストと詳細では、インストール前後：

- エントリで設定したフィールドについては、`plugin.json` が異なる値を設定している場合でも、ユーザーはエントリの値を表示されます。
- エントリが設定していないフィールドについては、ユーザーは `plugin.json` の値を表示されます。

インストール前に、Claude Code は**相対パスソース** (ページ 638) を持つエントリの `plugin.json` のみを読むことができます。そのプラグインファイルはマーケットプレイス内に存在します。他のソースタイプを持つエントリの場合、ユーザーはプラグインをインストールするまで、エントリ自体のフィールドのみを表示されます。

コンポーネント設定フィールド：

フィールド	タイプ	説明
`skills`	string|array	`<name>/SKILL.md` を含む skill ディレクトリへのカスタムパス
`commands`	string|array	フラットな `.md` skill ファイルまたはディレクトリへのカスタムパス
`agents`	string|array	agent ファイルへのカスタムパス
`hooks`	string|object	カスタム hooks 設定または hooks ファイルへのパス
`mcpServers`	string|object	MCP サーバー設定または MCP 設定ファイルへのパス
`lspServers`	string|object	LSP サーバー設定または LSP 設定ファイルへのパス

アーカイブ認証フィールド：

エントリが認証情報を必要とするサーバー上の *archive* **ソース** (ページ 642) を持つ場合、これらを設定します。

フィールド	タイプ	説明
`headers`	object	Claude Code がこのエントリのアーカイブをダウンロードするときに送信する HTTP ヘッダー。マーケットプレイスの同じ名前のヘッダーをオーバーライドします。Claude Code v2.1.238 以降が必要です。

次のページに続く

表 7 – 前のページからの続き

フィールド	タイプ	説明
`headersHelper`	string	このエントリのアーカイブダウンロード用の HTTP ヘッダーを 1 つの JSON オブジェクトとして出力するコマンド。有効期限が切れる認証情報用です。**アーカイブダウンロードの認証** (ページ 643) を参照してください。エントリは `"strict": false` (ページ 652) も設定する必要があります。Claude Code v2.1.238 以降が必要です。

24.6 プラグインソース

プラグインソースは、Claude Code にマーケットプレイスにリストされた各プラグインをどこから取得するかを指示します。これらは `marketplace.json` の各プラグインエントリの `source` フィールドで設定されます。

Claude Code は、インストール済みの各プラグインをローカルバージョン管理されたプラグインキャッシュ（`~/.claude/plugins/cache`）にコピーします。ただし、プラグインがその場で読み込まれる場合は例外です。**リンクモードの *command* ソース** (ページ 648) はその場で読み込まれ、**ローカルディレクトリから追加されたマーケットプレイスの相対パスソース** (ページ 638) も同様です。Claude Code はまた、**プラグインの対象となる *Node.js* パッケージ依存関係** (ページ 605) をキャッシュされたコピーにインストールします。ローカルディレクトリマーケットプレイスからその場で読み込まれたプラグインが編集内容を取得する方法については、**プラグインキャッシングとファイル解決** (ページ 605) を参照してください。

ソース	タイプ	フィールド	注記
相対パス	`string`（例：`"./my-plugin"`）	なし	マーケットプレイスリポジトリ内のローカルディレクトリ。`./` で始まる必要があります。ただし、***metadata.pluginRoot* の下に裸の名前を記述する** (ページ 638) 場合は除きます。Claude Code はパスを `.claude-plugin/` ディレクトリではなく、マーケットプレイスルートを基準に解決します
`github`	object	`repo`, `ref?`, `sha?`	
`url`	object	`url`, `ref?`, `sha?`	Git URL ソース
`git-subdir`	object	`url`, `path`, `ref?`, `sha?`	git リポジトリ内のサブディレクトリ。スパース部分クローンを使用して、モノレポの帯域幅を最小化します
`npm`	object	`package`, `version?`, `registry?`	npm パッケージ。npm クライアントでフェッチされ、インストールスクリプトを実行せずに解凍されます

次のページに続く

表 8 – 前のページからの続き

ソース	タイプ	フィールド	注記
`archive`	object	`url`, `sha256?`	HTTPS でダウンロードされた zip アーカイブ。ユーザーのマシンに git や npm がなくても動作します。Claude Code v2.1.224 以降が必要です
`command`	object	`command`, `timeout?`, `mode?`	ローカルコマンドを実行して生成されたプラグインディレクトリ。セッションごとに 1 回再実行して変更を反映します。Claude Code v2.1.229 以降が必要です

注釈:

マーケットプレイスソースとプラグインソース: これらは異なる概念で、異なるものを制御します。

- **マーケットプレイスソース**: `marketplace.json` カタログ自体をどこから取得するか。ユーザーが `/plugin marketplace add` を実行するか、`extraKnownMarketplaces` 設定で設定されます。Git ベースのマーケットプレイスソースは `ref`（ブランチ/タグ）をサポートしますが、`sha` はサポートしません。
- **プラグインソース**: マーケットプレイスにリストされた個別プラグインをどこから取得するか。`marketplace.json` 内の各プラグインエントリの `source` フィールドで設定されます。Git ベースのプラグインソースは `ref`（ブランチ/タグ）と `sha`（正確なコミット）の両方をサポートします。

例えば、`acme-corp/plugin-catalog`（マーケットプレイスソース）でホストされているマーケットプレイスは、`acme-corp/code-formatter`（プラグインソース）から取得されたプラグインをリストできます。マーケットプレイスソースとプラグインソースは異なるリポジトリを指し、独立して固定されます。

以下の Git ベースのソースタイプは `github`、`url`、および `git-subdir` です。`ref` と `sha` の両方が設定されている場合、`sha` が有効なピンになります。Claude Code はピンされたコミットを直接フェッチしてチェックアウトします。

GitHub、GitLab、Bitbucket を含むほとんどの git ホストでは、ブランチまたはタグが `ref` で指定されていても、その後アップストリームで削除されていても、コミットがリポジトリから到達可能である限り、インストールは成功します。AWS CodeCommit などの一部のサーバーは、SHA でコミットをフェッチすることをサポートしていません。これらのサーバーでは、`ref` が存在し、ピンされたコミットがそこから到達可能である必要があります。

組織設定 > プラグイン を通じてプラグインを配布する場合、一部のソースタイプのみが許可されます。**組織設定を通じた配布** (ページ 654) を参照してください。

24.6.1 相対パス

同じリポジトリ内のプラグインの場合、./ で始まるパスを使用します：

```
{
  "name": "my-plugin",
  "source": "./plugins/my-plugin"
}
```

パスはマーケットプレイスルート（.claude-plugin/ を含むディレクトリ）を基準に解決されます。上記の例では、marketplace.json が <repo>/.claude-plugin/marketplace.json にあっても、./plugins/my-plugin は <repo>/plugins/my-plugin を指します。マーケットプレイスルートの外のパスを参照するために ../ を使用しないでください。macOS と Linux では、Claude Code は先頭の ./ より後のどこかにバックスラッシュがあるエントリパスを拒否するため、すべてのプラットフォームで区切り文字を / として記述してください。

裸の名前は、"formatter" のように / を含まない単一のディレクトリ名です。./ パスの代わりに裸の名前を記述するには、*metadata.pluginRoot* (ページ 632) をそれらが解決されるディレクトリに設定します。"pluginRoot": "./plugins" の場合、Claude Code は "source": "formatter" を ./plugins/formatter に解決します。Claude Code v2.1.239 以降が必要です。

metadata.pluginRoot 自体はマーケットプレイス内の相対パスである必要があります。Claude Code は既に ./ で始まるソースに対しては無視します。team-a/formatter のように / を含むソースは裸の名前ではなく、metadata.pluginRoot が設定されていても ./ プレフィックスが必要です。

注釈:

Claude Code は相対パスをマーケットプレイスのローカルコピーに対して解決するため、ユーザーが git ソースまたはローカルディレクトリからマーケットプレイスを追加する場合に機能します。ユーザーが marketplace.json ファイルへの直接 URL を使用してマーケットプレイスを追加する場合、Claude Code はそのファイルのみをダウンロードするため、相対パスは解決されません。URL ベースの配布の場合は、代わりに他の**プラグインソース** (ページ 636) を使用してください。詳細は**トラブルシューティング** (ページ 675) を参照してください。

24.6.2 GitHub リポジトリ

```
{
  "name": "github-plugin",
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo"
  }
}
```

特定のブランチ、タグ、またはコミットにピンできます：

```
{
  "name": "github-plugin",
```

(次のページに続く)

(前のページからの続き)

```
  "source": {
    "source": "github",
    "repo": "owner/plugin-repo",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

フィールド	タイプ	説明
repo	string	必須。owner/repo 形式の GitHub リポジトリ
ref	string	オプション。Git ブランチまたはタグ（デフォルトはリポジトリのデフォルトブランチ）
sha	string	オプション。正確なバージョンにピンするための 40 文字の完全な git コミット SHA

24.6.3 Git リポジトリ

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git"
  }
}
```

特定のブランチ、タグ、またはコミットにピンできます：

```
{
  "name": "git-plugin",
  "source": {
    "source": "url",
    "url": "https://gitlab.com/team/plugin.git",
    "ref": "main",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

フィールド	タイプ	説明
url	string	必須。完全な git リポジトリ URL（https:// または git@）。.git サフィックスはオプションなので、サフィックスのない Azure DevOps と AWS CodeCommit URL が機能します

次のページに続く

表 10 – 前のページからの続き

フィールド	タイプ	説明
ref	string	オプション。Git ブランチまたはタグ（デフォルトはリポジトリのデフォルトブランチ）
sha	string	オプション。正確なバージョンにピンするための 40 文字の完全な git コミット SHA

24.6.4 Git サブディレクトリ

git-subdir を使用して、git リポジトリのサブディレクトリ内にあるプラグインを指します。Claude Code はスパース部分クローンを使用してサブディレクトリのみをフェッチし、大規模なモノレポの帯域幅を最小化します。

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin"
  }
}
```

特定のブランチ、タグ、またはコミットにピンできます：

```
{
  "name": "my-plugin",
  "source": {
    "source": "git-subdir",
    "url": "https://github.com/acme-corp/monorepo.git",
    "path": "tools/claude-plugin",
    "ref": "v2.0.0",
    "sha": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0"
  }
}
```

url フィールドは GitHub ショートハンド（owner/repo）または SSH URL（git@github.com:owner/repo.git）も受け入れます。

フィールド	タイプ	説明
url	string	必須。Git リポジトリ URL、GitHub owner/repo ショートハンド、または SSH URL
path	string	必須。プラグインを含むリポジトリ内のサブディレクトリパス（例："tools/claude-plugin"）

次のページに続く

表 11 – 前のページからの続き

フィールド	タイプ	説明
ref	string	オプション。Git ブランチまたはタグ（デフォルトはリポジトリのデフォルトブランチ）
sha	string	オプション。正確なバージョンにピンするための 40 文字の完全な git コミット SHA

24.6.5 npm パッケージ

npm ソースは、公開 npm レジストリまたはチームがホストするプライベートレジストリ上の任意のパッケージに名前を付けることができます。Claude Code は npm クライアントでパッケージを解決し、tarball をダウンロードして、プラグインキャッシュに解凍します。

パッケージのインストールスクリプト（preinstall や postinstall など）は実行されず、フェッチ中に依存関係はインストールされません。

パッケージが package.json の隣にサポートされているロックファイルを配布する場合、Claude Code はそれらの *Node.js* **パッケージ依存関係** (ページ 605) を別のステップでインストールし、スクリプトも無効にします。そうでない場合は、必要なすべてのものが既に構築されたプラグインを公開します。他のパッケージが必要な MCP サーバーは、npx を通じて起動でき、最初の実行時にそれらをインストールします。

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin"
  }
}
```

特定のバージョンにピンするには、version フィールドを追加します：

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
    "package": "@acme/claude-plugin",
    "version": "2.1.0"
  }
}
```

プライベートまたは内部レジストリからインストールするには、registry フィールドを追加します：

```
{
  "name": "my-npm-plugin",
  "source": {
    "source": "npm",
```

(次のページに続く)

(前のページからの続き)

```
    "package": "@acme/claude-plugin",
    "version": "^2.0.0",
    "registry": "https://npm.example.com"
  }
}
```

フィールド	タイプ	説明
package	string	必須。パッケージ名またはスコープ付きパッケージ(例:@org/plugin)
version	string	オプション。バージョンまたはバージョン範囲(例:2.1.0、^2.0.0、~1.5.0)
registry	string	オプション。カスタム npm レジストリ URL。デフォルトはシステム npm レジストリ(通常は npmjs.org)

24.6.6 Zip アーカイブ

archive を使用して、Claude Code が HTTPS でダウンロードする zip ファイルとしてプラグインを配布します。これにより、ユーザーのマシンに git や npm がなくてもインストールが機能します。S3 バケット、Artifactory 汎用リポジトリ、nginx などの静的ファイルサーバーまたはアーティファクトリポジトリでファイルをホストします。Claude Code v2.1.224 以降が必要です。v2.1.120 から v2.1.223 では、プラグインのインストールが This plugin uses a source type your Claude Code version does not support. Update Claude Code and try again. で失敗します。より古いバージョンでは、archive エントリを含むマーケットプレイス全体がロードに失敗します。

このエントリはアーティファクトサーバー上の zip ファイルからプラグインをインストールします:

```
{
  "name": "my-plugin",
  "source": {
    "source": "archive",
    "url": "https://artifacts.example.com/claude-plugins/my-plugin-2.1.0.zip"
  }
}
```

zip を構築するときは、プラグインのコンテンツを直接 zip するか、プラグインフォルダ自体を zip できます。Claude Code はアーカイブの最上部で .claude-plugin/ を探し、次に単一の最上位フォルダ内を探すため、両方のレイアウトがインストールされます:

```
my-plugin.zip          my-plugin.zip
├── .claude-plugin/    └── my-plugin/
│   └── plugin.json        ├── .claude-plugin/
└── commands/              │   └── plugin.json
                           └── commands/
```

Claude Code は 1 フォルダより深く探さないため、さらに下にネストされたプラグインはインストールに失敗します。Claude Code は 256 MiB より大きいアーカイブを拒否します。

正確なファイルにピンするには、アーカイブのダイジェストを含む sha256 フィールドを追加します：

```
{
  "name": "my-plugin",
  "source": {
    "source": "archive",
    "url": "https://artifacts.example.com/claude-plugins/my-plugin-2.1.0.zip",
    "sha256": "6bfa50e3d2e00c052b46abe51fff89346ac803e45771f76dcf6df1ab74cca5e1"
  }
}
```

ダウンロードされたファイルがピンと一致しない場合、Claude Code はインストールを拒否し、`Plugin archive integrity check failed` を報告します。

アーカイブソースはこれらのフィールドを受け入れます：

フィールド	タイプ	説明
`url`	string	必須。zip アーカイブの HTTPS URL。Claude Code は `http://` URL、ループバック、リンクローカル、クラウドメタデータホストを拒否します。すべてのリダイレクトホップが同じルールを満たす必要があります。そうでない場合、Claude Code はダウンロードを拒否します
`sha256`	string	オプション。アーカイブの SHA-256 ダイジェスト（64 文字の 16 進数、大文字または小文字）。Claude Code はすべてのダウンロードに対してこれを検証し、不一致の場合はインストールを拒否します

`sha256` ダイジェストは、`plugin.json` またはマーケットプレイスエントリが宣言していない場合、プラグインのバージョンとしても機能します。**バージョン管理** (ページ 624) を参照してください。`version` を宣言する場合、そのバージョン文字列が更新シグナルになるため、zip とそのダイジェストを変更した後、バージョンもバンプしてください。そうしないと、ユーザーはキャッシュされたコピーを保持し続けます。

アーカイブダウンロードの認証

プライベートレジストリからのダウンロードなど、アーカイブダウンロードを認証するには、Claude Code が送信する HTTP ヘッダーを設定します。マーケットプレイスを登録した `url` ソース（`extraKnownMarketplaces` エントリなど）で `headers` を設定します。Claude Code v2.1.238 以降では、プラグインのエントリで `source` の隣に設定できます。

`headers` に入れる値が短命の場合（レジストリがリクエストで生成するトークンなど）、代わりに同じ場所に `headersHelper` コマンドを設定します。Claude Code はコマンドを実行し、それが出力する JSON オブジェクトをその場所のヘッダーとして送信します。Claude Code v2.1.238 以降が必要です。

選択した場所は、どのダウンロードがヘッダーを取得し、Claude Code がコマンドをいつ実行するかを決定します：

場所	ヘッダーを取得するダウンロード	Claude Code が headersHelper をそこで実行する時期
マーケットプレイス url ソース	マーケットプレイス URL のオリジン上のアーカイブダウンロード（同じスキーム、ホスト、ポート）	マーケットプレイスの marketplace.json の各フェッチの前と、そのオリジン上の各アーカイブダウンロードの前。Claude Code は 1 回の実行の出力を最大 60 秒間再利用します
プラグインエントリ	そのエントリのダウンロードのみ	ユーザーがそのプラグインを単独でインストールまたは更新し、**コマンドを受け入れる** (ページ 646) 場合のみ

両方の場所が同じ名前のヘッダーを設定する場合、Claude Code はエントリの値を送信します。1 つの場所内で、コマンドが出力するヘッダーは同じ名前のリストされたヘッダーをオーバーライドします。

プラグインエントリに headersHelper を追加

このエントリは headersHelper を source の隣に設定します。また、"strict": false を設定します。これは Claude Code が headersHelper を設定する marketplace.json エントリに必要です。*"strict": false* (ページ 652) では、マーケットプレイスエントリはプラグインの完全な定義なので、ユーザーはコマンドを受け入れる前にプラグインに含まれるものを確認できます：

```
{
  "name": "my-plugin",
  "description": "Formatting commands for internal services",
  "strict": false,
  "commands": "./commands",
  "source": {
    "source": "archive",
    "url": "https://registry.example.com/plugins/my-plugin-2.1.0.zip"
  },
  "headersHelper": "/opt/bin/mint-registry-token.sh"
}
```

エントリを確認するには、claude plugin install my-plugin@your-marketplace を実行します。Claude Code はコマンドとアーカイブ URL を表示し、受け入れた後に zip をダウンロードします。

v2.1.238 より前では、Claude Code はエントリのアーカイブを headers または headersHelper なしでダウンロードしたため、それらに依存するインストールは HTTP 401 while downloading plugin archive from で失敗し、その後に URL が続き、レジストリのステータスコードが 401 の代わりに表示されました。

headersHelper コマンドを記述

マーケットプレイスの url ソースまたはプラグインエントリで headersHelper を設定するかどうかに関わらず、コマンドがこれらの要件を満たすように記述します：

- **コマンドテキスト**: 最大 500 文字の印字可能 ASCII、4 文字以上の連続スペースなし。

- **出力**: ヘッダー名と文字列値の 1 つの JSON オブジェクトを stdout に出力し、10 秒以内に終了コード 0 で終了します。
- **シェルと作業ディレクトリ**: Claude Code はコマンドを `sh` または Windows では `cmd.exe` を通じて実行し、設定ディレクトリ（`~/.claude` または `CLAUDE_CONFIG_DIR`）から実行します。相対パスはそのディレクトリに対して解決されるため（ユーザーのプロジェクトではなく）、絶対パスまたは `PATH` 上のコマンドを指定してください。
- **Claude Code が削除する変数**: `marketplace.json` エントリまたはプロジェクトの `.claude/settings.json` または `.claude/settings.local.json` で設定されたコマンドの環境から、Claude Code は `TOKEN`、`SECRET`、`KEY`、`AUTH` などの単語を含む名前を持つすべての変数を削除します（`ANTHROPIC_API_KEY` を含む）。Claude Code はこの削除をユーザー設定、`--settings` ファイル、または管理設定で設定されたコマンドには適用しません。
- **Claude Code が設定する変数**: `url` ソースのコマンドの場合は `CLAUDE_CODE_MARKETPLACE_URL` と `CLAUDE_CODE_MARKETPLACE_NAME`、エントリのコマンドの場合は `CLAUDE_CODE_PLUGIN_NAME` と `CLAUDE_CODE_PLUGIN_ARCHIVE_URL`。`CLAUDE_CODE_MARKETPLACE_NAME` は、ユーザーが URL でマーケットプレイスを追加した後の最初のフェッチでは設定されません。そのフェッチが名前を提供するためです。

ベアラートークンを生成するコマンドは、次のようなオブジェクトを出力します：

```
{"Authorization": "Bearer eyJhbGciOiJSUzI1NiJ9"}
```

Claude Code が headersHelper コマンドをスキップするか、その出力をドロップする場合

Claude Code は `headersHelper` コマンドを実行しないか、これらの状況で `headers` または コマンドの出力から来たヘッダーをドロップします：

- **コマンド失敗**: コマンドが 0 以外で終了する、10 秒を超えて実行される、または JSON 文字列値のオブジェクト以外を出力する場合、Claude Code はそれが実行されたフェッチまたはダウンロードを実行しません。
- **マーケットプレイス URL が `https://` で始まらない**: Claude Code はその `url` ソースのコマンドを実行せず、`headers` フィールドにリストされたヘッダーのみを送信します。
- **リダイレクトがオリジンを離れる**: ダウンロードがアーカイブ URL のオリジンからリダイレクトされる場合、Claude Code はマーケットプレイス `url` ソースとプラグインエントリの両方の `headers` 値とコマンド出力をドロップします。
- **エントリがルーティングまたはアイデンティティヘッダーを設定**: Claude Code は `Host`、`Cookie`、`X-Forwarded-*` などのリクエストルーティングおよびクライアントアイデンティティ名をエントリの `headers` とコマンド出力からドロップし、`Authorization` などの認証名を保持します。Claude Code はすべての `marketplace.json` エントリをこの方法でフィルタリングし、インライン設定エントリはそれを宣言するファイルに応じて。
- **`--add-dir` ディレクトリの設定で設定されたコマンド**: Claude Code はそれを無視し、`url` ソースとインラインプラグインエントリの両方で、そのファイルの `headers` のみを送信します。
- **管理設定がコマンドをブロック**: `disableCommandPluginSources` を `true` に設定すると `headersHelper` コマンドがブロックされ、`allowManagedHooksOnly` も `disableCommandPluginSources` が

明示的に `false` でない限りそれらをブロックします。どちらのブロックでも、Claude Code は管理設定自体が宣言するマーケットプレイスのコマンドを実行します。

ユーザーが headersHelper コマンドを受け入れる方法

ユーザーはプラグインエントリのコマンドを、そのプラグインを単独でインストールまたは更新するたびに受け入れます。これは `/plugin` のプラグイン自体のビューから、または `claude plugin install` または `claude plugin update` で行われます。Claude Code はコマンドとアーカイブ URL を表示し、ユーザーが受け入れた後にのみコマンドを実行します。

非対話型シェルでは、*`--yes`* (ページ 611) を `claude plugin install` または `claude plugin update` に渡してコマンドを受け入れます。前の `--json` 実行が表示したコマンドのみを受け入れるには、*`--accept-command`* (ページ 611) に `sha256` を渡します。

Claude Code は表示したコマンドのみを実行し、表示したアーカイブ URL に対してのみ実行します。その間にエントリのコマンドまたはアーカイブ URL が変更された場合、Claude Code はインストールまたは更新を拒否します。クエリ文字列のみの変更はカウントされません。

コマンドを要求する代わりに拒否するインストールと更新

単一プラグインのインストールまたは更新以外の操作では、Claude Code はエントリのコマンドを実行せず、そのアーカイブをダウンロードしないため、プラグインはインストール済みバージョンのままか、インストールされていないままです。ユーザーが見るものは操作によって異なります：

- **複数のプラグインを一度にインストール、プラグイン提案からインストール、または別のプラグインの依存関係としてインストール**: Claude Code はコマンドを持つプラグインを拒否し、ユーザーをそのプラグインの `/plugin` 内の独自のビューに指します。一括インストール内の他のプラグインはまだインストールされます。拒否されたプラグインに依存するプラグインは、ユーザーが拒否されたプラグインを単独でインストールするまでインストールに失敗します。
- **バックグラウンド自動更新、またはアーカイブがダウンロードされたことのないプラグインのセッション開始**: Claude Code は `/plugin` エラータブにプラグインをリストして、ユーザーが手動でインストールまたは更新することを知らせます。インストール済みバージョンをまだ宣伝している自動更新は何もリストしません。

マーケットプレイス `url` ソースのコマンドが実行される時期

マーケットプレイス `url` ソースの `headersHelper` は、マーケットプレイスが公開するカタログではなく、`extraKnownMarketplaces` エントリなどの設定ファイルで宣言されるため、Claude Code は各インストールまたは更新でユーザーに受け入れを求めません。それを宣言する設定ファイルが Claude Code がいつそれを実行するかを決定します：

設定ファイル	Claude Code がコマンドを実行する時期
ユーザー設定、`--settings` ファイル、またはマシン上の管理設定ファイル	バックグラウンドマーケットプレイス更新を含め、要求なし

次のページに続く

表 15 – 前のページからの続き

設定ファイル	Claude Code がコマンドを実行する時期
プロジェクトの .claude/settings.json または .claude/settings.local.json	ユーザーがそのフォルダ自体の**ワークスペーストラストダイアログ** (ページ 1066) を受け入れた後のみ。-p または SDK セッションはそれとしてカウントされず、親フォルダに付与された信頼もカウントされません
サーバー管理設定	ユーザーがセキュリティ承認ダイアログで配信された設定を承認した後のみ

-p または SDK セッションでは、Claude Code はセキュリティ承認ダイアログを表示できません。他の配信された設定を適用しますが、マーケットプレイスフェッチと、コマンドが必要なアーカイブダウンロードは、ユーザーが対話型セッションで承認するまで失敗します。

これらのファイルのインラインプラグインエントリの場合、Claude Code はそのファイル内のマーケットプレイスレベルのコマンドと同じフォルダ信頼または設定承認を要求し、ユーザーは各インストールまたは更新でエントリのコマンドも受け入れます。

24.6.7 コマンドソース

ローカルにインストールされたツールがプラグインディレクトリを生成する場合 (現在選択されているツールチェーンのプラグインをレンダリングする IDE など) に command を使用します。Claude Code はユーザーがプラグインをインストールするときにコマンドを実行し、セッションごとに 1 回バックグラウンドで再実行するため、ユーザーは再インストールなしでツールの変更された出力を取得します。Claude Code v2.1.229 以降が必要です。v2.1.120 から v2.1.228 では、プラグインのインストールが `This plugin uses a source type your Claude Code version does not support. Update Claude Code and try again.` で失敗し、より古いバージョンではマーケットプレイス全体がロードに失敗します。

このエントリはツールが出力するディレクトリからプラグインをインストールします：

```
{
  "name": "my-plugin",
  "source": {
    "source": "command",
    "command": "my-tool claude-plugin-path"
  }
}
```

Claude Code はプラットフォームシェル（macOS と Linux では sh、Windows では cmd.exe）を通じてコマンドを実行し、ユーザーのホームディレクトリから実行します。コマンドは stdout に正確に 1 行を出力し、終了コード 0 で終了する必要があります。その行は、コマンドが終了するまでに完全なプラグインを含むディレクトリの絶対パスであり、パスは実行間で変更される可能性があります。

Claude Code は timeout 秒より長く実行されるコマンドを停止し、インストールまたは更新は失敗します。Claude Code はこれらの場合にも出力されたパスを拒否し、インストールまたは更新は同じ方法で失敗します：

- ディレクトリの最上部にプラグインコンテンツがない（`.claude-plugin/` ディレクトリ、または `skills/`、`commands/`、`agents/`、`hooks/` ディレクトリなど）
- ディレクトリは Claude Code が開始されたディレクトリ、またはその親の 1 つ
- Windows では、パスは UNC パス

コマンドソースはこれらのフィールドを受け入れます：

フィールド	タイプ	説明
`command`	string	必須。プラグインディレクトリの絶対パスを stdout の単一行として出力し、0 で終了するシェルコマンド。ユーザーが受け入れるよう求められるコマンド全体を確認できるように、印字可能 ASCII で最大 500 文字、4 文字以上の連続スペースなし
`timeout`	number	オプション。コマンドを待つ秒数（デフォルト：60、最大：600）
`mode`	string	オプション。`"copy"`（デフォルト）は出力されたディレクトリをプラグインキャッシュにコピーします。`"link"` は出力されたディレクトリをその場で使用します。**コピーモードとリンクモード** (ページ 648) を参照してください

コピーモードとリンクモード

デフォルトの `"mode": "copy"` では、Claude Code は出力されたディレクトリをバージョン管理されたプラグインキャッシュにコピーし、ディレクトリのコンテンツのハッシュから**プラグインバージョン** (ページ 624) を導出します。ツールはコマンドが終了した後にディレクトリを削除または上書きでき、同じコンテンツを生成する再実行は最新とカウントされます。Claude Code は 256 MiB より大きいディレクトリまたは 20,000 を超えるエントリを含むディレクトリのインストールを拒否します。

大規模なプラグインディレクトリ（レンダリングされた SDK エクスポートなど）をコピーしてはいけない場合は、`"mode": "link"` を設定します。Claude Code は出力されたディレクトリの各最上位エントリへのリンクでプラグインのキャッシュエントリを埋め、ファイルをその場で使用するため、何もコピーされず、ファイルコンテンツはハッシュされず、サイズ制限は適用されません。最上位エントリが出力されたディレクトリの外を指すシンボリックリンクの場合、インストールは失敗します。Claude Code はリンクモードプラグインの *Node.js* **パッケージ依存関係インストール** (ページ 605) もスキップするため、プラグインが必要とする `node_modules` を既に含むディレクトリを出力します。

プラグインがインストール状態を保つ限り、出力されたディレクトリをその場に保ちます。Claude Code はすべての起動でそれらのリンクを通じてプラグインをロードするためです。Claude Code は**プラグインバージョン** (ページ 624) を出力されたディレクトリの実パスとその最上位エントリから導出し、内部のファイルからではないため、新しいコンテンツを通知するために異なるパスを出力します。出力されたディレクトリまたはその下のどこかで開始されたセッションでは、Claude Code はプラグインをロードしません。

Claude Code は Windows でリンクモードをサポートしておらず、そこでリンクモードプラグインのインストールを拒否します。代わりに `"mode": "copy"` を宣言します。

ユーザーがコマンドを受け入れる方法

Claude Code はユーザーのマシンでコマンドを実行するため、すべての実行をユーザーの明示的な受け入れにバインドします：

- ユーザーが `/plugin` のプラグインの詳細画面からプラグインをインストールするか、対話型ターミナルで `claude plugin install` または `claude plugin update` でインストールまたは更新する場合、Claude Code は最初に正確なコマンド文字列を表示し、そのインストールの受け入れられたコマンドを記録します。同じコマンドの受け入れで進行できる `claude plugin update` は何も表示しません。
- 非対話型シェルでは、`claude plugin install` または `claude plugin update` に `--yes` を渡してコマンドを受け入れます。前の `--json` 実行が表示したコマンドのみを受け入れるには、*`--accept-command`* (ページ 611) に `sha256` を渡します。
- 他のすべてのパスはユーザーが既に受け入れたコマンドのみを実行します。これには `/plugin` から開始された更新と、**コマンドが再実行される場合** (ページ 649) のバックグラウンド実行が含まれます。何も受け入れられていない場合、Claude Code はコマンドの実行を拒否し、ユーザーにそれを確認する方法を指示します。Claude Code は別のプラグインの依存関係としてコマンドソースプラグインをインストールしないため、ユーザーは最初にそれを自分でインストールします。
- エントリの `command` を変更するか、その `mode` を切り替える場合、ユーザーは既に持っているバージョンを保持し、Claude Code はコマンドの再実行を停止します。対話型セッションでは、`/plugin` エラータブは新しいコマンドを表示し、ユーザーが `claude plugin update <plugin>@<marketplace>` を実行して確認して受け入れるまで表示されます。

管理者は管理設定 `disableCommandPluginSources` を使用して、組織全体でコマンドソースをブロックできます。組織が `allowManagedHooksOnly` を設定する場合、Claude Code はデフォルトでコマンドソースをブロックします。

Claude Code がコマンドを再実行する場合

出力されたディレクトリはコマンドが実行された時点でのツールの状態を反映するため、Claude Code はこれらの時間にコマンドを再実行します：

- ユーザーがプラグインをインストールまたは更新するたびに
- セッションごとに 1 回、有効な各コマンドソースプラグインに対して、セッション開始直後にバックグラウンドで。この実行はマーケットプレイス自動更新を通じて行われないため、マーケットプレイスの**自動更新設定** (ページ 690) に依存しません
- 起動時または `/reload-plugins` で、有効なプラグインのインストール済みバージョンがプラグインキャッシュから欠落している場合

ユーザーが `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` を設定する場合、Claude Code は 2 つのバックグラウンド実行をスキップします。明示的なインストールと更新は、その変数が設定されていてもコマンドを実行します。

コマンドのハッシュされた出力が変更された場合、Claude Code は結果を新しいバージョンとしてインストールし、実行中の対話型セッションでそれをリロードし、*/reload-plugins* **が切り替わるのと同じコンポーネント** (ページ 602) を切り替えます。ユーザーはプラグインがリロードされたという通知を見ます。その場でリロードするとセッションのプロンプトキャッシュが無効になる場合、Claude Code は代わりにユー

ザーに /reload-plugins を実行するよう促し、キャッシュコストについて警告し、--force で再実行すると適用されます。

24.6.8 高度なプラグインエントリ

この例は、コマンド、エージェント、フック、MCP サーバーのカスタムパスを含む、多くのオプションフィールドを使用するプラグインエントリを示しています：

```
{
  "name": "enterprise-tools",
  "source": {
    "source": "github",
    "repo": "company/enterprise-plugin"
  },
  "description": "Enterprise workflow automation tools",
  "version": "2.1.0",
  "author": {
    "name": "Enterprise Team",
    "email": "enterprise@example.com"
  },
  "homepage": "https://docs.example.com/plugins/enterprise-tools",
  "repository": "https://github.com/company/enterprise-plugin",
  "license": "MIT",
  "keywords": ["enterprise", "workflow", "automation"],
  "category": "productivity",
  "commands": [
    "./commands/core/",
    "./commands/enterprise/",
    "./commands/experimental/preview.md"
  ],
  "agents": ["./agents/security-reviewer.md", "./agents/compliance-checker.md"],
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write|Edit",
        "hooks": [
          {
            "type": "command",
            "command": "${CLAUDE_PLUGIN_ROOT}/scripts/validate.sh"
          }
        ]
      }
    ]
  },
  "mcpServers": {
    "enterprise-db": {
```

(次のページに続く)

(前のページからの続き)

```
      "command": "${CLAUDE_PLUGIN_ROOT}/servers/db-server",
      "args": ["--config", "${CLAUDE_PLUGIN_ROOT}/config.json"]
    }
  },
  "strict": false
}
```

注意すべき重要な点：

- commands と agents: 複数のディレクトリまたは個別のファイルを指定できます。パスはプラグインルートを基準にしており、その内部に留まる必要があります。
  - Claude Code は、./../shared.md のようにプラグインディレクトリの外に解決されるパスを path escapes plugin directory エラーで拒否し、そのコンポーネントなしでプラグインをロードします
- ${CLAUDE_PLUGIN_ROOT}: フックコマンドと MCP サーバー設定でこの変数を使用して、プラグインのインストールディレクトリ内のファイルを参照します。
  - サーバータイプごとにどの設定フィールドがそれを置換するかについては、**置換テーブル** (ページ 602) を参照してください
  - プラグイン更新を生き残るべき依存関係または状態の場合は、代わりに *${CLAUDE_PLUGIN_DATA}* (ページ 603) を使用します
- strict: false: これが false に設定されているため、プラグインは独自の plugin.json を必要としません。マーケットプレイスエントリがすべてを定義します。**厳密モード** (ページ 652) を参照してください。

デフォルトでは、プラグインのスキルはそのソースの下の skills/ ディレクトリからロードされます。skills フィールドにリストされたパスはそのスキャンに追加されます：

```
"skills": ["./skills/", "./extra-skills/"]
```

複数のプラグインエントリがマーケットプレイスルート（source: "./"）で 1 つの skills/ フォルダを共有する場合、各エントリが独自のスキルのみをロードするように特定のサブディレクトリをリストします：

```
"source": "./",
"skills": ["./skills/code-review", "./skills/docs"]
```

マーケットプレイスルート source では、リストされたパスはそのエントリの完全なセットであり、共有 skills/ フォルダ内の他のディレクトリはロードされません。./skills/ 自体またはプラグインルートをリストすると、完全なスキャンが保持されます。リストされたパスが存在しない場合、デフォルトスキャンが代わりに実行されます。

24.6.9 厳密モード

strict フィールドは、plugin.json がコンポーネント定義（スキル、エージェント、フック、MCP サーバー、出力スタイル）の権限であるかどうかを制御します。

値	動作
true（デフォルト）	plugin.json が権限です。マーケットプレイスエントリは追加のコンポーネントで補足でき、両方のソースがマージされます。
false	マーケットプレイスエントリが完全な定義です。プラグインにコンポーネントを宣言する plugin.json もある場合、それは競合であり、プラグインはロードに失敗します。

各モードを使用する場合：

- strict: true: プラグインは独自の plugin.json を持ち、独自のコンポーネントを管理します。マーケットプレイスエントリは上に追加のスキルまたはフックを追加できます。これはデフォルトであり、ほとんどのプラグインで機能します。
- strict: false: マーケットプレイスオペレーターが完全な制御を望みます。プラグインリポジトリは生ファイルを提供し、マーケットプレイスエントリはプラグイン作成者の意図と異なる方法でプラグインのコンポーネントを再構成またはキュレートする場合に便利です。

24.7 マーケットプレイスのホストと配布

ユーザーが git リポジトリでホストされているマーケットプレイスを追加したり、そのマーケットプレイスがリストしている git ベースのプラグインをインストールしたりすると、Claude Code はそのマーケットプレイスまたはプラグインリポジトリをユーザーのマシンにクローンします。クローンは Git LFS コンテンツをダウンロードしないため、LFS で追跡されているファイルはポインタファイルとして到着します。プラグインが必要とするファイルを LFS の外に保つようにしてください。

24.7.1 GitHub でホストする（推奨）

GitHub はマーケットプレイスをホストして配布するための推奨される方法です。

1. **リポジトリを作成する**：マーケットプレイス用の新しいリポジトリを設定します
2. **マーケットプレイスファイルを追加する**：プラグイン定義を含む .claude-plugin/marketplace.json を作成します
3. **チームと共有する**：ユーザーは /plugin marketplace add owner/repo でマーケットプレイスを追加します

メリット：組み込みのバージョン管理、issue トラッキング、チームコラボレーション機能があります。

24.7.2 他の git サービスでホストする

GitLab、Bitbucket、自社ホストサーバーなど、任意の git ホスティングサービスが機能します。ユーザーは完全なリポジトリ URL で追加します。

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

24.7.3 プライベートリポジトリ

Claude Code はプライベートリポジトリからプラグインをインストールすることをサポートしています。代わりに Organization settings > Plugins を通じてマーケットプレイスを配布する場合、git 認証情報は関係ありません。organization sync は、claude.ai 上の organization の GitHub または GitLab 接続を通じてマーケットプレイスリポジトリを読み取ります。プライベートにできるプラグインソースについては、*Distribute through organization settings* (ページ 654) を参照してください。

実行するコマンド

`/plugin marketplace add`、`/plugin install`、`/plugin update`、または `/plugin marketplace update` を実行すると、Claude Code は既存の git 認証情報ヘルパーを使用するため、`gh auth login`、macOS Keychain、または `git-credential-store` 経由の HTTPS アクセスはターミナルと同じように機能します。SSH アクセスは、ホストが既に `known_hosts` ファイルにあり、キーが `ssh-agent` に読み込まれている限り機能します。Claude Code はホストフィンガープリントとキーパスフレーズの対話的な SSH プロンプトを抑制するためです。GitHub の `owner/repo` 短縮形ソースはデフォルトで SSH 経由でクローンされます。代わりに HTTPS 経由でクローンするには、`CLAUDE_CODE_PLUGIN_PREFER_HTTPS=1` を設定してください。

バックグラウンド自動更新

バックグラウンド更新チェックは、実行するコマンドと同じ方法で、設定された git 認証情報ヘルパーを使用してマーケットプレイスのリモートで新しいコミットをチェックします。SSH リモートの場合、`ssh-agent` に読み込まれたキーがチェックを認証します。Claude Code はチェックを非対話的に実行します。git のターミナルプロンプトと askpass プログラムをオフにし、認証情報ヘルパーにプロンプトを表示しないよう指示します。チェックが HTTPS 経由でプライベートリポジトリに認証できるかどうかは、ヘルパーに依存します。

- プロンプトなしで保存された認証情報を提供できるヘルパーはチェックを認証します。Git Credential Manager、macOS Keychain ヘルパー、および `git-credential-store` は、ホストの認証情報を保持すると、このように機能します。
- プロンプトが必要なヘルパーはバックグラウンドで応答できません。更新は静かに失敗し、既存のチェックアウトはそのままです。プラグインは最後に同期された状態から機能し続けます。`/plugin marketplace update <name>` を実行して、認証情報でマーケットプレイスを更新します。

チェックがチェックアウトが最新であることを見つけた場合、Claude Code はそれをそのままにします。チェックが新しいコミットを見つけた場合、またはリモートに到達または認証できないために失敗した場合、Claude Code はマーケットプレイスを再度クローンして新しいクローンと交換します。そのクローンが失敗した場合、既存のチェックアウトはそのままです。再クローンは **大規模なリポジトリでタイムアウト** (ページ 675) する可能性があります。

2 つの設定により、プライベートマーケットプレイスは予測可能に動作します。

- `CLAUDE_CODE_PLUGIN_KEEP_MARKETPLACE_ON_FAILURE=1` を設定して、バックグラウンドチェックがリモートに到達または認証できない場合、再クローンを試みずに既存のチェックアウトを保持します。プラグインは最後に同期された状態から機能し続け、`/plugin marketplace update` による手動更新は引き続き認証情報で認証されます。
- git 認証情報ヘルパーを設定します。例えば GitHub の場合は `gh auth setup-git` を使用して、バックグラウンドチェックと再クローンがプロンプトなしで認証できるようにします。

環境に `GITHUB_TOKEN` などのプロバイダートークンを設定しても、それ自体ではバックグラウンド認証は有効になりません。トークンは設定された認証情報ヘルパー（例えば `gh` CLI のヘルパー）を通じてのみ有効になり、これは `GH_TOKEN` と `GITHUB_TOKEN` を読み取ります。

注釈:

CI/CD 環境では、プライベートリポジトリからプラグインをインストールする前に git 認証情報ヘルパーを設定してください。GitHub Actions では、マーケットプレイスリポジトリへの読み取りアクセス権を持つトークンを `GH_TOKEN` としてエクスポートし、`gh auth setup-git` を実行します。デフォルトワークフロートークンはワークフロー自身のリポジトリにのみアクセスできるため、別のリポジトリ内のプライベートマーケットプレイスには個人用アクセストークンまたはアプリトークンが必要です。

24.7.4 organization settings を通じて配布する

Team または Enterprise プランで Organization settings > Plugins を通じてプラグインを配布する場合、これらのソースルールが適用されます。

- github.com と gitlab.com では、マーケットプレイスリポジトリはプライベートまたは内部である必要があります。Organization sync はホストに一致する接続を通じてリポジトリを読み取ります。
  - **github.com**：Claude GitHub App
  - **Your GitHub Enterprise Server host**：organization の GitHub Enterprise App
  - **gitlab.com または自社管理の GitLab インスタンス**：organization の *GitLab configuration* (ページ 655) のそのホストのアクセストークン
- 各プラグインソースは `github`、`url`、または `git-subdir` タイプ、または `./` で始まる **相対パス** (ページ 638) である必要があります。`metadata.pluginRoot` の下で裸の名前でプラグインをリストする場合、organization sync はそれをサポートされていないソースとして拒否するため、`./plugins/deploy-tools` などのパスを書き出してください。
- プラグインソースは 3 つの場合にプライベートにできます。
  - マーケットプレイスリポジトリの所有者を共有する github.com ソース
  - GHE App がリポジトリにインストールされている organization の GitHub Enterprise ホスト上のソース
  - マーケットプレイスリポジトリと同じ GitLab ホスト上の `url` または `git-subdir` ソース。gitlab.com では、ソースはマーケットプレイスリポジトリと同じトップレベルグループまたはユーザー名前空間の下にある必要があります。

- その他のプラグインソースは、github.com、gitlab.com、または bitbucket.org 上のパブリックリポジトリである必要があり、organization sync は認証情報なしでフェッチします。Organization sync はこれらのルールがカバーしていないホスト上のプラグインソースを拒否します。

admin ワークフローについては、Manage plugins for your organization を参照してください。

プライベートプラグインを含めるには、プラグインフォルダをマーケットプレイスリポジトリ内に配置し、**相対パス** (ページ 638) で参照します。Organization sync は配布中に各プラグインをパッケージ化するため、ユーザーは別のソースリポジトリへのアクセスが必要ありません。

例えば、この `marketplace.json` プラグインエントリは、マーケットプレイスリポジトリの `plugins/deploy-tools` にコミットしたプラグインを参照します。

```
{
  "name": "deploy-tools",
  "source": "./plugins/deploy-tools"
}
```

GitLab でホストされているマーケットプレイスを同期する

gitlab.com または自社管理の GitLab インスタンスからマーケットプレイスを同期するには、Owner が最初に Organization settings > Claude Code でそのホストの GitLab configuration を追加します。GitLab configuration はパブリックベータ版であり、プラグインマーケットプレイス同期にのみ適用されます。1 つを追加しても、GitLab リポジトリが *cloud sessions* (ページ 236) で利用可能になることはありません。セットアップ手順については、Manage plugins for your organization を参照してください。

マーケットプレイスを追加するときは、`https://gitlab.example.com/platform/claude-plugins` などのプロジェクトの HTTPS URL を入力します。ネストされたサブグループ内のプロジェクトが機能します。Organization sync はプロジェクトのデフォルトブランチを読み取ります。**Sync automatically** をオンにすると、デフォルトブランチへのプッシュのみが同期を開始します。

トップレベルの bin ディレクトリから実行可能ファイルを除外する

organization settings を通じて配布するプラグインにトップレベルの `bin/` ディレクトリを含めないでください。マーケットプレイス同期または直接アップロードのいずれかでプラグインが到着するかどうかに関わらず、claude.ai はそのようなプラグインを拒否します。

- **マーケットプレイス同期**：organization sync はそのプラグインを拒否し、マーケットプレイスの残りを同期します。エラーメッセージは `Plugin contains a top-level bin/ directory` で始まります。
- **直接アップロード**：代わりに Organization settings > Plugins でプラグインをアップロードする場合、claude.ai は同じメッセージで拒否します。

実行可能ファイルを `scripts/` などの別のディレクトリに保持し、*skills*、*hooks*、**または** *MCP* **サーバー** *configs* (ページ 602) から `${CLAUDE_PLUGIN_ROOT}/scripts/<name>` として参照してください。

24.7.5 チームのマーケットプレイスを必須にする

リポジトリを設定して、Claude Code がチームメンバーが **プロジェクトフォルダを信頼** (ページ 1066) した後、マーケットプレイスを追加するようにできます。別のプロンプトはありません。マーケットプレイスを `.claude/settings.json` に追加します。

```
{
  "extraKnownMarketplaces": {
    "company-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

デフォルトで有効にするプラグインを指定することもできます。

```
{
  "enabledPlugins": {
    "code-formatter@company-tools": true,
    "deployment-tools@company-tools": true
  }
}
```

完全な設定オプションについては、Plugin settings を参照してください。

注釈:

ローカル `directory` または `file` ソースを相対パスで使用する場合、パスはリポジトリのメインチェックアウトに対して解決されます。git worktree から Claude Code を実行する場合、パスはメインチェックアウトを指し続けるため、すべての worktree は同じマーケットプレイスの場所を共有します。マーケットプレイスの状態は、プロジェクトごとではなく、ユーザーごとに 1 回 `~/.claude/plugins/known_marketplaces.json` に保存されます。

24.7.6 コンテナ用にプラグインを事前入力する

コンテナイメージと CI 環境の場合、ビルド時にプラグインディレクトリを事前入力して、Claude Code がマーケットプレイスとプラグインを既に利用可能な状態で開始し、実行時にクローンしないようにできます。`CLAUDE_CODE_PLUGIN_SEED_DIR` 環境変数をこのディレクトリを指すように設定します。

複数のシードディレクトリをレイヤーするには、Unix では `:` で、Windows では `;` でパスを区切ります。Claude Code は各ディレクトリを順番に検索し、特定のマーケットプレイスまたはプラグインキャッシュを含む最初のシードを使用します。

シードディレクトリは `~/.claude/plugins` の構造をミラーリングします。

```
$CLAUDE_CODE_PLUGIN_SEED_DIR/
  known_marketplaces.json
  marketplaces/<name>/...
  cache/<marketplace>/<plugin>/<version>/...
```

シードディレクトリを構築するには、イメージビルド中に Claude Code を 1 回実行し、必要なプラグインをインストールしてから、結果の `~/.claude/plugins` ディレクトリをイメージにコピーして、`CLAUDE_CODE_PLUGIN_SEED_DIR` をそれを指すように設定します。

コピーステップをスキップするには、ビルド中に `CLAUDE_CODE_PLUGIN_CACHE_DIR` をターゲットシードパスに設定して、プラグインがそこに直接インストールされるようにします。

```
CLAUDE_CODE_PLUGIN_CACHE_DIR=/opt/claude-seed claude plugin marketplace add your-org/
↪plugins
CLAUDE_CODE_PLUGIN_CACHE_DIR=/opt/claude-seed claude plugin install my-tool@your-
↪plugins
```

次に、コンテナのランタイム環境で `CLAUDE_CODE_PLUGIN_SEED_DIR=/opt/claude-seed` を設定して、Claude Code がスタートアップ時にシードから読み取るようにします。

スタートアップ時に、Claude Code はシードの `known_marketplaces.json` にあるマーケットプレイスをプライマリ設定に登録し、`cache/` の下にあるプラグインキャッシュを再クローンせずに使用します。これは対話モードと `-p` フラグを使用した非対話モードの両方で機能します。

動作の詳細。

- **読み取り専用**：Claude Code はシードディレクトリに書き込みません。
- **自動更新が無効**：シードマーケットプレイスは自動更新されません。
- **シードエントリが優先**：シードで宣言されたマーケットプレイスは、スタートアップのたびにユーザーの設定内の一致するエントリを上書きします。シードプラグインをオプトアウトするには、マーケットプレイスを削除する代わりに `/plugin disable` を使用します。
- **パス解決**：Claude Code はシードの JSON 内に保存されたパスを信頼するのではなく、実行時に `$CLAUDE_CODE_PLUGIN_SEED_DIR/marketplaces/<name>/` をプローブしてマーケットプレイスコンテンツを見つけます。これは、シードが構築された場所とは異なるパスにマウントされている場合でも、シードが正しく機能することを意味します。
- **ミューテーションがブロック**：シード管理マーケットプレイスに対して `/plugin marketplace remove` または `/plugin marketplace update` を実行すると、管理者にシードイメージを更新するよう求めるガイダンスで失敗します。
- **設定と構成**：`extraKnownMarketplaces` または `enabledPlugins` がシードに既に存在するマーケットプレイスを宣言する場合、Claude Code はクローンする代わりにシードコピーを使用します。

24.7.7 マネージドマーケットプレイスの制限

プラグインソースの厳密な制御を必要とする organization の場合、管理者はマネージド設定の `strictKnownMarketplaces` 設定を使用して、ユーザーが追加できるプラグインマーケットプレイスを制限できます。単一実行のためにプラグイン、エージェント、MCP サーバーをサイドロードする CLI フラグも拒否するには、`disableSideloadFlags` とペアにします。マーケットプレイスのプラグインがコンテキスト内インストール提案として表示されるかをホワイトリストするには、`pluginSuggestionMarketplaces` を設定します。

`strictKnownMarketplaces` はプラグインが来るマーケットプレイスと一致し、その中のエントリではないため、ユーザーは許可されたマーケットプレイスから *command source* (ページ 647) を持つプラグインをインストールできます。コマンドソースもブロックするには、`disableCommandPluginSources` を設定します。

`strictKnownMarketplaces` がマネージド設定で設定されている場合、制限動作は値に依存します。

値	動作
未定義（デフォルト）	制限なし。ユーザーは任意のマーケットプレイスを追加できます
空の配列 `[]`	完全なロックダウン。公式 Anthropic マーケットプレイスを含むすべてのマーケットプレイスソースをブロックします
ソースのリスト	ホワイトリスト強制。ユーザーはエントリと一致するマーケットプレイスのみを追加できます

一般的な設定

公式 Anthropic マーケットプレイスを含むすべてのマーケットプレイス追加を無効にします。

```
{
  "strictKnownMarketplaces": []
}
```

Claude Code は *claude.ai* **から同期されたプラグイン** (ページ 590) をマーケットプレイスではなくアカウントからダウンロードするため、このロックダウンはそれらをカバーしません。それらも停止するには、マネージド設定で `syncClaudeAiPlugins` を `false` に設定するか、claude.ai で organization の Skills をオフにします。

公式 Anthropic マーケットプレイスのみを許可します。単一リポジトリエントリのマッチングは正確であるため、このエントリは同じリポジトリの `ref` または `path` バリアントをカバーしません。

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "anthropics/claude-plugins-official"
    }
  ]
}
```

このエントリを使用すると、Claude Code は既に登録されている公式マーケットプレイスを利用可能に保

ち、新しいマシンでは、Claude Code を対話的に初めて開始するときにマーケットプレイスを自動的に登録します。

自動登録はすべてのマシンをカバーしていません。最も一般的に見落とされるのは。

- マシンの最初の対話的な起動の前に実行される非対話環境。
- Claude Code が既に公式マーケットプレイスをブロックするポリシーの下で対話的に実行されたマシン（空の配列ロックダウンなど）。Claude Code はブロックされた試みを記録し、ポリシーが変更された後は再試行しません。

これらのマシンでは、マーケットプレイスを同じ `managed-settings.json` の `extraKnownMarketplaces` に追加して Claude Code が自動的に登録するようにするか、`claude plugin marketplace add anthropics/claude-plugins-official` を実行します。

特定のマーケットプレイスのみを許可します。

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "acme-corp/approved-plugins"
    },
    {
      "source": "github",
      "repo": "acme-corp/security-tools",
      "ref": "v2.0"
    },
    {
      "source": "url",
      "url": "https://plugins.example.com/marketplace.json"
    }
  ]
}
```

owner-wildcard エントリを使用して GitHub organization の下のすべてのマーケットプレイスリポジトリを許可します。Owner wildcards には Claude Code v2.1.223 以降が必要です。

```
{
  "strictKnownMarketplaces": [
    {
      "source": "github",
      "repo": "acme-corp/*"
    }
  ]
}
```

ホストの正規表現パターンマッチングを使用して、内部 git サーバーからすべてのマーケットプレイスを許可します。これは GitHub Enterprise Server または自社ホスト GitLab インスタンスの推奨アプローチです。

```
{
  "strictKnownMarketplaces": [
    {
      "source": "hostPattern",
      "hostPattern": "^github\\.example\\.com$"
    }
  ]
}
```

パスの正規表現パターンマッチングを使用して、特定のディレクトリからファイルシステムベースのマーケットプレイスを許可します。

```
{
  "strictKnownMarketplaces": [
    {
      "source": "pathPattern",
      "pathPattern": "^/opt/approved/"
    }
  ]
}
```

pathPattern として ".*" を使用して、ネットワークソースを hostPattern で制御しながら、任意のファイルシステムパスを許可します。

注釈:

strictKnownMarketplaces はユーザーが追加できるものを制限しますが、マーケットプレイスを登録しません。許可されたマーケットプレイスをユーザーに自動的に登録するには、同じ managed-settings.json の extraKnownMarketplaces に追加します。
公式 Anthropic マーケットプレイスは、Claude Code が独自に登録する唯一のマーケットプレイスであり、ホワイトリストがそれを許可する場合のみです。自動登録は非対話環境やポリシーがそれをブロックした場合など、一部のマシンも見落とします。これらのマシンをカバーするには、公式マーケットプレイスを extraKnownMarketplaces にも追加します。2 つの設定を並べて見るには、strictKnownMarketplaces reference を参照してください。

制限がどのように機能するか

制限はネットワークまたはファイルシステム操作の前にチェックされます。チェックはマーケットプレイス追加時およびプラグインのインストール、更新、更新、自動更新時に実行されます。マーケットプレイスがポリシーが設定される前に追加され、そのソースがホワイトリストと一致しなくなった場合、Claude Code はそれからプラグインをインストールまたは更新することを拒否します。同じ強制が blockedMarketplaces に適用されます。

GitHub 所有者の下のすべてのマーケットプレイスリポジトリをブロックするには、blockedMarketplaces エントリで owner-wildcard 形式を使用します。{ "source": "github", "repo": "untrusted-org/

*" }。Claude Code v2.1.223 以降が必要です。マッチングルールについては、ブロックリストとホワイトリストの間で異なり、Owner wildcards を参照してください。

ユーザーが Claude Code が **フェッチするのではなくクローンする** (ページ 683) `https://` リポジトリ URL（裸の `github.com` または `gitlab.com` リポジトリ URL など）を追加する場合、Claude Code は `blockedMarketplaces` の `url` エントリに対してもチェックします。Claude Code はエントリが同じ URL を名前付けする場合、追加をブロックします。その比較では、Claude Code は `.git` サフィックスと、ユーザーが `#` の後に追加する任意の ref を無視します。Claude Code v2.1.232 以降が必要です。v2.1.232 より前では、Claude Code は `url` エントリをホストされた `marketplace.json` ファイルとしてフェッチした URL に対してのみマッチしました。

ホワイトリストは owner-wildcard `github` エントリを除き、ほとんどのソースタイプに対して正確なマッチングを使用します。マーケットプレイスが許可されるには、すべての指定されたフィールドが一致する必要があります。

- GitHub ソースの場合：`repo` は必須であり、1 つのリポジトリを名前付けするか、owner-wildcard 形式 `owner/*` を使用してそのオーナーの下のすべてのリポジトリをカバーします。ワイルドカードエントリがどのようにマッチするか（大文字小文字ルールを含む）については、Owner wildcards を参照してください。単一リポジトリエントリの場合、`ref` は正確に一致する必要があるか、マーケットプレイスソースとホワイトリストエントリの両方に存在しない必要があり、同じルールが `path` に適用されます。
- URL ソースの場合：完全な URL は正確に一致する必要があります。
- `hostPattern` ソースの場合：マーケットプレイスホストは正規表現パターンに対してマッチされます。
- `pathPattern` ソースの場合：マーケットプレイスのファイルシステムパスは正規表現パターンに対してマッチされます。

ホワイトリストの正確なマッチングは、末尾のスラッシュ、`.git` サフィックス、または `ssh://` と `https://` スキームのみが異なる URL を異なる値として扱います。organization のマーケットプレイスが複数の URL 形式でクローンできる場合、リテラル URL よりも `hostPattern` エントリを優先して、`https://`、`ssh://`、および `user@host:path` 形式がすべてマッチするようにします。

claude.ai **でホストされているマーケットプレイス** (ページ 684) はホストでマッチされます。`hostPattern` エントリが `claude.ai` にマッチする場合、`strictKnownMarketplaces` と `blockedMarketplaces` の両方でそれを管理します。ホワイトリストでは、そのようなエントリはメンバーの個人的な claude.ai アップロードを許可しません。Claude Code v2.1.273 以降が必要です。

`strictKnownMarketplaces` は マネージド設定 で設定されるため、個々のユーザーとプロジェクト設定はこれらの制限をオーバーライドできません。

サポートされているすべてのソースタイプと `extraKnownMarketplaces` との比較を含む完全な設定詳細については、strictKnownMarketplaces reference を参照してください。

24.7.8 バージョン解決とリリースチャネル

プラグインバージョンはキャッシュパスと更新検出を決定します。解決されたバージョンがユーザーが既に持っているものと一致する場合、`/plugin update` と自動更新はプラグインをスキップします。git ベースのソースの場合、`version` を省略すると、Claude Code はソースの解決されたコミット SHA を使用するため、ユーザーはそのコミットが変更されるたびに更新を取得します。これは内部または積極的に開発さ

れているプラグインの最も簡単なセットアップです。完全な解決順序（`archive` ソースを含む）については、*Version management* (ページ 624) を参照してください。

警告:

`version` を設定すると、*command* (ページ 647) を除くすべてのソースタイプのプラグインがピン留めされます。その version には常に、コマンドが生成したもののハッシュが含まれます。マーケットプレイスから追加されたローカルディレクトリから *in place* **で読み込まれた** (ページ 605) プラグインもピン留めされません。`plugin.json` で `"version": "1.0.0"` を宣言し、その文字列を変更せずに新しいコミットをプッシュする場合、これらのソースの既存ユーザーはキャッシュされたコピーを保持します。Claude Code は同じバージョンを見るためです。すべてのリリースでフィールドをバンプするか、解決されたバージョンにフォールバックするために省略します。

`plugin.json` とマーケットプレイスエントリの両方で `version` を設定することを避けてください。Claude Code は常に警告なしに `plugin.json` 値を使用するため、古いマニフェストバージョンは `marketplace.json` で設定したバージョンをマスクできます。

リリースチャネルを設定する

プラグインの「stable」と「latest」リリースチャネルをサポートするには、同じリポジトリの異なる ref または SHA を指す 2 つのマーケットプレイスを設定できます。その後、マネージド設定を通じて各ユーザーグループに独自のマーケットプレイスを提供できます。2 つの方法のいずれかで。

- 各グループのデバイスに個別の endpoint-managed settings（マネージド設定ファイルまたは MDM プロファイルなど）をデプロイします。Claude Code がマネージドソースを組み合わせる方法 は、organization 全体のソースも持つデバイスでグループごとのファイルまたはプロファイルが適用されるかどうかを示します。
- グループごとに 1 つの Claude apps gateway policy を定義します。ゲートウェイは一致ルールが適合する最初のポリシーを適用するため、各ユーザーがグループのポリシーに到達するようにポリシーを順序付けます。グループポリシーの `extraKnownMarketplaces` はキャッチオールポリシーのマップを置き換えるため、グループが必要とするすべてのマーケットプレイスをグループのポリシーにリストします。チャネルマーケットプレイスのみではなく。

admin コンソールからのサーバー管理設定は organization 内のすべてのユーザーに適用 されるため、グループごとの割り当てを実行できません。

警告:

各チャネルは異なるバージョンに解決される必要があります。明示的なバージョンを使用する場合、`plugin.json` は各ピン留めされた ref で異なる `version` を宣言する必要があります。`version` を省略する場合、異なるコミット SHA は既にチャネルを区別します。2 つの ref が同じバージョン文字列に解決される場合、Claude Code はそれらを同一として扱い、更新をスキップします。

例

```
{
  "name": "stable-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "stable"
      }
    }
  ]
}
```

```
{
  "name": "latest-tools",
  "plugins": [
    {
      "name": "code-formatter",
      "source": {
        "source": "github",
        "repo": "acme-corp/code-formatter",
        "ref": "latest"
      }
    }
  ]
}
```

チャネルをユーザーグループに割り当てる

リリースチャネルを設定する (ページ 662) の下で説明されているグループごとの endpoint-managed settings またはゲートウェイポリシーを通じて、各マーケットプレイスをそのユーザーグループに割り当てます。例えば、stable グループは以下を受け取ります。

```
{
  "extraKnownMarketplaces": {
    "stable-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/stable-tools"
      }
    }
  }
}
```

早期アクセスグループは代わりに latest-tools を受け取ります。

```
{
  "extraKnownMarketplaces": {
    "latest-tools": {
      "source": {
        "source": "github",
        "repo": "acme-corp/latest-tools"
      }
    }
  }
}
```

依存関係バージョンをピン留めする

プラグインは依存関係を semver 範囲に制限して、依存関係への更新が依存プラグインを破壊しないようにできます。{plugin-name}--v{version} git-tag 規約、範囲構文、および同じ依存関係に対する複数の制約がどのように組み合わされるかについては、Constrain plugin dependency versions を参照してください。

24.7.9 プラグインの名前を変更または削除する

プラグインの name はその安定識別子です。ユーザーは enabledPlugins、pluginConfigs、および /plugin install コマンドでそれを参照するため、それを変更するとすべての既存インストールが破壊されます。UI に表示されるラベルを既存インストールを破壊せずに変更するには、*displayName* (ページ 634) を設定して name を変更しないままにします。

プラグインの name を変更する必要がある場合、または plugins 配列からプラグインを削除する場合、既存ユーザーが plugin-not-found エラーを見る代わりに移行するように、トップレベルの renames エントリを追加します。自動移行には Claude Code v2.1.193 以降が必要です。各前の名前を現在の名前にマップするか、プラグインが存在しなくなった場合は null にマップします。次の例は formatter を code-formatter に名前変更し、legacy-linter が削除されたことを記録します。

```
{
  "name": "acme-tools",
  "owner": { "name": "Acme" },
  "plugins": [
    { "name": "code-formatter", "source": "./plugins/code-formatter" }
  ],
  "renames": {
    "formatter": "code-formatter",
    "legacy-linter": null
  }
}
```

ユーザーが古い名前がまだ設定に含まれた状態で Claude Code を開始すると、Claude Code は renames マップに従います。

- エントリが新しい名前を指す場合、Claude Code はプラグインを新しい名前の下で読み込み、`Renamed to "code-formatter" in the "acme-tools" marketplace` などの 1 行の通知を表示します。その後、ユーザー、プロジェクト、ローカル設定スコープで `enabledPlugins` と `pluginConfigs` の両方の古いキーを新しいキーに書き直すため、通知は 1 回表示されます。
- `null` エントリの場合、Claude Code は古いキーをドロップし、通知はプラグインがマーケットプレイスから削除されたことを報告します。
- 名前変更されたプラグインが `github` または `npm` などのリモートソースを使用する場合、Claude Code は名前変更後に `plugin-cache-miss` を報告し、ユーザーは新しい名前の下でそれをフェッチするために 1 回 `/plugin install` を実行する必要があります。

`renames` を追加のみの履歴として扱います。すべてのユーザーが移行したと予想した後でも、古いエントリを所定の位置に保持します。Claude Code はチェーンに従うため、後で `code-formatter` を `formatter-pro` に名前変更する場合、最初のエントリを編集する代わりに 2 番目のエントリを追加します。元の `formatter` がまだ有効になっているユーザーは、両方のエントリを通じて `formatter-pro` に解決されます。

マップを編集した後、`claude plugin validate .` を実行します。チェーンがサイクルを形成するか、`null` または `plugins` にリストされた名前で終了しないエントリを拒否します。

注釈:

マネージドおよびポリシー設定は Claude Code に対して読み取り専用であるため、そこで有効になっているプラグインは自動的に書き直すことができません。名前変更されたプラグインは各セッションで引き続き読み込まれますが、管理者がマネージド設定ファイルの `enabledPlugins` を新しい名前を使用するように更新するまで、名前変更通知は繰り返されます。同じことが `--add-dir` などの他の読み取り専用ソースを通じて有効になっているプラグインに適用されます。

Claude Code の以前のバージョンは `renames` フィールドを無視し、古い名前に対して `plugin-not-found` を報告します。

24.8 検証とテスト

マーケットプレイスを共有する前にテストしてください。検証はファイル構造をチェックします。プラグインが現実的なプロンプトで Claude の動作を変更するかどうかをテストするには、新しいバージョンを公開する前に `claude plugin eval` を使用してその eval スイートを実行してください。

マーケットプレイスディレクトリから JSON 構文を検証します：

```
claude plugin validate .
```

または Claude Code 内から：

```
/plugin validate .
```

テスト用にマーケットプレイスを追加します：

```
/plugin marketplace add ./path/to/marketplace
```

すべてが機能することを確認するためにテストプラグインをインストールします：

```
/plugin install test-plugin@marketplace-name
```

完全なプラグインテストワークフローについては、**プラグインをローカルでテスト** (ページ 572) を参照してください。技術的なトラブルシューティングについては、**プラグインリファレンス** (ページ 579) を参照してください。

24.9 CLI からマーケットプレイスを管理する

Claude Code は、スクリプトと自動化のための非対話的な `claude plugin marketplace` サブコマンドを提供します。これらは、対話的なセッション内で利用可能な `/plugin marketplace` コマンドと同等です。

24.9.1 プラグインマーケットプレイス追加

GitHub リポジトリ、Git URL、リモート URL、またはローカルパスからマーケットプレイスを追加します。

```
claude plugin marketplace add <source> [options]
```

引数：

- <source>：GitHub owner/repo ショートハンド、Git URL、marketplace.json ファイルへのリモート URL、またはローカルディレクトリパス。ブランチまたはタグに固定するには、GitHub ショートハンドに `@ref` を追加するか、Git URL に `#ref` を追加します

URL はスキームを含める必要があります。Claude Code v2.1.196 以降、`gitlab.example.com/team/plugins` のようにスキームなしで入力されたホストは、無効な `owner/repo` ショートハンドとして拒否され、エラーメッセージは `https://` を追加するか、ローカルパスに `./` を使用するよう指示します。以前のバージョンでは、これを GitHub リポジトリパスとして誤読し、GitHub の見つからないエラーでクローン時に失敗します。

オプション：

オプション	説明	デフォルト
`--scope <scope>`	マーケットプレイスを宣言する場所：`user`、`project`、または `local`。**プラグインインストールスコープ** (ページ 589) を参照してください	`user`
`--sparse <paths...>`	Git スパースチェックアウト経由で特定のディレクトリにチェックアウトを制限します。モノレポに便利です	
`--claudeai`	引数をソースではなく、*claude.ai* **でホストされているマーケットプレイス** (ページ 684) の名前として読み取ります。Claude Code v2.1.273 以降が必要です	

GitHub から `owner/repo` ショートハンドを使用してマーケットプレイスを追加します。

```
claude plugin marketplace add acme-corp/claude-plugins
```

`@ref` を使用して特定のブランチまたはタグに固定します。

```
claude plugin marketplace add acme-corp/claude-plugins@v2.0
```

非 GitHub ホスト上の Git URL から追加します。

```
claude plugin marketplace add https://gitlab.example.com/team/plugins.git
```

marketplace.json ファイルを直接提供するリモート URL から追加します。

```
claude plugin marketplace add https://example.com/marketplace.json
```

テスト用にローカルディレクトリから追加します。

```
claude plugin marketplace add ./my-marketplace
```

マーケットプレイスをプロジェクトスコープで宣言して、.claude/settings.json 経由でチームと共有します。

```
claude plugin marketplace add acme-corp/claude-plugins --scope project
```

モノレポの場合、プラグインコンテンツを含むディレクトリにチェックアウトを制限します。

```
claude plugin marketplace add acme-corp/monorepo --sparse .claude-plugin plugins
```

claude plugin marketplace list の From claude.ai: セクションに表示されている名前で、*claude.ai* **でホストされているマーケットプレイス** (ページ 684) を追加します。

```
claude plugin marketplace add --claudeai claudeai-organization-library
```

--claudeai を使用すると、コマンドは --scope と --sparse を拒否します。マーケットプレイスはアカウント用にホストされており、設定ファイルで宣言されていないため、プロジェクトの .claude/settings.json 経由で共有することはできません。

24.9.2 プラグインマーケットプレイスリスト

設定されたすべてのマーケットプレイスをリストします。

```
claude plugin marketplace list [options]
```

オプション：

オプション	説明
--json	JSON として出力

--json を使用すると、各エントリには name、source、マーケットプレイスが保存されているローカルキャッシュパスを含む installLocation フィールド、およびソース固有のフィールドが含まれます：GitHub ソースの場合は repo、Git および URL ソースの場合は url、ローカルソースの場合は path。GitHub およ

び Git ソースには、マーケットプレイスが固定されたブランチまたはタグで追加された場合、ref フィールドも含まれます。

追加された *claude.ai* **マーケットプレイス** (ページ 684) にはローカルクローンがないため、そのエントリは installLocation の代わりに claude.ai 識別子である marketplaceId と organizationUuid を含みます。

プラグインが *claude.ai* **アカウントから同期される** (ページ 590) ターミナルセッションでは、テキストリストの末尾に From claude.ai: セクションがあり、追加したマーケットプレイスを超えて claude.ai がアカウント用にリストしているものを名前で示します。それらの 1 つを追加するには、*claude.ai* **から追加** (ページ 684) を参照してください。--json 出力は設定されたマーケットプレイスのみをカバーし、そのセクションは除外されます。Claude Code v2.1.273 以降が必要です。

24.9.3 プラグインマーケットプレイス削除

設定されたマーケットプレイスを削除します。エイリアス rm も受け入れられます。

```
claude plugin marketplace remove <name> [options]
```

引数：

- <name>：削除するマーケットプレイス名。claude plugin marketplace list で表示されます。これは渡したソースではなく、marketplace.json の name です

オプション：

オプション	説明	デフォルト
--scope <scope>	削除を単一の設定スコープに制限します：user、project、または local。**プラグインインストールスコープ** (ページ 589) を参照してください。省略した場合、宣言はすべての編集可能なスコープから削除されます。指定した場合、そのスコープの宣言のみが削除されます。マーケットプレイスが別のスコープで引き続き宣言されている場合、共有状態、キャッシュ、およびインストール済みプラグインデータは保持されます	(すべてのスコープ)

警告:

マーケットプレイスを最後に残ったスコープから削除すると、そこからインストールしたプラグインもアンインストールされます。インストール済みプラグインを失わずにマーケットプレイスを更新するには、claude plugin marketplace update を使用してください。

24.9.4 プラグインマーケットプレイス更新

マーケットプレイスをソースから更新して、新しいプラグインとバージョン変更を取得します。ブランチまたはタグ ref で追加されたマーケットプレイスは、リポジトリのデフォルトブランチではなく、その ref の最新コミットに更新されます。

```
claude plugin marketplace update [name]
```

引数：

- [name]：更新するマーケットプレイス名。claude plugin marketplace list で表示されます。省略した場合はすべてのマーケットプレイスを更新します

remove と update の両方は、読み取り専用のシード管理マーケットプレイスに対して実行すると失敗します。すべてのマーケットプレイスを更新する場合、シード管理エントリはスキップされ、他のマーケットプレイスは引き続き更新されます。シード提供プラグインを変更するには、管理者にシードイメージを更新するよう依頼してください。**コンテナ用にプラグインを事前入力する** (ページ 656) を参照してください。

24.10 トラブルシューティング

24.10.1 マーケットプレイスが読み込まれない

症状: マーケットプレイスを追加できない、またはそこからプラグインが見えない

解決策:

- マーケットプレイス URL がアクセス可能であることを確認してください
- .claude-plugin/marketplace.json が指定されたパスに存在することを確認してください
- claude plugin validate . または /plugin validate . をマーケットプレイスディレクトリから実行して JSON 構文が有効であることを確認してください。スキル、エージェント、コマンドのフロントマターを確認するには、**マニフェストなしでプラグインまたはディレクトリを検証する** (ページ 671) を参照してください
- プライベートリポジトリの場合は、アクセス権限があることを確認してください

24.10.2 マーケットプレイス検証エラー

マーケットプレイスディレクトリから claude plugin validate . または /plugin validate . を実行して、問題がないか確認してください。マーケットプレイスディレクトリを指定すると、バリデーターは marketplace.json のスキーマエラー、重複するプラグイン名、ソースパストラバーサルをチェックします。source がローカルパスである各エントリについて、そのプラグイン自体の plugin.json も検証し、エントリの version が plugin.json のものと一致しない場合に警告します。プラグインの plugin.json で見つかった問題には、エントリインデックスが plugins[2] plugin.json → の形式で付与されます。

Claude Code v2.1.196 以降、エントリごとのパスは以下も実行します:

- source が . であるプラグインを含める
- marketplace.json が .claude-plugin ディレクトリの外にある場合に実行し、ソースをファイル自体のディレクトリに対して解決する
- ファイルの別の部分にスキーマエラーがある場合でも、各エントリの問題を報告する

以前のバージョンではマーケットプレイスルートのプラグインをスキップし、.claude-plugin/marketplace.json からのみ下降します。

マーケットプレイスディレクトリから、Claude Code はプラグインのスキル、エージェント、コマンド、またはフックファイルを開きません。これらのファイルのエラーを見つけるには、**マニフェストなしでプラグインまたはディレクトリを検証する** (ページ 671) を参照してください。以下の表は、マーケットプレイスディレクトリからの最も一般的なエラーと、それぞれの原因と修正方法を示しています:

エラー	原因	解決策
`No manifest found in directory. Expected .claude-plugin/marketplace.json or .claude-plugin/plugin.json`	指定したディレクトリに `.claude-plugin/marketplace.json` または `plugin.json` がなく、チェックするスキル、エージェント、またはコマンドファイルもない	マーケットプレイスルートから実行するか、必須フィールドを含む `.claude-plugin/marketplace.json` を作成してください
`Invalid JSON syntax: Unexpected token...`	marketplace.json の JSON 構文エラー	不足しているコンマ、余分なコンマ、またはクォートされていない文字列がないか確認してください
`Duplicate plugin name "x" found in marketplace`	2 つのプラグインが同じ名前を共有している	各プラグインに一意の `name` 値を付与してください
`plugins[0].source: Path contains ".."`	ソースパスに `..` が含まれている	マーケットプレイスルートに対する相対パスを使用し、`..` を含めないでください。**相対パス** (ページ 638) を参照してください
`Marketplace name cannot contain control or bidirectional-formatting characters`	マーケットプレイス `name` に Unicode 双方向フォーマット文字またはエスケープや改行などの制御文字が含まれている	名前から文字を削除してください。v2.1.247 より前では、これらの文字は `Marketplace name impersonates an official Anthropic/Claude marketplace` エラーを生成していました
`Plugin name cannot contain control or bidirectional-formatting characters`	プラグイン `name` に Unicode 双方向フォーマット文字またはエスケープや改行などの制御文字が含まれている	名前から文字を削除してください。v2.1.247 より前では、Claude Code はこのチェックを実行していませんでした

警告（ブロッキングなし）:

- `Marketplace has no plugins defined`: `plugins` 配列に少なくとも 1 つのプラグインを追加してください
- `No marketplace description provided`: ユーザーがマーケットプレイスを理解するのに役立つよう、トップレベルの `description` を追加してください
- `Plugin name "x" is not kebab-case`: 小文字、数字、ハイフンのみを使用して名前を変更してください（例: `my-plugin`）。Claude Code は他の形式を受け入れますが、claude.ai マーケットプレイス同期はそれらを拒否します。
- `Marketplace name "x" is reserved in Claude Desktop`: マーケットプレイスが `org`、`org-provisioned`、または `unknown` という名前である（大文字小文字は問わない）。Claude Code はこ

れらの名前を受け入れますが、Claude Desktop の管理マーケットプレイス同期はマーケットプレイス全体を拒否します。マーケットプレイスの名前を変更してください。v2.1.221 より前では、`claude plugin validate` はこのチェックを実行していませんでした。

- `Marketplace name "x" is not accepted by Claude Desktop` または `Plugin name "x" is not accepted by Claude Desktop`: Claude Desktop は、文字、数字、.、_、- で構成され、文字または数字で始まる最大 128 文字の名前を受け入れます。Claude Code は他の形式を受け入れますが、Claude Desktop の管理マーケットプレイス同期は名前チェックに失敗したマーケットプレイスを拒否し、名前チェックに失敗したプラグインエントリを静かにドロップします。マーケットプレイスまたはプラグインの名前を変更してください。v2.1.221 より前では、`claude plugin validate` はこれらのチェックを実行していませんでした。

マニフェストなしでプラグインまたはディレクトリを検証する

フロントマターが解析されないスキル、エージェント、コマンドファイルを見つけるには、`claude plugin validate` を実行し、それらを保持するディレクトリを指定してください。Claude Code は指定したディレクトリの外を見ません。`plugin.json` を持つプラグインに対する 1 つを除くすべての実行には、Claude Code v2.1.233 以降が必要です。

指定するディレクトリを選択する

Claude Code は、指定したディレクトリに応じて異なるファイルをチェックします。最初の列で確認したいものを見つけ、その行のコマンドを実行してください:

確認対象	実行	Claude Code がチェックする内容
`plugin.json` を持つプラグイン	`claude plugin validate ./plugins/my-plugin`	`plugin.json`、`hooks/hooks.json`、およびプラグインルートの `skills`、`agents`、`commands` ディレクトリ
スキル、エージェント、またはコマンドの 1 つのディレクトリ（`plugin.json` がまだないプラグインなど）	`claude plugin validate .claude/skills`、`~/.claude/agents`、または `./my-plugin/agents`	そのディレクトリ内のすべてのスキル、エージェント、またはコマンドファイル
スキルがルート `SKILL.md` であるフォルダ	`claude plugin validate ./skills`（フォルダを保持する `skills` ディレクトリを指定）	各フォルダのルート `SKILL.md`。保持するディレクトリは `skills` という名前である必要があります。`plugins/` などの別の名前の下のフォルダには、ルート `SKILL.md` をチェックする実行がありません
プロジェクトの 3 つのディレクトリを一度に	`claude plugin validate .claude`、またはマニフェスト `.claude-plugin/` がないプロジェクトルート	`.claude/skills`、`.claude/agents`、`.claude/commands`

次のページに続く

表 23 – 前のページからの続き

確認対象	実行	Claude Code がチェックする内容
ユーザーレベルディレクトリ	`claude plugin validate ~/.claude`	`~/.claude/skills`、`~/.claude/agents`、`~/.claude/commands`

スキルがルート SKILL.md であるプラグインをチェックする

プラグインディレクトリに対して `claude plugin validate` を実行すると、Claude Code はプラグインルートの `SKILL.md` をチェックしません。プラグインが `skills` という名前のディレクトリにある場合は、コマンドを 2 回実行してください:

- その `skills` ディレクトリを指定して、プラグインのルート `SKILL.md` をチェックしてください。
- プラグインディレクトリを指定して、残りをチェックしてください。

プラグインが `plugins/` などの別の名前の下にある場合、`skills` ディレクトリの実行は利用できず、ルート `SKILL.md` をチェックする実行がありません。

シンボリックリンクの背後にあるファイルをチェックする

`claude plugin validate` を実行すると、Claude Code は指定したディレクトリ内のシンボリックリンクをフォローしません。リンクがどこにあるかによって、実行内容が異なります:

- **プラグインまたは `.claude` ルートの下にリンクされた `skills`、`agents`、または `commands` ディレクトリ**: Claude Code は、その中のものが何も読まれなかったことを警告します。
- **`skills`、`agents`、または `commands` ディレクトリ内のリンクされたエントリ**: Claude Code はそれをスキップし、ディレクトリごとにスキップしたエントリの数をセッションが読み込むことを警告します。
- **指定した `skills`、`agents`、または `commands` ディレクトリ自体がシンボリックリンク、またはその親 `.claude` ディレクトリがシンボリックリンク**: Claude Code はエラーを報告し、その中のものをチェックしません。代わりに実際のディレクトリを指定してください。

2 つのスキルケースでは、実行は警告付きで成功します。リンクされたファイルをチェックするには、再度実行し、それらを直接保持するディレクトリを指定してください:

- **`skills` ディレクトリが兄弟プラグインのスキルにリンク** (ページ 607) **しているプラグイン**: 兄弟プラグインのディレクトリを指定してください。
- **`~/.claude/skills` または `.claude/skills` のシンボリックリンクされたスキルエントリ** (ページ 279): Claude Code はセッションでエントリをフォローします。チェックするには、実際のフォルダを保持する `skills` という名前のディレクトリを指定してください。

検証結果を読む

クリーンな実行は `Validation passed` で終了します。

`No manifest found in directory` は、Claude Code がそこに `plugin.json` または `marketplace.json` を見つけず、その下で調査するディレクトリにスキル、エージェント、またはコマンドファイルがないこ

とを意味します。代わりに、ファイルを保持する `skills`、`agents`、または `commands` ディレクトリを指定してください。

Claude Code がこれらの実行から報告する 2 つのエラーと、それぞれの修正方法:

- `YAML frontmatter failed to parse: ...`: スキル、エージェント、またはコマンドファイルのフロントマターブロック内の YAML を修正してください。修正するまで、セッションはそのファイルからフロントマターフィールドを読み込みません
- `Invalid JSON syntax: ...` on `hooks/hooks.json`: JSON 構文を修正してください。修正するまで、セッションはそのファイルのフックなしでプラグインを読み込みます。Claude Code はこのエラーをプラグイン実行でのみ報告します

プラグイン実行では、Claude Code はプラグインルートの `CLAUDE.md` についても警告します。`plugin.json` の**コンポーネントパスフィールド** (ページ 596) を通じて設定したパスについては、Claude Code は各パスが存在することをチェックしますが、そこのファイルは読み込みません。

24.10.3 プラグインインストール失敗

症状: マーケットプレイスは表示されるがプラグインのインストールが失敗する

解決策:

- プラグインソース URL がアクセス可能であることを確認してください
- プラグインディレクトリに必須ファイルが含まれていることを確認してください
- GitHub ソースの場合は、リポジトリがパブリックであるか、アクセス権限があることを確認してください
- プラグインソースを手動でテストしてクローン/ダウンロードしてください
- ソースが `ref` と `sha` の両方をピンしている場合、削除されたアップストリームブランチまたはタグは、GitHub、GitLab、Bitbucket を含むほとんどの git ホストでのインストールをブロックしません。AWS CodeCommit などの SHA でのコミット取得をサポートしないサーバーでは、`ref` は依然として存在する必要があり、ピンされたコミットはそこから到達可能である必要があります。インストールが依然として失敗する場合は、ピンされたコミットがリポジトリに依然として存在することを確認してください

24.10.4 プライベートリポジトリ認証失敗

症状: プライベートリポジトリからプラグインをインストールするときに認証エラーが発生する

解決策:

手動インストールと更新の場合:

- git プロバイダーで認証されていることを確認してください（例: GitHub の場合は `gh auth status` を実行）
- 認証情報ヘルパーが設定されていることを確認してください: `git config --global credential.helper`

- `git ls-remote <marketplace-url>` を実行して、git が単独で認証できるかテストしてください。git がユーザー名またはパスワードを要求する場合は、最初に認証情報を保存してください: GitHub over HTTPS の場合は `gh auth setup-git` を実行し、SSH リモートの場合はキーを `ssh-agent` に読み込んでください

バックグラウンド自動更新の場合:

- バックグラウンドチェックは設定された git 認証情報ヘルパーを使用しますが、プロンプトを表示しません。そのため、ヘルパーは保存された認証情報で応答できる必要があります。`ssh-agent` に読み込まれたキーを持つ SSH リモートも認証します
- ヘルパーがプロンプトを表示する必要がある場合、バックグラウンド更新は静かに失敗し、既存のチェックアウトが所定の位置に留まります。ヘルパーに最初にサインインして、ホストの認証情報を保持するようにしてください。GitHub の場合は、`gh auth login` を実行してから `gh auth setup-git` を実行してください
- チェックが新しいコミットを見つけた場合、またはリモートに到達または認証できない場合、Claude Code は同じ認証情報でマーケットプレイスを再クローンします。再クローンは大規模なリポジトリでタイムアウトする可能性があります
- `CLAUDE_CODE_PLUGIN_KEEP_MARKETPLACE_ON_FAILURE=1` を設定して、バックグラウンドチェックがリモートに到達または認証できない場合に既存のチェックアウトを保持してください
- 大規模なリポジトリで再クローンがタイムアウトする場合は、*CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS* (ページ 675) で制限を増やしてください
- または、認証情報を使用する `/plugin marketplace update <name>` でプライベートマーケットプレイスを手動で更新してください

v2.1.280 より前では、バックグラウンドチェックは認証情報ヘルパーなしで実行され、HTTPS 経由でプライベートリポジトリに認証できませんでした。

24.10.5 オフライン環境でマーケットプレイス更新が失敗する

症状: オフラインまたはエアギャップ環境では、バックグラウンドマーケットプレイス更新がリモートに到達できず、Claude Code が成功できない再クローンを繰り返し試みます。

原因: バックグラウンド更新はマーケットプレイスのリモートで新しいコミットをチェックし、チェックがリモートに到達できない場合、Claude Code はマーケットプレイスを再度クローンしようとします。オフラインでは、クローンは同じ方法で失敗し、既存のチェックアウトが所定の位置に留まります。v2.1.274 より前では、更新は既存のチェックアウトで `git pull` を実行し、プルが失敗したときにチェックアウトを脇に移動して再クローンし、その後ベストエフォートベースで復元していました。

更新はスタートアップ後にバックグラウンドで実行されるため、スタートアップは遅延しません。各セッションは依然として失敗した試みを繰り返し、各 git 操作は *120* **秒のタイムアウト** (ページ 675) を待つことができます。

解決策: `CLAUDE_CODE_PLUGIN_KEEP_MARKETPLACE_ON_FAILURE=1` を設定して、チェックがリモートに到達できない場合に再クローン試行をスキップし、既存のチェックアウトを使用し続けてください:

```
export CLAUDE_CODE_PLUGIN_KEEP_MARKETPLACE_ON_FAILURE=1
```

リポジトリが到達不可能になる完全オフラインデプロイメントの場合は、代わりに *CLAUDE_CODE_PLUGIN_SEED_DIR* (ページ 656) を使用してビルド時にプラグインディレクトリを事前入力してください。

24.10.6 Git 操作がタイムアウトする

症状: プラグインのインストールまたはマーケットプレイスの更新が「Git clone timed out after 120s」などのタイムアウトエラーで失敗します。

原因: Claude Code は、プラグインリポジトリのクローンやマーケットプレイスの更新を含むすべての git 操作に 120 秒のタイムアウトを使用します。大規模なリポジトリまたは遅いネットワーク接続はこの制限を超える可能性があります。

解決策: `CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS` 環境変数を使用してタイムアウトを増やしてください。値はミリ秒単位です:

```
export CLAUDE_CODE_PLUGIN_GIT_TIMEOUT_MS=300000  # 5 minutes
```

24.10.7 URL ベースのマーケットプレイスで相対パスを持つプラグインが失敗する

症状: `https://example.com/marketplace.json` などの URL を通じてマーケットプレイスを追加しましたが、`"./plugins/my-plugin"` などの相対パスソースを持つプラグインが `its marketplace entry path does not stay inside the marketplace directory` でインストールに失敗します。既にインストールされているプラグインは `Plugin source path refused` で読み込みに失敗します。両方のメッセージにエラーリファレンスエントリがあります。

原因: URL ベースのマーケットプレイスを追加すると、`marketplace.json` ファイル自体のみがダウンロードされ、Claude Code はそのサーバーから相対パスでプラグインファイルをフェッチしません。マーケットプレイスエントリの相対パスは、ダウンロードされなかったリモートサーバー上のファイルを参照します。

解決策:

- **外部ソースを使用**: プラグインエントリを相対パス以外の任意の**プラグインソース** (ページ 636) に変更してください:

```
{ "name": "my-plugin", "source": { "source": "github", "repo": "owner/repo" } }
```

- **Git ベースのマーケットプレイスを使用**: マーケットプレイスを Git リポジトリでホストし、git URL で追加してください。Git ベースのマーケットプレイスはリポジトリ全体をクローンするため、相対パスが正しく機能します。

24.10.8 インストール後にファイルが見つからない

症状: プラグインはインストールされますが、ファイルへの参照が失敗します。特にプラグインディレクトリの外のファイル

原因: プラグインは、**リンクモードの *command* ソース** (ページ 648) を除き、その場で使用されるのではなく、キャッシュディレクトリにコピーされます。コピーされたプラグインのディレクトリの外のファイルを参照するパス（`../shared-utils` など）は、それらのファイルがコピーされないため機能しません。

解決策: **プラグインキャッシングとファイル解決** (ページ 605) を参照して、シンボリックリンクとディレクトリ再構成を含む回避策を確認してください。

追加のデバッグツールと一般的な問題については、**デバッグと開発ツール** (ページ 622) を参照してください。

24.11 関連項目

- **既成プラグインの検出とインストール** (ページ 677) - 既存のマーケットプレイスからプラグインをインストール
- **プラグイン** (ページ 565) - 独自のプラグインの作成
- **プラグインリファレンス** (ページ 579) - 完全な技術仕様とスキーマ
- プラグイン設定 - プラグイン設定オプション
- strictKnownMarketplaces リファレンス - 管理マーケットプレイス制限

第25章　マーケットプレイスから事前構築されたプラグインを発見してインストールする

> マーケットプレイスからプラグインを検索してインストールし、Claude Code を新しいスキル、エージェント、機能で拡張します。

プラグインは Claude Code をスキル、エージェント、フック、MCP サーバーで拡張します。プラグインマーケットプレイスは、これらの拡張機能を自分で構築することなく発見してインストールするのに役立つカタログです。

claude.ai でプラグインを有効にすることもできます。自分自身のため、または組織を通じて有効にできます。Claude Code はそれらをマーケットプレイスのインストールなしにセッションに同期します。*claude.ai から同期されたプラグイン* (ページ 590) で説明されているとおりです。

独自のマーケットプレイスを作成して配布したいですか？**プラグインマーケットプレイスを作成して配布する** (ページ 627) を参照してください。

25.1 マーケットプレイスの仕組み

マーケットプレイスは、他の誰かが作成して共有したプラグインのカタログです。マーケットプレイスを使用するのは 2 段階のプロセスです：

1. **マーケットプレイスを追加する**

   これにより、カタログが Claude Code に登録され、利用可能なものを参照できるようになります。プラグインはまだインストールされていません。

2. **個別のプラグインをインストールする**

   カタログを参照して、必要なプラグインをインストールします。

25.2 公式 Anthropic マーケットプレイス

Claude Code は初めて対話的に起動すると、公式 Anthropic マーケットプレイス（`claude-plugins-official`）を自動的に追加します。ネットワークがダウンロードをブロックしている場合や、**マーケットプレイスポリシー** (ページ 658) が以前の試行をブロックした場合など、Claude Code がそれを追加できない場合は、`/plugin marketplace add anthropics/claude-plugins-official` を使用して自分で追加してください。

利用可能なものを参照するには、`/plugin` を実行して **Discover** タブに移動するか、claude.com/plugins でカタログを表示してください。

公式マーケットプレイスからプラグインをインストールするには、`/plugin install <name>@claude-plugins-official` を使用します。たとえば、GitHub 統合をインストールするには：

```
/plugin install github@claude-plugins-official
```

`/plugin` はターミナル CLI で対話的なパネルを開きます。Claude が `/plugin` がこの環境では利用できないと返答する場合は、別の方法でプラグインをインストールしてください：

- **Claude デスクトップアプリ**：**プラグインブラウザー** (ページ 152) を使用してください。
- **VS Code 拡張機能**：*Manage plugins* **ダイアログ** (ページ 186) からインストールしてください。
- **クラウドセッション**：claude.ai アカウントのプラグインを有効にして、Claude Code が**同期されたプラグイン** (ページ 590) として読み込むようにしてください。

インストールが失敗した場合は、Claude Code が報告するメッセージと照合してください：

- `Marketplace "claude-plugins-official" not found`：`/plugin marketplace add anthropics/claude-plugins-official` でマーケットプレイスを追加してから、インストールを再試行してください。
- プラグインが**マーケットプレイスで見つかりません** (ページ 685)：プラグイン名を確認してください。

注釈:

公式マーケットプレイスは Anthropic によって管理されており、掲載は Anthropic の裁量です。アプリ内送信フォームはプラグインを**コミュニティマーケットプレイス** (ページ 680) に追加します。公式マーケットプレイスには追加されません。プラグインを独立して配布するには、**独自のマーケットプレイスを作成** (ページ 627) してユーザーと共有してください。

公式マーケットプレイスには、プラグインのいくつかのカテゴリが含まれています：

25.2.1 コード インテリジェンス

コード インテリジェンス (ページ 678) プラグインは Claude Code の組み込み LSP ツールを有効にし、Claude が定義にジャンプしたり、参照を見つけたり、編集直後に型エラーを確認したりできるようにします。これらのプラグインは Language Server Protocol 接続を構成します。これは VS Code のコード インテリジェンスを強化する同じテクノロジーです。**クラウドセッション** (ページ 223) では、Claude Code はプラグイン言語サーバーを起動しないため、Claude はそこで LSP ツールを取得しません。

これらのプラグインを使用する前に、以下の表から言語サーバーバイナリをインストールしてください。プラグインはそれをインストールしません。言語サーバーが既にインストールされている場合、プロジェクトを開くと Claude は対応するプラグインをインストールするよう促す場合があります。

言語	プラグイン	必要なバイナリ
C/C++	`clangd-lsp`	`clangd`
C#	`csharp-lsp`	`csharp-ls`
Go	`gopls-lsp`	`gopls`
Java	`jdtls-lsp`	`jdtls`
Kotlin	`kotlin-lsp`	`kotlin-language-server`
Lua	`lua-lsp`	`lua-language-server`
PHP	`php-lsp`	`intelephense`

次のページに続く

表 1 – 前のページからの続き

言語	プラグイン	必要なバイナリ
Python	`pyright-lsp`	`pyright-langserver`
Rust	`rust-analyzer-lsp`	`rust-analyzer`
Swift	`swift-lsp`	`sourcekit-lsp`
TypeScript	`typescript-lsp`	`typescript-language-server`

他の言語用に独自の *LSP* プラグインを作成 (ページ 584) することもできます。

注釈:

プラグインをインストール後に `/plugin` Errors タブに `Executable not found in $PATH` が表示される場合は、上記の表から必要なバイナリをインストールしてください。

コード インテリジェンス プラグインから Claude が得られるもの

コード インテリジェンス プラグインがインストールされ、その言語サーバーバイナリが利用可能になると、Claude は 2 つの機能を得られます：

- **自動診断**：Claude が行うすべてのファイル編集後、言語サーバーはエラーと警告を報告し、Claude はコンパイラやリンターを実行することなく、型エラー、不足しているインポート、構文の問題を確認します。Claude がエラーを導入した場合、それに気付いて同じターンで修正します。
- **コード ナビゲーション**: Claude は言語サーバーを使用して定義にジャンプしたり、参照を見つけたり、ホバーで型情報を取得したり、シンボルをリストしたり、実装を見つけたり、呼び出し階層をトレースしたりできます。これらの操作により、Claude は grep ベースの検索よりも正確なナビゲーションが可能になりますが、言語と環境によって可用性が異なる場合があります。

診断を自分で読むには、Claude Code が **Found 3 new diagnostic issues in 2 files** などのインジケーターを表示するときに **Ctrl+O** を押してください。

問題が発生した場合は、**コード インテリジェンスのトラブルシューティング** (ページ 693) を参照してください。

25.2.2 外部統合

これらのプラグインは事前構成された *MCP* **サーバー** (ページ 363) をバンドルしているため、手動セットアップなしで Claude を外部サービスに接続できます：

- **ソース管理**: `github`、`gitlab`
- **プロジェクト管理**: `atlassian`（Jira/Confluence）、`asana`、`linear`、`notion`
- **デザイン**: `figma`
- **インフラストラクチャ**: `vercel`、`firebase`、`supabase`
- **コミュニケーション**: `slack`
- **監視**: `sentry`

25.2.3 自動セキュリティレビュー

`security-guidance` プラグインは Claude が行う各変更を一般的な脆弱性についてレビューし、Claude に見つかったものを修正するよう指示します。Claude がコードを書く際にセキュリティの問題をキャッチするを参照して、何をチェックするか、プロジェクト固有のルールを追加する方法を確認してください。

25.2.4 開発ワークフロー

一般的な開発タスク用のスキルとエージェントを追加するプラグイン：

- **commit-commands**: コミット、プッシュ、PR 作成を含む Git コミット ワークフロー
- **pr-review-toolkit**: プルリクエストをレビューするための特化したエージェント
- **agent-sdk-dev**: Claude Agent SDK で構築するためのツール
- **plugin-dev**: 独自のプラグインを作成するためのツールキット

25.2.5 出力スタイル

Claude の応答方法をカスタマイズします：

- **explanatory-output-style**: 実装の選択に関する教育的な洞察
- **learning-output-style**: スキル構築のためのインタラクティブな学習モード

25.3 コミュニティ マーケットプレイス

`anthropics/claude-plugins-community` のコミュニティ マーケットプレイスは、Anthropic の自動検証とセキュリティ スクリーニングに合格したサードパーティ プラグインをホストしています。各プラグインはカタログ内の特定のコミット SHA に固定されています。公式マーケットプレイスとは異なり、手動で追加します：

```
/plugin marketplace add anthropics/claude-plugins-community
```

その後、`claude-community` マーケットプレイス名を使用してプラグインをインストールします：

```
/plugin install <plugin-name>@claude-community
```

独自のプラグインをコミュニティ マーケットプレイスに送信するには、プラグイン作成ガイドの**プラグインをコミュニティ マーケットプレイスに送信する** (ページ 574) を参照してください。

25.4 試してみる: デモマーケットプレイスを追加する

Anthropic は、プラグインシステムで何が可能かを示す例プラグインを含む デモプラグインマーケットプレイス（`claude-code-plugins`）も管理しています。公式マーケットプレイスとは異なり、このマーケットプレイスは手動で追加する必要があります。

1. **マーケットプレイスを追加する**

   Claude Code 内から、`anthropics/claude-code` マーケットプレイスの `plugin marketplace add` コマンドを実行します：

   ```
   /plugin marketplace add anthropics/claude-code
   ```

   これにより、マーケットプレイスカタログがダウンロードされ、そのプラグインが利用可能になります。

2. **利用可能なプラグインを参照する**

   `/plugin` を実行してプラグインマネージャーを開きます。これにより、**Tab**（または後方に移動するには **Shift+Tab**）を使用して循環できるタブ付きインターフェースが開きます：

   - **Discover**: すべてのマーケットプレイスから利用可能なプラグインを参照
   - **Installed**: インストール済みプラグインを表示および管理
   - **Marketplaces**: 追加したマーケットプレイスを追加、削除、または更新
   - **Errors**: プラグイン読み込みエラーを表示
   - **Stats**: **各スキルのコンテキストコストと使用頻度を確認** (ページ 307) します。`/skill-doctor` が利用可能なセッションで確認できます

   Discover タブに移動して、追加したばかりのマーケットプレイスからプラグインを確認してください。管理者が `pluginSuggestionMarketplaces` マネージド設定を通じてマーケットプレイスをホワイトリストに登録している場合、現在の作業ディレクトリに関連するとマークされたプラグインは、**suggested for this directory** ラベル付きで上部に固定されます。

3. **プラグインをインストールする**

   プラグインを選択してその詳細を表示します。詳細ペインには、プラグインに含まれるもの、およびそのコストが表示されます：

   - **Context cost** の推定値。毎ターン コンテキストウィンドウ にプラグインが追加するトークン数を確認できます
   - プラグインの **Last updated** 日付
   - プラグインのコマンド、エージェント、スキル、フック、MCP および LSP サーバーをリストアップする **Will install** セクション。インストール前に正確に何が追加されるかを確認できます

   すべてのプラグインがこれらのフィールドの背後にあるデータを提供するわけではありません。ローカルまたはカスタムマーケットプレイスのプラグインの場合、**Context cost** および **Last updated** 行が表示されない場合があり、**Will install** セクションは代わりに **Components will be discovered at installation** を表示する場合があります。

   インストールスコープを選択します：

   - **User scope**: すべてのプロジェクト全体で自分用にインストール
   - **Project scope**: このリポジトリのすべてのコラボレーター用にインストール
   - **Local scope**: このリポジトリ内で自分用にのみインストール

たとえば、**commit-commands**（git ワークフロースキルを追加するプラグイン）を選択して、ユーザースコープにインストールします。

コマンドラインから直接インストールすることもできます：

```
/plugin install commit-commands@claude-code-plugins
```

スコープの詳細については、**設定ファイル** (ページ 711) を参照してください。

4. **新しいプラグインを使用する**

   インストール概要が `Run /reload-plugins to activate.` と報告されている場合、Claude Code はそのリロードを自動的に実行します。リロードが会話を再読み込みすることを警告する場合は、`/reload-plugins --force` を実行してプラグインをアクティベートしてください。

   プラグインスキルはプラグイン名でネームスペース化されているため、**commit-commands** は `/commit-commands:commit` のようなスキルを提供します。

   ファイルに変更を加えて、以下を実行して試してみてください：

   ```
   /commit-commands:commit
   ```

   これにより、変更がステージされ、コミットメッセージが生成され、コミットが作成されます。

   各プラグインは異なる方法で機能します。**Discover** タブのプラグインの詳細を確認して、提供されるコマンドとスキルを確認するか、使用方法のガイダンスについてそのホームページにアクセスしてください。

25.5 マーケットプレイスを追加する

`/plugin marketplace add` コマンドを使用して、異なるソースからマーケットプレイスを追加します。

Tip:

ショートカット: `/plugin marketplace` の代わりに `/plugin market` を使用でき、`remove` の代わりに `rm` を使用できます。

- **GitHub リポジトリ**: `owner/repo` 形式（例：`anthropics/claude-code`）
- **Git URL**: 任意の git リポジトリ URL（GitLab、Bitbucket、自己ホスト）
- **ローカル パス**: ディレクトリまたは `marketplace.json` ファイルへの直接パス
- **リモート URL**: ホストされた `marketplace.json` ファイルへの直接 URL
- **claude.ai**: claude.ai でホストされているマーケットプレイス（組織のプラグイン ライブラリなど）。これらは *Marketplaces* **タブまたはシェルから名前で追加** (ページ 684) します。ソースではなく名前で追加します。

25.5.1 GitHub から追加する

`.claude-plugin/marketplace.json` ファイルを含む GitHub リポジトリを `owner/repo` 形式を使用して追加します。ここで `owner` は GitHub ユーザー名または組織で、`repo` はリポジトリ名です。

たとえば、`anthropics/claude-code` は `anthropics` が所有する `claude-code` リポジトリを指します：

```
/plugin marketplace add anthropics/claude-code
```

25.5.2 他の Git ホストから追加する

git マーケットプレイス リポジトリを完全な URL を提供することで追加します。`https://` URL の場合、`.git` サフィックスを含めるかどうかはホストによって異なります：

- **`github.com` と `gitlab.com`**: Claude Code はリポジトリ URL を `.git` サフィックスの有無にかかわらず認識し、クローンします。サフィックスなしで `gitlab.com` URL を追加するには、Claude Code v2.1.232 以降が必要です。v2.1.232 より前では、Claude Code はこれをホストされた `marketplace.json` ファイルへの直接リンクとして扱いました。
- **Azure DevOps**: サフィックスを省略します。Claude Code はパスに `/_git/` を含む任意の URL をクローンします。`/_git/` パスに `.git` を追加すると、クローンが失敗します。
- **自己管理 GitLab サーバーを含む他のすべてのホスト**: `.git` サフィックスを含めて、Claude Code がリポジトリをクローンするようにしてください。URL をホストされた `marketplace.json` ファイルへの直接リンクとして扱うのではなく。AWS CodeCommit のようにクローン URL にサフィックスを含まないホストの場合は、代わりに `extraKnownMarketplaces` に git エントリとしてマーケットプレイスを追加してください。Claude Code は git エントリを URL が `.git` で終わるかどうかにかかわらずクローンします。

Claude Code は `https://gitlab.com/group/subgroup/project` のようなネストされたサブグループを持つ `gitlab.com` URL もクローンします。

`https://` プレフィックスを含めてください。Claude Code v2.1.196 以降は、`gitlab.com/company/plugins.git` のようにプレフィックスなしで入力されたホストを無効な GitHub `owner/repo` ショートハンドとして拒否し、エラーメッセージでプレフィックスを追加するよう指示します。以前のバージョンでは、これを GitHub リポジトリパスとして誤読し、クローン時に失敗します。

HTTPS を使用する場合：

```
/plugin marketplace add https://gitlab.com/company/plugins.git
```

SSH を使用する場合：

```
/plugin marketplace add git@gitlab.com:company/plugins.git
```

Claude Code は SSH アドレスを `.git` で終わるかどうかにかかわらずクローンします。

特定のブランチまたはタグを追加するには、`#` の後に ref を追加します：

```
/plugin marketplace add https://gitlab.com/company/plugins.git#v1.0.0
```

25.5.3 ローカル パスから追加する

`.claude-plugin/marketplace.json` ファイルを含むローカル ディレクトリを追加します：

```
/plugin marketplace add ./my-marketplace
```

`marketplace.json` ファイルへの直接パスを追加することもできます：

```
/plugin marketplace add ./path/to/marketplace.json
```

25.5.4 リモート URL から追加する

URL 経由でリモート `marketplace.json` ファイルを追加します：

```
/plugin marketplace add https://example.com/marketplace.json
```

注釈:

URL ベースのマーケットプレイスは、Git ベースのマーケットプレイスと比べていくつかの制限があります。URL ベースのマーケットプレイスからのプラグインのインストールが失敗する場合は、**トラブルシューティング** (ページ 675) を参照してください。

25.5.5 claude.ai から追加する

プラグインが *claude.ai* アカウントから同期されるターミナル セッション (ページ 590) では、claude.ai はマーケットプレイスをリストアップすることもできます。組織のプラグイン ライブラリや独自の claude.ai アップロードなど。`claude plugin marketplace list` は `From claude.ai:` セクションでそれらを出力し、`/plugin` **Marketplaces** タブにもリストされます。そこから 1 つを選択して追加します。claude.ai からマーケットプレイスを追加するには、Claude Code v2.1.273 以降が必要です。

シェルから追加するには、`claude plugin marketplace add` を `--claudeai` フラグとリストに表示されている名前で実行します：

```
claude plugin marketplace add --claudeai claudeai-organization-library
```

Claude Code はマーケットプレイスをローカル名で登録します。これは claude.ai がリストアップしている名前から派生した `claudeai-` で始まる名前です。「Organization library」としてリストされているマーケットプレイスは `claudeai-organization-library` として登録されます。その名前でプラグインをインストールします。たとえば `claude plugin install <plugin>@claudeai-organization-library` を使用します。

サインアウトするか別のアカウントでサインインすると、マーケットプレイスは設定されたままですがプラグインが表示されず、既にそこからインストールしたプラグインは引き続き読み込まれます。

`From claude.ai:` セクションは、claude.ai を通じて共有されている git ベースのマーケットプレイスもリストアップできます。これらは通常の `marketplace add` コマンドで追加します。リストが出力するソースを使用します。

25.6 プラグインをインストールする

マーケットプレイスを追加したら、プラグインを名前でインストールできます。まだ追加していないマーケットプレイスの場合は、代わりに *1* **つのコマンドでマーケットプレイスを追加してインストール** (ページ 686) できます。

名前でインストールするには：

```
/plugin install plugin-name@marketplace-name
```

このコマンドはそのプラグインの詳細を開き、**インストール スコープ** (ページ 711) を選択できます。`/plugin` を実行して **Discover** タブに移動し、プラグインで **Enter** を押すと、同じ選択肢が表示されます：

- **User scope**: すべてのプロジェクト全体で自分用にインストール
- **Project scope**: このリポジトリのすべてのコラボレーター用にインストール（`.claude/settings.json` に追加）
- **Local scope**: このリポジトリ内で自分用にのみインストール（コラボレーターと共有されない）

インタラクティブなステップなしでインストールするには、*`claude plugin install`* (ページ 611) シェルコマンドを使用します。このコマンドはユーザー スコープにインストールします。`--scope` を渡さない限り、ユーザー スコープにインストールされます。*command* **ソース** (ページ 649) を持つプラグインの場合、`--yes` を渡して、それが表示するコマンドを受け入れます。

managed スコープのプラグインも表示される場合があります。これらは管理者が管理設定経由でインストールしたもので、変更することはできません。

Claude Code はローカル マーケットプレイス カタログのコピーでプラグインを検索します。プラグインの名前の付け方によって、Claude Code がそのコピーを最初に更新するかどうかが決まります：

- **マーケットプレイス名を含む場合**: セッションで `plugin-name@marketplace-name` をインストールするか、`claude plugin install` で実行すると、Claude Code はルックアップの前にそのマーケットプレイスを更新します。Claude Code は、マーケットプレイスの**自動更新** (ページ 690) をオフにしたか、`DISABLE_AUTOUPDATER` を設定した場合でも、更新を実行します。v2.1.232 より前では、Claude Code はルックアップの前にマーケットプレイスを更新しませんでした。Claude Code は以下の場合、この更新をスキップします：
  - マーケットプレイスが *GitHub*、**別の** *Git* **ホスト、リモート** *URL* **から追加** (ページ 682) されていない、または *Claude.ai* **から追加** (ページ 684) されていない。
  - **シード ディレクトリ** (ページ 656) がマーケットプレイスを提供している。
  - Claude Code が過去 30 秒以内にマーケットプレイスを更新した。
  - `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` を設定した。
  - **管理設定** (ページ 658) がマーケットプレイスをブロックしている。この場合、Claude Code はインストールも拒否します。
- **プラグイン名のみ**: セッションで `/plugin install plugin-name` を実行すると、Claude Code は**バックグラウンドでも更新** (ページ 690) するマーケットプレイスのみを更新し、ルックアップが失敗した後のみです。`claude plugin install plugin-name` を実行すると、Claude Code は更新なしでキャッシュされたカタログを読み取ります。最後の更新後に公開されたプラグインをインストールす

るには、セッションで /plugin marketplace update <marketplace-name> を実行するか、シェルで *claude plugin marketplace update <marketplace-name>* (ページ 668) を実行してから、インストールを再試行します。

名前付きインストール前の更新が失敗した場合（例えば、オフラインの場合）、Claude Code はキャッシュされたカタログでプラグインを検索します。claude plugin install は成功メッセージで marketplace not refreshed を報告し、/plugin install はプラグインの詳細の上または見つからないメッセージでエラーを表示します。

/plugin インターフェイスからインストールすると、インストール サマリーは、プラグインが現在のセッションでアクティブかどうかを示します：

- Plugin is now active.: Claude Code はインストールの一部としてプラグインをアクティブにしました。
- Run /reload-plugins to activate.: プラグインはまだアクティブではありません。これは、アクティブ化するとプロンプト キャッシュが無効になるか、アクティブ化の試行が失敗したためです。Claude Code はその後、/reload-plugins を実行します。そのリロードがプロンプト キャッシュについて警告する場合、**プラグインをとにかくアクティブにする** (ページ 689) には /reload-plugins --force を実行してください。
- プラグインの読み込みに失敗した場合、サマリーは失敗を報告し、/plugin **Errors** タブに詳細が表示されます。

v2.1.221 より前では、/reload-plugins を実行するか再起動するまで、現在のセッションでインストールが有効になりませんでした。

claude plugin install シェル コマンドはセッションで実行されないため、Claude Code は次回 Claude Code を起動するか、既に開いているセッションで /reload-plugins を実行するときに、インストールしたプラグインを読み込みます。

警告:

プラグインをインストールする前に、それを信頼していることを確認してください。Anthropic はプラグインに含まれる MCP サーバー、ファイル、またはその他のソフトウェアを制御せず、意図したとおりに機能することを確認できません。詳細については、各プラグインのホームページを確認してください。

25.6.1 1 つのコマンドでマーケットプレイスを追加してインストールする

まだ追加していないマーケットプレイスからプラグインをインストールするには、--marketplace でマーケットプレイス ソースを指定します。Claude Code v2.1.275 以降が必要です。

```
/plugin install quality-review-plugin --marketplace your-org/plugins
```

ソースは*/plugin marketplace add* (ページ 682) と同じ形式を取ります。例えば、GitHub の owner/repo、git URL、またはローカル パスなど。ただし、スペースを含むことはできません。プラグイン名は @marketplace サフィックスなしで指定します。

Claude Code はソースが解決されたことを表示し、マーケットプレイスを追加する前に確認を求めます。キャンセルするとインストールがキャンセルされ、何も追加されません。マーケットプレイスが追加されると、プラグインの詳細が開き、**インストール スコープ** (ページ 711) を選択できます。ソースが既に追加

したマーケットプレイスと一致する場合、Claude Code は確認をスキップし、そのマーケットプレイスでプラグインの詳細を開きます。

25.7 インストール済みプラグインを管理する

`/plugin` を実行して **Installed** タブに移動し、プラグインを表示、有効化、無効化、またはアンインストールします。リストはスコープでグループ化され、問題が最初に表示されるようにソートされます。読み込みエラーまたは未解決の依存関係を持つプラグインが上部に表示され、その後にお気に入りが続き、無効化されたプラグインは下部の折りたたまれたヘッダーの後ろに折りたたまれます。

リストから以下を実行できます：

- `f` を押して、選択したプラグインをお気に入りに追加またはお気に入りから削除
- 入力してプラグイン名または説明でフィルタリング
- Enter を押してプラグインの詳細ビューを開き、有効化、無効化、またはアンインストール

Claude Code はまた、*claude.ai* **アカウントから同期されたプラグイン** (ページ 590) を **Installed** タブに一覧表示します。ソースは `synced` です。組織がそれを必須としてマークしていない限り、そこで有効化または無効化できます。削除するには、claude.ai でオフにします。同期されたプラグインは Claude Code v2.1.273 以降のターミナルセッションに表示されます。

プロジェクトの `.claude/settings.json` が有効化しているプラグインをアンインストールすると、Claude Code はどのスコープを意図しているかを尋ねます。自分だけのために無効化する場合は、`.claude/settings.local.json` にオーバーライドを書き込み、プラグインはプロジェクトにインストールされたままになります。または、すべてのユーザーのためにアンインストールする場合は、共有の `.claude/settings.json` から削除されます。

詳細ビューには、プラグインが提供するコンポーネントが表示されます。コマンド、skills、agents、hooks、MCP サーバー、LSP サーバーです。同じインベントリは、コマンドラインから `claude plugin details` で利用できます。

Claude Code はまた、マーケットプレイスから自分でインストールしたが、少なくとも 2 週間以上、かつ少なくとも 10 セッション以上にわたって使用していないプラグインも一覧表示します。これらは **Installed** タブの **Not used recently** ヘッダーの下に表示されます。詳細ビューには、各プラグインの **Last used** 行が表示されます。これらを使用して、もう使用していないが、それでもスタートアップとコンテキストコストを追加しているプラグインを見つけ、無効化またはアンインストールできます。

2 種類のプラグインは、未使用として一覧表示されることはありません：

- 組織が管理するプラグイン、または `--plugin-dir` で読み込むプラグイン
- テーマ、出力スタイル、モニター、またはワークフローを提供するプラグイン。これらは追跡する呼び出しなしで価値を提供するため

Not used recently ヘッダーと **Last used** 行は、組織が `strictKnownMarketplaces` でマーケットプレイスを制限している場合、両方とも非表示になります。

プラグインの **言語サーバー** (ページ 570) は、診断を提供するか、コード ナビゲーション リクエストに応答するときに使用済みとしてカウントされます。そのため、LSP プラグインのサーバーがセッションでアクティブな場合、未使用として一覧表示されません。v2.1.203 より前では、言語サーバー アクティビティ

を使用としてカウントできなかったため、LSP サーバーを提供するプラグインはグループ全体から除外されていました。これはテーマと出力スタイル プラグインと同じ方法です。

言語サーバー アクティビティをカウントするバージョンの最初のセッションは、まだ使用を記録していない各 LSP プラグインの使用記録もリセットするため、Claude Code は、サーバー アクティビティが追跡される前に記録されたデータに基づいて、以前にインストールしたプラグインを未使用と判断しません。

依存関係を宣言するプラグインをインストールすると、インストール出力には、それと共に自動インストールされた依存関係が一覧表示されます。

直接コマンドでプラグインを管理することもできます：

- `/plugin disable`、`/plugin enable`、または `/plugin uninstall` を実行すると、Claude Code はプラグインパネルを開いて変更を適用し、パネルを開いたままにします。別のコマンドを入力する前に **Esc** を押してパネルを閉じます。**プラグインの変更をリスタートなしで適用する** (ページ 689) では、変更がセッションでいつ有効になるかについて説明しています。
- スクリプティングの場合は、代わりに `claude plugin` シェルコマンドを使用します。これらはパネルを開きません。

メニューを開かずにインストール済みプラグインを一覧表示します：

```
/plugin list
```

`--enabled` または `--disabled` を渡して、その状態のプラグインのみを表示します。

プラグインをアンインストールせずに無効化します：

```
/plugin disable plugin-name@marketplace-name
```

無効化されたプラグインを再度有効化します：

```
/plugin enable plugin-name@marketplace-name
```

これらの識別子では、`plugin-name` は **マーケットプレイスエントリ** (ページ 633) の `name` であり、プラグイン自体の `plugin.json` の `name` と異なる場合があります。

Claude Code v2.1.195 以降では、`/plugin` インターフェイスの **Enable** と **Disable** は、2 つの名前が異なるプラグインに対して機能し、`/plugin enable` と `/plugin disable` はどちらの名前でも受け入れます。以前のバージョンでそのようなプラグインを無効化すると、Claude Code は `already disabled` を報告し、有効なままにします。

プラグインを完全に削除します：

```
/plugin uninstall plugin-name@marketplace-name
```

`--scope` オプションを使用すると、CLI コマンドで特定のスコープをターゲットにできます：

```
claude plugin install formatter@your-org --scope project
claude plugin uninstall formatter@your-org --scope project
```

25.7.1 プラグインの変更をリスタートなしで適用する

`/plugin` メニューを閉じると、Claude Code はインストール、有効化、無効化、アンインストールなど、メニューで行った変更を適用するために `/reload-plugins` を実行します。リロードが プロンプトキャッシュを無効にする 場合、警告を表示し、変更を保留のままにします。代わりに `/reload-plugins --force` を実行して適用します。Claude がまだ応答中の場合、リロードは応答が終了した後に実行されます。

メニューの外で発生するプラグイン変更については、`/reload-plugins` を自分で実行します。これらの変更には以下が含まれます：

- 別のターミナルで実行した `claude plugin` コマンド
- 開発中に *`--plugin-dir`* (ページ 572) で読み込んだプラグインへの編集
- 再度読み込むよう求める通知を表示するプラグイン **自動更新** (ページ 690)
- *claude.ai* **アカウントからの同期** (ページ 590) がプラグインを追加、更新、または削除し、再度読み込むよう求める通知を表示した
- Claude Code が保留していた *--plugin-dir* **フォルダ** (ページ 572) の変更。これは、適用するとプロンプトキャッシュが無効になるため

v2.1.268 より前では、メニューで有効化、無効化、またはアンインストールしたプラグイン、およびインストール中にアクティブ化されなかったインストールは、`/reload-plugins` を実行するまで保留のままでした。

`/reload-plugins` はまた、デスクトップアプリ、Agent SDK、および *-p* **を使用した非対話型モード** (ページ 695) など、対話型ターミナルのないセッションでも実行されます。Claude Code v2.1.260 以降が必要です。これらのセッションでは 2 つの制限が適用されます：

- コマンドは、`-p` プロンプトやデスクトップアプリのプロンプトボックスなど、セッションに直接入力した場合にのみ実行されます。代わりに Remote Control やリレーされたチャットメッセージなどのリモート接続を介して送信する場合、コマンドは何もリロードせずに拒否されます。
- リロードはプラグイン MCP サーバーを接続または切断しません。これらの変更は次のセッションで有効になります。

Claude Code はすべてのアクティブなプラグインをリロードし、プラグイン、skills、agents、hooks、プラグイン MCP サーバー、プラグイン LSP サーバーのカウントを表示します。対話型ターミナルのないセッションではプラグイン MCP サーバーのカウントを省略します。skills のカウントでは、Claude Code はプラグインが提供するすべての skill を含みます。`commands/` エントリと `SKILL.md` skills の両方です。v2.1.246 より前では、Claude Code は `commands/` エントリのみをカウントしていたため、プラグインの `SKILL.md` skills をリロードしても、概要で `0 skills` を報告する可能性がありました。

リロードには次のリクエストでトークンコストがあります。新しくロードされたコンポーネントは会話に追加されたコンテンツで自身を発表し、既存の履歴はプロンプトキャッシュから読み込まれたままです。MCP サーバーを提供するプラグインは、そのツールが *tool search* (ページ 406) によって遅延されていない場合、より多くのコストがかかります。変更はキャッシュを無効にし、次のリクエストは会話全体を再度読み込みます。詳細については、プラグインの有効化または無効化 を参照してください。

25.8 マーケットプレイスを管理する

インタラクティブな /plugin インターフェースまたは CLI コマンドを使用してマーケットプレイスを管理できます。

25.8.1 インタラクティブ インターフェースを使用する

/plugin を実行して **Marketplaces** タブに移動して、以下を実行します：

- 追加したすべてのマーケットプレイスをそのソースとステータスで表示
- 新しいマーケットプレイスを追加
- マーケットプレイス リストを更新して最新のプラグインを取得
- 不要になったマーケットプレイスを削除

25.8.2 CLI コマンドを使用する

直接コマンドでマーケットプレイスを管理することもできます。

構成されたすべてのマーケットプレイスをリストします：

```
/plugin marketplace list
```

マーケットプレイスからプラグイン リストを更新します：

```
/plugin marketplace update marketplace-name
```

マーケットプレイスを削除します：

```
/plugin marketplace remove marketplace-name
```

警告:

マーケットプレイスを削除すると、そこからインストールしたプラグインがアンインストールされます。

25.8.3 自動更新を構成する

Claude Code はスタートアップ時にマーケットプレイスとそのインストール済みプラグインをバックグラウンドで自動的に更新できます。マーケットプレイスで自動更新が有効になっている場合、Claude Code はマーケットプレイス データを更新し、インストール済みプラグインをディスク上の最新バージョンに更新します。

Claude Code はセッション開始後、最大 10 分のランダムな遅延でマーケットプレイスとプラグインの更新をチェックするため、実行中のセッションは起動時に読み込んだバージョンを使用し続けます。プラグインが更新された場合、/reload-plugins を実行するよう促すメッセージが表示されるか、次回の起動時に新しいバージョンが読み込まれます。

自動更新は、マーケットプレイス エントリが `headersHelper` を宣言するプラグインも除外します。Claude Code は **そのパス上のコマンドを実行したり、アーカイブをダウンロードしたりしません** (ページ 646)。そのセクションでは、Claude Code がプラグインを `/plugin` Errors タブにリストする時期について説明しており、独自のビューから更新できます。

Claude Code は *command* **ソース** (ページ 647) を持つプラグインを、マーケットプレイス自動更新設定および `DISABLE_AUTOUPDATER` とは異なるペースで更新します。代わりに、**セッションごとに *1* 回コマンドを再実行** (ページ 649) し、その **ハッシュ** (ページ 624) が変更された場合、出力を新しいプラグイン バージョンとしてインストールします。

UI を通じて個別のマーケットプレイスの自動更新を切り替えます：

1. `/plugin` を実行してプラグイン マネージャーを開く
2. **Marketplaces** を選択
3. リストからマーケットプレイスを選択
4. **Enable auto-update** または **Disable auto-update** を選択

`claude-plugins-official`、その他のほとんどの公式 Anthropic マーケットプレイス、および *claude.ai* **から追加されたマーケットプレイス** (ページ 684) はデフォルトで自動更新が有効になっています。その他のサードパーティ マーケットプレイスおよびローカル開発マーケットプレイスはデフォルトで自動更新が無効になっています。

管理者は、マネージド設定で各 `extraKnownMarketplaces` エントリに `"autoUpdate": true` を設定して、各ユーザーが切り替える必要なく、組織マーケットプレイスの自動更新を有効にすることもできます。

Claude Code とマーケットプレイスから取得されたプラグインの自動更新を無効にするには、`DISABLE_AUTOUPDATER` 環境変数を設定します。*command* **ソース** (ページ 647) を持つプラグインは独自のセッションごと 1 回の再解決に従います。詳細については、**自動更新** (ページ 24) を参照してください。

Claude Code の自動更新を無効化しながらプラグイン自動更新を有効化したままにするには、`DISABLE_AUTOUPDATER` と共に `FORCE_AUTOUPDATE_PLUGINS=1` を設定します：

```
export DISABLE_AUTOUPDATER=1
export FORCE_AUTOUPDATE_PLUGINS=1
```

25.9 チーム マーケットプレイスを構成する

チーム管理者は、`.claude/settings.json` にマーケットプレイス構成を追加することで、プロジェクトの自動マーケットプレイス インストールを設定できます。チーム メンバーが**リポジトリ フォルダを信頼** (ページ 1066) すると、Claude Code はこれらのマーケットプレイスをさらなるプロンプトなしで追加します。

Claude Code v2.1.195 以降、マーケットプレイスを追加しても、プラグインを読み込むすべてのパスで外部ソースから提供されるプラグインはインストールされません。プロジェクトの `.claude/settings.json` のみで有効にされ、GitHub リポジトリや npm パッケージなどの外部ソースから提供されるプラグインは、チーム メンバーがインストールするまで読み込まれません。それまでの間、Claude Code はプラグインがインストールされていないと報告し、実行する `claude plugin install` コマンドを表示します。

プロジェクトの `.claude/settings.json` に `extraKnownMarketplaces` を追加します：

```
{
  "extraKnownMarketplaces": {
    "my-team-tools": {
      "source": {
        "source": "github",
        "repo": "your-org/claude-plugins"
      }
    }
  }
}
```

extraKnownMarketplaces と enabledPlugins を含む完全な構成オプションについては、プラグイン設定を参照してください。

25.10 セキュリティ

プラグインとマーケットプレイスは、ユーザー権限でマシン上で任意のコードを実行できる、非常に信頼されたコンポーネントです。信頼できるソースからのみプラグインをインストールし、マーケットプレイスを追加してください。組織は、**管理マーケットプレイス制限** (ページ 658) を使用してユーザーが追加できるマーケットプレイスを制限できます。

25.11 トラブルシューティング

25.11.1 /plugin コマンドが認識されない

「unknown command」が表示されるか、/plugin コマンドが表示されない場合：

1. **バージョンを確認する**: claude --version を実行して、インストールされているものを確認します。
2. **Claude Code を更新する**:
   - **Homebrew**: brew upgrade claude-code、または brew upgrade claude-code@latest をインストールした場合は brew upgrade claude-code@latest
   - **npm**: npm install -g @anthropic-ai/claude-code@latest
   - **ネイティブ インストーラー**: **セットアップ** (ページ 19) からインストール コマンドを再実行します。
3. **Claude Code を再起動する**: 更新後、ターミナルを再起動して claude を再度実行します。

25.11.2 一般的な問題

プラグイン スキルが表示されない場合は、rm -rf ~/.claude/plugins/cache でキャッシュをクリアし、Claude Code を再起動して、プラグインを再度インストールしてください。

詳細なトラブルシューティングとソリューションについては、マーケットプレイス ガイドの **トラブルシューティング** (ページ 669) を参照してください。デバッグ ツールについては、**デバッグと開発ツール** (ページ 622) を参照してください。

25.11.3 コード インテリジェンスの問題

- **言語サーバーが起動しない**: バイナリがインストールされており、`$PATH` で利用可能であることを確認してください。詳細については、`/plugin` Errors タブを確認してください。
- **メモリ使用量が多い**: `rust-analyzer` や `pyright` などの言語サーバーは、大規模なプロジェクトで大量のメモリを消費する可能性があります。メモリの問題が発生した場合は、`/plugin disable <plugin-name>` でプラグインを無効化し、代わりに Claude の組み込み検索ツールを使用してください。
- **モノレポでの誤検知診断**: ワークスペースが正しく構成されていない場合、言語サーバーは内部パッケージの未解決インポート エラーを報告する可能性があります。これらはコードを編集する Claude の能力に影響しません。

25.12 次のステップ

- **独自のプラグインを構築する**: スキル、エージェント、フックを作成するには、**プラグイン** (ページ 565) を参照してください。
- **マーケットプレイスを作成する**: チームまたはコミュニティにプラグインを配布するには、**プラグイン マーケットプレイスを作成** (ページ 627) を参照してください。
- **技術リファレンス**: 完全な仕様については、**プラグイン リファレンス** (ページ 579) を参照してください。

第26章　Claude Code をプログラムで実行する

> Agent SDK を使用して、CLI、Python、または TypeScript からプログラムで Claude Code を実行します。

Agent SDK は、Claude Code を支える同じツール、エージェントループ、およびコンテキスト管理を提供します。スクリプトと CI/CD 用の CLI として、または完全なプログラムによる制御のための Python および TypeScript パッケージとして利用できます。

Claude Code を非対話型モードで実行するには、プロンプトと任意の *CLI* **オプション** (ページ 45) を指定して `-p` を渡します。

```
claude -p "Find and fix the bug in auth.py" --allowedTools "Read,Edit,Bash"
```

このページでは、CLI（`claude -p`）経由で Agent SDK を使用することについて説明しています。構造化された出力、ツール承認コールバック、およびネイティブメッセージオブジェクトを備えた Python および TypeScript SDK パッケージについては、完全な Agent SDK ドキュメント を参照してください。

26.1 基本的な使用方法

任意の `claude` コマンドに `-p`（または `--print`）フラグを追加して、非対話的に実行します。すべての *CLI* **オプション** (ページ 45) が `-p` と組み合わせられるわけではありません。Claude Code は `--bg` を拒否し、タスク説明付きの `--cloud` を拒否します。競合を示すエラーが表示されます。セッション ID 付きの `--cloud` と `-p` は代わりに **そのクラウドセッションにメッセージをキューイングして終了します** (ページ 228)。`-p` と組み合わせることが多いオプションには以下が含まれます。

- `--continue` は **会話を続行する** (ページ 707) ため
- `--allowedTools` は **ツールを自動承認する** (ページ 704) ため
- `--output-format` は **構造化された出力を取得する** (ページ 699) ため

この例は、コードベースについて Claude に質問し、応答を出力します。

```
claude -p "What does the auth module do?"
```

Claude Code は成功時にコード 0 で終了し、実行が失敗した場合は 0 以外のコードで終了するため、スクリプトは終了ステータスで分岐できます。無効なフラグを渡すと、Claude Code は実行開始前にエラーを stderr に報告します。実行内で認証の欠落など障害が発生した場合、Claude Code は障害を stdout の結果として出力します。

26.1.1 ベアモードで高速に開始する

--bare を追加して、hooks、skills、カスタムコマンド、*subagents* (ページ 317)、インストール済みプラグイン、MCP サーバー、自動メモリ、CLAUDE.md の自動検出をスキップすることで、起動時間を短縮します。これがない場合、claude -p は対話的セッションと同じ コンテキスト を読み込みます。これには、作業ディレクトリまたは ~/.claude で設定されたすべてが含まれます。

ベアモードは、すべてのマシンで同じ結果が必要な CI とスクリプトに役立ちます。チームメイトの ~/.claude のフック、またはプロジェクトの .mcp.json の MCP サーバーは実行されません。ベアモードはそれらを読み込まないためです。--add-dir で指定するディレクトリは部分的な例外です。ベアモードはその .claude/skills/ フォルダから skills を読み込みますが、その .claude/commands/ と .claude/agents/ フォルダはスキップします。**追加ディレクトリからの** *Skills* (ページ 281) は、何が読み込まれ、何が読み込まれないかについて説明しています。

--bare がない場合、-p セッションはプロジェクトの .claude/settings.json のフックを実行し、その .mcp.json のサーバーに接続します。これは、信頼したことのないフォルダでも同様です。-p セッションはワークスペース信頼ダイアログもサーバーごとの承認プロンプトも表示しません。**フォルダを信頼する前に実行されるもの** (ページ 1066) は、-p の下での各種リポジトリコンテンツと、それを除外する方法について説明しています。

この例は、ベアモードで 1 回限りの要約タスクを実行し、Read ツールを事前承認して、権限プロンプトなしで呼び出しが完了するようにします。ベアモードはサブスクリプションログインを使用しないため、実行前に ANTHROPIC_API_KEY を設定してください。

```
claude --bare -p "Summarize README.md" --allowedTools "Read"
```

ベアモードでは、Claude Code は OAuth 認証情報またはシステムキーチェーンを読み込みません。Anthropic API の場合、環境で ANTHROPIC_API_KEY を設定します。Claude Console で作成されたキーを使用するか、--settings JSON で apiKeyHelper を指定します。Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry は、通常どおり独自のプロバイダー認証情報を読み込み続けます。

ベアモードでは、Claude は Bash、ファイル読み取り、ファイル編集ツールにアクセスできます。フラグで必要なコンテキストを渡します。

読み込むもの	使用
システムプロンプト追加	--append-system-prompt、--append-system-prompt-file
設定	--settings <file-or-json>
MCP サーバー	--mcp-config <file-or-json>
カスタムエージェント	--agents <json>
プラグイン	--plugin-dir <path>、--plugin-url <url>

bare モードでは、セッションの実行中に行われる処理も制限されます。

- **MCP サーバー**: コマンドラインで指定されたサーバー（例えば --mcp-config で指定したもの）のみが接続されます。対話型セッションでは、--ide を渡さない限り、Claude Code は IDE への自動接続もスキップします。
- **システムリマインダー**: Claude はプロンプトとツールの結果を受け取りますが、Claude Code が通常それらと一緒に追加する **システムリマインダー** は含まれません。例えば、以前に読み取ったファイ

ルがディスク上で変更されても Claude には通知されず、`--add-dir` フォルダのスキルを含む利用可能なスキルの一覧も Claude には渡されません。

- **バックグラウンドタスク**: 実行されません。タイムアウト に達したコマンドは、バックグラウンドに移行する 代わりに停止します。

v2.1.286 より前は、これらの制限は部分的にしか適用されていませんでした。対話型の `--bare` セッションは通常のセッションと同じ MCP サーバーに接続し、すべての `--bare` セッションがシステムリマインダーを送信し、バックグラウンドタスクも引き続き利用可能でした。

注釈:

`--bare` はスクリプト化および SDK 呼び出しの推奨モードであり、将来のリリースで `-p` のデフォルトになります。

26.1.2 終了時のバックグラウンドタスク

Claude が `claude -p` 実行中に バックグラウンド Bash タスク （例えば、開発サーバーまたはウォッチビルド）を開始した場合、Claude が最終結果を返し、stdin が閉じられてから約 5 秒後に、そのシェルは終了します。猶予期間により、結果の直後に終了するタスクでも出力を配信できます。

Claude がバックグラウンド *subagent* (ページ 317) またはワークフローを開始した場合、`claude -p` は代わりにその作業が完了するまで開いたままになります。その結果は最終出力の一部だからです。

デフォルトでは、待機は 10 分間の継続的なアイドル待機後に終了するため、スタックした subagent またはワークフローがプロセスを無期限に開いたままにすることはできません。その時点で Claude Code は実行中のすべてを停止し、その部分的な結果をドロップします。制限を変更するには、`CLAUDE_CODE_PRINT_BG_WAIT_CEILING_MS` を設定するか、`0` に設定して制限なしで待機します。

Claude が `claude -p` 実行中に Monitor ウォッチを開始した場合、Claude Code はウォッチがタイムアウトするか、10 分の上限が待機を終了するまで、どちらか先に来た方まで待機します。待機中、Claude はウォッチが報告することに応答し続けます。デフォルトでは、ウォッチは Claude が開始してから 5 分後にタイムアウトします。

26.1.3 SIGTERM で実行を停止する

`claude -p` 実行を SIGTERM で停止した場合 (例えば、`kill` またはプロセススーパーバイザーから)、Claude Code はコード 143 で終了します。Claude Code は進行中のターンを未完了のままにし、そのための結果を記録しません。ターンを終了するには、SIGINT を送信するか、Agent SDK の `interrupt()` を呼び出してから、プロセスを停止します。

SIGTERM では、Claude Code はまだ実行中の Bash コマンドのプロセスツリーを終了します。Claude Code は *SessionEnd hooks* (ページ 548) を実行して終了します。終了中、Claude Code は新しいツール呼び出しを開始せず、新しいモデルリクエストを送信せず、`SessionEnd` 以外のフックを実行しません。実行が権限プロンプトへの回答を待機しているコマンドの途中にあった場合、Claude Code はそのステップを次のように処理します。

- **コマンドを実行中**: Claude Code はコマンドをセッションで強制終了として記録します。

- **権限プロンプトへの回答を待機中**: SIGTERM をプロセスに送信した場合、Claude Code はプロンプトを未回答のままにします。プログラムが Agent SDK を通じてセッションを閉じた場合、SDK はシグナルを送信する前に Claude Code の入力を終了し、Claude Code は入力が終了するとすぐにプロンプトをキャンセルします。

セッションを再開 (ページ 707) すると、Claude Code は中断されたターンをそのままにし、次のプロンプトが会話を駆動します。再開時に Claude Code が中断されたターンを続行するようにするには、`CLAUDE_CODE_RESUME_INTERRUPTED_TURN=1` を設定します。

26.1.4 作業ディレクトリが削除された場合

`claude -p` または Agent SDK セッションの作業ディレクトリがセッション中に削除された場合、セッションは実行を続けます。ディレクトリが見つからない間にターンが開始されると、Claude Code は `stream-json` 出力で 警告メッセージ を発行し、ディレクトリが再度存在するまでシェルコマンドは失敗します。

26.2 例

これらの例は、一般的な CLI パターンを強調しています。`auth.py` や `build-error.txt` などのファイルを指定するコマンドの場合は、自分のプロジェクトのファイルに置き換えてください。CI やその他のスクリプト環境では、*`--bare`* (ページ 696) を追加して、Claude Code がホストの hooks、plugins、auto memory、または `CLAUDE.md` を読み込まずに起動するようにしてください。

26.2.1 Claude にデータをパイプする

非対話モードは stdin を読み込むため、他のコマンドラインツールと同様にデータをパイプして応答をリダイレクトできます。

この例は、ビルドログを Claude にパイプして、説明をファイルに書き込みます。

```
cat build-error.txt | claude -p 'concisely explain the root cause of this build error
↪' > output.txt
```

`--output-format json` を使用すると、応答ペイロードに `total_cost_usd` とモデルごとのコスト内訳が含まれるため、スクリプト呼び出し元は *usage dashboard* (ページ 911) を参照せずに支出を追跡できます。`--continue` または `--resume` で以前の会話を続ける場合、実行は会話全体の合計を報告し、以前の実行の支出を含めて。両方の数値は client-side estimates であり、実際の請求額と異なる場合があります。

注釈:

パイプされた stdin は 10MB に制限されています。制限を超えた場合、Claude Code は明確なエラーメッセージを表示して終了し、ゼロ以外のステータスを返します。より大きな入力を処理するには、コンテンツをファイルに書き込み、パイプする代わりにプロンプトでファイルパスを参照してください。

Claude Code が stdin を読み込めない場合（例えば、それを開始したプロセスが終了した場合）、Claude Code は stderr に警告を出力して、コマンドラインからのプロンプトで続行します。v2.1.211 より前では、Windows で読み込み不可能な stdin はセッションをクラッシュさせるか、出力なしで静かに終了していました。

26.2.2 ビルドスクリプトに Claude を追加する

非対話呼び出しをスクリプトでラップして、Claude をプロジェクト固有のリンターまたはレビュアーとして使用できます。

この package.json スクリプトは main に対する diff をパイプして Claude に渡し、タイプミスを報告するよう指示します。diff をパイプすることで、Claude は Bash 権限がなくても読み込むことができ、エスケープされたダブルクォートはスクリプトを Windows に対応させます。

```
{
  "scripts": {
    "lint:claude": "git diff main | claude -p \"you are a typo linter. for each typo␣
↪in this diff, report filename:line on one line and the issue on the next. return␣
↪nothing else.\""
  }
}
```

npm run lint:claude で実行します。

26.2.3 構造化された出力を取得する

--output-format を使用して、応答の返され方を制御します。

- text（デフォルト）：プレーンテキスト出力
- json：結果、セッション ID、メタデータを含む構造化 JSON
- stream-json：リアルタイムストリーミング用の改行区切り JSON

この例は、プロジェクト概要を JSON で返し、セッションメタデータを含め、テキスト結果は result フィールドに入ります。

```
claude -p "Summarize this project" --output-format json
```

特定のスキーマに準拠した出力を取得するには、--output-format json を --json-schema と JSON Schema 定義と共に使用します。応答には、リクエストに関するメタデータ（セッション ID、使用状況など）が含まれ、構造化出力は structured_output フィールドに入ります。

この例は、関数名を抽出して文字列の配列として返します。

```
claude -p "Extract the main function names from auth.py" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"functions":{"type":"array","items":{
↪"type":"string"}}},"required":["functions"]}'
```

値が有効な JSON Schema でない場合、claude は Error: --json-schema is not a valid JSON Schema で終了し、その後にバリデータの診断が続きます。Claude Code は format キーワード（例："format": "email"）を使用するスキーマを受け入れますが、format を注釈として扱い、強制しません。v2.1.205 より前では、Claude Code は無効なスキーマを静かに無視して非構造化テキストを返し、format を含むスキーマを無効として扱っていました。

Tip:

jq などのツールを使用して応答を解析し、特定のフィールドを抽出します。

```
# テキスト結果を抽出
claude -p "Summarize this project" --output-format json | jq -r '.
↪result'

# 構造化出力を抽出
claude -p "Extract function names from auth.py" \
  --output-format json \
  --json-schema '{"type":"object","properties":{"functions":{"type
↪":"array","items":{"type":"string"}}},"required":["functions"]}
↪' \
  | jq '.structured_output'
```

26.2.4 応答をストリーミングする

`--output-format stream-json` を `--verbose` と `--include-partial-messages` と共に使用して、生成されるトークンをリアルタイムで受け取ります。各行は、イベントを表す JSON オブジェクトです。

```
claude -p "Explain recursion" --output-format stream-json --verbose --include-partial-
↪messages
```

ストリームの最後の行は、最終応答テキスト、コスト、セッションメタデータを含む `result` メッセージです。

コンシューマーがストリームをゆっくり読む場合、Claude Code はキューに入った出力がドレインされるまで待機し、待機時間をまだキューに入っているもの量に応じてスケーリングし、最大 30 秒に制限されます。v2.1.214 より前では、終了待機は約 2 秒に制限されており、大きな応答の終わりが切り取られる可能性がありました。

次の例は jq を使用してテキストデルタをフィルタリングし、ストリーミングテキストのみを表示します。`-r` フラグは生の文字列（引用符なし）を出力し、`-j` は改行なしで結合するため、トークンは継続的にストリーミングされます。

```
claude -p "Write a poem" --output-format stream-json --verbose --include-partial-
↪messages | \
  jq -rj 'select(.type == "stream_event" and .event.delta.type? == "text_delta") | .
↪event.delta.text'
```

コールバックとメッセージオブジェクトを使用したプログラマティックストリーミングについては、Agent SDK ドキュメントの Stream responses in real-time を参照してください。

サブエージェントメッセージをフォローする

subagents (ページ 317) からのメッセージは、ストリームに `assistant` および `user` メッセージとして表示され、その `parent_tool_use_id` フィールドはサブエージェントを生成したツール呼び出しの ID です。メインの会話からのメッセージは、そのフィールドに `null` を含みます。

foreground (ページ 344) で実行されているサブエージェントからの最初のメッセージは、それを駆動するプロンプトを含む `user` メッセージです。その最初のメッセージの後、Claude Code は以下を発行します。

- **デフォルト**：サブエージェントの `tool_use` および `tool_result` ブロック。
- *`--forward-subagent-text`* (ページ 51) **または** `CLAUDE_CODE_FORWARD_SUBAGENT_TEXT` **を使用**：サブエージェントのテキストおよび thinking ブロックも含まれるため、各サブエージェントのトランスクリプトを再構築できます。これには Claude Code v2.1.211 以降が必要です。

いずれかのオプションを有効にすると、Claude Code は *subagents at every nesting depth* (ページ 349) からのメッセージを転送します。Agent ツールまたは *forked skill* (ページ 303) として開始された場合、サブエージェントが生成されたかどうかに関わらず。フォークされたスキルが生成するサブエージェントのメッセージ、およびサブエージェント内またはフォークされたスキル内で開始されたフォークされたスキルには、Claude Code v2.1.275 以降が必要です。`parent_tool_use_id` では、ネストされたサブエージェントのメッセージは、それを開始した Agent または Skill ツール呼び出しの ID を含むため、これらの ID をフォローして完全なネストツリーを再構築できます。v2.1.219 より前では、ネストされたサブエージェントからのメッセージはストリームに表示されていませんでした。

subagent **で実行される** (ページ 303) Skills は、ストリームに同じ方法で表示されます。フォークされたスキルの最初のメッセージは、実行を駆動するスキルコンテンツを含む `user` メッセージです。いずれかのオプションを有効にすると、ストリームはフォークされたスキルのテキストおよび thinking ブロックも含みます。v2.1.265 より前では、フォークされたスキルの `tool_use` および `tool_result` ブロックのみがストリームに表示されていました。

API 再試行を処理する

API リクエストが再試行可能なエラーで失敗すると、Claude Code は再試行する前に `system/api_retry` イベントを発行します。v2.1.246 以降では、`401` または `403` が `apiKeyHelper` 認証情報を拒否する場合、Claude Code は最初の 2 回の再試行を静かに行い、イベントなしで実行してから、3 回目の連続再試行からイベントを通常通り発行します。静かな再試行は依然として `attempt` にカウントされます。イベントを使用して、独自のインターフェースで再試行の進行状況を表示できます。

フィールド	型	説明
`type`	`"system"`	メッセージタイプ
`subtype`	`"api_retry"`	これを再試行イベントとして識別
`attempt`	integer	現在の試行番号（1 から開始）
`max_retries`	integer	このエラーの原因に許可される再試行の合計（セッション全体の予算より少ない場合がある）
`retry_delay_ms`	integer	次の試行までのミリ秒
`error_status`	integer または null	失敗した試行の HTTP ステータスコード、または試行が API から HTTP レスポンスを受け取らなかった場合は `null`

次のページに続く

表 2 – 前のページからの続き

フィールド	型	説明
no_response	object、optional	失敗した試行が no response headers in time を受け取った場合にのみ存在します。waited_ms はその試行が待機した時間で、retry_wait_ms は再試行が待機する時間です。これらのイベントでは、max_retries はセッション全体の予算ではなく、この原因が通常受け取る 1 回の再試行を反映しています。Claude Code v2.1.261 以降が必要です
error	string	エラーカテゴリ：authentication_failed、oauth_org_not_allowed、account_on_hold、billing_error、rate_limit、overloaded、invalid_request、model_not_found、server_error、max_output_tokens、cloud_credential_error、または unknown
uuid	string	一意のイベント識別子
session_id	string	イベントが属するセッション

セッションメタデータを読む

system/init イベントは、モデル、ツール、MCP サーバー、読み込まれたプラグインを含むセッションメタデータを報告します。スタートアップイベントが先行しない限り、ストリームの最初のイベントです。

- plugin_install イベント（CLAUDE_CODE_SYNC_PLUGIN_INSTALL が設定されている場合）。
- hook_started、hook_progress、および hook_response イベント（設定された *SessionStart* (ページ 481) または *Setup* (ページ 485) hook が実行されている間）。これらは hook が生成するときにストリーミングされます。Claude Code v2.1.169 から v2.1.203 はそれらを hook 完了後に 1 つのバッチで配信し、依然として system/init より前でした。v2.1.204 はライブ配信を復元しました。

イベントは、このバージョンの Claude Code が実装するプロトコル動作を命名する文字列の optional capabilities 配列も含みます（例：interrupt_receipt_v1 または interrupt_cancel_queued_v1）。バージョン文字列を比較する代わりに、これを使用して機能を検出し、認識しない値は無視してください。フィールドは Claude Code v2.1.205 以降が必要で、以前のバージョンでは存在しません。機能リストについては SDKSystemMessage を参照してください。

プラグインまたは MCP サーバーが読み込まれない場合に CI を失敗させる

system/init イベントのプラグインフィールドを使用して、読み込まれなかったプラグインをキャッチします。

フィールド	型	説明
plugins	array	正常に読み込まれたプラグイン（各々 name と path を含む）

次のページに続く

表 3 – 前のページからの続き

フィールド	型	説明
plugin_errors	array	プラグイン読み込み時エラー（各々 plugin、type、message を含む）。満たされていない依存関係バージョンと --plugin-dir 読み込み失敗（パスの欠落やアーカイブが無効など）を含みます。影響を受けたプラグインは plugins から削除されます。エラーがない場合、キーは省略されます

--plugin-dir ディレクトリまたはアーカイブ自体が読み込みに失敗した場合、その plugin_errors エントリは解決された絶対パスを path として含みます。複数の --plugin-dir 値のどれが失敗したかを判断するために使用します。path フィールドは Claude Code v2.1.283 以降が必要です。

MCP サーバーフィールドも同じ方法で使用します。 -p で *--mcp-config* (ページ 51) を渡す場合、Claude Code は最初のターンを実行する前に、まだ保留中のサーバーを待機します（MCP_TIMEOUT スタートアップタイムアウト（デフォルト 30 秒）まで）。cached tool list を持つリモートサーバーは待機をスキップし、system/init に pending を表示し、最初のツール呼び出しで接続します。待機には Claude Code v2.1.221 以降が必要です。

Claude Code は起動時に各 --mcp-config エントリを検証し、検証に失敗したエントリをスキップします（例えば、type のない url エントリ）。実行は続行され、クリーンに終了するため、これらのフィールドをチェックして、読み込まれなかったサーバーをキャッチします。

フィールド	型	説明
mcp_servers	array	セッション内の MCP サーバー（各々 name と status を含む）
mcp_server_errors	array	設定検証によってスキップされた --mcp-config エントリ（各々 name、type、message を含む）。type はスキップカテゴリ（unknown_type、url_missing_type、invalid_config、reserved_name など）です。認識しない値は汎用スキップとして扱ってください。影響を受けたサーバーは mcp_servers から削除されます。エラーがない場合、キーは省略されるため、CI ゲートは空でない配列で失敗できます。Claude Code v2.1.219 以降が必要です

コマンドを手でターミナルで実行する場合、Claude Code は stderr にスタートアップ警告も出力します（例：Warning: 1 MCP server skipped due to invalid config:）。その後に各スキップエントリの理由が続きます。stderr をリダイレクトする場合、または CI ランナーなどのプログラムがそれをキャプチャする場合、Claude Code は警告を出力せず、スキップされたエントリを mcp_server_errors フィールドでのみ報告します。警告には Claude Code v2.1.219 以降が必要です。

プラグインインストールを追跡する

CLAUDE_CODE_SYNC_PLUGIN_INSTALL が設定されている場合、Claude Code は最初のターンの前にマーケットプレイスプラグインがインストールされている間、system/plugin_install イベントを発行します。これらを使用して、独自の UI にインストール進行状況を表示します。

フィールド	型	説明
type	"system"	メッセージタイプ
subtype	"plugin_install"	これをプラグインインストールイベントとして識別
status	"started"、"installed"、"failed"、または "completed"	started と completed は全体的なインストールをブラケットします。installed と failed は個別のマーケットプレイスを報告します
name	string、optional	マーケットプレイス名（installed と failed に存在）
error	string、optional	失敗メッセージ（failed に存在）
uuid	string	一意のイベント識別子
session_id	string	イベントが属するセッション

26.2.5 ツールを自動承認する

--allowedTools を使用して、Claude が特定のツールをプロンプトなしで使用できるようにします。Read と Edit をリストすると、Claude はファイルを読み書きできます。Bash をリストすると、シェルコマンドについても同じことができます。ただし、auto mode で開始される実行では、Claude Code は広いアロールールとして裸の Bash エントリをドロップし、auto mode が代わりに各コマンドを評価します。この例はテストスイートを実行して失敗を修正し、これら 3 つのツールをリストします。

```
claude -p "Run the test suite and fix any failures" \
  --allowedTools "Bash,Read,Edit"
```

個別のツールをリストする代わりにセッション全体のベースラインを設定するには、permission mode を渡します。権限モードを設定しない実行は、built-in starting permission mode を取得します。これは auto の場合があるため、必要な権限モードを渡します。

- auto：--permission-mode auto を渡して、ほとんどのアクションをあなたの代わりに分類器にレビューさせます
- dontAsk：Claude Code はそれ以外の場合はプロンプトするすべての呼び出しを拒否します。これはロックダウンされた CI 実行に役立ちます。Manual モードで承認が不要なアクション（作業ディレクトリでのファイル読み取りや *read-only command set* (ページ 1049)）は依然として実行され、--allowedTools エントリまたは permissions.allow ルールがカバーするアクションも実行されます。AskUserQuestion、connector tools *your organization set to ask* (ページ 399)、および *requiresUserInteraction* (ページ 404) とマークされた MCP ツールは、許可ルールが一致する場合でも拒否されます
- acceptEdits：Claude はプロンプトなしでファイルを書き込み、Claude Code は mkdir、touch、mv、

cp などの一般的なファイルシステムコマンドを自動承認します。actions no mode auto-approves は依然として適用されます。read-only command set を除き、他のシェルコマンドとネットワークリクエストは依然として --allowedTools エントリまたは permissions.allow ルールが必要です。what acceptEdits auto-approves を参照して、完全なリストを確認してください

この例は acceptEdits をベースラインとしてリント修正を適用します。

```
claude -p "Apply the lint fixes" --permission-mode acceptEdits
```

26.2.6 無人実行で権限プロンプトをオフにする

誰も権限プロンプトに答えられない場合（例えば、スケジュール済みジョブ）、--permission-prompts none を渡します。フラグは、実行に権限ホストがある場合に最も重要です。Agent SDK アプリ（canUseTool callback を含む）、または *--permission-prompt-tool* (ページ 51) で渡す MCP ツール。フラグなしでは、実行は各権限リクエストに対してそのホストを待機します。

フラグを使用すると、実行はホストを参照せず、それを待機しません。プロンプトするすべてのものは、PermissionRequest hook が許可しない限り拒否され、Claude には誰も要求を承認できず、再試行しないことが伝えられ、実行は続行されます。ホストのない -p 実行では、これらのリクエストはいずれにせよ拒否され、フラグは Claude に再試行しないことも伝えます。権限ルール、*PermissionRequest hooks* (ページ 506)、および設定した権限モードは依然としてすべての呼び出しを最初に決定します。Claude Code は、他に何も解決しないリクエストのみを拒否します。

この例は auto mode で無人タスクを実行します。分類器は通常通り各アクションをレビューし、Claude Code はプロンプトにフォールバックしたであろうすべてのものを拒否します。

```
claude -p "Update the dependency pins and run the tests" --permission-mode auto --
↪permission-prompts none
```

--permission-prompts none を使用すると、Claude Code は AskUserQuestion など、人からの答えが必要なツールを削除するため、Claude はそれらを呼び出すことができません。*Elicitation hook* (ページ 549) が答えない *MCP elicitation request* (ページ 405) はキャンセルされます。

--output-format stream-json を使用すると、拒否は permission_denied システムメッセージとして表示され、最終結果メッセージは permission_denials にそれらをリストします。

注釈:

--permission-prompts フラグには Claude Code v2.1.259 以降が必要です。以前のバージョンは不明なオプションエラーで拒否します。

26.2.7 コミットを作成する

この例はステージされた変更をレビューして、適切なメッセージでコミットを作成します。

```
claude -p "Look at my staged changes and create an appropriate commit" \
  --allowedTools "Bash(git diff *),Bash(git log *),Bash(git status *),Bash(git commit␣
↪*)"
```

--allowedTools フラグは permission rule syntax を使用します。末尾の * はプレフィックスマッチングを有効にするため、Bash(git diff *) は git diff で始まるすべてのコマンドを許可します。スペースは * の前に重要です。なければ、Bash(git diff*) は git diff-index も一致させます。

注釈:

コマンドサポートは -p モードで異なります。

- ユーザーが呼び出した *skills* (ページ 275) とカスタムコマンドは機能します。プロンプト文字列に /skill-name を含めると、Claude Code は実行前にそれを展開します。
- ターミナルインターフェースでのみ実行される /login などの組み込みコマンドは利用できません。
-

/model、/effort、/fast、/color、/rename は値を引数として受け入れます（例：/model sonnet）。/mcp は引数なしでサーバーステータスのテキスト概要を出力します。これらのフォームには Claude Code v2.1.205 以降が必要で、各コマンドの availability notes に従います。

- 設定を変更するには、/config に key=value を渡します（例：/config thinking=false）。
-

/output-style <style> は *output styles* (ページ 821) を切り替え、/output-style のみはそれらをリストします。Claude Code v2.1.269 以降が必要です。

26.2.8 システムプロンプトをカスタマイズする

--append-system-prompt を使用して、Claude Code のデフォルト動作を保持しながら指示を追加します。この例は PR diff を Claude にパイプして、セキュリティ脆弱性をレビューするよう指示します。シェルスクリプトとして保存します（例：review.sh）。

```
gh pr diff "$1" | claude -p \
  --append-system-prompt "You are a security engineer. Review for vulnerabilities." \
  --output-format json
```

スクリプトでは、"$1" はコマンドラインで渡す最初の引数を表します。bash review.sh 123 を実行すると、シェルは "$1" を 123 に置き換えるため、スクリプトは PR 123 の diff をフェッチします。Claude Code はレビューを JSON として出力し、テキストは result フィールドにあります。

詳細については、*system prompt flags* (ページ 72) を参照してください。--system-prompt を使用してデフォルトプロンプトを完全に置き換えるオプションも含まれています。

26.2.9 会話を続ける

`--continue` を使用して最新の会話を続けるか、`--resume` をセッション ID と共に使用して特定の会話を続けます。 Claude Code v2.1.257 以降では、`--continue` を渡すと、Claude Code は完了した background session を開きますが、まだ実行中のセッションは開きません。この例はレビューを実行してから、フォローアッププロンプトを送信します。

```
# 最初のリクエスト
claude -p "Review this codebase for performance issues"

# 最新の会話を続ける
claude -p "Now focus on the database queries" --continue
claude -p "Generate a summary of all issues found" --continue
```

複数の会話を実行している場合は、セッション ID をキャプチャして特定のセッションを再開します。

```
session_id=$(claude -p "Start a review" --output-format json | jq -r '.session_id')
claude -p "Continue that review" --resume "$session_id"
```

2 つのコマンドを異なるディレクトリから実行できます。Claude Code は finds the session by its ID をこのマシン上の任意のプロジェクトで実行します。v2.1.223 より前では、Claude Code は現在のプロジェクトディレクトリとその git worktrees でのみ ID を探していたため、両方のコマンドを同じディレクトリから実行する必要がありました。

セッション ID の代わりに、`--resume` にセッションの `.jsonl` transcript file への絶対パスを渡すことができます。Claude Code はそのファイルに保存されている会話を続けます。

26.3 次のステップ

- Agent SDK クイックスタート：Python または TypeScript で最初のエージェントを構築します
- *CLI* **リファレンス** (ページ 45)：すべての CLI フラグとオプション
- *GitHub Actions* (ページ 247)：GitHub ワークフローで Agent SDK を使用します
- *GitLab CI/CD* (ページ 261)：GitLab パイプラインで Agent SDK を使用します

第V部

設定

第 27 章　設定ファイルと優先順位

Claude Code の設定を変更し、キーが属するスコープを選択し、変更を確認し、複数の場所でキーが設定されている場合に Claude Code が使用する値を学びます。

設定は、Claude Code の動作を変更する JSON キーです。どのモデルで開始するか、何を確認なしで実行できるか、どのファイルを読み取れないか、ターミナルでどのように見えるか、および組織が強制する内容を決定します。

Tip:

特定のキーを検索するには、すべての設定に移動してください。すべてのキーが、設定するファイル、デフォルト、および例とともにリストされています。

Claude Code は `~/.claude/settings.json` などの JSON 設定ファイルから設定を読み込みます。いくつかの場所でそれらを探し、**設定を読み込むファイルが、その設定が誰に適用されるかを決定します** (ページ 711)。このページでは、これらのファイルについて説明します。設定をどのファイルに入れるか、設定を変更して適用されたことを確認する方法、および同じキーが複数のファイルで設定されている場合に Claude Code が使用する値について説明します。**権限を構成する** (ページ 1039) では、Claude Code が確認なしで実行できる内容と、`allow`、`ask`、および `deny` ルールを記述する方法について説明します。

注釈:

このページでは、マシンで実行されている Claude Code について説明します。ターミナル、*VS Code* (ページ 173) および *JetBrains* (ページ 205) 拡張機能、および **デスクトップアプリ** (ページ 137) は、すべて同じ設定ファイルを読み込みます。*Claude Code on the web* (ページ 223) のクラウドセッションは別のマシンで実行され、そのうちのいくつかのみを読み込みます。**クラウドセッションの設定** (ページ 724) を参照してください。

27.1 設定ファイルと誰に影響するか

Claude Code は 4 つのファイルから設定を読み込み、組織は claude.ai コンソールから管理設定を配信することもできます。各ソースにはスコープがあります。スコープは、その中に保存された設定が適用される人とプロジェクトのセット。それがあなただけ、プロジェクト内のすべての人、またはあなたの組織内のすべての人であるかどうか。

スコープ	ファイル	誰に影響するか	用途
ユーザー	`~/.claude/settings.json`	あなた、このマシン上のすべてのプロジェクト	個人設定：テーマ、エディターモード、デフォルトモデル、独自の権限ルール
共有プロジェクト	`.claude/settings.json`	このフォルダを含むフォルダで作業しているすべての人。git リポジトリでは、コミットしてチームメイトが取得できるようにします	チーム権限、hooks、プラグイン、およびプロジェクトが必要とする環境変数
プロジェクトローカル	`.claude/settings.local.json`	あなた、このプロジェクトのみ。Claude Code はファイルを作成するときに git から除外します。手動で作成する場合は、自分で `.gitignore` に追加してください	1 つのプロジェクトの個人的なオーバーライド、および共有する前のテスト
管理	`managed-settings.json` およびその他の管理ソース	組織がデプロイするすべての人。あなたが設定するものは何もそれをオーバーライドしません。いくつかの**セキュリティに敏感な例外** (ページ 722) を除いて	セキュリティポリシーおよびコンプライアンス要件

ファイル列では、`~/.claude` はホームディレクトリの `.claude` フォルダ、ベアの `.claude` はプロジェクト内の `.claude` フォルダです。

27.1.1 各設定ファイルのスコープを比較する

マシン上に 3 つのプロジェクト `website/`、`api/`、および `acme-app/` があり、チームメイトが `acme-app/` の独自のクローンを持ち、`acme-app/` で**クラウドセッション** (ページ 724) を開始するとします。

下のグラフィックは、それらのフォルダから Claude Code を開始するときに設定が適用されるフォルダを示しています。設定ファイルをクリックして、到達するフォルダを確認してください。

注釈:

公式サイトでは、ここに対話型の図があります。本書では省略しています。

- `~/.claude/settings.json`：マシン上のすべてのプロジェクト、チームメイトまたはクラウドセッションには何もありません
- `acme-app/.claude/settings.json`：あなたの `acme-app/`。バージョン管理にファイルをコミットする場合のみ、チームメイトのクローンとクラウドセッションに到達します。それまでは、他のファイルのようにディスク上のファイルであり、誰も持っていません
- `acme-app/.claude/settings.local.json`：あなたの `acme-app/` のみ。Claude Code はファイルを

初めて書き込むときに、グローバル git 除外に追加するため、コミットから除外されます。手動でファイルを作成する場合は、**自分で *.gitignore* に追加してください** (ページ 714)

- **管理設定**。`managed-settings.json` ファイル、MDM ポリシー、または claude.ai コンソールからのサーバー管理設定：組織がデプロイするすべてのマシン上のすべてのプロジェクト、またはあなたの組織アカウントでサインインするマシン。サーバー管理設定のみがクラウドセッションに到達します

27.1.2 設定ファイルを見つけるか作成する

Claude Code をインストールしても、設定ファイルは作成されません。マシンまたはプロジェクトに既に 1 つある場合は、これらのソースの 1 つから来ました：

- **管理**：組織がデプロイします。作成または編集しません。
- **共有プロジェクト**：Claude Code を既に使用しているプロジェクトにはコミットされたものがあるかもしれません。ない場合は、プロジェクトフォルダに `.claude/settings.json` で作成してください。
- **ユーザー**および**プロジェクトローカル**：自分で作成するか、Claude Code に作成させます。テーマなどのユーザー設定に保存する `/config` メニューのオプションを初めて変更するときに `~/.claude/settings.json` を書き込み、Bash コマンドに対して「はい、今後は聞かないでください」などの権限プロンプトで立ったままの承認を初めて与えるときに `.claude/settings.local.json` を書き込みます。**ヒントを表示**を含むいくつかの `/config` オプションは、ユーザーファイルの代わりに `.claude/settings.local.json` に保存されます。

注釈:

Windows では、`~/.claude` は `%USERPROFILE%\.claude` を意味します。ホームディレクトリファイルを別の場所に保つには、`CLAUDE_CONFIG_DIR` を設定してください。Claude Code はその代わりに設定、セッション履歴、およびプラグインをそこに保存します。

Claude Code は、`~/.claude.json` という 5 番目のファイルも保持します。これは自分で書き込みます。編集する必要はありません。サインインセッション、*MCP* **サーバー** (ページ 363) 構成、信頼決定などのプロジェクトごとの状態、および `/config` があなたのために書き込むグローバル構成キーを保持します。

27.1.3 チームと設定を共有する

`.claude/settings.json` をコミットして、リポジトリをクローンするすべての人が同じ権限、hooks、およびプラグインを取得するようにします。各チームメイトは、個人的な例外がコミットを必要としないように、独自の `.claude/settings.local.json` でそれをオーバーライドできます。完全なチームファイルについては、チームの共有設定を参照してください。

コミットするものの一部は、各チームメイトが**フォルダを信頼する** (ページ 1064) まで待機し、いくつかのキーはリポジトリファイルから効果を発揮しません。**適用されない設定をトラブルシューティングする** (ページ 721) は両方をカバーしています。

27.1.4 リポジトリから個人設定を除外する

プロジェクト内で自分の設定を変更し、チームメイトの設定を変更しないようにするには、プロジェクト内の `.claude/settings.local.json` に保存してください。Claude Code はそのファイルをコミットされた `.claude/settings.json` に適用するため、チームのファイルが `"model": "claude-sonnet-5"` を設定し、Opus が必要な場合は、ローカルファイルに `"model": "claude-opus-5-5"` を入れて、セッションのみを変更します。

Claude Code もこのファイルに書き込み、コミットから除外し、信頼ステップなしに allow ルールを適用します：

- **Claude Code もそれを書き込みます。** Claude が Bash コマンドを実行する権限を求め、「はい、今後は聞かないでください」を選択すると、Claude Code はその**権限承認** (ページ 1039) をここに `allow` ルールとして保存します。
- **手動で作成した場合を除き、自分で gitignore する必要はありません。** Claude Code がリポジトリでファイルを初めて書き込むときに、既にそれを無視していない場合、グローバル git 除外ファイルに `**/.claude/settings.local.json` を追加するため、ファイルはすべてのリポジトリのコミットから除外されます。そのファイルは、グローバル git 構成がそれを絶対パスまたは ~ プレフィックス付きパスに設定する場合は `core.excludesFile`。それ以外の場合は `$XDG_CONFIG_HOME/git/ignore`、または `XDG_CONFIG_HOME` が設定されていない場合は `~/.config/git/ignore`。手動でファイルを作成し、Claude Code がまだそれに書き込んでいない場合は、自分で `.gitignore` に追加してください。
- **ファイルが追跡されていない間、その allow ルールは信頼を待ちません。** ファイルはリポジトリのものではなくあなたのものであるため、Claude Code はコミットされたファイルが必要とする**ワークスペース信頼** (ページ 1064) ステップなしにその `allow` ルールを適用します。ファイルが git で追跡されている場合、信頼ステップもそれに適用されます。**ローカル設定ファイルが信頼を必要とする場合** (ページ 1065) を参照してください。

Claude Code が git リポジトリでローカルファイルを保持する場所

Claude が Bash コマンドを実行する権限を求め、「はい、今後は聞かないでください」を選択すると、Claude Code はその承認を `.claude/settings.local.json` の `allow` ルールとして保存します。git リポジトリのサブディレクトリから Claude Code を開始する場合、リポジトリルートでそのファイルを読み取り、書き込み、リポジトリ全体に承認を適用します。worktree では、メインチェックアウトのルートのファイルを使用します。

2 つのルールがルートの場所を適格にします：

- **ファイルが `.claude/settings.json` の代わりに留まる場合**：git リポジトリの外、リポジトリルートがホームディレクトリ、Windows、またはリポジトリルート、その `.git` またはその `.claude` エントリがユーザーによって所有されていない場合。
- **ファイル内のパスはリポジトリルートに固定されません**：/ で始まる権限ルール、または相対サンドボックスパスは、**セッションのプライマリ作業ディレクトリ** (ページ 1051) に固定されます。

v2.1.211 より前では、Claude Code は開始ディレクトリにファイルを保持していました。以前のバージョンが残したファイルをルートファイルと並行して読み込みます。両方が同じキーを設定する場合、ルートの値が適用され、両方のファイルからの権限ルールが適用されます。Agent SDK の `resolveSettings()` ヘルパーは常に開始ディレクトリからファイルを読み込みます。

Claude Code は共有 `.claude/settings.json` をセッションの**プライマリ作業ディレクトリ** (ページ 1060) から読み込むため、リポジトリルートにコミットされたファイルを使用するには、そこから Claude Code を開始してください。*/cd* (ページ 1061) でセッションを移動した後、Claude Code は代わりに新しいディレクトリから両方のプロジェクトファイルを読み込み、同じルールでローカルファイルを配置します。移動したディレクトリから読み込むには Claude Code v2.1.246 以降が必要です。

27.1.5 組織が強制する内容を確認する

組織が Claude Code を管理する場合、いくつかの設定はあなたのために決定され、独自のファイルに入れるものは何もそれらを変更しません。どれを確認するには、`/status` を実行してください。`Setting sources` 行は、あなたに適用される管理ソースの名前を付けます。管理設定はこのマシンで Claude Code が実行される場所に到達します。開発者が変更できる内容はローカル管理者権限と Claude Code 以外のツールをカバーしています。

管理設定は管理設定ページの配信メカニズムを通じてあなたに到達します。最も一般的には：

- サーバー管理設定。Claude Code が claude.ai 管理コンソールまたは自己ホスト型 Claude apps gateway から取得します
- MDM または OS レベルのポリシー、およびシステムディレクトリの `managed-settings.json` ファイル
- Claude Desktop などの埋め込みホスト。SDK `managedSettings` オプション経由。埋め込みホストからポリシーを制御するを参照してください

Claude Desktop アプリで実行される Cowork セッションでは、Claude Code は claude.ai 管理コンソールからサーバー管理設定を取得しません。組織の Claude Desktop 構成が `requireCoworkFullVmSandbox` を設定しない限り、デバイスにデプロイされたポリシーを読み込みます。ポリシーが適用される場所と時期は Cowork とクラウドセッションをカバーしています。

管理者の場合、組織向けに Claude Code をセットアップするは何を強制するかを選択する手順を説明し、管理設定をデプロイするは配信と、ポリシーが有効であることを確認する方法をカバーしています。

27.2 設定を変更する

`/config` メニューから、設定ファイルを編集して、または 1 つのセッションのコマンドラインから設定を変更できます。

Claude Code のシステムプロンプトは公開されていません。Claude に立ったままの命令を与えるには、*CLAUDE.md* **ファイル** (ページ 997) または `--append-system-prompt` フラグを使用してください。

27.2.1 /config メニューを使用する

Claude Code 内で `/config` を実行し、**Config** タブを開きます。テーマ、エディターモード、詳細出力などの短いセットの個人的なオプションをリストします。すべての設定キーではありません。オプションを選択して変更します。Claude Code はあなたのために保存します：

- **ほとんどのオプション**：`~/.claude/settings.json`
- **ヒントを表示などのいくつかのオプション**：`.claude/settings.local.json`

- グローバル構成オプション：~/.claude.json

1 つのオプションをメニューなしで設定するには、/config verbose=true などの key=value を渡してください。

注釈:

/config はターミナルインターフェースの一部です。*VS Code* (ページ 173) チャットパネルと**デスクトップアプリ** (ページ 137) はそれを開きません。設定ファイルを編集するか、これらのアプリ独自の設定を通じて設定を変更してください。

27.2.2 設定ファイルを編集する

エディターで必要なスコープの設定ファイルを開き、キーを追加または変更します。設定ファイルは厳密な JSON です。// コメントまたは末尾のコンマは構文エラーであり、Claude Code はファイルを**設定エラー** (ページ 718) として次の開始時に報告します。たとえば、Claude Code が確認なしに lint とテストコマンドを実行し、.env ファイルの読み取りを停止するようにするには、これを ~/.claude/settings.json に追加してください：

リスト 1: ~/.claude/settings.json

```
{
  "$schema": "https://json.schemastore.org/claude-code-settings.json",
  "permissions": {
    "allow": [
      "Bash(npm run lint)",
      "Bash(npm run test *)"
    ],
    "deny": [
      "Read(./.env)",
      "Read(./.env.*)"
    ]
  }
}
```

permissions の下の各エントリは、ツールとそれが何ができるかを名前付けするルールです。**権限を構成する** (ページ 1039) は構文を説明します。$schema 行は Claude Code 設定の公開 JSON スキーマを指し、VS Code、Cursor、および JSON スキーマをサポートする他のエディターでオートコンプリートとインライン検証を提供します。スキーマは最新の CLI リリースより遅れることができるため、最近ドキュメント化されたキーの検証警告は、構成が無効であることを意味しません。

保存した後、Claude Code 内で /status を実行してファイルが読み込まれたことを確認してください。**読み込まれたものを確認する** (ページ 718) は Setting sources 行が表示するものと、壊れたファイルがどのように報告されるかを説明します。

完全な個人ファイル、チームファイル、および組織ファイルについては、それが設定するすべてのキーのコメント付きで、設定ファイルの例を参照してください。

27.2.3 1 つのセッションの設定を変更する

値を保存せずに試すには、Claude Code を開始するときに設定します。値はそのセッションに適用され、設定ファイルはそのままです。3 つの方法があります：

- `--settings`：JSON としてキーを渡します。インラインまたはファイルへのパス。Claude Code はそれをユーザー、プロジェクト、およびローカルファイルの上に適用し、管理設定の下に適用します。ユーザー設定ファイルが設定できるキーを設定できます。`Managed` または `Global config` キーを設定することはできません。
- **そのキーのフラグ**：いくつかのキーには独自のフラグがあります。`model` の `--model` や `effortLevel` と `modelSettings` の `--effort` など。
- **環境変数**：`model` の `ANTHROPIC_MODEL` など、キーのペアリングされた変数をエクスポートしてから `claude` を実行してください。

各キーの設定リファレンスのエントリは、その 1 セッションのオーバーライドと、どれが優先されるかをリストします。変更したいキーのエントリを確認してください。

セッション内で実行するコマンドはほとんどあなたの選択を保存します。`/config` で設定を変更すると、Claude Code はそれを設定ファイルに書き込み、`/model` は値を新しいセッションのデフォルトとして保存します。

`/model` ピッカーで `s` を押すと、Claude Code はモデルを切り替えますが、ユーザーデフォルトとして保存しません。**努力レベルを調整する** (ページ 747) は、Claude Code が使用しているモデルのデフォルトとして保存する `/effort` ピックと、現在のセッションのみに適用するものを説明します。

たとえば、デフォルトを変更せずに 1 つのセッションで Opus で開始するには：

```
claude --settings '{"model": "claude-opus-5-5"}'
```

27.2.4 編集がいつ有効になるか

Claude Code は設定ファイルを監視し、変更時に再読み込みするため、`permissions`、`hooks`、および `apiKeyHelper` などの認証情報ヘルパーへの編集を含む、ほとんどの編集は再起動なしで実行中のセッションに適用されます。Claude Code は、セッションの途中で作成した設定ファイルも、そのフォルダがセッション開始時に存在していれば読み込みます。プロジェクトの `.claude/` フォルダについては、同じセッション内でフォルダを作成した場合でもファイルを読み込みます。

再読み込みはユーザー、プロジェクト、ローカル、および管理設定をカバーし、Claude Code は検出した各設定ファイル変更に対して *ConfigChange hook* (ページ 531) を実行します。MDM または claude.ai コンソールから到達する管理設定ではありません。MDM または claude.ai コンソールから到達する管理設定は、保存時ではなくスケジュールに従って実行中のセッションに到達します。配信テーブルはソースごとにそれを提供します。

Claude Code はセッション開始時に一度だけいくつかのキーを読み込むため、そのうちの 1 つへの編集は実行中のセッションに到達しません。`requiredMinimumVersion` などの管理者側のキーも再起動を待ちます。ポリシーが適用される場所と時期の下にリストされています。セッション中に編集する可能性が最も高いもの：

- `model`：セッション中に切り替えるには */model* (ページ 728) を使用してください。各モデルには独自のプロンプトキャッシュがあるため、切り替え後の最初のリクエストは会話全体をキャッシュなしで再読み込みします。モデルを切り替えるを参照してください
- `effortLevel` および `modelSettings`：セッション中に努力を変更するには */effort* (ページ 747) を使用してください

27.2.5 読み込まれたものを確認する

Claude Code 内で `/status` を実行して、どの設定ソースがアクティブであるかを確認してください。**Status** タブには、Claude Code が現在のセッション用に読み込んだ各設定ファイルをリストする `Setting sources` 行が含まれています。`User settings` または `Project local settings` など。管理設定が有効な場合、管理設定エントリは括弧内にそれらがマシンに到達した方法を示します。

行は Claude Code が読み込んだファイルを確認します。各ファイルがどのキーを供給したかは表示されません。Claude Code が拒否したエントリをリストするには、`claude doctor` を実行してください。プロジェクトまたは管理設定が設定するモデルについては、スタートアップヘッダーがそれを設定したファイルの名前を付けます。`/status` と `/config` は異なるタブで同じダイアログを開き、**Config** タブは `settings.json` コンテンツのビューではありません。

27.2.6 壊れた設定ファイルを修正する

JSON をタイプミスするか、Claude Code が受け入れない値にキーを設定する場合、Claude Code はインタラクティブセッションの開始時にあなたに伝えます。表示される内容は、ファイルのどの程度が影響を受けるかによって異なります：

- **設定エラー**：ユーザー、プロジェクト、またはローカルファイルに無効な JSON またはスキーマが拒否する値があります。インタラクティブセッションの開始時に Claude Code はダイアログを表示し、Claude の助けでファイルを修正するか、終了するか、壊れた設定なしで続行できます。
- **設定警告**：個別のエントリのみが失敗します。不正な形式の権限ルールまたは不明な hook イベント名など。Claude Code はそれらの値をスキップし、ファイルの残りを有効に保ちます。
- **管理設定**：Claude Code はファイルの残りを強制し続けます。管理設定の無効なエントリは、それが削除するものと、修正するまでどのキーがより厳密な値にフォールバックするかを説明します。有効な JSON ではない管理設定ドキュメントについては、管理設定ドキュメントを解析できませんでしたを参照してください。
- **構成エラー**：`~/.claude.json` を解析できません。Claude Code は壊れたファイルを `~/.claude/backups/.claude.json.corrupted.<timestamp>` にコピーし、終了して手動で修正するか、デフォルト構成にリセットするかを尋ねます。`-p` 実行はエラーを出力して終了します。以前の状態を復旧するには、`~/.claude/backups/` の 5 つの最新 `.claude.json.backup.<timestamp>` ファイルの 1 つをコピーして戻してください。Claude Code はファイルを書き込む前に保存します。

続行した後、`/status` を実行して影響を受けたファイルを確認し、各エラーの詳細について `claude doctor` を実行してください。

`-p` 実行はダイアログを表示しません。管理設定ドキュメントを解析できない場合を除き、Claude Code は壊れたファイルまたは値をスキップして残りで続行するため、設定をスキップした `-p` 実行の後、`claude doctor` を実行して削除したものを確認してください。

27.3 設定の優先順位

同じキーが複数の場所に表示される場合、Claude Code はそれを設定する最も高いレベルの値を使用します。以下のスタックはレベルを示しており、上が最も高い優先度です。より高いレベルのキーは、その下のどこにある同じキーよりも優先されます。

注釈:

公式サイトでは、ここに対話型の図があります。本書では省略しています。

優先度が高い順に：

1. **管理設定**：組織が `managed-settings.json` ファイル、MDM ポリシー、または claude.ai コンソールからのサーバー管理設定 によってデプロイする設定です。設定したものは何もこれをオーバーライドしません。`--settings` で渡すキーは同じ管理キーをオーバーライドしませんし、`--model` などのフラグは組織が許可するモデルからのみ選択します。管理 `model` は各セッションが開始するモデルを設定し、`/model` で切り替えることはできます。ロックは `availableModels` で、これは `/model`、`--model`、および独自ファイルの `model` キーを制限します。組織が複数の管理ソースを提供する場合、管理層内の優先順位 のルールが Claude Code が各ソースから読み込むものを指定します。

2. **コマンドライン引数**：ターミナルから `claude` を開始するときに渡すフラグで、1 つのセッション用です。*1* **つのセッションの設定を変更する** (ページ 717) を参照してください。Claude Code は `--settings <file-or-json>` で渡す JSON を他のレベルと同じルールで設定ファイルとマージします。ここで設定したキーはローカル、プロジェクト、またはユーザー設定の同じキーより優先され、省略したキーについては下位レベルの値を保持します。

3. **プロジェクトローカル設定** （`.claude/settings.local.json`）：このプロジェクトの個人設定です。

4. **共有プロジェクト設定** （`.claude/settings.json`）：チームがソース管理にチェックインする設定です。

5. **ユーザー設定** （`~/.claude/settings.json`）：すべてのプロジェクトの個人設定です。

環境変数はこのスタックのレベルではありません。動作がシェル変数と設定キーの両方を持つ場合、どちらが適用されるかはレベルではなくペアごとに決定されます。シェルでエクスポートされた `ANTHROPIC_MODEL` はどのファイルからの `model` キーよりも優先されますが、`ANTHROPIC_DEFAULT_MODEL` はファイルが `model` を設定していない場合にのみ適用されます。環境変数リファレンス はどのキーがペアを持ち、Claude Code がどちらを最初に読み込むかを示しています。設定ファイル内の `env` ブロックは通常のキーであり、上記のレベルに従います。

いくつかのセキュリティに敏感なキーについて、Claude Code は下位レベルからのより厳密な値を管理値より優先します。**管理設定の優先順位の例外** (ページ 722) がそれらをリストしています。

27.3.1 リストはオーバーライドではなくマージされます

`permissions.allow` などの同じリストキーを複数のファイルで設定する場合、Claude Code はリストを結合し、1 つを選択しません。そのため、各ファイルは別のファイルのエントリを削除することなくエントリを追加できます。モデルリストまたはモデルごとのエントリを保持する 4 つのキーは独自のルールに従います：

- `fallbackModel` は位置が意味を持つ順序付きチェーンであるため、Claude Code は最も優先度の高いファイルからの全体の値を取得します。
- `modelPicker` は 1 つの順序付きリストの行とリプレースフラグを保持するため、Claude Code は 2 つのソースから行をマージしません。管理設定、`--settings`、およびユーザー設定の最も高いものから全体の値を取得し、プロジェクトおよびローカル設定のキーを無視します。Claude Code v2.1.242 以降が必要です。
- `availableModels`：Claude Code が適用する管理設定がこれを定義する場合、Claude Code はそのリストをそのまま適用し、ユーザー、プロジェクト、またはローカル設定で追加するエントリを無視します。ただし、Claude Code を埋め込むアプリが独自のモデルリストを提供する場合を除きます。**管理設定の優先順位の例外** (ページ 722) を参照してください。管理ソース全体でリストはマージされません。Claude Code が管理ソースを結合する方法 はどのソースのリストが適用されるかを示しています。非管理スコープ全体で Claude Code は通常どおり配列をマージします。
- `modelSettings`：Claude Code はそれを `effortLevel` と一緒に一度に 1 つのモデルで解決します。`modelSettings` エントリはどのファイルの値がモデルに適用されるかを示しています。

27.3.2 優先順位の例

Claude が動作している間、Claude Code はスピナーの下に 1 行のヒントを表示します。例えば「/config を使用してデフォルト権限モード（Plan Mode を含む）を変更します」。`spinnerTipsEnabled` を `~/.claude/settings.json` で `false` に設定したいとします。以下の各シナリオはそれらをオンに戻す可能性があるもので、それについて何ができるかを示しています。

チーム設定が個人設定をオーバーライドします

チームの `.claude/settings.json` がそれを `true` に設定しています。Claude Code はプロジェクト値を使用します。共有プロジェクトはユーザーより上にあるため、そのプロジェクトではヒントが表示され、他の場所では表示されません。

値を取り戻すことができます。そのプロジェクトの `.claude/settings.local.json` に `"spinnerTipsEnabled": false` を追加します。プロジェクトローカルは共有プロジェクトより上にあるため、そこでのセッションはヒントを表示しなくなり、チームメイトのセッションは変わりません。

組織設定がすべてをオーバーライドします

組織の管理設定がそれを `true` に設定しています。ユーザー、プロジェクト、またはローカル設定に何を入れてもヒントをオフにすることはできませんし、`--settings` でもできません。管理は最上位レベルです。

値を取り戻すことはできません。`/status` を実行して、どの管理ソースが適用されるかを確認し、ポリシーを変更する必要があるかどうかを管理者に尋ねてください。

コマンドラインが 1 つのセッションのファイルをオーバーライドします

`claude --settings '{"spinnerTipsEnabled": true}'` でセッションを開始しました。コマンドラインは管理を除くすべてのファイルより上にあるため、そのセッションはファイルが `false` と言っていても、ヒントを表示します。

次のセッションで値を取り戻します。`--settings` は 1 つのセッション限りで、ファイルに書き込みません。

フラグまたは環境変数が同じものを設定します

一部のキーには、設定値をオーバーライドするコマンドラインフラグまたは環境変数があります。これはどのファイルが設定したかに関係なく機能します。`ANTHROPIC_MODEL` は `model` 設定をオーバーライドし、`--model` はセッション用に両方をオーバーライドします。

値を取り戻せるかどうかはキーによって異なります。変数をアンセットするか、フラグをドロップし、設定リファレンス のキーのエントリと 環境変数リファレンス の変数の行を確認して、Claude Code がどちらを使用するかを確認してください。

27.3.3 適用されない設定をトラブルシューティングします

キーを設定し、Claude Code がそのように動作しない場合は、`/status` で読み込まれたファイルを確認してから、以下で症状を見つけてください。設定をデバッグする はより広いチェックをカバーしており、クリーン設定テストを含みます。

設定した値が無視されます

別のものが同じキーを設定しているか、ファイルがその値を設定できないか、またはファイルが読み込まれませんでした：

- **より高いレベルがそれを設定しています。** 別の設定ファイル、`--settings` フラグ、または管理ソースがキーをあなたのより上に設定しています。**スタック** (ページ 719) はどちらを示しています。フラグまたは環境変数もキーごとに決定されて、キーをそれ自体でオーバーライドできます。設定リファレンス のキーのエントリはどちらを Claude Code が使用するかを示し、env エントリ は管理 env 値対シェルエクスポートをカバーしています。
- **セキュリティキーは厳密な値を保持しています。** いくつかのキーについて Claude Code は任意のファイルからの制限値を優先するため、プロジェクト `true` は `disableClaudeAiConnectors` のままです。**管理設定の優先順位の例外** (ページ 722) を参照してください。
- **ファイルはその値を設定できません。** `permissions.defaultMode` の値 `auto` および `bypassPermissions` はプロジェクトまたはローカル設定から有効になりません。代わりにユーザーまたは管理設定で設定するか、1 つのセッション用に `--permission-mode` を渡してください。v2.1.257 より前では、`bypassPermissions` はどのファイルからでも有効になりました。

  `env` ブロック内のテレメトリエクスポート変数もプロジェクトまたはローカル設定からは有効になりません。いくつかのオフ値を除きます。Claude Code が env で無視する変数 は変数とそれらの値をリストしています。
- **ファイルが壊れています。** 無効な JSON または拒否された値により、Claude Code はファイルまたはエントリをスキップします。**壊れた設定ファイルを修正する** (ページ 718) を参照してください。

Claude Code で行った変更が新しいセッションで失われます

Claude Code 内から新しいセッション用に選択を保存する場合（例えば `/model` でデフォルトモデルを設定する場合）、Claude Code はそれをユーザー設定ファイル `~/.claude/settings.json` に書き込みます。そのファイルに書き込むことができない場合（例えば別のツールが生成するか、読み取り専用コピーにリンクしている場合）、変更は現在のセッションに適用され、次のセッションでは失われます。ファイルを生成するツールでキーを設定するか、ファイルを書き込み可能なものに置き換えてください。

ファイルに書き込むことができ、変更がまだ続かない場合は、変更が *1* **つのセッション限り** (ページ 717) であったか、**より高いレベルが同じキーを設定** (ページ 721) しているかを確認してください。`model` キーについては、**新しいセッションが選択したものとは異なるモデルで開始します** (ページ 732) がより多くの原因をリストしています。

管理変更があなたに到達していません

管理ソースは 配信テーブル のスケジュールで実行中のセッションに到達するため、最初にセッションを再開してください。`/status` がその後、管理者が変更したものとは異なるソースを名前付けする場合、より優先度の高いソースが適用されます。Claude Code が管理ソースを結合する方法 は順序を示しています。

コミットされたキーがチームメイトに到達しません

2 つのことが `.claude/settings.json` のキーがそれをクローンするすべての人に適用されるのを防ぎます：

- **Claude Code はリポジトリファイルのキーを無視します。** 設定インデックス のスコープ列で `User, local, or managed`、`User or managed`、`Managed`、または `Global config` を探してください。これらのキーは共有ファイルから適用されません。ただし、リポジトリファイルがまだオフにできるいくつかを除きます。これらのエントリのそれぞれはスコープ行でそう言っています。`Global config` キーは `~/.claude.json` からのみ適用されます。

  `env` キー内では、テレメトリエクスポート変数も共有ファイルから適用されません。いくつかのオフ値を除きます。Claude Code が env で無視する変数 を参照してください。

- **キーは信頼を待っています。** `permissions.allow` ルール、`permissions.additionalDirectories`、`extraKnownMarketplaces`、およびほとんどの env 値は、各チームメイトが **フォルダを信頼** (ページ 1064) した後にのみ適用されます。それまで、彼らはプロンプトを見続け、ファイルが宣言するマーケットプレイスからプラグインを取得しません。`deny` および `ask` ルールはすぐに適用されます。

権限ルールが予想と異なる方法で結合されます

- **権限プロンプトで「はい、今後は聞かないでください」を選択しましたが、同じツールのプロンプトが表示されます。** その選択はローカルファイルに `allow` ルールを保存し、そこの `allow` ルールはプロジェクトまたは管理ファイルからの `ask` ルールより優先されません。**権限ルールがどのように結合されるか** (ページ 1064) は順序を説明しています。VS Code 拡張機能では、承認カードはプロジェクトの共有ファイルを含む宛先ファイルを選択できます。これはすべての人のルールを変更します。CLI では、Claude Code はローカルファイルにのみ書き込みます。

- **組織の許可ルールはあなたのルールと一緒に適用されます。** これは予想されています。Claude Code は `permissions.allow` をスコープ全体でマージします。ただし、組織が `allowManagedPermissionRulesOnly` を設定していない限り。

27.3.4 管理設定の優先順位の例外

セッションを制限する値を持つ少数のキーについて、Claude Code は通常は管理設定をオーバーライドできないスコープからの制限値を優先します。このテーブルでキーを見つけて、どの値を優先し、どこから優先するかを確認してください。

キー	Claude Code が優先する値	注記
disableClaudeAiConnectors	任意のスコープからの true	管理ソースが false を設定する場合でも優先されます
enableArtifact	任意のスコープからの false、および任意のスコープからの disableArtifact: true	管理ソースが true を設定する場合でも優先されます。何も Artifact ツール をオンに戻しません。Claude Code v2.1.242 以降が必要です
isolatePeerMachines	任意のスコープからの true	管理ソースが false を設定する場合でも優先されます
remoteControlAtStartup	.claude/settings.json または .claude/settings.local.json からの false	管理ソースが true を設定する場合でも優先されます。プロジェクトまたはローカル true は無視されます
crossSessionInbound	.claude/settings.json または .claude/settings.local.json からのより厳密な値、accept < hold < refuse ラダーで	管理、--settings、およびユーザー値より優先されます。より厳密でないプロジェクトまたはローカル値は無視されます
useAutoModeDuringPlan	任意の管理ソース、--settings、~/.claude/settings.json、または .claude/settings.local.json からの false	勝利した管理ソースが true を設定する場合でも優先されます。.claude/settings.json の false は無視されます
syncClaudeAiSkills	任意の管理ソース、--settings、~/.claude/settings.json、または .claude/settings.local.json からの false	勝利した管理ソースが true を設定する場合でも優先されます。.claude/settings.json の false は無視されます
syncClaudeAiPlugins	任意の管理ソース、--settings、~/.claude/settings.json、または .claude/settings.local.json からの false	勝利した管理ソースが true を設定する場合でも優先されます。.claude/settings.json の false は無視されます
maxEffortLevel	--settings を含む任意のスコープからのより低いキャップ	Claude Code が適用する管理設定がより高いキャップを設定する場合でも優先されます。最も低いキャップが適用されます。Claude Code v2.1.267 以降が必要です

Claude Code を内部で実行し、CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST を設定するアプリも例外です。Claude Code はそのアプリのモデル設定をすべての管理ソースからの model、fallbackModel、modelPicker、および modelOverrides キーより優先し、管理 env ブロック内のモデル選択変数（ANTHROPIC_MODEL および ANTHROPIC_DEFAULT_*_MODEL ファミリーなど）より優先します。Claude Code は管理 availableModels 許可リストを有効に保ちます。ただし、アプリが独自のものを提供する場合を除きます。

27.4 クラウドセッションの設定

クラウドセッション (ページ 223) はクラウド環境で実行され、マシン上ではなくリポジトリの新しいクローンで実行されます。これにより、どの設定がそこに到達するかが変わります：

- **共有プロジェクト設定**（`.claude/settings.json`）：1 つのリポジトリを持つセッションで読み込まれます。ファイルはクローンの一部であり、セッションはその内部で開始されるためです。その設定をコミットして、それらのセッションに適用してください。複数のリポジトリを持つセッションはクローンの上で開始され、各リポジトリの `.claude/settings.json` から `enabledPlugins` と `extraKnownMarketplaces` キーのみを読み込み、権限ルール、hooks、env、またはその他のキーは読み込みません。これら 2 つのキーが宣言するマーケットプレイスとプラグインは、それでもクラウドセッションでは読み込まれません。
- **ユーザーおよびプロジェクトローカル設定**（`~/.claude/settings.json` および `.claude/settings.local.json`）：読み込まれません。両方ともマシンに留まり、ローカルファイルはクローンにありません。
- **管理設定**：デバイスの `managed-settings.json` ファイルまたは MDM プロファイルはクラウドセッションに到達しません。組織のサーバー管理設定は到達します。**サーフェスカバレッジ** (ページ 735) はどのクラウドセッションがそれらを受け取るかをリストします。自己ホスト型環境もランナーイメージの管理設定ファイルを読み込みます。Claude Code が管理ソースを結合する方法はそのファイルがいつ適用されるかを説明します。
- `/config`：ブラウザの claude.ai/code では、値を変更する代わりに Claude Code セクションの claude.ai 設定を開きます。クラウドセッションの設定を変更するには、環境で環境変数を設定するか、1 つのリポジトリを持つセッションでは、そのリポジトリの `.claude/settings.json` にキーをコミットしてください。

セットアップから何が引き継がれるかは残りをリストします：`CLAUDE.md`、skills、MCP サーバー、plugins、および認証情報。

27.5 次は何か

- すべての設定：すべてのキー。設定する場所と例
- 設定ファイルの例：個人ファイル、チームファイル、および組織の管理ファイル
- **権限を構成する** (ページ 1039)：allow、ask、および deny ルール。Claude Code が確認なしで実行するもの
- 環境変数：Claude Code が読み込む変数および env ブロック
- 構成をデバッグする：設定が適用されない場合
- Claude ディレクトリリファレンス：Claude Code が読み込むすべてのファイル。subagents、MCP サーバー、プラグイン、および `CLAUDE.md` を含む

第 28 章　モデル設定

Claude Code が使用するモデル、effort レベル、拡張コンテキスト、自動圧縮ウィンドウを設定します

28.1 利用可能なモデル

Claude Code の `model` 設定には、次のいずれかを設定できます。

- **モデルエイリアス**
- **モデル名**
  - Anthropic API: 完全な モデル名
  - Amazon Bedrock: 推論プロファイル ARN
  - Microsoft Foundry: デプロイ名
  - Google Cloud's Agent Platform: バージョン名

さまざまな種類の作業にどのモデルと effort レベルが適しているかについては、ブログの Choosing a Claude model and effort level in Claude Code を参照してください。

注釈:

`ANTHROPIC_BASE_URL` はリクエストの送信先を変更するものであり、どのモデルが応答するかを変更するものではありません。LLM ゲートウェイ経由で Claude にルーティングするには、*LLM* **ゲートウェイ** (ページ 841) を参照してください。

28.1.1 モデルエイリアス

モデルエイリアスを使用すると、正確なバージョン番号を覚えなくてもモデル設定を選択できます。

モデルエイリアス	動作
`default`	モデルの上書きをすべてクリアし、**アカウントのランタイムデフォルト** (ページ 742) に戻す特別な値。それ自体はモデルエイリアスではありません
`best`	Fable が利用可能な場合は *fable* **エイリアスが解決されるモデル** (ページ 726) を使用し、それ以外の場合は `opus` と同じモデルを使用します
`fable`	最も難しく長時間実行されるタスク向けに、**プロバイダーの** *Fable* **モデル** (ページ 726) を使用します

次のページに続く

表 1 – 前のページからの続き

モデルエイリアス	動作
sonnet	日常的なコーディングタスク向けに最新の Sonnet モデルを使用します
opus	複雑な推論タスク向けに最新の Opus モデルを使用します
haiku	シンプルなタスク向けに高速で効率的な Haiku モデルを使用します
sonnet[1m]	長いセッション向けに 100 万トークンのコンテキストウィンドウを備えた Sonnet を使用します。sonnet がネイティブで 1M ウィンドウを持つ Sonnet 5.5 または Sonnet 5 にすでに解決される場合は効果がありません
opus[1m]	長いセッション向けに 100 万トークンのコンテキストウィンドウを備えた Opus を使用します
opusplan	plan モードでは opus を使用し、実行時には sonnet に切り替える特別なモード

opus および sonnet エイリアスが解決されるバージョンは、プロバイダーによって異なります。

プロバイダー	opus	sonnet
Anthropic API	Opus 5.5	Sonnet 5.5
Claude Platform on AWS	Opus 5.5	Sonnet 4.6
Amazon Bedrock、Google Cloud's Agent Platform	Opus 5.5	Sonnet 4.5
Microsoft Foundry	Opus 4.6	Sonnet 4.5

ANTHROPIC_DEFAULT_FABLE_MODEL を設定しない限り、fable エイリアスは Fable 5.1 に解決されます。ただし Claude apps gateway のセッションでは、fable と best は Fable 5 に解決されます。

claude-fable-5-1 を提供するように設定されていないゲートウェイは、そのモデルへのリクエストを拒否します。Fable 5.1 を提供しているゲートウェイ経由で使用するには、/model claude-fable-5-1 で選択してください。

エイリアスが古いモデルに解決される場合、完全なモデル名を明示的に選択するか、ANTHROPIC_DEFAULT_OPUS_MODEL または ANTHROPIC_DEFAULT_SONNET_MODEL を設定することで、新しいモデルを利用できます。

以前のバージョンでは、これらのエイリアスはより古いモデルに解決されます。各エイリアスがどのバージョンで変更されたかについては、**バージョン履歴** (ページ 763) を参照してください。

エイリアスはプロバイダーの推奨バージョンを指しており、時間の経過とともに更新されます。特定のバージョンに固定するには、完全なモデル名（例: claude-opus-5-5）を使用するか、ANTHROPIC_DEFAULT_OPUS_MODEL などの対応する環境変数を設定してください。

注釈:

Sonnet 5.5 には Claude Code v2.1.284 以降、Opus 5.5 には v2.1.280 以降が必要です。古いバージョンからこれらのモデルへのリクエストが失敗する場合は、Claude Code does not support this model を参照してください。アップグレードするには claude update を実行します。

28.1.2 Fable を使用する

Claude Fable 5.1 と Claude Fable 5 は Claude Code で最も高性能なモデルであり、1 回の作業時間に収まらない規模のタスクに適しています。長時間の自律的なセッションを維持し、行動する前に調査を行い、小規模なモデルよりも頻繁に自身の作業を検証します。Fable 5.1 は新しいリリースです。

どちらの Fable モデルも、いずれのプランやプロバイダーにおいてもアカウント種別のデフォルトではありません。明示的に選択してください。

- **Fable 5.1**: `/model fable` を実行するか、`claude --model fable` で起動します。エイリアスが Fable 5 に解決される Claude apps gateway のセッションでは、代わりに `/model claude-fable-5-1` を実行します。
- **Fable 5**: モデル ID で選択します。Anthropic API では、`/model claude-fable-5` を実行するか、`claude --model claude-fable-5` で起動します。その他のプロバイダーでは、プロバイダーの Fable 5 モデル ID を使用するか、`ANTHROPIC_DEFAULT_FABLE_MODEL` で**固定** (ページ 758) します。

Anthropic API に直接接続していて、ユーザー設定にモデルとして `claude-fable-5` または `claude-fable-5[1m]` が保存されている場合（たとえば v2.1.257 より前に `/model` ピッカーで Fable を選択した場合）、v2.1.257 以降を初めて実行したときに、Claude Code はその保存値を `fable` または `fable[1m]` エイリアスに変更します。起動時のモデル行には `(auto-updated)` が一度だけ表示されます。プロジェクト設定、ローカル設定、または管理設定にある `claude-fable-5` の値はそのまま残ります。

Fable モデルの安全性分類器によって警告されたリクエスト（主にサイバーセキュリティや生物学の分野）は、**自動モデルフォールバック** (ページ 744) をトリガーします。

Fable を最大限に活用するには:

- **手順ではなく結果を説明する**: 望む結果を伝え、その道筋は Fable に計画させます。その結果に向けて作業を続けさせるには、ゴールを設定します。
- **曖昧な問題を任せる**: 根本原因の調査、障害のデバッグ、アーキテクチャの決定などは、追加の調査と検証が効果を発揮する場面です。
- **検証のリマインダーを省く**: Fable は少ない指示で自身の作業を検証するため、テストや確認を促すリマインダーは通常不要です。
- **より大きなタスクを任せる**: 通常なら分割するような作業を与えてください。Fable は長いセッションでも文脈を見失いません。

注釈:

Fable 5.1 には Claude Code v2.1.257 以降が必要です。古いバージョンからのリクエストが失敗する場合は、Claude Code does not support this model を参照してください。アップグレードするには `claude update` を実行します。ゼロデータ保持での利用可否については、ZDR でのモデルの利用可否を参照してください。

Anthropic API では、*`availableModels`* (ページ 732) または**組織のモデル制限** (ページ 739) によって除外されない限り、Fable モデルは `/model` ピッカーに表示されます。組織が Fable をまったく使用できない場合（たとえばゼロデータ保持の下にある場合）、その行はグレーアウトされた状態でピッカーに残り、理由が注記されます。

Fable と使用クレジット

プランやシート階層によっては、Fable の使用量がプランに含まれる上限から差し引かれるのではなく、使用クレジットに課金される場合があります。その場合、`/model` ピッカーの Fable の行に「Requires usage credits」と表示されます。使用クレジットを管理するには、**サブスクリプションに使用クレジットを追加する** (ページ 914) を参照してください。

対話セッションでは、Fable のリクエストが使用クレジットに課金される前に、Claude Code が同意プロンプトを表示します。組織請求を利用している Enterprise プランのメンバーには、このプロンプトは表示されません。使用クレジットを使って Fable で続行するか、デフォルトモデルに切り替えることができます。プロンプトを閉じることもできます。

- `/model` で Fable モデルを選択した場合は、現在のモデルが維持されます。
- セッションの途中では、Claude Code はデフォルトモデルでそのターンを続行します。

使用クレジットを使って Fable で続行することを選択すると、Claude Code はそれ以降プロンプトを表示しません。

Remote Control が接続されているセッション、バックグラウンドセッション、またはエージェントチームのチームメイトのセッションでは、ターミナルの前に誰もいない可能性があるため、Claude Code はセッション途中の同意プロンプトを `dialogExpiry` の期限（デフォルトは 5 分）まで保持します。期限までに誰も応答しなかった場合、Claude Code はリクエストを送信せずにターンを終了し、トランスクリプトに通知を追加します。この通知は Remote Control クライアントにも表示されます。モデルの選択は変更されず、次のメッセージで Claude Code は再度同意を求めます。

プロンプトが待機している間にできることは、セッションによって異なります。

- Remote Control が接続されている場合やチームメイトのセッションでは、ターミナルで任意のキーを押すと期限がキャンセルされ、Claude Code は応答を待ちます。
- バックグラウンドセッションでは、期限までに応答してください。
- ターミナルで誰かが入力する前にリモートクライアントから新しいメッセージを送信した場合、Claude Code は同様にターンを終了し、新しいメッセージが次のターンを開始します。ターミナルで誰かが入力した後は、Claude Code は応答を待ち続け、新しいメッセージはその後ろにキューイングされます。

別のアプリケーションが Agent SDK を通じてホストしているセッションでは、プロンプトを表示するかどうかはそのアプリケーション次第です。プロンプトが表示され、同じ `dialogExpiry` の期限までに誰も応答しなかった場合、Claude Code はリクエストを送信せずにターンを終了します。

`-p` フラグを使用した**非対話モード** (ページ 695) や、プロンプトを表示しない Agent SDK アプリケーションでは、Claude Code は同意を求めません。そこで Fable のリクエストが使用クレジットに課金される場合、Claude Code は確認なしで課金します。

28.1.3 モデルの設定

モデルはいくつかの方法で設定できます。以下は優先順位の高い順です。

1. **セッション中**: `/model <alias|name>` を使用してすぐに切り替えるか、引数なしで `/model` を実行してピッカーを開きます。Claude Code が切り替えの確認を求める場合を参照してください
2. **起動時**: `claude --model <alias|name>` で起動します

3. **環境変数**: ANTHROPIC_MODEL=<alias|name> を設定します
4. **設定**: 設定ファイルの model フィールドで永続的に設定します
5. **新しいセッションのデフォルト** (ページ 731): ANTHROPIC_DEFAULT_MODEL=<alias|name> を設定します

/model は、ユーザー設定の model フィールドに書き込むことで、選択内容を新しいセッションのデフォルトとして保存します。ピッカーでは次のキーを使用します。

- Enter: モデルを切り替え、デフォルトとして保存します
- s: このセッションのみモデルを切り替え、デフォルトは変更しません。別のキーを使用するには、*modelPicker:thisSessionOnly* (ページ 775) を再割り当てします

/model <name> を直接入力した場合は、Enter と同じ動作になります。このセッションのみ切り替えるには、/model でピッカーを開き、そのモデルの行で s を押します。

Enterprise プランで claude.ai アカウントでログインしており、/model でデフォルトを保存した場合、Claude Code はその選択をアカウントにも記録します。これには Claude Code v2.1.280 以降が必要です。

- 管理者が**組織のデフォルトモデル** (ページ 740) を設定していない場合、*Default* **オプション** (ページ 742) は記録されたモデルに解決されることがあり、その場合はピッカーの Default 行にそのモデル名が表示されます。
- **モデル制限** (ページ 732) によって記録されたモデルが除外されている場合、またはそのモデルがアカウントで利用できない場合で、かつ管理者が組織のデフォルトモデルを設定していない場合、Default オプションは何も記録されていないかのように解決されます。
- /model で Default または opusplan を選択した場合、記録された選択は変更されません。

/model でモデルを切り替えると、その切り替えは**メインの会話のモデルを継承するサブエージェント** (ページ 329) にも反映されます。Claude がサブエージェントを起動する際、Claude Code はセッションで使用しているモデルからそのモデルを解決するためです。Claude が調査やテスト実行をサブエージェントに委任する前に Opus に切り替えると、その作業も Opus で実行されます。カスタムサブエージェントを小さなモデルのままにするには、その定義で model を設定してください。

-p フラグを使用した**非対話モード** (ページ 695) で /model によってモデルを設定した場合、選択は現在のセッションにのみ適用され、デフォルトとしては保存されません。このモードでの /model には Claude Code v2.1.205 以降が必要です。プロジェクト設定と管理設定は引き続き優先され、次回の起動時に再適用されます。管理者がユーザーの選択を上書きするように設定した**組織のデフォルトモデル** (ページ 740) も、次回の起動時に再適用されます。

v2.1.144 から v2.1.152 では、/model は現在のセッションにのみ適用され、ピッカーで d を押すとデフォルトが保存されていました。

--model フラグと ANTHROPIC_MODEL 環境変数は、それらを指定して起動したセッションにのみ適用されます。異なるターミナルで異なるモデルを同時に実行するには、/model で切り替えるのではなく、それぞれ独自の --model フラグを指定して起動してください。

/model ピッカーの価格は、Claude Code が Anthropic API と直接、またはそれをプロキシする *LLM* **ゲートウェイ** (ページ 841) 経由で通信している場合に表示され、各行の価格はその行が選択するモデルの価格です。Amazon Bedrock などの**サードパーティプロバイダー** (ページ 847) や Claude apps gateway では、支払う金額はプロバイダーまたはゲートウェイによって決まるため、ピッカーの行には価格が表示されません。

価格は表示用のラベルにすぎず、行が選択するモデルやプロバイダーの請求額には影響しません。v2.1.206 より前は、Claude Platform on AWS とゲートウェイのセッションで Anthropic の定価が表示され、行が選択するモデルとは異なるモデルの価格が表示されることがありました。

`claude --resume`、`--continue`、または `/resume` ピッカーで再開したセッションは、現在の `model` 設定にかかわらず、トランスクリプトが保存された時点で使用していたモデルを維持します。復元されたモデルが廃止されている場合や *`availableModels`* (ページ 732) によって除外されている場合、セッションは通常の優先順位に従います。これにより、別のセッションでの `/model` の選択が再開時のモデルを変更することを防ぎます。Amazon Bedrock、Google Cloud's Agent Platform、Microsoft Foundry など、Anthropic のモデル ID ではなくプロバイダー固有のデプロイ ID を使用するプロバイダーでは、トランスクリプトのモデルはまったく復元されず、セッションは通常の優先順位に従ってモデルを解決します。

新しい起動時に `--model` または `ANTHROPIC_MODEL` で選択したモデルは、引き続き復元されたモデルより優先されます。v2.1.195 以降は、*`ANTHROPIC_DEFAULT_OPUS_MODEL`* (ページ 757) 系の変数も同様に優先されます。*`ANTHROPIC_DEFAULT_MODEL`* (ページ 731) も、そのセクションに記載されている条件の下で優先されます。

起動時のアクティブなモデルが自分の選択ではなくプロジェクト設定または管理設定に由来する場合、起動時のヘッダーにどの設定ファイルで設定されたかが表示されます。上書きするには `/model` を実行します。プロジェクト設定または管理設定は次回の起動時に再適用されます。Claude Code を組み込み、`CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST` を設定しているプラットフォームでは、ホストのモデル設定が管理設定のモデル設定より優先されます。一方、管理設定の `availableModels` 許可リストは、ホストが独自のものを提供しない限り有効なままです。ホストがどのキーと変数を上書きするかについては、**管理設定の優先順位の例外** (ページ 722) を参照してください。

ユーザーまたは組織が *PreModelSwitch* **フック** (ページ 542) を設定している場合、それらは要求された切り替えが適用される前に実行され、切り替えをブロックしたり確認を求めたりできます。

組織の管理プラグインがどの PreModelSwitch フックを提供しているかを Claude Code が判断できない場合（たとえば管理プラグインの読み込みに失敗した場合）、Claude Code は確認されないまま切り替えを適用するのではなく切り替えを拒否し、新しい試行のたびに再度確認します。メッセージと復旧方法については、Model switch was blocked by a PreModelSwitch hook を参照してください。

Agent SDK の `setModel()` メソッド、**デスクトップアプリ** (ページ 137) などのアプリ、または Remote Control 経由で接続されたデバイスからモデルを切り替える場合、Claude Code は切り替え時に値を確認します。

- **Agent SDK またはアプリ**: Claude Code v2.1.268 以降では、**カスタムモデルオプション** (ページ 756) の場合のように Claude Code がモデル ID をローカルで受け入れる場合を除き、セッションがそのモデルに初めて切り替わるときにプロバイダーに ID を確認します。この確認はすべてのプロバイダーで実行され、プロバイダーが提供していない ID は、次のリクエストで失敗するのではなく切り替え時に拒否されます。
- **Remote Control**: Anthropic API では、Claude Code は値をローカルで確認し、リクエストは送信しません。

メッセージについては、Model is not a recognized model id および Model not found を参照してください。

`--model` フラグ、`ANTHROPIC_MODEL` 環境変数、または `model` 設定でモデルを設定した場合、Claude Code は事前に確認を行わないため、値を誤入力すると最初のリクエストで There's an issue with the selected model が発生します。

要求されたモデルに廃止予定日が設定されている場合、または新しいバージョンに自動的に再マッピング

される場合、Claude Code は要求されたモデル名を示す警告を表示します。対話セッションでは、起動時の通知として表示されます。v2.1.182 以降、デフォルトのテキスト出力形式を使用している場合、**非対話モード** (ページ 695) でも同じ警告が stderr に書き込まれます。このチェックは、**サブエージェントのフロントマター** (ページ 317) で設定された `model` も対象とします。`--output-format json` および `stream-json` では stderr への警告は抑制されます。代わりに、**結果メッセージ** (ページ 699) の `modelUsage` フィールドから実際のモデルを読み取ってください。

たとえば、Opus でセッションを開始します。

```
claude --model opus
```

次に、セッション内からモデルを切り替えます。

```
/model sonnet
```

設定ファイルの例:

```
{
    "permissions": {
        "allow": ["Bash(npm run lint)"]
    },
    "model": "opus"
}
```

新しいセッションのデフォルトモデルを設定する

セッションがデフォルトで開始するモデルを選択するには、`ANTHROPIC_DEFAULT_MODEL=<alias|name>` を設定します。Claude Code v2.1.236 以降が必要です。

Claude Code が新しいセッションをこの変数のモデルで開始するのは、次のいずれもモデルを選択していない場合のみです。

- `--model` フラグ
- `ANTHROPIC_MODEL`
- いずれかの設定ファイルにある `model` の値（`/model` で保存した選択を含む）
- **組織のデフォルトモデル** (ページ 740)

`/model` で保存した選択は、以降の起動でもこの変数より優先されます。代わりに `ANTHROPIC_MODEL` を設定している場合、`/model` で何を保存したかにかかわらず、Claude Code は次回の起動時にその変数のモデルに戻ります。

組織のデフォルトモデルが適用されない限り、Claude Code は Default オプションもこの変数のモデルに解決します。Default オプションがこの変数のモデルに解決される場合、`/model` ピッカーの Default 行には Set by ANTHROPIC_DEFAULT_MODEL というラベルが表示されます。

次の場合、Claude Code はこの変数を無視し、Default オプションは変数を設定していないかのように解決されます。

- `default`、`inherit`、`opusplan`、または `haiku` に設定している

- *enforceAvailableModels* (ページ 736) がオンになっている
- 組織の**モデル制限** (ページ 732) によってそのモデルが除外されている
- そのモデルがアカウントで利用できない

新しいセッションがこの変数のモデルで開始される場合、claude --resume、--continue、または /resume ピッカーで再開したセッションもそのモデルで開始されます。Claude Code はそのセッションのトランスクリプトに保存されたモデルを復元しません。それ以外の場合、**セッションを再開** (ページ 728) するときに Claude Code はこの変数を使用しません。

新しいセッションが選択したものとは異なるモデルで開始される

/model でモデルを選択したのに次のセッションが別のモデルで開始される場合、通常は次のような原因があります。

- **1 つのセッションのみを対象に選択した。** ピッカーで s を押す、--model で起動する、非対話モードで /model を実行する、のいずれも現在のセッションにのみ適用され、保存されたデフォルトは変更されません。
- **より優先順位の高いものがモデルを設定している。** プロジェクト設定や管理設定にある model の値、シェルの ANTHROPIC_MODEL、または管理者がユーザーの選択を上書きするように設定した**組織のデフォルト** (ページ 740) は、起動のたびに再適用されます。/model での選択は保存されていますが、優先順位で負けています。プロジェクト設定または管理設定がモデルを設定している場合、起動時のヘッダーにそのファイル名が表示されます。
- **Claude Code が選択を保存できなかった。** /model は ~/.claude/settings.json に model を書き込みます。別のツールがそのファイルを生成している、または読み取り専用のコピーにリンクしているなどの理由でそのファイルに書き込めない場合、選択したモデルはそのセッションの間だけ有効で、次回の起動時には古い値が読み込まれます。ファイルを生成しているツールで model を設定するか、ファイルを書き込み可能にしてください。*Claude Code* **で行った変更が新しいセッションで失われる** (ページ 721) を参照してください。
- **セッションを再開した。** claude --resume または --continue で再開したセッションは、通常、現在のデフォルトではなく**使用していたモデルを維持** (ページ 728) します。

28.2 モデル選択を制限する

管理者は、管理設定またはポリシー設定の availableModels を使用して、ユーザーが選択できるモデルを制限できます。エントリは、sonnet のようなモデルファミリー、claude-sonnet-4-5 のようなバージョンプレフィックス、または claude-sonnet-4-5-20250929 のような完全なモデル ID に一致します。バージョンプレフィックスは、それにさらにセグメントを付け加えた後続のモデル ID にも一致するため、claude-fable-5 は Fable 5 と Fable 5.1 の両方を許可し、claude-fable-5-1 は Fable 5.1 のみを許可します。リストが許可するモデルをブロックする方法や、各モデル ID エントリがその ID で指定されたバージョンのみを許可するようにする方法については、**特定のモデルやバージョンをブロックする** (ページ 738) を参照してください。

Claude Code を組み込み、CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST を設定しているプラットフォームでは、ホストのモデル設定が管理モデル設定よりも優先されます。ただし、管理設定の availableModels 許

可リストは、ホストが独自の許可リストを提供しない限り引き続き有効です。ホストがどのキーと変数を上書きするかについては、**管理設定の優先順位の例外** (ページ 722) を参照してください。

availableModels が設定されている場合、許可リストはユーザーがモデルを指定できるすべての場所に適用されます。

- **メインセッションのモデル**: /model、--model フラグ、ANTHROPIC_MODEL 環境変数、model 設定、*ANTHROPIC_DEFAULT_MODEL* (ページ 731)、および**セッションの再開** (ページ 728) 時に復元されるモデル
- **エイリアスの解決**: ANTHROPIC_DEFAULT_OPUS_MODEL、ANTHROPIC_DEFAULT_SONNET_MODEL、ANTHROPIC_DEFAULT_HAIKU_MODEL、ANTHROPIC_DEFAULT_FABLE_MODEL 環境変数を使って、許可されたエイリアスをリスト外のモデルにリダイレクトすることはできません
- **fast mode**: リスト外の Opus モデルに暗黙的に切り替わる場合、/fast は "is not in your organization's allowed models" というメッセージを表示して切り替えを拒否します
- **サブエージェントとチームメイトのモデル**: **サブエージェント** (ページ 329) のフロントマターの model フィールド、Agent ツールの model パラメータ、エージェントチームのチームメイトのモデル、CLAUDE_CODE_SUBAGENT_MODEL、および v2.1.197 以前では /agents ウィザードのモデルピッカー
- **スキルとコマンドのモデル**: **スキルとコマンド** (ページ 275) の model フロントマター
- **アドバイザーモデル**: 設定された advisorModel 設定と --advisor フラグ
- **バックグラウンドエージェントのモデル**: Dispatch ピッカーで選択されたモデル

Anthropic API と Claude Platform on AWS では、モデルファミリーのエイリアス（opus、sonnet、haiku、fable）は、許可リストがそのモデルを許可している場合、通常のモデルに解決されます。許可リストがそのモデルをブロックしている場合、Claude Code は許可リストが許可するそのファミリーの最新バージョンに置き換え、要求されたモデルと置き換え後のモデルの両方を示す通知を表示します。たとえば ["sonnet", "claude-opus-4-6"] の場合、/model opus と --model opus はどちらも、許可されている最新の Opus である Claude Opus 4.6 を選択します。v2.1.205 より前は、最新のリリースバージョンがリスト外にあるエイリアスは、リストが古いバージョンを許可していても、他のブロックされた選択と同様に拒否または置き換えられていました。

この置き換えには、置き換え先となる許可されたバージョンが必要です。許可リストがエイリアスのファミリーのどのバージョンも許可していない場合、そのエイリアスは他のブロックされた値と同様に、以下の拒否および置き換えの動作に従います。

Claude Code は、その他のブロックされた選択を、モデルが設定された場所に応じて次のように処理します。

- /model: Claude Code はエラーを表示して切り替えを拒否します
- **--model フラグ、ANTHROPIC_MODEL、または model 設定**: Claude Code は起動時に値を置き換え、要求されたモデルと置き換え後のモデルの両方を示す警告を表示します。セッションはデフォルトモデルで開始されます
- *ANTHROPIC_DEFAULT_MODEL* (ページ 731): Claude Code はこの変数を無視します
- **サブエージェントまたはチームメイトの上書き**: Claude Code はリクエストを失敗させるのではなく、フォールバックモデルでサブエージェントまたはチームメイトを実行します。サブエージェントのフォールバックについては**モデルを選択する** (ページ 329) を、チームメイトのフォールバックについてはチームメイトとモデルを指定するを参照してください。

インタラクティブセッションでは、このフォールバックまたは上記の許可された最新バージョンへの置き換えによってサブエージェントのモデルが置き換えられた場合、Claude Code は要求されたモデルと置き換え後のモデルを示す警告を表示します。チームメイトのフォールバックについては報告しません。

上記の許可された最新バージョンへの置き換えが機能する環境では、ブロックされたファミリーエイリアスはそちらに従います。v2.1.222 より前は、すべてのプロバイダーで、エイリアスは他のブロックされた値と同様にフォールバックしていました

- **スキルまたはコマンドの上書き**: Claude Code は、ブロックされたファミリーエイリアスを含めて上書きを無視し、スキルまたはコマンドはセッションのモデルで実行されます。**サブエージェントで実行される** (ページ 303) スキルまたはコマンドは、代わりに上記のサブエージェントの動作に従います
- `advisorModel` **設定**: そのセッションではアドバイザーが無効になります
- `--advisor` **フラグ**: Claude Code は起動時にエラーで終了します。バックグラウンドセッションでは、終了する代わりにアドバイザーなしでセッションを開始します

Claude Code は、除外されたモデルを `/model` ピッカーから非表示にします。リストに含めたモデル ID が独自の行も持つかどうかは、プロバイダーによって異なります。

- **Anthropic API、Claude Platform on AWS、Claude apps gateway、または `ANTHROPIC_BASE_URL` で設定した *LLM* ゲートウェイ** (ページ 841): リストに含めた Anthropic のモデル ID のうち、組み込みのピッカー行がないものは、独自のラベル付き行として表示されます。Claude Code は、リストで固定された古いバージョンなど、Opus、Sonnet、Haiku のバージョンに対してこのような行を追加します。`modelPicker` のラインナップで `replaceBuiltInOptions` を設定している場合、その行は表示されません。v2.1.199 より前は、そのような ID は `/model <id>` と入力することでのみ選択できました。
- **Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry**: リストに含めたモデル ID が `anthropic.` で始まらない限り、それが Anthropic のモデル ID であってもプロバイダー固有のものであっても、Claude Code はその行を追加しません。*Mantle* **のモデル** *ID* (ページ 738) にはこのプレフィックスが付いています。組み込みの行がないリスト内のバージョンを表示するには、`modelPicker` のラインナップにも追加してください。ラインナップはプロバイダーの形式の ID を受け付けます。

Claude Code がユーザーに代わって行うモデル変更も、同じ方法でチェックされます。

- **フォールバックモデルチェーン** (ページ 743): 許可リスト外のエントリは除外されます
- **plan モードのアップグレード**: Anthropic API と Claude Platform on AWS では、*`opusplan`* (ページ 742) のように除外されたモデルへのアップグレードは、アップグレード先ファミリーの許可された最新バージョンを使用します。プロバイダー固有のモデル ID を持つプロバイダーの場合、および許可されたバージョンがない場合は、アップグレードはスキップされ、計画はセッションのモデルで続行されます
- **自動モデルフォールバック** (ページ 744): フォールバック先が除外されている場合、フォールバックは実行されないため、警告されたリクエストは代わりに拒否で終了します
- auto モードの分類器: 分類器のデフォルトである Claude Sonnet 5 は、許可リストが Sonnet 5 を許可している場合にのみ適用されます。除外されている場合、分類器はセッションのモデル（すでに許可リストの制御下にあります）で実行されるか、セッションが *Fable* **モデル** (ページ 727) で実行されている場合は Opus モデルで実行されます。Anthropic API 以外のプロバイダーでは、その Opus フォー

ルバックは許可リストを参照せずに、`ANTHROPIC_DEFAULT_OPUS_MODEL` で設定したモデル、または設定していない場合は Opus 5 で実行されます。Claude Code v2.1.210 以降が必要です

- fast mode: 有効化後にセッションが実行されるモデルが許可リスト外である場合、fast mode の有効化は拒否されます

```
{
  "availableModels": ["sonnet", "haiku"]
}
```

28.2.1 サーフェスごとの適用範囲

すべてのサーフェスは、受け取った許可リストを適用します。各サーフェスに届く配信メカニズムは次のように異なります。

配信メカニズム	CLI と IDE	Desktop のローカルセッション	Web、モバイル、クラウドセッション	Agent SDK と非インタラクティブ	Cowork
管理コンソールからのサーバー管理設定	適用される	適用される	適用される（Claude Tag セッションを除く）	適用される	リモート Cowork セッション: サーバーがモデルをチェックします。ユーザーのマシン上: 配信されません。
MDM または管理設定ファイル	適用される	適用される	Anthropic がホストする環境には配信されません。セルフホスト環境では、Claude Code が管理ソースを組み合わせる方法に従ってランナーイメージから適用されます	適用される	デプロイされている場所で適用される

- Desktop アプリから開始したものを含む**クラウドセッション** (ページ 223) は、デフォルトで Anthropic が管理する VM 上で実行されます。デバイスにデプロイされた設定はこれらに届かないため、許可リストはサーバー管理設定を通じて配信してください。組織がセルフホスト環境にルーティングするセッションは独自のコンピューティング上で実行され、ランナーイメージ内の管理設定ファイルも読み取ります。そのファイルがいつ適用されるかについては、Claude Code が管理ソースを組み合わせる方法を参照してください。クラウドセッションでのセッション途中のモデル切り替えは、要求されたモデルが許可リストによって除外されている場合は拒否されます。サーバー管理設定の `availableModels` リストが空でない場合、サーバーは、リストが除外するモデルで claude.ai/code または Desktop アプリからクラウドセッションを開始するリクエストを拒否します。

- Claude Tag セッションはクラウド環境で実行されますが、サーバー管理設定を受け取りません。セルフホスト環境では、引き続きランナーイメージ内の管理設定ファイルを読み取ります。これらのセッションのモデルを設定するには、Claude Tag 管理者ガイドのスコープのモデルを選択するを参照してください。

- Claude Desktop アプリのエージェント型作業タブである Cowork は、セッションを Claude Code 上で実行しますが、設計上、claude.ai 管理コンソールからサーバー管理設定を受け取りません。サーバー管理設定の `availableModels` リストが空でなく、ユーザーがリスト外のモデルを選択した場合、サーバーはリモート Cowork セッションでそのモデルを拒否します。管理設定ファイルは、セッションが実行される場所に存在する場合に Cowork セッションに適用されます。リモート Cowork セッションは Anthropic が管理する VM 上で実行され、そこにはデバイスにデプロイされたファイルは存在しません。

- Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry、Claude Platform on AWS などのサードパーティプロバイダー上のセッションはサーバー管理設定を受け取らないため、それらの環境では MDM または管理設定ファイルを通じて許可リストを配信してください。
- サーバー管理による配信では、セッションが対象となるログインまたはキーで認証されている必要もあります。`apiKeyHelper` スクリプトを通じてのみキーを生成するフリートでは、MDM または管理設定ファイルを通じて許可リストを配信してください。
- Desktop の Code タブは *SSH* **セッション** (ページ 159) もホストしており、これらは実行先のリモートホストから管理設定ファイルを読み取ります。*Desktop* **の管理設定** (ページ 161) を参照してください。
- claude.ai および Desktop アプリのモデルピッカーは、組織の許可リストによって除外されたモデルを非表示にするかグレーアウトします。ピッカーの状態はユーザーの利便性のためのものであり、許可リストを適用するものではありません。

28.2.2 デフォルトモデルの動作

デフォルトのプレフィックス一致では、`availableModels` 単独では、*`enforceAvailableModels`* (ページ 736) も設定するまで、Default オプションはアカウントに対するシステムの**ランタイムデフォルト** (ページ 742) のままになります。そのデフォルトが制限したいモデルである場合は、`enforceAvailableModels` も設定するか、**そのモデルをブロック** (ページ 738) してください。

`availableModels: []` の場合、名前を指定したモデル選択はブロックされ、`enforceAvailableModels` は効果を持ちません。

28.2.3 Default モデルに許可リストを適用する

管理設定で、空でない `availableModels` とともに `enforceAvailableModels: true` を設定すると、許可リストが Default オプションにも適用されます。これには Claude Code v2.1.175 以降が必要です。

```
{
  "availableModels": ["sonnet", "haiku"],
  "enforceAvailableModels": true
}
```

アカウントにモデルが記録されていない (ページ 728) メンバーの場合、Default オプションはアカウントタイプのデフォルト、または管理者が設定している場合は**組織のデフォルトモデル** (ページ 740) に解決されます。そのモデルが許可リストにない場合、Default オプションは代わりに、許可されていて利用可能なモデルを指定する最初の `availableModels` エントリに解決され、`/model` ピッカーの Default 行にはそのモデルが表示されます。これは、デフォルトに到達するすべての場所に適用されます。つまり、セッションの起動時、`/model` で Default を選択した場合、**フォールバックモデルチェーン** (ページ 743) の `"default"` キーワード、および除外された選択が除外されたときに使用されるフォールバックです。メンバーのアカウントに記録されたモデルも `availableModels` に照らしてチェックされます。Default オプションがそれをどのように扱うかについては、**モデルを設定する** (ページ 728) を参照してください。

`enforceAvailableModels` は、`availableModels` が空でない場合にのみ Default オプションを再マッピングします。`availableModels` が空でないものの、許可されていて利用可能なモデルに解決されるエントリがない場合、適用はスキップされ、`--debug` でのみ表示される警告が出されます。これを避けるには、確実に利用可能なエントリを少なくとも 1 つリストに含めてください。

両方のキーは、配信する管理ソースのうち最も順位の高いものに一緒にデプロイしてください。デフォルトでは Claude Code はそのソースのみを読み取るため、管理コンソールが何らかの設定を配信している場合、管理設定ファイルに配置したペアは無視されます。Claude Code が管理ソースを組み合わせる方法のオプトインのマージを使用する場合でも、Claude Code は `availableModels` を設定するソースより下位のソースからの `modelOverrides` マップを無視します。

28.2.4 ユーザーが実行するモデルを制御する

`model` 設定は初期選択であり、適用ではありません。これはセッション開始時にどのモデルがアクティブになるかを設定しますが、ユーザーは引き続き `/model` を開いて Default を選択できます。Default は、*enforceAvailableModels* (ページ 736) または**特定のバージョンをブロックするキー** (ページ 738) が適用されない限り、`model` の設定値に関係なくシステムの**ランタイムデフォルト** (ページ 742) に解決されます。

モデルの動作を完全に制御するには、次の設定を組み合わせます。

- `availableModels`: ユーザーが切り替えられる名前付きモデルを制限します
- `enforceAvailableModels`: `availableModels` 許可リストを Default オプションに拡張し、Default がリスト外のモデルに解決されないようにします
- `deniedModels` と `availableModelsMatch`: `availableModels` エントリがなければ許可されてしまう**特定のバージョンをブロック** (ページ 738) します
- `model`: セッション開始時の初期モデル選択を設定します
- `ANTHROPIC_DEFAULT_SONNET_MODEL` / `ANTHROPIC_DEFAULT_OPUS_MODEL` / `ANTHROPIC_DEFAULT_HAIKU_MODEL` / `ANTHROPIC_DEFAULT_FABLE_MODEL`: `sonnet`、`opus`、`haiku`、`fable` エイリアスが何に解決されるか、および**アカウントタイプのデフォルト** (ページ 742) がどのバージョンを使用するかを制御します

この例では、ユーザーを Sonnet 4.5 で開始させ、ピッカーを Sonnet と Haiku に制限し、Default がティアのデフォルトではなく許可リスト上のモデルに解決されるようにします。

```
{
  "model": "claude-sonnet-4-5",
  "availableModels": ["claude-sonnet-4-5", "haiku"],
  "enforceAvailableModels": true,
  "env": {
    "ANTHROPIC_DEFAULT_SONNET_MODEL": "claude-sonnet-4-5"
  }
}
```

`enforceAvailableModels` や `env` ブロックがない場合、ピッカーで Default を選択したユーザーには、`model` で固定されたバージョンではなく**ランタイムデフォルト** (ページ 742) が適用されます。この 2 つの設定は異なる範囲をカバーします。`enforceAvailableModels` は Default を許可リストに従わせ、`env` ブロックは `sonnet` などの許可されたエイリアスがどのバージョンに解決されるかを固定します。モデルファミリーを制限するだけで十分な場合は `enforceAvailableModels` のみを使用し、特定のバージョンも固定する必要がある場合は `env` ブロックを追加してください。

28.2.5 マージの動作

Claude Code が適用する管理設定で `availableModels` が定義されている場合、**独自のものを提供するホストプラットフォーム** (ページ 722) を除き、そのリストのみが適用されます。ユーザー設定、プロジェクト設定、ローカル設定のエントリでそれを拡張することはできず、Claude Code は管理ソース間で `availableModels` をマージすることもありません。どのソースのリストが適用されるかについては、Claude Code が管理ソースを組み合わせる方法を参照してください。それ以外の場合、ユーザー設定、プロジェクト設定、ローカル設定のリストは、他の配列設定と同様に**連結され、重複が除去されます** (ページ 719)。Claude Code v2.1.175 より前は、優先順位の低いスコープのエントリは、管理リストに置き換えられるのではなく、管理リストにマージされていました。

有効なリスト内で、ファミリー内の特定のモデルを指定するエントリ（バージョンプレフィックスまたは完全なモデル ID）は、そのファミリーのワイルドカードエントリを無効にします。`["sonnet", "claude-sonnet-4-5"]` は、すべての Sonnet モデルではなく、Sonnet 4.5 のバージョンのみを許可します。

28.2.6 Mantle のモデル ID

`availableModels` 内の `anthropic.` で始まるエントリは、カスタムオプションとして `/model` ピッカーに追加されます。これは、**サードパーティのデプロイ向けにモデルを固定する** (ページ 758) で説明されているエイリアス一致の例外です。*Amazon Bedrock Mantle* **エンドポイント** (ページ 875) が有効な場合、Claude Code は Mantle 形式に一致するエントリをそのエンドポイントにルーティングします。この設定は引き続きピッカーをリストされたエントリに制限し、Mantle ID にはファミリー名が含まれているため、特定のエントリとしてカウントされ、そのファミリーのワイルドカードを無効にします。Mantle ID と併せて、選択可能なままにしたいバージョンプレフィックスまたは完全な ID をリストしてください。**マージの動作** (ページ 738) を参照してください。

28.2.7 特定のモデルやバージョンをブロックする

`claude-opus-5` のような `availableModels` エントリは、Opus 5.5 など、それを拡張する後続のリリースも、Claude Code がサポートした時点で許可します。リリースを保留するための管理設定が 2 つあり、どちらも Claude Code v2.1.283 以降が必要です。

- `deniedModels`: ブロックするモデルをリストします。リストされたモデルは、`availableModels` が許可している場合でもブロックされ、このキーは許可リストがまったくない場合でも機能します。どのエントリにもブロックされないリリースは許可されたままです
- `availableModelsMatch`: これを `"exact"` に設定すると、`availableModels` 内の各モデル ID は、その ID で指定されたバージョンのみを許可します。リストされたモデル ID の新しいバージョンは、リストに追加するまでブロックされたままになります

以前のバージョンはどちらのキーも無視するため、それらのバージョンが起動しないように `requiredMinimumVersion` も設定してください。

この例では、Opus と Sonnet のモデルを許可し、日付付きの ID やプロバイダー固有の ID を含むあらゆる表記の Opus 5.5 をブロックします。

```
{
  "availableModels": ["opus", "sonnet"],
```

(次のページに続く)

(前のページからの続き)

```
  "deniedModels": ["claude-opus-5-5"]
}
```

ブロックされたモデル（deniedModels で指定されたもの、または "exact" リストで省略されたもの）は、**許可リストが適用される** (ページ 732) すべての場所で、ブロックされた選択として扱われます。/model ピッカーから非表示になり、/model <name> はそれを拒否します。--model、ANTHROPIC_MODEL、または model 設定でブロックされたモデル ID を指定すると、Claude Code は起動時にそれを除外し、代わりに Default オプションを解決します。**フック** (ページ 443) やバックグラウンドのリクエストが deniedModels によってブロックされたモデルを指定した場合（エージェントフックの model フィールドなど）、そのリクエストは代わりにセッションのモデルで実行されます。

Default オプションも、*enforceAvailableModels* (ページ 736) を設定しているかどうかにかかわらず、両方のキーに従います。空でない availableModels とともにそれを設定している場合、ブロックされたデフォルトは許可リスト外のモデルとしてカウントされます。それ以外の場合、ブロックされたモデルに解決されることになる Default オプションは、次の順序で段階的に切り替わります。

1. 同じファミリーの許可された最新バージョン
2. より低コストの各ファミリーの許可された最新モデル（順に Sonnet、次に Haiku）
3. 許可されたモデルを指定する最初の availableModels エントリ

これらのいずれも許可されていない場合、Default オプションで開始するセッションは、修正すべきキーを示すエラーとともに起動を拒否します。"exact" リストが Default オプションに影響するのは、管理設定の availableModels リストが少なくとも 1 つのモデルまたはファミリーを指定している場合のみです。

Claude Code は、両方のキーを管理設定からのみ読み取ります。ユーザー設定、プロジェクト設定、ローカル設定、または --settings でいずれかを設定した場合、Claude Code は警告を表示してそれを無視します。

28.2.8 組織のモデル制限

Claude Enterprise プランの組織管理者は、claude.ai 管理コンソールで個々のモデルを無効にすることで、メンバーが実行できるモデルを制限します。この制限は、Claude Code の認証時にアカウントのエンタイトルメントとともに配信され、設定内の availableModels リストとは別のものです。また、サーバーはセッション作成時に同じ制限を独立して適用します。Claude Code v2.1.187 以降が必要です。

この制限は、メンバーがサインインした場合、または自身の API キーを使用した場合に適用されます。組織サービスキーなどの組織スコープの認証情報はユーザーに紐付いていないため、制限は適用されません。

Claude Console にはモデル制限の制御機能はありません。Anthropic API を通じてメンバーが認証する組織を含め、Claude Enterprise プランを持たない組織は、代わりに管理設定の *availableModels* (ページ 732) でモデルを制限し、Default オプションをカバーするために *enforceAvailableModels* (ページ 736) を追加します。各サーフェスがこれらの設定をどのように受け取り適用するかについては、**サーフェスごとの適用範囲** (ページ 735) を参照してください。

制限されたモデルは /model ピッカーから非表示になります。--model、ANTHROPIC_MODEL 環境変数、または model 設定で名前を指定してそれを選択すると、Model "<name>" is restricted by your organization's settings. Using <model> instead. という通知が表示され、セッションは許可されたモデルで開始されます。制限されたモデルに対して /model <name> と入力すると、Model '<name>' is

`restricted by your organization's settings. Run /model to choose a different model.` と表示されて拒否され、セッションは現在のモデルを維持します。

opus などの**モデルファミリーのエイリアス** (ページ 732) は、組織がそのモデルを許可している場合、通常のモデルに解決されます。組織がそのモデルを制限している場合、Claude Code は組織が許可するそのファミリーの最新バージョンに置き換え、同じ置き換え通知を表示します。`/model <alias>` が拒否されるのは、そのファミリーのすべてのバージョンが制限されている場合のみです。その場合でも、`--model`、`ANTHROPIC_MODEL`、または `model` 設定で設定されたエイリアスは、起動時に置き換えられます。v2.1.205 より前は、古いバージョンが許可されていても、ファミリーエイリアスは最新のリリースバージョンのみに基づいて置き換えまたは拒否されていました。

制限は組織全体またはロールごとに適用されます。

- 組織レベルでモデルを無効にすると、すべてのメンバーからそのモデルが削除されます。
- ロールレベルのアクセスでは、カスタムロールごとに異なるモデルを付与でき、複数のロールを持つメンバーは、いずれかのロールが付与するモデルを使用できます。
- Haiku モデルは常に利用可能で無効にできないため、すべてのメンバーは少なくとも 1 つの使用可能なモデルを保持します。
- アクセスの変更は、約 1 分以内に新しいリクエストに反映されます。`/model` ピッカーには、次回のセッション開始時に反映されます。

両方の制限は同時に適用されます。モデルが選択可能になるのは、`availableModels` で許可されていて、かつ組織によって制限されていない場合のみです。組織の制限が適用されるのは、Anthropic API および *LLM* **ゲートウェイ** (ページ 841) のデプロイ上のセッションのみです。その他のプロバイダーでは、代わりに `availableModels` を使用してください。

28.3 組織のデフォルトモデル

Claude Enterprise プランの組織管理者は、claude.ai の管理コンソールから、組織全体またはカスタムロールごとに Claude Code メンバー向けのデフォルトモデルを設定できます。設定されている場合、Default オプションはそのモデルに解決されます。Claude Code v2.1.196 以降が必要です。

`/model` ピッカーの Default 行には、組織のデフォルトモデルの名前が Org default というラベル付きで表示されます。管理者が組織全体に対してデフォルトを設定した場合でも、ユーザーのロールに対して設定した場合でも、ラベルは Org default と表示されます。ロールのデフォルトはそのカスタムロールのメンバーに適用され、組織全体のデフォルトよりも優先されます。ユーザーの複数のロールで異なるデフォルトが設定されている場合は、最も高性能なモデルが適用されます。

組織のデフォルトは出発点であり、制限ではありません。次の選択は組織のデフォルトよりも優先されます。

- `--model` フラグおよび `ANTHROPIC_MODEL` 環境変数
- 管理設定内の `model` の値、または `--settings` で指定された `model` の値
- ユーザー、プロジェクト、またはローカル設定内の `model` の値（`/model` で保存したモデルを含む）

管理者は、組織のデフォルトがユーザーの選択を上書きするように設定することもできます。上書きが有効な場合、組織のデフォルトはユーザー、プロジェクト、ローカル設定内の `model` の値よりも優先されるため、`/model` で保存したモデルは現在のセッションにのみ適用され、次回の起動時には組織のデフォルトに戻ります。ユーザーの選択が異なる場合、`/model` には `Your organization's default (<model>)`

`applies on restart` と表示されます。上書きが有効な場合でも、`--model` フラグ、`ANTHROPIC_MODEL`、管理設定、`--settings` は引き続き優先されます。

メンバーが選択できるモデルを制限するには、代わりに**組織のモデル制限** (ページ 739) または *`available Models`* (ページ 732) を使用してください。

Claude Code は起動時に一度だけ組織のデフォルトを読み込むため、セッション中に管理者がデフォルトを変更した場合は、次回の起動時に反映されます。

組織のデフォルトがユーザーの選択を上書きしない場合、管理者がデフォルトを変更した後の最初の対話型起動時に、ユーザー設定から `model` キーが一度だけ削除され、新しいデフォルトが適用されます。ファイル内のそれ以外の内容は変更されず、その起動後に `/model` で保存したモデルは保持されます。

組織のデフォルトは、採用される前に次の制限チェックを通過します。

- デフォルトのプレフィックスマッチングでは、*`availableModels`* (ページ 732) 単体では組織のデフォルトに適用されないため、許可リストに含まれない組織のデフォルトも引き続き適用されます。*`enforceAvailableModels`* (ページ 736) も設定されている場合は、許可リストに含まれない組織のデフォルトも許可リストの最初のエントリに再マッピングされます
- **組織のモデル制限** (ページ 739) によってユーザーのアカウントで拒否されている組織のデフォルトは、同じファミリー内で許可されている最新のモデルに置き換えられます。そのファミリーのすべてのバージョンが制限されている場合は、より低コストのファミリーに置き換えられます
- `deniedModels` または `"exact"` リストによってブロックされる組織のデフォルトについては、**特定のモデルまたはバージョンをブロックする** (ページ 738) を参照してください
- ユーザーのアカウントでまったく利用できない組織のデフォルトはスキップされ、Default オプションは**組織のデフォルトがない場合** (ページ 742) と同様に解決されます

v2.1.199 以降では、組織のデフォルトがユーザーのアカウントタイプの通常のデフォルトとは異なるモデルファミリーである場合、`/model` ピッカーにはその通常のファミリー用の行が別途残るため、セッション中にそのファミリーへ切り替えることもできます。v2.1.196 から v2.1.198 では、この行はピッカーに表示されません。

組織のデフォルトは、Anthropic API で認証されたセッションにのみ適用されます。*LLM* **ゲートウェイ** (ページ 841) のデプロイを含むその他の環境でデフォルトを設定するには、代わりに管理設定の `model` キーを使用してください。

28.4 組織の effort 上限

組織は、*effort* **レベル** (ページ 747) に 2 つの方法で上限を設定できます。Claude Enterprise プランでは、組織の管理者が以下で説明するロールごとの effort 上限を設定します。Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry を含むすべてのプランとプロバイダーでは、代わりに `maxEffortLevel` 管理設定によってクライアント側で effort に上限が設定されます。1 つのモデルに両方が適用される場合は、低いほうの上限が適用されます。

Claude Enterprise プランの組織管理者は、ロールレベルの**組織のモデル制限** (ページ 739) とあわせて、カスタムロールごとにモデル単位で *effort* **レベル** (ページ 747) の最大値を設定できます。上限を超えるレベルは `/effort` ピッカーに表示されず、`--effort` または `/effort` でより高いレベルを指定した場合は、代わりに上限のレベルで実行されます。対話型セッションとプレーンテキストの `--print` 実行では、要求されたレベルと適用されたレベルを示す警告が表示されます。`json` または `stream-json` 出力の場合やバックグ

ラウンドエージェントでは、上限への制限は警告なしで適用されます。上限はモデルごとに設定されるため、モデルを切り替えると利用可能なレベルが変わることがあります。ユーザーの複数のロールが同じモデルを許可している場合は、最も制限の緩い上限が適用されます。Claude Code v2.1.195 以降が必要です。

effort 上限は**組織のモデル制限** (ページ 739) と一緒に配信され、同じセッションに適用されます。

28.5 特殊なモデルの動作

28.5.1 `default` モデル設定

`default` の動作はアカウントの種類によって異なります。

- **Pro、Max、Team、Enterprise、Anthropic API**：デフォルトは Opus 5.5
- **Claude Platform on AWS、Amazon Bedrock、Google Cloud's Agent Platform**：デフォルトは Opus 5.5
- **Microsoft Foundry**：デフォルトは Sonnet 4.5

v2.1.280 より前は、`default` は Pro と Team Standard では Sonnet 5 に、Max、Team Premium、Enterprise、Anthropic API では Opus 5 に解決され、Claude Platform on AWS、Amazon Bedrock、Google Cloud's Agent Platform では v2.1.219 以降 Opus 5 に解決されていました。v2.1.219 より前は、`default` は Anthropic API、Max、Team Premium、Enterprise の従量課金では v2.1.154 以降 Opus 4.8 に解決され、Claude Platform on AWS、Amazon Bedrock、Google Cloud's Agent Platform では v2.1.207 以降 Opus 4.8 に解決されていました。v2.1.207 より前は、`default` は Claude Platform on AWS では Opus 4.7 に、Amazon Bedrock と Google Cloud's Agent Platform では Sonnet 4.5 に解決されていました。

管理者が**組織のデフォルトモデル** (ページ 740) を設定している場合、`default` は上記のアカウント種類ごとのデフォルトではなく、そのモデルに解決されます。Claude Code v2.1.196 以降が必要です。また、`default` は、該当セクションに記載された条件のもとで *`ANTHROPIC_DEFAULT_MODEL`* (ページ 731) で設定したモデルに解決されることや、**アカウントに記録されている** (ページ 728) モデルに解決されることもあります。

アカウントに何も記録されておらず、管理設定で *Default* **モデルに許可リストが強制** (ページ 736) されていて、かつアカウント種類ごとのデフォルトが `availableModels` に含まれていない場合、`default` は上記のアカウント種類ごとのデフォルトではなく、強制された Default に解決されます。組織のデフォルトと強制の両方が適用される場合は、まず組織のデフォルトがアカウント種類ごとのデフォルトを置き換え、その後に強制が適用されます。許可リストに含まれる組織のデフォルトはそのまま維持され、リスト外のものは強制された Default に解決されます。

Fable モデルは、どのプランやプロバイダーでもアカウント種類ごとのデフォルトにはなりません。`/model` で Fable モデルを選ぶと、ユーザー設定に選択中のモデルとして保存されるため、以降のセッションはそのモデルで開始されます。v2.1.257 で Claude Code が保存済みの Fable 5 の選択に対して行う一度限りの変更については、*Fable* **を使用する** (ページ 727) を参照してください。

28.5.2 `opusplan` モデル設定

`opusplan` モデルエイリアスは、自動化されたハイブリッドアプローチを提供します。

- **plan モードの場合**：複雑な推論やアーキテクチャ上の判断に `opus` を使用します
- **実行モードの場合**：コード生成と実装のために自動的に `sonnet` に切り替えます

これにより、計画には Opus の推論能力を、実行には Sonnet の効率性を組み合わせられます。

plan モードの Opus フェーズは `opus` モデル設定と同じコンテキストウィンドウを使用し、実行フェーズは `sonnet` と同じウィンドウを使用します。`opus` と `sonnet` が、Anthropic API 上の現行モデルのようにデフォルトで *1M* **コンテキストウィンドウ** (ページ 752) で動作するモデルに解決される場合は、両方のフェーズがそのウィンドウで動作します。そうでない場合に両方のフェーズで 1M コンテキストを要求するには、たとえば `/model opusplan[1m]` のように、**モデルを** (ページ 728) `opusplan[1m]` に設定します。`/model` での設定には Claude Code v2.1.265 以降が必要です。それより前のバージョンでは、代わりに `--model` フラグまたは `model` 設定を使用してください。

`availableModels` (ページ 732) が最新の Opus を除外しつつ古いバージョンを許可している場合（たとえば `["sonnet", "claude-opus-4-6"]`）、`opusplan` は許可されている最新の Opus を計画に使用し、すべての Opus が除外されている場合にのみ Sonnet のままになります。同様に、通常は plan モードで Sonnet にアップグレードされる Haiku セッションは、許可されている最新の Sonnet を使用し、すべての Sonnet が除外されている場合にのみ Haiku のままになります。v2.1.205 より前は、アップグレード先ファミリーの最新バージョンが除外されていると、許可リストが古いバージョンを許可していても、plan モードはセッションのモデルのままでした。

許可されている古いバージョンへの置き換えは、Anthropic API と Claude Platform on AWS に適用されます。デプロイでプロバイダー固有のモデル ID を使用する Amazon Bedrock、Google Cloud's Agent Platform、Microsoft Foundry、Mantle では、アップグレード先のモデルが除外されている場合、plan モードはセッションのモデルのままになります。

計画の境界で切り替えるのではなく、タスクの途中で Claude が 2 つ目のモデルに相談するタイミングを判断するハイブリッドアプローチについては、advisor ツールを参照してください。

28.5.3 フォールバックモデルチェーン

プライマリモデルが過負荷状態、利用不可、またはその他の再試行不可能なサーバーエラーを返した場合、Claude Code はリクエストを失敗させる代わりにフォールバックモデルに切り替えることができます。認証、請求、レート制限、リクエストサイズ、トランスポートのエラー、および組織のポリシーチェックによる拒否では切り替えは発生せず、通常の再試行とエラー処理に従います。*Amazon Bedrock* (ページ 871) または *Google Cloud's Agent Platform* (ページ 887) がアカウントで呼び出せないモデルを拒否した場合は切り替えが発生します。Claude Code はこれを認証エラーではなく、モデルが利用不可であるものとして扱います。

1 つ以上のフォールバックモデルを設定すると、Claude Code はそれらを順に試し、切り替え時に通知を表示します。切り替えは現在のターンにのみ有効なため、次のメッセージでは再びプライマリモデルが最初に試されます。Claude Code は重複を除いたうえでチェーンを 3 モデルまでに制限し、それを超えるエントリは無視します。

1 つのセッションにチェーンを設定するには、カンマ区切りのリストを受け付ける `--fallback-model` フラグを使用します。

```
claude --fallback-model sonnet,haiku
```

セッションをまたいでチェーンを保持するには、**設定** (ページ 711) で `fallbackModel` を配列として設定します。

```
{
  "fallbackModel": ["claude-sonnet-5", "claude-haiku-4-5"]
}
```

`--fallback-model` フラグは `fallbackModel` 設定より優先されます。各エントリにはモデル名またはエイリアスを指定でき、`"default"` はデフォルトモデルに展開されます。

Claude Code は起動時にチェーンを確認せず、`/status` にも表示されません。切り替えが発生したときに表示される通知が、フォールバックが設定されていることを示す最初の目に見えるサインです。

リクエストがフェイルオーバーすると、Claude Code はいずれかのエントリがリクエストを受け付けるまで、各エントリを順に試します。設定に固定された廃止済みモデルなど、到達できないエントリも同様に次のエントリへフェイルオーバーします。Claude Code は、この順次試行を始める前に 2 種類のエントリを除外します。

- **許可リスト外**：Claude Code はチェーンを読み込む際に、*availableModels* (ページ 732) で許可されていないエントリを除外します。
- **コンテキスト圧縮中のより小さいコンテキストウィンドウ**：チェーンはコンテキスト圧縮にも適用されますが、Claude Code はプライマリモデルより小さいコンテキストウィンドウを持つモデルにはフォールバックしません。そこで要約すると、会話の一部が先に切り捨てられてしまうためです。すべてのフォールバックがより小さい場合、圧縮は元のエラーを表示し、再試行できます。

Claude Code はチェーンを**サブエージェント** (ページ 317) にも適用します。サブエージェントのリクエストがフェイルオーバーすると、Claude Code は設定されたフォールバックモデルを順に試し、サブエージェントはリクエストを受け付けたモデルで処理を続行します。セッションのモデルは変わりません。v2.1.247 より前は、チェーンの対象となる失敗が発生するとサブエージェントが終了していました。

28.5.4 自動モデルフォールバック

このセクションでは、Fable モデル、Opus 5.5、Sonnet 5.5、Opus 5 からのコンテンツに基づくフォールバックについて説明します。モデルが過負荷状態または利用不可の場合の可用性に基づくフォールバックについては、**フォールバックモデルチェーン** (ページ 743) を参照してください。

Fable モデル、Opus 5.5、Sonnet 5.5、Opus 5 は安全性分類器とともに動作し、分類器が最も頻繁に警告するのはサイバーセキュリティと生物学に関するコンテンツです。分類器がリクエストを警告し、警告されたカテゴリにフォールバックモデルがある場合、Claude Code はそのモデルでリクエストを再実行し、トランスクリプトに通知を表示します。この 2 つのカテゴリについて、フォールバックモデルは拒否したモデルによって異なります。

- **Fable 5.1、Fable 5、Opus 5.5**：生物学で警告されたリクエストは Opus 5 で、サイバーセキュリティで警告されたリクエストは Opus 4.8 で再実行されます。
- **Sonnet 5.5**：サイバーセキュリティで警告されたリクエストは Sonnet 5 で再実行されます。Sonnet 5.5 には生物学のフォールバックモデルがないため、生物学で警告されたリクエストは代わりに拒否で終了します。
- **Opus 5**：サイバーセキュリティで警告されたリクエストは Opus 4.8 で再実行されます。Opus 5 は独自の生物学分類器をフォールバックモデルなしで実行するため、生物学で警告されたリクエストは代わりに拒否で終了します。

Amazon Bedrock、Google Cloud's Agent Platform、Microsoft Foundry では、Claude Code はこれらのターゲットを代わりにデプロイのモデル ID を通じて解決します。*Bedrock、Agent Platform、Foundry* **でフォールバックを有効にする** (ページ 746) を参照してください。

フォールバック後、セッションはフォールバックモデルで続行されます。元のモデルに戻るには、*`/model`* (ページ 728) を実行します。

カテゴリに基づくフォールバックには Claude Code v2.1.219 以降が必要です。v2.1.219 より前は、警告された Fable 5 のリクエストはすべてプロバイダーのデフォルトの Opus モデルで再実行され、Opus 5 はフォールバック元ではありませんでした。

フォールバックモデルは *`availableModels`* (ページ 732) と照合されます。ブロックされている場合、フォールバックは発生しません。拒否は通常のエラーとして表示され、セッションのモデルは変わりません。

フォールバック後の effort レベル

Claude Code がセッションをフォールバックモデルに切り替える際は、そのモデルのデフォルトの effort ではなく、警告されたリクエストが実行されていた effort レベルを維持します。たとえば、デフォルトの `medium` で動作している Opus 5.5 のセッションが Opus 4.8 にフォールバックした場合、Opus 4.8 のデフォルトは `high` ですが、`medium` のままになります。

次のような場合は、別のレベルが適用されます。

- **設定または組織のデフォルト**：フォールバックモデルに適用される設定内のレベル、または組織がそのモデルに設定したデフォルトの effort が代わりに適用されます。
- **ユーザー自身による変更**：effort レベルを選択したり、`/model` でモデルを選んだり、後でセッションを再開したりすると、警告されたリクエストのレベルは引き継がれなくなります。
- **スキルの effort**：スキルの `effort` フロントマターが警告されたリクエストに設定したレベルはそのターンに適用され、以降のターンは *effort* **の解決順序** (ページ 747) がフォールバックモデルに与えるレベルで実行されます。

セッションヘッダーには、有効なレベルがモデル名の横に表示されます。変更するには、セッション内で `/effort` を実行します。

フォールバックのきっかけを確認する

フォールバックは、特殊な内容を何も送信する前に、セッションの最初のリクエストで発生することがあります。最初のリクエストには、CLAUDE.md の内容や git status などのワークスペースのコンテキストが含まれるためです。セキュリティや生物学に関する資料を含むリポジトリでは、そのコンテキストだけで分類器が反応することがあります。

カスタマイズがきっかけかどうかを確認するには、`claude --safe-mode` でセッションを開始します。これにより、CLAUDE.md、スキル、MCP サーバー、フックなどのカスタマイズが無効になります。git status やディレクトリ名はカスタマイズではないため、引き続き含まれます。

切り替える前に確認する

自動的に切り替えるのではなく、リクエストが警告されるたびにどうするかを判断したい場合は、`/config` を実行して **Switch models when a message is flagged** をオフにするか、設定ファイルで `switchModelsOnFlag` を `false` に設定します。すると、警告されたリクエストはセッションを一時停止し、フォールバックモデルに切り替えるか、プロンプトを編集して現在のモデルで再試行するかの 2 つの選択肢を表示します。

次の場合は動作が異なります。

- Opus 5 や Sonnet 5.5 での生物学の警告のように、警告されたカテゴリにフォールバックモデルがない場合、Claude Code は確認を表示せず、リクエストは拒否で終了します。
- 両方のモデルが同じリクエストを警告した場合は、プロンプトを編集して再試行するか、新しいセッションを開始できます。
- モバイルアプリ上の**クラウドセッション** (ページ 223) では、編集して再試行することはできません。モデルを切り替えるか、デスクトップのブラウザまたはデスクトップアプリからセッションを続行してください。
- 確認を表示できない**非対話モード** (ページ 51) や SDK 統合では、警告されたリクエストは代わりに拒否でターンを終了します。
- フォールバック先が *`availableModels`* (ページ 732) によってブロックされている場合、Claude Code は確認を表示しません。ターゲットがブロックされている場合の自動フォールバックと同様に、警告されたリクエストは拒否で終了します。

Bedrock、Agent Platform、Foundry でフォールバックを有効にする

Amazon Bedrock (ページ 863)、*Google Cloud's Agent Platform* (ページ 881)、*Microsoft Foundry* (ページ 891) では、モデル ID がプロバイダー固有であるため、自動フォールバックは Claude Code が関係する各モデルを識別できる場合にのみ動作します。

- Claude Code が現在のモデルをフォールバック元として認識する必要があります。Fable 5.1 と Fable 5 は、モデル ID に `claude-fable-5` が含まれる場合、`ANTHROPIC_DEFAULT_FABLE_MODEL` の値と一致する場合、または *`modelOverrides`* (ページ 761) でマッピングされている場合に認識されます。Opus 5.5、Sonnet 5.5、Opus 5 は、プロバイダーのモデル ID または *`modelOverrides`* (ページ 761) のマッピングによって認識されます。
- どのモデルが拒否したかにかかわらず、Opus のターゲットがデプロイで解決される必要があります。`ANTHROPIC_DEFAULT_OPUS_MODEL` を設定するか、プロバイダーのモデルリストに Opus 4.8 のエントリを残してください。どちらもない場合、Sonnet 5.5 を含むすべてのフォールバック元モデルでフォールバックはオフのままとなり、警告されたリクエストは拒否で終了します。
- 警告されたカテゴリのフォールバックモデルがデプロイで解決される必要があります。Fable モデル、Opus 5.5、Opus 5 からの場合、`ANTHROPIC_DEFAULT_OPUS_MODEL` を設定すると、フォールバックがあるすべてのカテゴリについて、警告されたリクエストはそのモデルで再実行されます。ただし、Opus 5 での生物学の警告は引き続き拒否で終了します。設定しない場合、サイバーセキュリティで警告されたリクエストは Opus 4.8 のエントリで再実行され、Fable モデルまたは Opus 5.5 からの生物学で警告されたリクエストは Opus 5 のエントリで再実行されます。Sonnet 5.5 からの場合、サイバーセキュリティで警告されたリクエストは `ANTHROPIC_DEFAULT_SONNET_MODEL` で設定したモデルで再実行されるか、設定しない場合はプロバイダーのモデルリストにある Sonnet 5 のエントリで再実行されます。

いずれかのモデルを識別できない場合、Claude Code は切り替えません。警告されたリクエストは拒否メッセージで終了し、*/model* (ページ 728) でモデルを切り替えて再試行できます。両方のモデルを識別できるようにするには、フォールバック元モデルに応じて次の固定設定を行います。

- **Fable モデル**：Claude Code がフォールバック元として認識できるよう、`ANTHROPIC_DEFAULT_FABLE_MODEL` に Fable のモデル ID を設定します。
- **すべてのフォールバック元モデル**：フォールバックをオンにし、警告されたカテゴリにターゲットを与えるため、`ANTHROPIC_DEFAULT_OPUS_MODEL` に Opus のモデル ID を設定します。Opus ファミリー以外のモデルや、拒否したモデル自体を指定した場合は、拒否がそのまま残ります。
- **Sonnet 5.5**：Opus の固定設定に加えて、`ANTHROPIC_DEFAULT_SONNET_MODEL` を設定するか、プロバイダーのモデルリストに Sonnet 5 のエントリを残して、リクエストを再実行するモデルを用意します。Sonnet ファミリー以外のモデルや Sonnet 5.5 自体を指定した Sonnet の固定設定では、拒否がそのまま残ります。

セキュリティ研究と生物学のワークロード

ペネトレーションテスト、Capture the Flag（CTF）演習、生物学に関連するコードベースなど、攻撃的セキュリティや生物学のワークロードでは、フォールバックが頻繁に、多くの場合最初のリクエストで発生します。Fable 5.1、Fable 5、Opus 5.5 で本格的な生物学の作業を行う場合、Claude Code は最初に警告されたリクエストの時点でセッションを Opus 5 に移行し、Opus 5 には生物学のフォールバックがないため、その後の生物学で警告されたリクエストはそこで拒否となります。Opus 5 と Sonnet 5.5 では、最初に警告されたリクエストからその拒否が発生します。

これはこれらの分野における想定どおりのルーティングであり、アカウントに対する警告ではありません。組織がこの作業に Fable クラスの能力を必要とする場合は、Anthropic のアカウントチームに信頼済みアクセスプログラムについてお問い合わせください。

28.5.5 effort レベルを調整する

effort レベルはアダプティブ推論を制御します。アダプティブ推論では、タスクの複雑さに基づいて、各ステップで思考するかどうか、どの程度思考するかをモデルが判断します。低い effort は単純なタスクに対してより高速かつ低コストで、高い effort は複雑な問題に対してより深い推論を提供します。

利用可能な effort レベルはモデルによって異なります。ここに記載されていないモデルは effort をサポートしていません。

モデル	レベル
Fable 5.1 と Fable 5	`low`、`medium`、`high`、`xhigh`、`max`
Opus 5.5、Sonnet 5.5、Opus 5、Sonnet 5、Opus 4.8、Opus 4.7	`low`、`medium`、`high`、`xhigh`、`max`
Opus 4.6 と Sonnet 4.6	`low`、`medium`、`high`、`max`

アクティブなモデルがサポートしていないレベルを設定した場合、Claude Code は設定したレベル以下でサポートされている最も高いレベルにフォールバックします。たとえば、Opus 4.6 では `xhigh` は `high` として動作します。組織またはユーザー自身の設定によって、モデルが提供するレベルに上限を設けることもできます。**組織の *effort* 制限** (ページ 741) を参照してください。

Claude Code は、次の順序でセッションの effort レベルを解決し、最初に該当したものを採用します。

1. 明示的な選択：CLAUDE_CODE_EFFORT_LEVEL 環境変数、--effort を付けた起動、またはセッション内での /effort（**非対話の *effort* は効果の範囲が狭くなります** (ページ 748)）
2. 設定：モデルに対して保存したレベルまたは effortLevel キー。これらの間および設定ファイル間の優先順位は modelSettings に記載されています
3. モデルのデフォルトの effort：effort をサポートするすべてのモデルで high。ただし、Opus 5.5 と Sonnet 5.5 のデフォルトは medium、Opus 4.7 のデフォルトは xhigh です。また、組織が**組織のデフォルトモデル** (ページ 740) にデフォルトの effort レベルを設定している場合、そのモデルを実行するときはそのレベルがデフォルトになります。自動モデルフォールバックの後に適用されるレベルについては、**フォールバック後の *effort* レベル** (ページ 745) を参照してください

Opus 5.5 は、上記のいずれかのソースでレベルが設定されていない限り medium で開始され、ユーザー設定ファイルのトップレベルの effortLevel は Opus 5.5 には適用されません。このキーは、Claude Code がモデルごとにレベルを保存するようになる前に /effort が書き込んでいた古い形式です。Opus 5、Fable 5.1、およびそれ以前のモデルでは以前と同様に適用され続けますが、Opus 5.5 とそれ以降にリリースされたモデルは、/effort または /model ピッカーでレベルを選ぶまで、それぞれのデフォルトで開始されます。プロジェクト設定、ローカル設定、管理設定のトップレベルの effortLevel、または --settings で渡されたものは、すべてのモデルに適用されます。

自分のマシン上の対話セッションで low、medium、high、xhigh を設定する場合、確定の方法によって有効期間を選べます。

- /effort スライダーまたは /model ピッカーでの Enter、または /effort の後にレベルを入力：レベルをデフォルトとして保存し、以降のセッションにも適用します
- /effort スライダーまたは /model ピッカーでの s：このセッションにのみレベルを適用します。Claude Code v2.1.257 以降が必要です

Claude Code はユーザー設定の modelSettings キーの下にモデルごとのレベルを保存するため、各モデルはそれぞれ独自の保存済みレベルを保持します。

max は最も深い推論レベルです。CLAUDE_CODE_EFFORT_LEVEL 環境変数で設定しない限り、Claude Code は max を現在のセッションにのみ適用します。

注釈:

Remote Control で接続したスマートフォンやブラウザの effort コントロールから選んだレベルは、そのセッションにのみ適用されます。

-p **実行** (ページ 695) で /effort を使ってレベルを設定した場合、Claude Code はそのセッションにのみ適用し、デフォルトとしては保存しません。

/effort スライダーには **Ultracode** トグルもあります。Ultracode はモデルの effort レベルではなく Claude Code の設定です。オンにすると、Claude はセッションの effort レベルにかかわらず、本格的なタスクに対して動的ワークフローをオーケストレーションします。永続的に設定できる場所については、ultracode 設定を参照してください。

/effort または ultracode 設定で ultracode をオンまたはオフにしても、effort レベルは変わりません。

`--effort ultracode` フラグと Agent SDK の `effortLevel: "ultracode"` 値は、ultracode をオンにすると同時にレベルを `xhigh` に設定します。`/effort` スライダーや `/model` ピッカーでレベルを選んでも、ultracode はそのままです。

ultracode は次のいずれかの方法でオンにできます。

- `/effort`：`/effort ultracode` を実行すると現在のセッションでオンになり、`/effort ultracode off` を実行するとオフになります。`/effort` スライダーでは、`Tab` を押して **Ultracode** トグルを切り替え、`Enter` で適用します
- `--effort` **フラグ**：`claude --effort ultracode` で起動すると、`xhigh` の effort で ultracode をオンにした状態でセッションが開始されます
- `ultracode` **設定**：設定ファイル、`--settings`、または Agent SDK のコントロールリクエストで `"ultracode": true` を設定します。`applyFlagSettings()` リクエストは `effortLevel: "ultracode"` も受け付け、ultracode をオンにして effort レベルを `xhigh` に設定します

`/effort ultracode off` 形式、スライダーのトグル、`xhigh` 以外の effort レベルで ultracode をオンのまま維持することには、Claude Code v2.1.284 以降が必要です。v2.1.284 より前は、ultracode をオンにするとセッションが `xhigh` の effort に設定され、別のレベルを選ぶとオフになり、`xhigh` 未満の effort 上限があると利用できませんでした。

`--effort` フラグまたは Agent SDK の `effortLevel` 値に `ultracode` を渡すには、Claude Code v2.1.203 以降が必要です。v2.1.203 より前は、`--effort ultracode` は `Unknown --effort value 'ultracode'` を出力し、セッションはデフォルトの effort で開始されていました。

永続化される `effortLevel` 設定と `CLAUDE_CODE_EFFORT_LEVEL` 環境変数は `ultracode` を受け付けません。`CLAUDE_CODE_EFFORT_LEVEL` または *effort* **上限** (ページ 741) によってセッションのレベルが設定されている場合、ultracode はそのレベルでオンのままになります。

次の場合、Ultracode は利用できません。

- ワークフローがオフになっている
- モデルが `xhigh` の effort をサポートしていない

これらの場合、`--effort ultracode` は ultracode をオフにした状態で、モデルと上限が許容する最も高い effort レベル（最大 `xhigh`）でセッションを開始します。

effort レベルを選ぶ

各レベルは、トークン消費と能力のトレードオフです。デフォルトはほとんどのコーディングタスクに適しています。異なるバランスが必要な場合に調整してください。

レベル	使用する場面
`low`	ブレインストーミング、最初の下書き、名前の変更のような小さな変更など、結果を 1 つずつ確認する素早いやり取り
`medium`	Opus 5.5 と Sonnet 5.5 のデフォルトで、新機能の実装など、範囲が明確な日常的なエンジニアリング作業に適しています。その他のモデルでは、ある程度の知能と引き換えにできるコスト重視の作業でトークン使用量を削減します

次のページに続く

表 5 – 前のページからの続き

レベル	使用する場面
high	既存のコードベースのバグ修正など、検証が重要な作業やエッジケースが発生しやすい作業。Opus 5.5、Sonnet 5.5、Opus 4.7 を除くすべてのモデルのデフォルト
xhigh	より多くのトークン消費でより深い推論。Opus 4.7 のデフォルト
max	セキュリティ脆弱性の発見など、ユーザーの関与なしに Claude に取り組ませたい難しい問題。max は収穫逓減を示すことがあり、考えすぎる傾向があるため、広く採用する前にテストしてください
ultracode	レベルではなく Claude Code の設定：任意の effort レベルで、本格的なタスクごとに動的ワークフローを計画します

Opus 5.5 と Fable 5.1 でのテストでは、高いレベルの Claude は回答前により多くのエッジケースをテストし、作業のより多くの部分を検証しました。また、自らの判断でより多くの選択を行いました。低いレベルでは、Claude はより早く出発点を返しました。これは、結果を 1 つずつ確認して次のステップを方向付ける作業に適しています。同じタスクを各レベルで実行した例については、ブログの Using Claude Code: Spending your effort をお読みください。

effort のスケールはモデルごとに調整されているため、同じレベル名でもモデル間で同じ基礎値を表すわけではありません。

Opus 5.5 は**デフォルトで** *medium* (ページ 747) であり、Opus 5 のデフォルトである high より 1 段階低くなっています。Anthropic のテストでは、medium の Opus 5.5 は、コーディングやナレッジワークの評価で high の Opus 5 と同等以上の結果を示しています。同じレベルでは、Opus 5.5 は Opus 5 よりもターンあたりの思考量が多くなる傾向があります。Opus 5 から Opus 5.5 に移行する際は、Opus 5 で使用していたレベルを引き継ぐのではなく、medium から始めてください。自分の作業に対してレベルをテストするには、Opus 5.5 のプロンプティングガイドの Calibrate effort を参照してください。

一度限りの深い推論に ultrathink を使う

プロンプトのどこかに ultrathink を含めると、セッションの effort 設定を変更せずに、そのターンでより深い推論を要求できます。Claude Code はこのキーワードを認識し、コンテキスト内に指示を追加します。API に送信される effort レベルは変わりません。「think」、「think hard」、「think more」などのその他の表現は、Claude Code が通常のプロンプトテキストとしてそのまま渡し、キーワードとしては認識しません。

effort レベルを設定する

effort は次のいずれかの方法で変更できます。

- /effort：引数なしで /effort を実行するとインタラクティブなスライダーが開き、/effort の後にレベル名を続けると直接設定でき、/effort auto を実行するとアクティブなモデルの保存済みレベルがクリアされます。Claude が作業中でも実行でき、Claude Code がキャッシュの警告を表示した場合はそれを確認すると、Claude Code はターン内の次のリクエストに新しいレベルを適用します
- /model 内：モデルを選択する際に、左右の矢印キーで effort スライダーを調整します
- --effort フラグ：Claude Code の起動時にレベル名を渡して、1 つのセッションに設定します
- **環境変数**：CLAUDE_CODE_EFFORT_LEVEL にレベル名または auto を設定します

- **設定**：modelSettings でモデルごとのレベルを設定するか、レベルが設定されていないモデルのデフォルトとして effortLevel に low、medium、high、xhigh のいずれかを設定します。どちらのキーでも max はレベルとして受け付けられず、ultracode には専用の ultracode キーがあります
- **接続されたデバイスから**：Remote Control セッションで、スマートフォンまたはブラウザの effort コントロールからレベルを選びます。レベルは現在のセッションにのみ適用されます。Claude Code v2.1.234 以降が必要です
- **スキルとサブエージェントのフロントマター**：**スキル** (ページ 288) または**サブエージェント** (ページ 326) の markdown ファイルで effort を設定すると、そのスキルまたはサブエージェントの実行時に effort レベルを上書きします

フロントマターの effort は、そのスキルまたはサブエージェントがアクティブなときに適用され、セッションのレベルは上書きしますが、環境変数は上書きしません。maxEffortLevel または**組織の** *effort* **上限** (ページ 741) は、スキルやサブエージェントが実行されるレベルを引き続き制限します。

管理設定で effortLevel を設定した場合、Claude Code は *effort* **の解決順序** (ページ 747) の設定のステップでそれを適用し、ユーザーは引き続き /effort や --effort でレベルを変更できます。ユーザーを特定のレベル以下に制限するには、maxEffortLevel を設定します。

サポートされているモデルが選択されている場合、effort スライダーは /model に表示されます。現在の effort レベルはセッションヘッダーのモデル名の横にも「with low effort」のように表示されるため、/model を開かずにどの設定が有効かを確認できます。フッターにも、起動時と変更時に effort レベルが短時間表示されます。

アダプティブ推論と固定の思考予算

アダプティブ推論では、各ステップで思考が任意になるため、Claude は日常的なプロンプトにはより速く応答し、より深い思考はそれが役立つステップのために取っておけます。現在のレベルよりも思考の頻度を増やしたり減らしたりしたい場合は、プロンプトまたは CLAUDE.md で直接そう伝えることができます。モデルは effort 設定の範囲内でその指示に応えます。

Fable モデル、Sonnet 5 以降、Opus 4.7 以降は常にアダプティブ推論を使用します。固定の思考予算モードと CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING はこれらのモデルには適用されません。

Opus 4.6 と Sonnet 4.6 では、CLAUDE_CODE_DISABLE_ADAPTIVE_THINKING=1 を設定すると、MAX_THINKING_TOKENS で制御される以前の固定の思考予算に戻せます。環境変数を参照してください。

28.5.6 拡張思考

拡張思考は、Claude が応答する前に出力する推論です。**アダプティブ推論** (ページ 747) をサポートするモデルでは、effort レベルが思考量の主な制御手段です。以下の設定は、思考のオン/オフと表示方法を制御します。Anthropic API で思考をオフにしている場合、Opus 5 などその組み合わせを受け付けないことを Claude Code が把握しているモデルには、より高いレベルではなく effort high を送信します。

制御	設定方法
現在のセッションで切り替える	macOS では Option+T、Windows と Linux では Alt+T を押します

次のページに続く

表 6 – 前のページからの続き

制御	設定方法
グローバルのデフォルトを設定する	`/config` を実行して思考モードを切り替えます。`~/.claude/settings.json` に `alwaysThinkingEnabled` として保存されます
環境変数で無効にする	`MAX_THINKING_TOKENS=0` を設定します。これにより、Opus 5.5、Sonnet 5.5、Fable モデルを除き、Anthropic API で思考がオフになります。**サードパーティプロバイダー** (ページ 847) では、Claude Code は代わりに `thinking` パラメータを省略するため、アダプティブ推論モデルは引き続き思考する場合があります。その他の値は**固定の思考予算** (ページ 751) の場合にのみ適用されます

Opus 5.5、Sonnet 5.5、Fable モデルでは思考をオフにできません。これらのモデルでは、セッションのトグルと `/config` の行に切り替えの代わりに `Thinking can't be turned off` が表示され、保存済みの `alwaysThinkingEnabled: false` や `MAX_THINKING_TOKENS=0` は効果がありません。これらのモデルでは、effort レベルに基づいて、モデルがステップごとにどの程度思考するかを判断します。保存済みの設定は、それを受け付けるモデルに切り替えると再び適用されます。

Claude Code はデフォルトで思考の出力を折りたたみます。`Ctrl+O` を押して詳細モードを切り替えると、推論がグレーの斜体テキストで表示されます。Anthropic API 上の対話セッションはデフォルトで編集済みの思考ブロックを受け取るため、展開時に完全な要約を表示したい場合は、**設定** (ページ 711) で `showThinkingSummaries: true` を設定してください。折りたたまれていても編集済みであっても、生成されたすべての思考トークンに対して課金されます。

28.5.7 拡張コンテキスト

Fable 5.1、Fable 5、Sonnet 5 以降、Opus 4.6 以降、Sonnet 4.6 は、大規模なコードベースでの長いセッション向けに 100 万トークンのコンテキストウィンドウをサポートしています。

Anthropic API では、Fable 5.1、Fable 5、Sonnet 5 以降、Opus 4.7 以降は、Pro を含むすべてのプランで 1M ウィンドウで動作します。これらのモデルでは、1M ウィンドウのために `[1m]` バリアントを選択したり、使用クレジットをオンにしたりする必要はありません。Fable の利用自体は、一部のプランでは使用クレジットに請求される場合があります。*Fable* **と使用クレジット** (ページ 728) を参照してください。

Opus 4.6 と Sonnet 4.6 が 1M に到達するのは `[1m]` バリアントを通じてのみで、そのバリアントへのアクセスはプランによって異なります。Team Standard と Team Premium の両方のシートを含む Max、Team、Enterprise プランでは、1M コンテキストの Opus 4.6 はサブスクリプションに含まれています。1M コンテキストの Sonnet 4.6 は、Max を含むすべてのサブスクリプションプランで使用クレジットが必要です。

プラン	1M コンテキストの Opus 4.6	1M コンテキストの Sonnet 4.6
Max、Team、Enterprise	サブスクリプションに含まれる	使用クレジットが必要
Pro	使用クレジットが必要	使用クレジットが必要
API と従量課金	フルアクセス	フルアクセス

Claude Code がこれらのプラン要件を確認するのは、Anthropic API に直接接続する場合のみです。

`ANTHROPIC_BASE_URL` を *LLM* **ゲートウェイ** (ページ 842) に向け、保存済みの claude.ai ログインがアクティブな認証情報のままである場合、Claude Code はプランの使用クレジットを確認しません。`[1m]` オプションは `/model` で引き続き利用でき、リクエストが成功するかどうかはゲートウェイが判断します。v2.1.229 より前は、この構成でアカウントの使用クレジットを確認できない場合、Claude Code は `/model sonnet[1m]` を拒否していました。

`ANTHROPIC_BASE_URL` を *LLM* **ゲートウェイ** (ページ 841) やその他のプロキシに設定した場合、Claude Code は認識する各モデルに、Anthropic API 上と同じコンテキストウィンドウを割り当てます。Fable 5.1、Fable 5、Sonnet 5 以降、Opus 4.7 以降は、`[1m]` バリアントを選択しなくても 1M ウィンドウを使用でき、Opus 4.6 のように `[1m]` バリアントを通じてのみ 1M に到達するモデルは、バリアントなしでは 200K で動作します。Claude Code は、ゲートウェイやその背後のサーバーが強制するより低い制限を検出できません。ゲートウェイが 200K トークンを超えるリクエストを拒否する場合は、Claude Code を起動する環境で `CLAUDE_CODE_AUTO_COMPACT_WINDOW=200000` を設定して、すべてのモデルのセッションが**その境界で圧縮される** (ページ 754) ようにしてください。

1M コンテキストをオフにするには、`CLAUDE_CODE_DISABLE_1M_CONTEXT=1` を設定します。Claude Code はモデルピッカーから 1M のモデルバリアントを削除します。Sonnet 5 や Fable モデルなど、ネイティブで 1M ウィンドウを持つモデルでは、そのモデルのコンテキストウィンドウを 200K として扱います。

- 自動圧縮がオンの場合、セッションは**自動圧縮** (ページ 754) によって 200K の境界で圧縮されます。Claude Code は自動圧縮のウィンドウをモデルのコンテキストウィンドウに制限するため、自動圧縮のウィンドウを 200K より大きく設定しても制限は解除されません。
- 自動圧縮がオフの場合、セッションは圧縮されず、200K の境界でコンテキスト制限エラーにより停止します。

v2.1.223 より前は、Claude Code が 200K に制限していたのは Sonnet 5、Opus 4.8、Opus 5 のセッションのみでした。環境変数を参照してください。

1M コンテキストウィンドウは標準のモデル料金を使用し、200K を超えるトークンに対する割増料金はありません。拡張コンテキストがサブスクリプションに含まれるプランでは、使用量は引き続きサブスクリプションでカバーされます。使用クレジットを通じて拡張コンテキストにアクセスするプランでは、トークンは使用クレジットに請求されます。

アカウントが 1M コンテキストをサポートしている場合、最新バージョンの Claude Code では `/model` ピッカーにそのオプションが表示されます。表示されない場合は、セッションを再起動してください。サードパーティプロバイダーでは、デプロイが `ANTHROPIC_DEFAULT_*_MODEL` 変数で**モデルを固定** (ページ 758) していないか確認してください。

`[1m]` サフィックスは、モデルエイリアスや完全なモデル名とともに使用することもできます。

```
# Use the opus[1m] or sonnet[1m] alias
/model opus[1m]
/model sonnet[1m]

# Or append [1m] to a full model name
/model claude-opus-4-8[1m]
```

Sonnet 5.5 と Sonnet 5 のコンテキストウィンドウ

Anthropic API では、Sonnet 5.5 と Sonnet 5 は常に 1M コンテキストウィンドウで動作します。200K バリアントはなく、選択する `[1m]` サフィックスもなく、どのプランでも使用クレジットは不要です。セッションはウィンドウが埋まる前に、デフォルトで約 967K トークンで自動圧縮されます。別のしきい値を選ぶには、`CLAUDE_CODE_AUTO_COMPACT_WINDOW` を設定してください。

Claude Code は、*LLM* **ゲートウェイ** (ページ 841) やその他のカスタム `ANTHROPIC_BASE_URL` の背後でも、Sonnet 5.5 と Sonnet 5 に同じ 1M ウィンドウを割り当てます。ゲートウェイがより低い制限を強制する場合は、**ゲートウェイの背後でのコンテキストウィンドウ** (ページ 753) を参照してください。

次の設定では、代わりにウィンドウを 200K に制限します。

- `CLAUDE_CODE_DISABLE_1M_CONTEXT=1`：ネイティブで 1M ウィンドウを持つすべてのモデルのセッションを 200K ウィンドウに制限します。制限がどのように適用されるかについては**拡張コンテキスト** (ページ 752) を参照してください。コンテキストに上限を設ける必要があるデプロイに役立ちます。

28.6 コンテキストウィンドウと自動圧縮

自動圧縮ウィンドウとは、Claude Code が会話を圧縮する前にコンテキストウィンドウがどこまで埋まってよいかを示すものです。各メカニズムでコンテキスト圧縮が何を保持し何を破棄するかについては、コンテキスト圧縮後に残るものを参照してください。

28.6.1 自動圧縮ウィンドウを設定する

自動圧縮ウィンドウは次の場所で設定できます。

- **現在のモデルについて、現在のセッションとそれ以降のセッション**：`/autocompact 500k` のように、値を指定して `/autocompact` を実行します。Claude Code はこの値をユーザー設定の `modelSettings` に現在のモデル用として保存し、現在のセッションに適用します。管理設定など、より優先順位の高い**設定スコープ** (ページ 719) がそのモデルまたはすべてのモデルに対して独自のウィンドウを設定している場合、コマンドは値を保存しますが、セッションではそのスコープのウィンドウが維持され、コマンドはその旨を表示します。使用中のモデル向けに調整されたウィンドウに戻すには、`/autocompact auto` を実行します。v2.1.288 より前は、このコマンドはすべてのモデルに共通の 1 つのウィンドウを、トップレベルの `autoCompactWindow` として保存していました。
- **すべてのモデル**：設定ファイルで `autoCompactWindow` を設定します（例：`~/.claude/settings.json` に `"autoCompactWindow": 200000` を記述）。`/autocompact` でモデルごとに保存したウィンドウは、そのモデルについては同じファイル内のこのキーよりも優先されます。
- **1 回の起動のみ**：Claude Code の起動時に *`--autocompact`* (ページ 51) を渡します。このフラグは、保存済みの設定を変更することなく、その起動に限って設定を上書きします。また、`claude --autocompact auto` を実行すると、保存済みの設定に値があっても、調整済みのウィンドウでセッションが実行されます。`/autocompact` とは異なり、このフラグは管理設定など、より優先順位の高い設定スコープによって無効化されることはありません。
- **スクリプトおよびクラウド環境**：`CLAUDE_CODE_AUTO_COMPACT_WINDOW` を設定します。この環境変数が設定されている間は、コマンド、フラグ、設定よりも優先され、`/autocompact` はウィンドウを変更する代わりに上書きされていることを報告します。

コマンドとフラグには、100K から 1M トークンまでのウィンドウサイズを次のいずれかの形式で指定できます。

- `200000` のような単純なトークン数
- `500k` や `1M` のような `k` または `M` の接尾辞付きの値
- 100 から 1000 までの数値のみ（千単位を意味し、`200` は 200,000 に設定されます）

環境変数には単純なトークン数のみを指定できます。Claude Code は、ウィンドウの上限をモデルのコンテキストウィンドウに制限します。

28.6.2 デフォルトの自動圧縮のしきい値

自動圧縮ウィンドウを設定していない場合、Claude Code は会話がモデルのコンテキスト上限に達した時点で圧縮します。ただし、次のセッションは例外です。

- **クラウドセッション** (ページ 223) は、会話がモデルの上限に近づいた時点で圧縮します
- **拡張コンテキスト** (ページ 752) を使用しない Sonnet 4.6 と Opus 4.6 は 200K の境界で圧縮します。Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry など、200K のコンテキストウィンドウで実行される Opus 4.8 以降も同様です
- `CLAUDE_CODE_DISABLE_1M_CONTEXT=1` を設定すると、Sonnet 5 や Fable モデルなど、ネイティブで 1M のウィンドウを持つモデルは 200K の境界で圧縮します
- ネイティブの 1M ウィンドウで実行されるモデルは、ウィンドウが埋まる前に、デフォルトで約 967K トークンの時点で圧縮します。Anthropic API では、Sonnet 5、Fable モデル、Opus 4.7 以降がこれに該当します。Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry でどのモデルがこのウィンドウで実行されるかについては、**サードパーティのデプロイでモデルを固定する** (ページ 758) を参照してください。カスタムの `ANTHROPIC_BASE_URL` を使用している場合は、**ゲートウェイ経由のコンテキストウィンドウ** (ページ 753) を参照してください
- *LLM* **ゲートウェイ** (ページ 841) のエイリアスなど、Claude Code が認識できないモデル ID を使用するセッションは、Claude Code がその ID に対して想定するコンテキストウィンドウで圧縮します。**ゲートウェイまたはカスタムモデル *ID* のウィンドウを修正する** (ページ 755) を参照してください

28.6.3 ゲートウェイまたはカスタムモデル ID のウィンドウを修正する

LLM **ゲートウェイ** (ページ 841) やその他のカスタムデプロイでは、Claude Code がその ID を Claude モデルに解決するかどうかにかかわらず、モデル ID に対してモデルの実際のウィンドウとは異なるコンテキストウィンドウを想定することがあります。Claude Code が代わりに想定すべきウィンドウを `CLAUDE_CODE_MAX_CONTEXT_TOKENS` に設定してください。

この変数の適用方法は ID によって異なります。Claude Code は、ID が（大文字小文字を問わず）`claude-` で始まらない場合、または Google Cloud の Agent Platform で使用される `@YYYYMMDD` の日付のように、ID の読み取り時に Claude Code が取り除く接尾辞が付いている場合に、その ID をプロバイダー固有またはカスタムの表記として扱います。v2.1.259 より前の Claude Code は取り除かれた接尾辞を考慮しなかったため、日付の接尾辞が付いた認識できない `claude-` ID は、接尾辞のない `claude-` 名として扱われていました。

認識できないプロバイダー固有またはカスタムの表記、[1m] が付いた同じ表記、およびそれ以外のすべての ID は、次の 3 つの異なるケースとして扱われます。

- Claude Code がプロバイダー固有またはカスタムの表記を認識できるモデルに解決できず、ID に [1m] が含まれていない場合、この変数は直接適用され、宣言されたウィンドウでプロアクティブなコンテキスト圧縮が継続されます。
- Claude Code がプロバイダー固有またはカスタムの表記を認識できるモデルに解決できず、ID に（大文字小文字を問わず）[1m] が含まれている場合、Claude Code はその ID に 1M のウィンドウを想定し、この変数は単独では適用されません。プロアクティブなコンテキスト圧縮を維持しながらウィンドウを修正するには、CLAUDE_CODE_DISABLE_1M_CONTEXT=1 も設定してください。この変数を設定すると、Claude Code はその ID を [1m] のない同じ表記と同様にサイズ設定するため、CLAUDE_CODE_MAX_CONTEXT_TOKENS は、タグのない表記に適用される場合と同じ条件で適用されます。

  宣言されたウィンドウが 200K を超える場合、Claude Code は 200K の上限が適用されないことを示す起動時の警告を表示します。この構成では、この警告は想定どおりの動作です。
- ID が Claude Code の認識するモデルに解決される場合、または ID が（大文字小文字を問わず）Claude Code が取り除く接尾辞のない claude- 名のみの場合、この変数は DISABLE_COMPACT も設定したときにのみ有効になります。DISABLE_COMPACT はすべてのコンテキスト圧縮を無効にします。

  たとえば、anthropic/claude-opus-4-8、us.anthropic.claude-⋯-v1:0、日付付きの claude-sonnet-4-5@20250929 など、Claude Code が認識している Claude モデル名を含む ID は、そのモデルに解決されます。これには [1m] も含む ID も含まれます。CLAUDE_CODE_DISABLE_1M_CONTEXT が設定されている場合でも、Claude Code は claude-opus-4-8[1m] を Opus 4.8 に解決します。

Claude Code が認識できないモデル ID の場合、CLAUDE_CODE_DISABLE_UNKNOWN_MODEL_WINDOW_ENFORCEMENT=1 を設定すると、API が Claude Code の認識する長すぎるエラーで会話を拒否した後にのみ、Claude Code が圧縮するようになります。ゲートウェイがエラーを Claude Code の認識できない文言に書き換える場合、Claude Code はこの復旧処理を実行しません。

28.7 現在のモデルの確認

現在使用しているモデルは、2 つの場所で確認できます。

- **ステータスライン** (ページ 785) 内（設定されている場合）
- /status 内。アカウント情報も表示されます。

28.8 カスタムモデルオプションを追加する

ANTHROPIC_CUSTOM_MODEL_OPTION を使用すると、組み込みのエイリアスを置き換えることなく、/model ピッカーにカスタムエントリを 1 つ追加できます。これは、Claude Code がデフォルトで一覧表示しないモデル ID をテストする場合に便利です。LLM ゲートウェイのデプロイでは、CLAUDE_CODE_ENABLE_GATEWAY_MODEL_DISCOVERY=1 が設定されていると、Claude Code はゲートウェイの /v1/models エンドポイントからピッカーに項目を取り込めます。そのため、この環境変数が必要になるのは、検出が無効になっている場合、または検出で目的のモデルが返されない場合のみです。ゲートウェイのモデル検出を参照してください。

代わりに複数のモデルを、独自の順序で、選択したラベルを付けて一覧表示するには、modelPicker を設定します。その項目では、このラインナップが組み込みのラインナップを置き換える際に、ピッカーがどの行を保持するかを説明しています。

この例では、3 つの環境変数すべてを設定して、ゲートウェイ経由でルーティングされる Opus のデプロイを選択可能にします。Claude Code は起動時に環境変数を読み込むため、claude を起動する前に export を実行するか、既存のセッションを再起動して変数を反映させてください。

```
export ANTHROPIC_CUSTOM_MODEL_OPTION="my-gateway/claude-opus-5-5"
export ANTHROPIC_CUSTOM_MODEL_OPTION_NAME="Opus via Gateway"
export ANTHROPIC_CUSTOM_MODEL_OPTION_DESCRIPTION="Custom deployment routed through␣
↪the internal LLM gateway"
```

ANTHROPIC_CUSTOM_MODEL_OPTION_NAME と ANTHROPIC_CUSTOM_MODEL_OPTION_DESCRIPTION は省略可能です。

- 名前を省略した場合、Claude Code が *ID を認識する* (ページ 760) ときはエントリにモデルの名前が表示され、それ以外の場合はモデル ID が表示されます。
- 説明を省略した場合、Claude Code は Custom model (<model-id>) を使用します。

Claude Code はカスタムエントリを組み込みエントリの後に表示し、追加した modelPicker の行はその後に表示されます。

Claude Code は ANTHROPIC_CUSTOM_MODEL_OPTION に設定されたモデル ID の検証をスキップするため、API エンドポイントが受け付ける任意の文字列を使用できます。

availableModels (ページ 732) が設定されている場合は、カスタムモデル ID も許可リストに含めてください。含めない場合、Claude Code はカスタムエントリをピッカーから除外し、--model でそれを選択しても、除外された他のモデルと同様に拒否します。

my-gateway/claude-opus-5-5 のようにファミリー名を含むカスタム ID は、そのファミリーの特定のエントリとして扱われ、そのファミリーのワイルドカードが無効になります。そのため、選択可能なままにしたいバージョンも併せて一覧に含めてください。**マージの動作** (ページ 738) を参照してください。

28.9 環境変数

エイリアスがマッピングされるモデル名を制御するには、次の環境変数を使用します。各値は完全なモデル名、または API プロバイダーにおける同等の識別子である必要があります。セッション開始時のモデルを選択するには、この表には記載されていない *ANTHROPIC_DEFAULT_MODEL* (ページ 731) を設定します。

環境変数	説明
ANTHROPIC_DEFAULT_FABLE_MODEL	fable に使用するモデル。また、サードパーティプロバイダーでの**自動モデルフォールバック** (ページ 744) において Claude Code が Fable モデルとして認識するモデル ID
ANTHROPIC_DEFAULT_OPUS_MODEL	opus に使用するモデル、または Plan Mode がアクティブな場合の opusplan に使用するモデル。

次のページに続く

表 8 – 前のページからの続き

環境変数	説明
ANTHROPIC_DEFAULT_SONNET_MODEL	sonnet に使用するモデル、または Plan Mode がアクティブでない場合の opusplan に使用するモデル。
ANTHROPIC_DEFAULT_HAIKU_MODEL	haiku または**バックグラウンド機能** (ページ 923) に使用するモデル
CLAUDE_CODE_SUBAGENT_MODEL	他の方法でモデルが割り当てられていない**サブエージェント** (ページ 329)、エージェントチームのチームメイト、ワークフローエージェントのデフォルトモデル。haiku などのエイリアスまたは完全なモデル名を指定できます。呼び出しごとのモデル指定や、定義の model フィールド（inherit を含む）が優先されます。これを変更するには、*CLAUDE_CODE_SUBAGENT_MODEL_FORCE* (ページ 331) を設定します

サードパーティプロバイダーにおいて、固定したモデルの行が /model ピッカーにどのように表示されるかについては、**固定モデルの表示と機能のカスタマイズ** (ページ 760) を参照してください。

注: ANTHROPIC_SMALL_FAST_MODEL は非推奨となり、ANTHROPIC_DEFAULT_HAIKU_MODEL に置き換えられました。

28.9.1 サードパーティデプロイ向けのモデルの固定

Amazon Bedrock (ページ 863)、*Google Cloud's Agent Platform* (ページ 881)、*Microsoft Foundry* (ページ 891)、または Claude Platform on AWS を通じて Claude Code をデプロイする場合は、ユーザーに配布する前にモデルバージョンを固定してください。

固定しない場合、Claude Code は fable、opus、sonnet、haiku などのモデルエイリアスを使用し、これらは各プロバイダーの組み込みのデフォルトモデル ID に解決されます。このデフォルトは最新の Anthropic リリースより遅れている場合があり、参照先のモデルがユーザーのアカウントでまだ有効になっていないこともあります。デフォルトが利用できない場合、Amazon Bedrock と Google Cloud's Agent Platform のユーザーには通知が表示され、セッションはデフォルトモデルの以前のバージョンにフォールバックします。デフォルトが Opus モデルで利用可能な Opus バージョンがない場合は、デフォルトの Sonnet モデルにフォールバックします。Microsoft Foundry には同等の起動時チェックがないため、Microsoft Foundry のユーザーには代わりにエラーが表示されます。

Amazon Bedrock と Google Cloud's Agent Platform では、ユーザーが --model、ANTHROPIC_MODEL、または model 設定などで特定の Sonnet または Opus バージョンでセッションを開始すると、そのバージョンが対応するエイリアスのセッションのデフォルトとして固定されます。起動時チェックは置き換えられた組み込みのデフォルトをスキップし、フォールバック通知は表示されません。v2.1.211 より前は、セッションモデルが明示的に設定されていてもチェックが実行され、通知が表示されることがありました。

警告:

初期セットアップの一環として、モデルの環境変数を特定のバージョン ID に設定してください。固定することで、ユーザーが新しいモデルに移行するタイミングを制御できます。

プロバイダーに応じたバージョン固有のモデル ID を指定して、次の環境変数を使用します。

プロバイダー	例
Amazon Bedrock	`export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-8'`
Google Cloud's Agent Platform	`export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-8'`
Microsoft Foundry	`export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-8'`

`ANTHROPIC_DEFAULT_FABLE_MODEL`、`ANTHROPIC_DEFAULT_SONNET_MODEL`、`ANTHROPIC_DEFAULT_HAIKU_MODEL` にも同じパターンを適用します。すべてのプロバイダーにおける現行およびレガシーのモデル ID については、モデルの概要を参照してください。ユーザーを新しいモデルバージョンにアップグレードするには、これらの環境変数を更新して再デプロイします。

固定したモデルで**拡張コンテキスト** (ページ 752) を有効にするには、`ANTHROPIC_DEFAULT_OPUS_MODEL`、`ANTHROPIC_DEFAULT_SONNET_MODEL`、または `ANTHROPIC_DEFAULT_FABLE_MODEL` のモデル ID に `[1m]` を付加します。

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-8[1m]'
```

`[1m]` サフィックスを付けると、1M コンテキストウィンドウは固定したエイリアスのすべての使用に適用されます。これには *opusplan* (ページ 742) の plan モードにおける Opus フェーズや、`model` フロントマターでそのエイリアスを指定している**サブエージェント** (ページ 329) も含まれます。

- Claude Code はモデル ID をプロバイダーに送信する前にサフィックスを取り除きます。
- `[1m]` は、基盤となるモデルが 1M コンテキストをサポートしている場合にのみ付加してください。
- サフィックスはモデルごとではなく、変数ごとに読み取られます。Amazon Bedrock、Google Cloud's Agent Platform、Microsoft Foundry では、ある変数で `[1m]` なしのモデル ID を指定すると、別の変数で同じモデルにサフィックスを付けて設定していても、200K コンテキストが使用されます。Sonnet 5 はこれらのプロバイダーでは常に 1M ウィンドウで動作し、サフィックスは必要ありません。

`ANTHROPIC_DEFAULT_*_MODEL` 変数を設定すると、`/model` ピッカーには、そのファミリーの組み込みの行（1M コンテキストの行を含む）の代わりに、そのモデルの行が 1 つ表示されます。その変数にサフィックスを追加せずに 1M ウィンドウを利用するには、ユーザーが `/model opus[1m]` を実行します。すると Claude Code は、変数で指定されたモデルにサフィックスを適用します。`/model sonnet[1m]` も同様に動作します。

注釈:

MDM または管理設定ファイルを通じて配布される `availableModels` 許可リストは、サードパーティプロバイダーを使用する場合でも適用されます。サーバー管理設定はそこには配布されません。
フィルタリングは、`opus` などのモデルエイリアス、`claude-opus-4-8` などのバージョンプレフィックス、またはプロバイダー形式の完全なモデル ID に対して照合されます。`us.anthropic.` などのプロバイダー固有のプレフィックスは取り除かれないため、特定のモデルを許可するには、プロバイダー形式の完全な ID を記載するか、*`modelOverrides`* (ページ 761) を通じてマッピングしてください。固定したモデルの場合、その ID は対応する `ANTHROPIC_DEFAULT_*_MODEL` 変数に設定した値です。照合の前に、許可リストのエントリと要求されたモデルの両方から `[1m]` サフィックスが取り除かれます。

28.9.2 固定モデルの表示と機能のカスタマイズ

サードパーティプロバイダーでモデルを固定すると、`/model` ピッカーのその行には、Claude Code が固定した ID を認識する場合はデフォルトでモデル名が表示され、認識しない場合は生の ID が表示されます。

- **認識される場合**: Claude Code が把握しているモデルの正確な ID。Anthropic API の ID や、プロバイダーまたはゲートウェイでのその形式などで、`[1m]` サフィックスの有無は問いません。`us.anthropic.claude-sonnet-4-5-20250929-v1:0` を固定すると、行には `Sonnet 4.5` と表示されます。
- **認識されない場合**: アプリケーション推論プロファイル ARN や Claude Code が把握していないモデルバージョンなど、その他の ID。ただし、*`modelOverrides`* (ページ 761) のエントリがモデルをその正確な文字列にマッピングしている場合を除きます。Microsoft Foundry ではデプロイ名がユーザー定義であるため、マッピングの有無にかかわらず Claude Code は固定した ID を認識せず、行にはデフォルトでデプロイ名が表示されます。

行にモデル名が表示される場合、そのデフォルトの説明には固定した ID が含まれるため、どの ID が固定されているかを確認できます。

Claude Code は、固定したモデルがどの機能をサポートしているかを認識できない場合もあります。固定した各モデルについて、表示名と説明を自分で設定し、付随する環境変数で機能を宣言できます。

これらの変数は、Amazon Bedrock、Google Cloud's Agent Platform、Microsoft Foundry などのサードパーティプロバイダーで有効になります。`_NAME` と `_DESCRIPTION` 変数は、`ANTHROPIC_BASE_URL` が *LLM* **ゲートウェイ** (ページ 841) を指している場合にも有効になります。`api.anthropic.com` に直接接続する場合は効果がありません。

環境変数	説明
`ANTHROPIC_DEFAULT_OPUS_MODEL_NAME`	`/model` ピッカーにおける固定した Opus モデルの表示名。設定されていない場合、Claude Code が固定した ID を認識すればモデル名が、認識しなければ固定した ID が行に表示されます
`ANTHROPIC_DEFAULT_OPUS_MODEL_DESCRIPTION`	`/model` ピッカーにおける固定した Opus モデルの表示説明。設定されていない場合、`Custom Opus model` で始まるデフォルトの説明が行に表示されます

次のページに続く

表 10 – 前のページからの続き

環境変数	説明
ANTHROPIC_DEFAULT_OPUS_MODEL_SUPPORTED_CAPABILITIES	固定した Opus モデルがサポートする機能のカンマ区切りリスト

同じ _NAME、_DESCRIPTION、_SUPPORTED_CAPABILITIES サフィックスは、ANTHROPIC_DEFAULT_SONNET_MODEL、ANTHROPIC_DEFAULT_HAIKU_MODEL、ANTHROPIC_DEFAULT_FABLE_MODEL、ANTHROPIC_CUSTOM_MODEL_OPTION でも使用できます。

Claude Code は、モデル ID を既知のパターンと照合することで、*effort* **レベル** (ページ 747) や**拡張思考** (ページ 751) などの機能を有効にします。Amazon Bedrock の ARN やカスタムデプロイ名などのプロバイダー固有の ID はこれらのパターンに一致しないことが多く、サポートされている機能が無効のままになります。_SUPPORTED_CAPABILITIES を設定して、モデルが実際にサポートしている機能を Claude Code に伝えてください。

機能の値	有効になるもの
effort	*effort* **レベル** (ページ 747) と /effort コマンド
xhigh_effort	xhigh effort レベル
max_effort	max effort レベル
thinking	**拡張思考** (ページ 751)
adaptive_thinking	タスクの複雑さに応じて思考を動的に割り当てる適応型推論
interleaved_thinking	ツール呼び出し間の思考

_SUPPORTED_CAPABILITIES が設定されている場合、Claude Code は対応する固定モデルについて、記載された機能を有効にし、記載されていない機能を無効にします。変数が設定されていない場合、Claude Code はモデル ID に基づく組み込みの検出にフォールバックします。

次の例では、Opus を Amazon Bedrock のカスタムモデル ARN に固定し、わかりやすい名前を設定して、その機能を宣言しています。

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='arn:aws:bedrock:us-east-1:123456789012:custom-
↪model/abc'
export ANTHROPIC_DEFAULT_OPUS_MODEL_NAME='Opus via Bedrock'
export ANTHROPIC_DEFAULT_OPUS_MODEL_DESCRIPTION='Opus 4.7 routed through a Bedrock␣
↪custom endpoint'
export ANTHROPIC_DEFAULT_OPUS_MODEL_SUPPORTED_CAPABILITIES='effort,xhigh_effort,max_
↪effort,thinking,adaptive_thinking,interleaved_thinking'
```

28.9.3 バージョンごとのモデル ID の上書き

Claude Code を組み込み、CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST を設定しているプラットフォームでは、ホストのモデル設定が管理設定のモデル設定より優先されます。一方、管理設定の availableModels 許可リストは、ホストが独自のものを提供しない限り引き続き有効です。ホストがどのキーと変数を上書きするかについては、**管理設定の優先順位の例外** (ページ 722) を参照してください。

上記のファミリーレベルの環境変数は、ファミリーエイリアスごとに 1 つのモデル ID を設定します。同じファミリー内の複数のバージョンをそれぞれ異なるプロバイダー ID にマッピングする必要がある場合は、代わりに `modelOverrides` 設定を使用します。

`modelOverrides` は、個々の Anthropic モデル ID を、Claude Code がプロバイダーの API に送信するプロバイダー固有の文字列にマッピングします。ユーザーが `/model` ピッカーでマッピングされたモデルを選択すると、Claude Code は組み込みのデフォルトの代わりに、設定された値を使用します。

これにより、エンタープライズの管理者は、ガバナンス、コスト配分、またはリージョンルーティングのために、各モデルバージョンを特定の Amazon Bedrock 推論プロファイル ARN、Google Cloud's Agent Platform のバージョン名、または Microsoft Foundry のデプロイ名にルーティングできます。

設定ファイル (ページ 711) で `modelOverrides` を設定します。

```
{
  "modelOverrides": {
    "claude-opus-4-7": "arn:aws:bedrock:us-east-2:123456789012:application-inference-
↪profile/opus-prod",
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-inference-
↪profile/opus-46-prod",
    "claude-sonnet-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-
↪inference-profile/sonnet-prod"
  }
}
```

キーは、モデルの概要に記載されている Anthropic モデル ID である必要があります。日付付きのモデル ID の場合は、そこに記載されているとおりに日付サフィックスを含めてください。不明なキーは無視されます。

ゲートウェイエイリアスなどの ID に対する `[claude-code:unrecognized_model]` 診断行を止めるには、その ID を値とするエントリを追加します。

上書きは、`/model` ピッカーの各エントリの背後にある組み込みのモデル ID を置き換えます。Amazon Bedrock では、`modelOverrides` のエントリは、Claude Code が起動時に自動的に検出する推論プロファイルよりも優先されます。Amazon Bedrock の推論プロファイル ARN や Microsoft Foundry のデプロイ名など、すでにプロバイダーネイティブな値は、Claude Code がそのままプロバイダーに渡します。

上書きは、`--model`、`ANTHROPIC_MODEL` 環境変数、または `ANTHROPIC_DEFAULT_*_MODEL` 環境変数を通じて Anthropic モデル ID を直接渡した場合にも適用されます。Amazon Bedrock、Google Cloud's Agent Platform、*Mantle* (ページ 875) では、`modelOverrides` エントリのない Anthropic モデル ID は、プロバイダーがそのバージョンをサポートしている場合、そのバージョンの `/model` ピッカーの行と同じプロバイダー固有の ID に解決されます。Mantle はバージョンの一部のみをサポートしています。その範囲外の Anthropic モデル ID の場合、`modelOverrides` エントリで対応していない限り、Claude Code はマッピングせずに生の ID を Mantle に送信します。v2.1.200 より前は、`--model` と環境変数の値は上書きマップを経由せず、そのままプロバイダーに渡されていました。

`modelOverrides` は `availableModels` と連携して動作します。許可リストは上書き値ではなく Anthropic モデル ID に対して評価されるため、Opus バージョンが ARN にマッピングされている場合でも、`availableModels` の `"opus"` のようなエントリは引き続き一致します。管理設定で `enforceAvailableModels` が設定されている場合、強制される Default は管理設定の `modelOverrides` のみを通じて解決されます。推論

プロファイル ARN に固定したバージョンなど、管理者のマッピングは強制される Default に反映されます。ユーザー設定やプロジェクト設定の上書きはこれに影響しません。

管理設定で `availableModels` が設定されている場合、`--model` または上記の環境変数を通じて直接渡された Anthropic モデル ID には、管理設定の `modelOverrides` のみが適用されます。Claude Code はそれらの ID に対するユーザー設定やプロジェクト設定の上書きを無視し、管理リストで除外された ID は、どの設定ソースの `modelOverrides` を通じても解決しません。この管理ソースの制限には Claude Code v2.1.200 以降が必要です。ブロックされた ID の扱いについては、**モデル選択の制限** (ページ 732) を参照してください。

28.9.4 プロンプトキャッシュの設定

Claude Code は、パフォーマンスを最適化しコストを削減するために、自動的にプロンプトキャッシュを使用します。プロンプトキャッシュは、グローバルに、または特定のモデル階層ごとに無効にできます。

環境変数	説明
`DISABLE_PROMPT_CACHING`	1 に設定すると、すべてのモデルでプロンプトキャッシュを無効にします。モデルごとの設定より優先されます
`DISABLE_PROMPT_CACHING_HAIKU`	1 に設定すると、デフォルトの Haiku モデルのプロンプトキャッシュを無効にします
`DISABLE_PROMPT_CACHING_SONNET`	1 に設定すると、デフォルトの Sonnet モデルのプロンプトキャッシュを無効にします
`DISABLE_PROMPT_CACHING_OPUS`	1 に設定すると、デフォルトの Opus モデルのプロンプトキャッシュを無効にします
`DISABLE_PROMPT_CACHING_FABLE`	1 に設定すると、Fable モデルのみのプロンプトキャッシュを無効にします

メインの会話とサブエージェントのキャッシュ TTL を個別に選択するには、TTL を自分で選択するを参照してください。キャッシュミスが発生する原因については、Claude Code がプロンプトキャッシュを使用する方法を参照してください。

28.10 バージョン履歴

この表は、各モデルエイリアスの解決先モデルが変更された Claude Code のバージョンを、新しい順に示しています。

バージョン	変更内容
v2.1.284	Anthropic API で `sonnet` が Sonnet 5.5 に解決されるようになりました
v2.1.280	Anthropic API、Claude Platform on AWS、Amazon Bedrock、Google Cloud の Agent Platform で `opus` が Opus 5.5 に解決されるようになりました
v2.1.257	Claude apps ゲートウェイのセッションを除き、`fable` が Fable 5.1 に解決されるようになりました
v2.1.219	Anthropic API、Claude Platform on AWS、Amazon Bedrock、Agent Platform で `opus` が Opus 5 に解決されるようになりました

次のページに続く

表 13 – 前のページからの続き

バージョン	変更内容
v2.1.207	Claude Platform on AWS、Amazon Bedrock、Agent Platform で `opus` が Opus 4.8 に解決されるようになりました
v2.1.197	Anthropic API で `sonnet` が Sonnet 5 に解決されるようになりました
v2.1.154	Anthropic API で `opus` が Opus 4.8 に解決されるようになりました
それ以前	`opus` は Claude Platform on AWS では Opus 4.7 に、Amazon Bedrock と Agent Platform では Opus 4.6 に解決されます。`fable` はすべてのプロバイダーで Fable 5 に解決されます

第29章　キーボードショートカットのカスタマイズ

キーボードショートカットをカスタマイズして、Claude Code でキーバインディング設定ファイルを使用します。

Claude Code はカスタマイズ可能なキーボードショートカットをサポートしています。`/keybindings` を実行して、`~/.claude/keybindings.json` に設定ファイルを作成または開きます。

29.1 設定ファイル

キーバインディング設定ファイルは、`bindings` 配列を持つオブジェクトです。各ブロックはコンテキストとキーストロークからアクションへのマップを指定します。

注釈:

キーバインディングファイルへの変更は自動的に検出され、Claude Code を再起動することなく適用されます。

フィールド	説明
`$schema`	エディタのオートコンプリート用のオプション JSON スキーマ URL
`$docs`	オプションのドキュメント URL
`bindings`	コンテキスト別のバインディングブロックの配列

この例では、チャットコンテキストで `Ctrl+E` を外部エディタを開くにバインドし、`Ctrl+S` をアンバインドします。

```
{
  "$schema": "https://www.schemastore.org/claude-code-keybindings.json",
  "$docs": "https://code.claude.com/docs/ja/keybindings",
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+e": "chat:externalEditor",
        "ctrl+s": null
      }
```

(次のページに続く)

(前のページからの続き)

```
    }
  ]
}
```

29.2 コンテキスト

各バインディングブロックは、バインディングが適用される**コンテキスト**を指定します。

コンテキスト	説明
Global	アプリ全体に適用
Chat	メインチャット入力エリア
Autocomplete	オートコンプリートメニューが開いている
Settings	設定メニュー
Confirmation	権限と確認ダイアログ
Tabs	タブナビゲーションコンポーネント
Help	ヘルプメニューが表示されている
Transcript	トランスクリプトビューア
HistorySearch	履歴検索モード（Ctrl+R）
Task	フォアグラウンドで実行中のタスク
ThemePicker	テーマピッカーダイアログ
Attachments	選択ダイアログ内の画像添付ファイルナビゲーション
Footer	フッターインジケータナビゲーション（タスク、チーム、diff、Artifacts）
MessageSelector	巻き戻しと要約ダイアログのメッセージ選択
DiffDialog	**差分ビューア** (ページ 773) のナビゲーション
DiffPanel	*diff* **パネル** (ページ 97) が開いている
ModelPicker	モデルピッカー努力レベル
EffortSlider	/effort で開かれた努力スライダー
Select	汎用選択/リストコンポーネント
Plugin	プラグインダイアログ（参照、発見、管理）
Pane	mod が描画したペインにキーボードフォーカスがある
PaneField	mod のペイン内の入力フィールドまたは選択にキーボードフォーカスがある
Agents	エージェントビュー（claude agents）
Scroll	会話スクロールとフルスクリーンモードでのテキスト選択

v2.1.205 より前では、Doctor コンテキストと /doctor 診断スクリーン用の doctor:fix アクションが存在していました。

29.3 利用可能なアクション

アクションは namespace:action 形式に従います。例えば、メッセージを送信する chat:submit や、タスクリストを表示する app:toggleTodos などです。各コンテキストには特定のアクションが利用可能です。

29.3.1 アプリケーションアクション

Global コンテキストで利用可能なアクション：

アクション	デフォルト	説明
app:interrupt	Ctrl+C	現在の操作をキャンセル
app:exit	Ctrl+D	Claude Code を終了します。800ms 以内に 2 回押して確認
app:redraw	（バインドなし）	ターミナルの再描画を強制
app:toggleTodos	Ctrl+T	Claude のやることリストの表示を切り替えます。これは /tasks バックグラウンドタスクビューではありません
app:toggleTranscript	Ctrl+O	詳細トランスクリプトの表示を切り替え

29.3.2 履歴アクション

コマンド履歴をナビゲートするためのアクション：

アクション	デフォルト	説明
history:search	Ctrl+R	履歴検索を開く
history:previous	Up	前の履歴項目
history:next	Down	次の履歴項目

29.3.3 チャットアクション

Chat コンテキストで利用可能なアクション：

アクション	デフォルト	説明
chat:cancel	Escape	現在の入力をキャンセル
chat:clearInput	Ctrl+L	入力と会話を保持したまま、フルスクリーン再描画を強制
chat:clearScreen	Cmd+K	chat:clearInput と同じです。Cmd+K が iTerm2 と Terminal.app でどのように動作するかについては、会話をクリア を参照してください
chat:killAgents	Ctrl+X Ctrl+K	このセッション内のすべての実行中の **バックグラウンドサブエージェント** (ページ 344) を停止し、残りのセッションの アーティファクト自動返信 をオフにします
chat:cycleMode	Shift+Tab*	権限モードをサイクル
chat:modelPicker	Meta+P	モデルピッカーを開く
chat:fastMode	Meta+O	高速モードを切り替え

次のページに続く

表 5 – 前のページからの続き

アクション	デフォルト	説明
chat:thinkingToggle	Meta+T	拡張思考を切り替え
chat:submit	Enter	メッセージを送信
chat:queueSubmit	Ctrl+X Enter	メッセージを送信し、順番を待つようにマーク：Claude が作業中の場合、Claude Code は **それをキューに入れ** (ページ 90)、ターンを中断することはありません。chat:submit とは異なり、オートコンプリート候補がハイライトされている場合でもドラフトを送信します。v2.1.247 以降が必要
chat:sendNow	Ctrl+Enter、Ctrl+X Ctrl+S	**キューに入れたメッセージ** (ページ 90) とドラフトをすぐに送信します。*Claude Code* **がキューに入れたものを送信する場合** (ページ 90) は、Claude が作業中のターンに何が起こるかについて説明しています。何も実行されていない場合、キーはドラフトを送信し、**シェルモード** (ページ 89) ではコマンドのみをキューに入れます。拡張キーを報告しないターミナルは Ctrl+Enter をプレーン Enter として配信するため、Ctrl+X Ctrl+S はどのターミナルでも機能するバインディングです。v2.1.275 以降が必要
chat:newline	Ctrl+J	送信せずに改行を挿入
chat:undo	Ctrl+_、Ctrl+Shift+-	最後のアクションを元に戻す
chat:externalEditor	Ctrl+G、Ctrl+X Ctrl+E	外部エディタで開きます。エージェントビューディスパッチ入力 もこのアクションの単一キーストロークバインディングに従います
chat:stash	Ctrl+S	現在のプロンプトを保存
chat:imagePaste	Ctrl+V（Windows と WSL では Alt+V）	クリップボードから画像を貼り付けます。WSL では、両方のショートカットがデフォルトでバインドされています

*VT モードなし Windows （Node <24.2.0/<22.17.0、Bun <1.2.23） では、デフォルトは Meta+M です。

29.3.4 オートコンプリートアクション

Autocomplete コンテキストで利用可能なアクション：

アクション	デフォルト	説明
autocomplete:accept	Tab	候補を受け入れ
autocomplete:dismiss	Escape	メニューを閉じる
autocomplete:previous	Up	前の候補

次のページに続く

表 6 – 前のページからの続き

アクション	デフォルト	説明
autocomplete:next	Down	次の候補

29.3.5 確認アクション

Confirmation コンテキストで利用可能なアクション：

アクション	デフォルト	説明
confirm:yes	Enter	アクションを確認
confirm:no	Escape	アクションを拒否
confirm:previous	Up	前のオプション
confirm:next	Down	次のオプション
confirm:nextField	Tab	次のフィールド
confirm:previousField	（バインドなし）	前のフィールド
confirm:toggle	Space	選択を切り替え
confirm:cycleMode	Shift+Tab*	権限モードをサイクルします。ファイル権限プロンプトでは、開いている **コメントフィールド** (ページ 1040) を閉じます。フィールドが開いていない場合は、プロンプトがそのオプションを提供する場合、セッションの残りの間アクションを許可するオプションを選択します

*VT モードなし Windows （Node <24.2.0/<22.17.0、Bun <1.2.23） では、デフォルトは Meta+M です。

v2.1.257 より前では、confirm:toggleExplanation アクション（デフォルトでは Ctrl+E にバインド）が Bash と PowerShell 権限プロンプトでモデル生成の説明を表示していました。

ダイアログは confirm:yes と confirm:no を使用して、はい/いいえの質問をしていない場合でも受け入れてキャンセルします。このコンテキストで y や n などのベアレターをバインドすると、そのレターをキーとして表示しないダイアログにも作用します。y と n をキーとして表示するダイアログはそれらのレターを自分で読み取り、バインディングは必要ありません。

ほとんどのダイアログでは、Ctrl+C または Ctrl+D を 2 回押すと、Claude Code を終了する代わりにダイアログを閉じます。最初の押下後のヒントは、2 番目の押下がダイアログを閉じるか Claude Code を終了するかを示します。両方のキーは **予約済み** (ページ 782) であり、再バインドできません。

この例は y を confirm:yes に、n を confirm:no にバインドします：

```
{
  "bindings": [
    {
      "context": "Confirmation",
      "bindings": {
        "y": "confirm:yes",
        "n": "confirm:no"
```

(次のページに続く)

(前のページからの続き)

```
      }
    }
  ]
}
```

これらのバインディングを使用すると、**テキストフィールド** (ページ 782) がフォーカスを持つ間、y と n はまだレターとして入力されます。

v2.1.280 より前では、y はデフォルトで confirm:yes に、n は confirm:no にバインドされていました。v2.1.280 より前に /keybindings で keybindings.json を作成した場合、ファイルは両方のバインディングをリストし、それらの 2 行を削除するまで有効なままです。

29.3.6 権限アクション

権限ダイアログの Confirmation コンテキストで利用可能なアクション：

アクション	デフォルト	説明
permission:toggleDebug	（バインドなし）	権限デバッグ情報を切り替えます。v2.1.146 で app:exit をシャドウしたため、Ctrl+D の以前のデフォルトが削除されました

29.3.7 トランスクリプトアクション

Transcript コンテキストで利用可能なアクション：

アクション	デフォルト	説明
transcript:toggleShowAll	Ctrl+E	すべてのコンテンツの表示を切り替え
transcript:exit	q、Ctrl+C、Escape	トランスクリプトビューを終了

transcript:toggleShowAll はクラシックレンダラーのみで適用されます。フルスクリーンレンダリングでは、トランスクリプトビューアは show-all トグルを提供しません。

29.3.8 履歴検索アクション

HistorySearch コンテキストで利用可能なアクション：

アクション	デフォルト	説明
historySearch:next	Ctrl+R	次のマッチ
historySearch:accept	Escape、Tab	選択を受け入れ
historySearch:cancel	Ctrl+C	検索をキャンセル
historySearch:execute	Enter	選択したコマンドを実行

次のページに続く

表 10 – 前のページからの続き

アクション	デフォルト	説明
historySearch:cycleScope	Ctrl+S	スコープをサイクル：セッション、プロジェクト、すべて

historySearch:next、historySearch:accept、historySearch:cancel、および historySearch:execute のデフォルトは、クラシックレンダラーのインライン履歴検索に適用されます。これは常にすべてのプロジェクトからプロンプトを検索します。historySearch:cycleScope は フルスクリーンレンダリング でのみ有効です。ここで Ctrl+R は検索ダイアログを開き、Ctrl+S はそのスコープをサイクルします。ダイアログの他のキーは固定されており、再バインドできません：Enter または Tab はハイライトされたマッチをプロンプト入力に配置し、Esc はキャンセルします。

29.3.9 タスクアクション

Task コンテキストで利用可能なアクション：

アクション	デフォルト	説明
task:background	Ctrl+B、Ctrl+X Ctrl+B	現在のタスクをバックグラウンドに移動します。Ctrl+X Ctrl+B コードは tmux プレフィックスの競合を回避します

29.3.10 テーマアクション

ThemePicker コンテキストで利用可能なアクション：

アクション	デフォルト	説明
theme:toggleSyntaxHighlighting	Ctrl+T	構文ハイライトを切り替え

29.3.11 ヘルプアクション

Help コンテキストで利用可能なアクション：

アクション	デフォルト	説明
help:dismiss	Escape	ヘルプメニューを閉じる

29.3.12 タブアクション

Tabs コンテキストで利用可能なアクション：

アクション	デフォルト	説明
tabs:next	Tab、Right	次のタブ
tabs:previous	Shift+Tab、Left	前のタブ

タブ付きダイアログでは、タブ行がフォーカスを持つ間、tabs:next と tabs:previous がタブを切り替えます。/help や /sandbox などの一部のダイアログでは、タブ切り替えキーはタブのコンテンツ内からも機能します。

Up と Down はタブ行とタブのコンテンツ間でフォーカスを移動し、コンテンツ内のリストはフォーカスを持つ間のみキーに応答します。

29.3.13 添付ファイルアクション

Attachments コンテキストで利用可能なアクション：

アクション	デフォルト	説明
attachments:next	Right	次の添付ファイル
attachments:previous	Left	前の添付ファイル
attachments:remove	Backspace、Delete	選択した添付ファイルを削除
attachments:exit	Down、Escape	添付ファイルナビゲーションを終了

29.3.14 フッターアクション

Footer コンテキストで利用可能なアクション：

アクション	デフォルト	説明
footer:next	Right	次のフッター項目
footer:previous	Left	前のフッター項目
footer:up	Up	フッター内を上に移動（上部で選択解除）
footer:down	Down	フッター内を下に移動
footer:openSelected	Enter	選択したフッター項目を開く
footer:clearSelection	Escape	フッター選択をクリア
footer:dismiss	（バインドなし）	このアクションにキーをバインドしても効果がなく、それを名前付けする keybindings.json は有効なままです。v2.1.281 より前では、Backspace と Delete はフッターから選択したアーティファクトリンクを削除しました。

フッター項目が選択されている場合（プロンプトの下のエージェントパネルの行など）、chat:submit を chat:queueSubmit または chat:newline に再バインドしても、Enter はそれを開きます。

Chat バインディングは、Shift+Tab を chat:cycleMode のように Footer コンテキストがバインドしないキーで機能し、項目が選択されている間も機能します。

29.3.15 メッセージセレクターアクション

リワインドメニュー (ページ 1019) のメッセージリストでは、**選択アクション** (ページ 776) とそのデフォルトキーを使用してメッセージを移動して選択します。これらのアクションの `Select` バインディングもそこに適用されます。`MessageSelector` コンテキストには、独自のアクションまたはデフォルトバインディングはありません。このリストのみのキーを変更するには、`MessageSelector` ブロックで `select:accept` などの選択アクションをバインドします。

この例は、リワインドメニューでハイライトされたメッセージを選択するために `o` を `select:accept` にバインドします。他のリストは変更しません：

```
{
  "bindings": [
    {
      "context": "MessageSelector",
      "bindings": {
        "o": "select:accept"
      }
    }
  ]
}
```

v2.1.283 より前では、このリストは `Select` バインディングを無視し、独自のアクション `messageSelector:up`、`messageSelector:down`、`messageSelector:top`、`messageSelector:bottom`、および `messageSelector:select` を持っていました。`keybindings.json` がそれらの名前の 1 つをバインドする場合、バインディングはこのリストで同じことを行う選択アクションとして機能し続けます。`Home` と `End` はリストのいずれかの端にジャンプします。v2.1.283 より前では、`Shift+K` や `Shift+J` などのキーがデフォルトでそれを行いました。

29.3.16 差分アクション

これらのアクションは、Claude Code の以前の差分ビューアにのみ作用します。このビューアは、`/plugin` で cc-plugin-diff mod を無効にした後、フルスクリーンレンダリング 以外で `/diff` を実行すると開きます。この mod が有効な間は、`/diff` は代わりに **差分ダイアログ** (ページ 98) を開きます。これらのアクションを指定する `keybindings.json` は、どちらの場合でもエラーなく読み込まれます。

`DiffDialog` コンテキストで利用可能なアクション：

アクション	デフォルト	説明
`diff:dismiss`	Escape	差分ビューアを閉じます。詳細ビューから、ファイルリストに戻ります
`diff:previousSource`	Left	前の差分ソース
`diff:nextSource`	Right	次の差分ソース
`diff:previousFile`	Up、K	ファイルリスト内の前のファイル。詳細ビューで 1 行上にスクロール
`diff:nextFile`	Down、J	ファイルリスト内の次のファイル。詳細ビューで 1 行下にスクロール

次のページに続く

表 17 – 前のページからの続き

アクション	デフォルト	説明
diff:back	（バインドなし）	差分ビューアで戻ります。Escape は diff:dismiss を介してバック アクションを実行します。詳細ビューの Left の以前のデフォルトは v2.1.203 で削除されました

ファイルリストは **選択アクション** (ページ 776) にも応答し、そのデフォルトキーと Select バインディングを通じて応答します。select:previous と select:next は前のファイルと次のファイルに移動し、Enter は select:accept を通じて選択したファイルの差分を開きます。ファイルリストのみのそれらのキーの 1 つを変更するには、DiffDialog ブロックで選択アクションをバインドします。

v2.1.283 より前では、ファイルリストは Select バインディングを無視し、Enter は別の diff:viewDetails アクションを通じて選択したファイルの差分を開きました。keybindings.json が diff:viewDetails をバインドする場合、バインディングはファイルリストで select:accept として機能し続けます。

差分詳細ビューは、ページャースタイルキーを標準 **スクロールアクション** (ページ 777) にバインドします。これらのバインディングは DiffDialog コンテキストの一部であり、詳細ビューのみで適用されます。**スクロールアクション** (ページ 777) の下にリストされている Scroll コンテキストのデフォルトは変更されません。

アクション	デフォルト	説明
scroll:pageUp	PageUp	ビューポートの半分上にスクロール
scroll:pageDown	PageDown	ビューポートの半分下にスクロール
scroll:fullPageUp	Shift+Space、B	フルビューポート上にスクロール
scroll:fullPageDown	Space	フルビューポート下にスクロール
scroll:top	G、Home	トップにジャンプ
scroll:bottom	Shift+G、End	ボトムにジャンプ

29.3.17 差分パネルアクション

/diff がフルスクリーンレンダリングで開く **差分パネル** (ページ 97) のアクション。app:cycleDiffBase は DiffPanel コンテキストにあり、パネルが開いている間はアクティブです。その他は Global です。

組み込みの cc-plugin-diff mod がこのパネルを描画し、app:cycleDiffBase、app:diffFileListUp、app:diffFileListDown を処理します。app:toggleReplTab、app:toggleDiffNoiseFilter、app:toggleDiffPreSession は、Claude Code の以前のパネルにのみ作用します。このパネルは、/plugin で cc-plugin-diff を無効にした後に /diff で開きます。

アクション	デフォルト	説明
app:toggleReplTab	（バインドなし）	差分パネルを開くか閉じる
app:cycleDiffBase	Ctrl+X B	パネルの比較ベースをサイクル：このセッション、コミットされていない、その後ブランチ

次のページに続く

表 19 – 前のページからの続き

アクション	デフォルト	説明
app:diffFileListUp	Ctrl+Up、Meta+Up	パネルのファイルリストがオーバーフローするときに上にスクロール
app:diffFileListDown	Ctrl+Down、Meta+Down	パネルのファイルリストがオーバーフローするときに下にスクロール
app:toggleDiffNoise Filter	（バインドなし）	パネルのテストファイルと生成ファイルを表示または非表示
app:toggleDiffPreSession	（バインドなし）	このセッション前の変更を展開または折りたたむ

29.3.18 モデルピッカーアクション

ModelPicker コンテキストで利用可能なアクション：

アクション	デフォルト	説明
modelPicker:decreaseEffort	Left	努力レベルを低下
modelPicker:increaseEffort	Right	努力レベルを増加
modelPicker:thisSessionOnly	s	ハイライトされたモデルをこのセッションのみに適用

29.3.19 努力スライダーアクション

EffortSlider コンテキストで利用可能なアクション。引数なしで /effort を実行するときに開くスライダー。スライダーの Enter と Escape キーは再バインドできません。

アクション	デフォルト	説明
effortSlider:decreaseEffort	Left	スライダーを次の低い努力レベルに移動します。v2.1.284 以降が必要
effortSlider:increaseEffort	Right	スライダーを次の高い努力レベルに移動します。v2.1.284 以降が必要
effortSlider:toggleUltracode	Tab	このセッションの ultracode をオンまたはオフにします。スライダーが **それを提供する** (ページ 749) 場合。v2.1.284 以降が必要
effortSlider:thisSessionOnly	s	フォーカスされた **努力レベル** (ページ 747) をこのセッションのみに適用します。v2.1.257 以降が必要

29.3.20 選択アクション

Select コンテキストで利用可能なアクション：

アクション	デフォルト	説明
select:next	Down、J、Ctrl+N	次のオプション
select:previous	Up、K、Ctrl+P	前のオプション
select:pageUp	PageUp	オプションの 1 ページ上に移動
select:pageDown	PageDown	オプションの 1 ページ下に移動
select:first	Home	最初のオプション
select:last	End	最後のオプション
select:accept	Enter	選択を受け入れ
select:cancel	Escape	選択をキャンセル

/skills、/mcp、/tasks などのリストパネルでは、Claude Code は select:pageUp、select:pageDown、select:first、および select:last バインディングを適用します。/model ピッカーなどのほとんどの他のリストでは、select:first と select:last バインディングが適用されます。PageUp と PageDown は、バインディングに関係なく、これらのリストのオプションをページングします。

v2.1.280 より前では、これらの他のリストは Home、End、および select:first と select:last バインディングを無視していました。

v2.1.283 より前では、/mcp ツールリストは、バインディングに関係なく固定 PageUp と PageDown キーでページングしました。

29.3.21 プラグインアクション

Plugin コンテキストで利用可能なアクション：

アクション	デフォルト	説明
plugin:toggle	Space	プラグイン選択を切り替え
plugin:install	I	選択したプラグインをインストール
plugin:favorite	F	選択したプラグインをお気に入りにして、インストール済みタブの上部付近でソート

29.3.22 設定アクション

Settings コンテキストで利用可能なアクション。select:accept と confirm:no アクションは **選択** (ページ 776) と **確認** (ページ 769) コンテキストから再利用されます。設定固有の動作：変更は各設定を変更するとすぐに適用されるため、Escape はパネルを閉じ、変更を保存します。拒否するのではなく。

アクション	デフォルト	説明
settings:search	/	検索モードに入る

次のページに続く

表 24 – 前のページからの続き

アクション	デフォルト	説明
settings:retry	R	エラー時に使用データの読み込みを再試行
select:accept	Enter、Space	選択した設定を変更またはそのサブメニューを開く
confirm:no	Escape	パネルを閉じます。変更は既に保存されています

29.3.23 エージェントアクション

Agents コンテキストで利用可能なアクション。claude agents で開く エージェントビュー で適用されます。v2.1.257 以降が必要です。

アクション	デフォルト	説明
agents:switchView	Ctrl+S	セッショングループ化 を状態とディレクトリ間で切り替え
agents:togglePin	Ctrl+T	選択したセッションを ピン留めまたはピン留め解除

エージェントビューが開いている間、Claude Code は Agents コンテキストがバインドするキーに対して Agents バインディングを使用し、同じキーの Chat または Global バインディングを無視します。例えば、エージェントビューで Ctrl+S を押すと、デフォルトの chat:stash をトリガーするのではなく、セッショングループ化を切り替えます。

ディスパッチ入力の外部エディタショートカットは Agents アクションではありません。エージェントビューは Chat コンテキストの chat:externalEditor バインディング（デフォルトでは Ctrl+G）に従います。

バインディングはエージェントビューで単一キーストロークで発火するため、chat:externalEditor にバインドされた Ctrl+X Ctrl+E コードはエディタをそこで開きません。

29.3.24 音声アクション

音声ディクテーション が有効な場合、Chat コンテキストで利用可能なアクション：

アクション	デフォルト	説明
voice:pushToTalk	Space	プロンプトをディクテーション。/voice モードに応じて保持またはタップ

29.3.25 スクロールアクション

フルスクリーンレンダリング が有効な場合、Scroll コンテキストで利用可能なアクション：

アクション	デフォルト	説明
scroll:lineUp	wheelup	1 行上にスクロール。マウスホイールスクロールはこのアクションをトリガー
scroll:lineDown	wheeldown	1 行下にスクロール。マウスホイールスクロールはこのアクションをトリガー
scroll:pageUp	PageUp	ビューポート高さの半分上にスクロール
scroll:pageDown	PageDown	ビューポート高さの半分下にスクロール
scroll:top	Ctrl+Home	会話の開始にジャンプ
scroll:bottom	Ctrl+End	最新のメッセージにジャンプして自動フォローを再度有効化
scroll:halfPageUp	（バインドなし）	ビューポート高さの半分上にスクロール。scroll:pageUp と同じ動作。vi スタイルの再バインドのために提供
scroll:halfPageDown	（バインドなし）	ビューポート高さの半分下にスクロール。scroll:pageDown と同じ動作。vi スタイルの再バインドのために提供
scroll:fullPageUp	（バインドなし）	フルビューポート高さ上にスクロール
scroll:fullPageDown	（バインドなし）	フルビューポート高さ下にスクロール
selection:copy	Ctrl+Shift+C / Cmd+C	選択したテキストをクリップボードにコピー
selection:clear	（バインドなし）	アクティブなテキスト選択をクリア。v2.1.234 以降が必要
selection:extendLeft	Shift+Left	アクティブな選択を 1 列左に拡張
selection:extendRight	Shift+Right	アクティブな選択を 1 列右に拡張
selection:extendUp	Shift+Up	アクティブな選択を 1 行上に拡張。選択が上端に達するとビューポートをスクロール
selection:extendDown	Shift+Down	アクティブな選択を 1 行下に拡張。選択が下端に達するとビューポートをスクロール
selection:extendLineStart	Shift+Home	アクティブな選択を行の開始に拡張
selection:extendLineEnd	Shift+End	アクティブな選択を行の終了に拡張

29.4 キーストロークシンタックス

29.4.1 モディファイア

+ セパレータでモディファイアキーを使用します。

- ctrl または control - Control キー
- shift - Shift キー
- alt、opt、option、または meta - Windows と Linux の Alt キー、macOS の Option キー
- cmd、command、super、または win - macOS の Command キー、Windows の Windows キー、Linux の Super キー

cmd グループは Super モディファイアを報告するターミナル（Kitty キーボードプロトコルまたは xterm の modifyOtherKeys モードをサポートするターミナルなど）でのみ検出されます。ほとんどのターミナルは

これを送信しないため、すべての場所で機能するバインディングには ctrl または meta を使用してください。

例えば：

```
ctrl+k          Ctrl + K
shift+tab       Shift + Tab
meta+p          macOS の Option + P、その他の場所では Alt + P
ctrl+shift+c    複数のモディファイア
```

29.4.2 大文字

Claude Code はキー名を大文字と小文字を区別せずに解析するため、K は k と同じバインディングであり、ctrl+K は ctrl+k と同じです。Shift とレターをバインドするには、shift+k と記述します。

29.4.3 非 US キーボードレイアウト

Ctrl ショートカットのキー名は、アクティブなキーボードレイアウトが他の文字を入力する場合でも、ラテン文字として記述してください。

Claude Code がキーボードで押したキーをバインディングにマッチさせる方法は、レイアウトの種類によって異なります。

- キリル文字などの非ラテンレイアウトの場合、Claude Code はターミナルが Kitty キーボードプロトコルを使用し、その位置を報告する場合、キーの US レイアウト位置で Ctrl ショートカットをマッチさせます。そのようなターミナルでロシア語レイアウトがアクティブな場合、Ctrl キーと物理的な W キーを押すと ctrl+w がトリガーされます。位置を報告しないターミナルでは、Claude Code はターミナルがキープレスに対して送信するものをマッチさせます。ASCII 制御コードは Latin ショートカットをトリガーし、キリル文字として到着するキープレスはバインディングにマッチしません。
- AZERTY などのラテン文字を並べ替えるレイアウトの場合、Claude Code はキーが入力する文字をマッチさせるため、Ctrl キーと A というラベルのキーを押すと ctrl+a がトリガーされます。

29.4.4 コード

コードはスペースで区切られたキーストロークのシーケンスです。

```
ctrl+k ctrl+s   Ctrl+K を押して、リリースしてから Ctrl+S
```

各キーストロークは、その前のキーストロークから 3 秒以内に押してください。それより長く待つと、Claude Code はコードをキャンセルし、そのことを示す簡潔な通知を表示します。

29.4.5 特殊キー

- escape または esc - Escape キー
- enter または return - Enter キー
- tab - Tab キー

- space - スペースバー
- up、down、left、right - 矢印キー
- pageup、pagedown - Page Up キーと Page Down キー
- home、end - Home キーと End キー
- backspace、delete - Delete キー
- wheelup、wheeldown - マウスホイールスクロールイベント

29.5 デフォルトのショートカットのバインドを解除する

アクションを null に設定すると、デフォルトのショートカットのバインドを解除できます。

```
{
  "bindings": [
    {
      "context": "Chat",
      "bindings": {
        "ctrl+s": null
      }
    }
  ]
}
```

これは chord（連続キー入力）バインディングにも有効です。プレフィックスを共有するすべての chord のバインドを解除すると、そのプレフィックスを単一キーのバインディングとして使用できるようになります。アクティブなコンテキストのいずれかに chord が 1 つでも残っていれば、そのプレフィックスは予約されたままになります。そのため、各 chord は、それを定義しているコンテキストでバインドを解除する必要があります。

Claude Code は、ctrl+x プレフィックスに以下のデフォルトの chord をコンテキストごとにバインドしています。

- Chat：ctrl+x ctrl+k、ctrl+x ctrl+e、ctrl+x enter、ctrl+x ctrl+a、ctrl+x ctrl+s、ctrl+x tab
- Task：ctrl+x ctrl+b
- DiffPanel：ctrl+x b
- Pane：ctrl+x left、ctrl+x right、ctrl+x up、ctrl+x down、ctrl+x x
- PaneField：ctrl+x x

ctrl+x enter の chord には v2.1.247 以降が、ctrl+x b、ctrl+x ctrl+a、ctrl+x tab には v2.1.260 以降が、ctrl+x ctrl+s には v2.1.275 以降が必要です。

ctrl+x 自体を単一キーのバインディングとして取り戻すには、これらすべてのバインドを解除します。

```
{
  "bindings": [
    {
      "context": "Task",
      "bindings": {
        "ctrl+x ctrl+b": null
      }
    },
    {
      "context": "DiffPanel",
      "bindings": {
        "ctrl+x b": null
      }
    },
    {
      "context": "Pane",
      "bindings": {
        "ctrl+x left": null,
        "ctrl+x right": null,
        "ctrl+x up": null,
        "ctrl+x down": null,
        "ctrl+x x": null
      }
    },
    {
      "context": "PaneField",
      "bindings": {
        "ctrl+x x": null
      }
    },
    {
      "context": "Chat",
      "bindings": {
        "ctrl+x ctrl+k": null,
        "ctrl+x ctrl+e": null,
        "ctrl+x enter": null,
        "ctrl+x ctrl+a": null,
        "ctrl+x ctrl+s": null,
        "ctrl+x tab": null,
        "ctrl+x": "chat:newline"
      }
    }
  ]
}
```

プレフィックス上の chord の一部だけのバインドを解除し、すべてを解除しなかった場合、そのプレフィッ

クスを押すと、残りのバインディングのために引き続き chord 待機モードに入ります。

29.6 予約済みショートカット

これらのショートカットは再バインドできません。

ショートカット	理由
Ctrl+C	ハードコードされた割り込み/キャンセル
Ctrl+D	ハードコードされた終了
Ctrl+M	Claude Code は常に Enter として受け取ります
Ctrl+[	Claude Code は常に Escape として受け取ります。Kitty キーボードプロトコルを使用するターミナルでは、v2.1.242 以降が必要です
Ctrl+I	Claude Code は常に Tab として受け取ります
Ctrl+H	ASCII バックスペースバイトを送信します。*Claude Code* **が** *Windows* **で読み込む方法** (ページ 832) は、ターミナルと `CLAUDE_CODE_BS_AS_CTRL_BACKSPACE` 環境変数によって異なります
Caps Lock	ターミナルアプリケーションに配信されない

29.7 ターミナルの競合

一部のショートカットはターミナルマルチプレクサと競合する可能性があります。

ショートカット	競合
Ctrl+B	tmux プレフィックス（2 回押して送信）
Ctrl+A	GNU screen プレフィックス
Ctrl+Z	Unix プロセス一時停止（SIGTSTP）

29.8 テキストフィールド

ベアレター、数字、またはスペースをバインドした場合でも、ダイアログまたはパネル内のテキストフィールドでそのキャラクターを入力できます。このようなフィールドの 1 つは、Claude が質問する「Other」の回答です。フィールドがフォーカスを持っている間、Ctrl、Alt、または Cmd なしで押したプリント可能なキーはフィールドに送られ、Claude Code はそれをバインディングと照合しません。

これらのキーはフィールドがフォーカスを持っている間でもバインディングを実行します。

- Enter、Escape、Tab、矢印キーなど、キャラクターを入力しないキー
- Ctrl、Alt、または Cmd で押されたキー
- 既に進行中の**コード** (ページ 779) の 2 番目のキーストローク

メインプロンプトでは、Claude Code はすべてのキーを `Chat` などのアクティブなコンテキストと照合し、バインディングがそれを取らない場合にのみキーを入力します。

29.9 Vim モードの相互作用

Vim モードが `/config` → エディタモードで有効な場合、キーバインディングと Vim モードは独立して動作します。

- **Vim モード** はテキスト入力レベルで入力を処理します（カーソル移動、モード、モーション）
- **キーバインディング** はコンポーネントレベルでアクションを処理します（todos を切り替え、送信など）
- Vim モードの Escape キーは INSERT から NORMAL モードに切り替わります。`chat:cancel` をトリガーしません
- ほとんどの Ctrl+key ショートカットは Vim モードを通過してキーバインディングシステムに渡されます
- Vim キーはキーバインディングファイルを通じて再マップできません。`jj` を Escape にマップするような 2 キーの INSERT モードシーケンスをマップするには、*`vimInsertModeRemaps`* (ページ 84) 設定を使用してください
- Vim NORMAL モードでは、`?` はヘルプメニューを表示します（Vim の動作）
- Vim NORMAL モードでは、`/` は履歴検索を開きます。標準モードの Ctrl+R と同じです

29.10 検証

Claude Code はキーバインディングを検証し、以下の警告をデバッグログに書き込みます。

- 解析エラー（無効な JSON または構造）
- スペルミスのあるモディファイア（例：`ctl+k`）。Claude Code は認識できない部分を削除し、残りのキーストロークにバインディングを適用します。この例では `k` です。
- 無効なコンテキスト名
- 無効なアクション値（アクションが文字列または `null` ではない場合など）
- 不明なアクション名（登録されたアクションのタイプミスなど）。Claude Code はバインディングをスキップし、そのキーのデフォルトバインディングを有効に保ちます。
- 予約済みショートカットの競合
- 同じコンテキスト内の重複バインディング

Claude Code を *`--debug`* (ページ 51) で起動して、詳細を確認してください。

第30章　ステータスラインをカスタマイズする

> Claude Code でコンテキストウィンドウの使用状況、コスト、git ステータスを監視するカスタムステータスバーを設定します

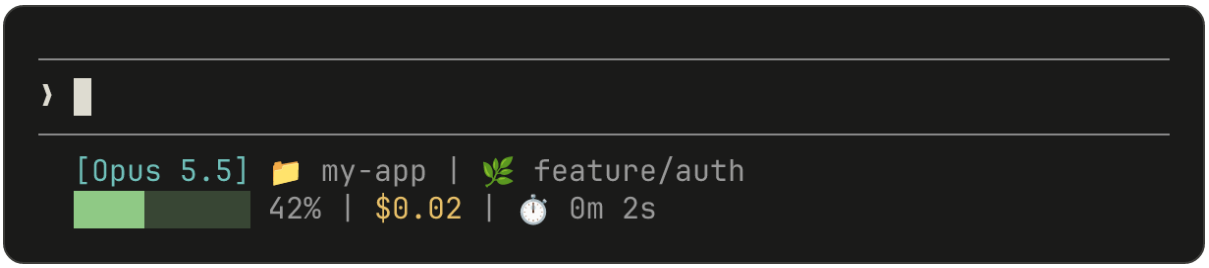
ステータスラインは Claude Code の下部にあるカスタマイズ可能なバーで、設定したシェルスクリプトを実行します。stdin 経由で JSON セッションデータを受け取り、スクリプトが出力したものを表示し、コンテキスト使用状況、コスト、git ステータス、またはその他の追跡したい情報を一目で確認できる永続的なビューを提供します。

ステータスラインは以下の場合に便利です：

- 作業中にコンテキストウィンドウの使用状況を監視したい
- セッションコストを追跡する必要がある
- 複数のセッション間で作業し、それらを区別する必要がある
- git ブランチとステータスを常に表示したい

ステータスラインは組み込みのフッターバッジの上にある独自の行にレンダリングされ、それらを置き換えません。カスタムステータスラインが設定されている場合、Claude Code はフッターのキーボードヒントのほとんどを表示しなくなります。これには `esc to interrupt`、`? for shortcuts` フォールバック、および `hold space to speak` 音声入力 ヒントが含まれます。会話内に ID が表示されたときにフッターにクリック可能なリンクバッジを追加する場合は、スクリプトを記述せずに `footerLinksRegexes` を設定してください。

以下は、最初の行に git 情報を表示し、2 番目の行にカラーコード化されたコンテキストバーを表示する **複数行ステータスライン** (ページ 804) の例です。



このページでは、**基本的なステータスラインの設定** (ページ 786) について説明し、Claude Code からスクリプトへの **データフロー** (ページ 788) について説明し、**表示できるすべてのフィールド** (ページ 789) をリストアップし、git ステータス、コスト追跡、プログレスバーなどの一般的なパターンの **すぐに使える例** (ページ 799) を提供します。

30.1 ステータスラインを設定する

/statusline **コマンド** (ページ 786) を使用して Claude Code にスクリプトを生成させるか、**手動でスクリプトを作成** (ページ 786) して設定に追加します。

30.1.1 /statusline コマンドを使用する

`/statusline` コマンドは、表示したい内容を説明する自然言語の指示を受け入れます。Claude Code は `~/.claude/` にスクリプトファイルを生成し、設定を自動的に更新します：

```
/statusline show model name and context percentage with a progress bar
```

セットアップ中に Claude Code が権限を求める場合は、ファイル編集プロンプトを承認してください。

30.1.2 ステータスラインを手動で設定する

ユーザー設定（`~/.claude/settings.json`、~ はホームディレクトリ）または **プロジェクト設定** (ページ 711) に `statusLine` フィールドを追加します。`type` を `"command"` に設定し、`command` をスクリプトパスまたはインラインシェルコマンドに指定します。スクリプト作成の完全なチュートリアルについては、**ステータスラインをステップバイステップで構築する** (ページ 787) を参照してください。

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh",
    "padding": 2
  }
}
```

`command` フィールドはシェルで実行されるため、スクリプトファイルの代わりにインラインコマンドを使用することもできます。この例では `jq` を使用して JSON 入力を解析し、モデル名とコンテキスト割合を表示します：

```
{
  "statusLine": {
    "type": "command",
    "command": "jq -r '\"[\\(.model.display_name)] \\(.context_window.used_percentage␣
↪// 0)% context\"'"
  }
}
```

オプションの `padding` フィールドは、ステータスラインコンテンツに追加の水平スペース（文字単位）を追加します。デフォルトは `0` です。このパディングはインターフェイスの組み込みスペースに加えて追加されるため、ターミナルエッジからの絶対距離ではなく相対的なインデントを制御します。

オプションの `refreshInterval` フィールドは、**イベント駆動更新** (ページ 788) に加えて、N 秒ごとにコマンドを再実行します。最小値は 1 です。ステータスラインが時計などの時間ベースのデータを表示する

場合、またはメインセッションがアイドル状態の間にバックグラウンドサブエージェントが git 状態を変更する場合に設定します。イベントのみで実行する場合は設定しないままにします。

オプションの `hideVimModeIndicator` フィールドは、プロンプトの下にある組み込みの `-- INSERT --` テキストを非表示にします。スクリプトが *vim.mode* (ページ 789) 自体をレンダリングする場合は、これを `true` に設定して、モードが 2 回表示されないようにします。

30.1.3 ステータスラインを無効にする

`/statusline` を実行し、ステータスラインを削除またはクリアするよう指示します（例：`/statusline delete`、`/statusline clear`、`/statusline remove it`）。settings.json から `statusLine` フィールドを手動で削除することもできます。

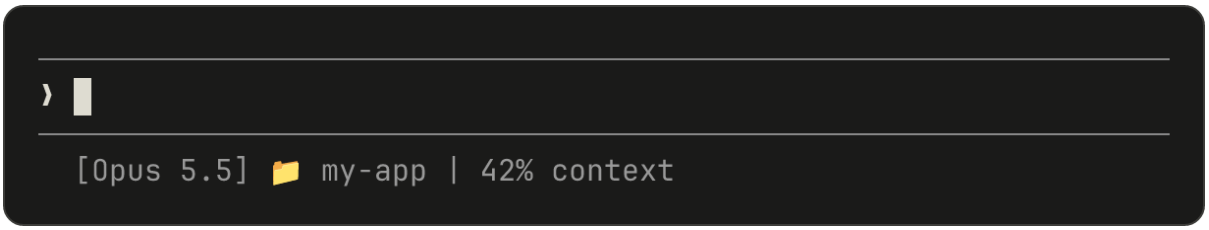
30.2 ステータスラインをステップバイステップで構築する

このチュートリアルでは、現在のモデル、作業ディレクトリ、コンテキストウィンドウ使用状況の割合を表示するステータスラインを手動で作成することで、`/statusline` が内部で何をセットアップするかを示します。

注釈:

/statusline (ページ 786) を実行して、表示したい内容を説明すると、これらすべてが自動的に設定されます。

これらの例では Bash スクリプトを使用しており、macOS と Linux で動作します。Windows では、*Windows 設定* (ページ 814) で PowerShell と Git Bash の例を参照してください。



1. **JSON を読み取り、出力を出力するスクリプトを作成する**

   Claude Code は stdin 経由でスクリプトに JSON データを送信します。このスクリプトは `jq`（コマンドラインの JSON パーサーで、インストールが必要な場合があります）を使用して、モデル名、ディレクトリ、コンテキスト割合を抽出し、フォーマットされた行を出力します。

   これを `~/.claude/statusline.sh` に保存します（~ はホームディレクトリ、macOS では `/Users/username`、Linux では `/home/username` など）：

```
#!/bin/bash
# Claude Code が stdin に送信する JSON データを読み取る
input=$(cat)

# jq を使用してフィールドを抽出する
```

(次のページに続く)

(前のページからの続き)

```
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
# "// 0" はフィールドが null の場合のフォールバックを提供します
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

# ステータスラインを出力します - ${DIR##*/} はフォルダ名のみを抽出します
echo "[$MODEL] ☒ ${DIR##*/} | ${PCT}% context"
```

2. **実行可能にする**

   スクリプトを実行可能にマークして、シェルが実行できるようにします：

   ```
   chmod +x ~/.claude/statusline.sh
   ```

3. **設定に追加する**

   Claude Code にスクリプトをステータスラインとして実行するよう指示します。この設定を `~/.claude/settings.json` に追加します。これは `type` を `"command"`（「このシェルコマンドを実行する」という意味）に設定し、`command` をスクリプトに指定します：

   ```
   {
     "statusLine": {
       "type": "command",
       "command": "~/.claude/statusline.sh"
     }
   }
   ```

   ステータスラインはインターフェイスの下部に表示されます。Claude Code は設定を自動的に再読み込みし、ファイルを保存するとすぐにスクリプトを実行します。

30.3 ステータスラインの仕組み

Claude Code はスクリプトを実行し、stdin 経由で *JSON* **セッションデータ** (ページ 789) をパイプします。スクリプトが stdout に出力したものを Claude Code が表示します。

更新のタイミング

スクリプトはセッション開始時（再開時を含む）に 1 回実行されます。その後、以下の場合に再度実行されます：

- 新しいアシスタントメッセージが到着したとき
- `/compact` が完了したとき
- パーミッション権限モードが変更されたとき
- Vim モードが切り替わったとき
- `statusLine` 設定で `command` を変更したとき
- *`refreshInterval`* (ページ 786) タイマーが経過したとき（設定した場合）

- スクリプトが最後に受け取ったデータ内の **レート制限ウィンドウ** (ページ 809) が `resets_at` 時刻に到達したとき
- スクリプトが最後に受け取ったデータ内のウォーム **プロンプトキャッシュ** (ページ 797) が `expires_at` 時刻に到達したとき

Claude Code は更新を 300ms でデバウンスするため、急速な変更がバッチ処理され、スクリプトは変更が停止した後に 1 回実行されます。`command` 自体への変更はデバウンスをスキップします。Claude Code は新しいコマンドをすぐに実行します。スクリプトがまだ実行中に新しい更新がトリガーされた場合、Claude Code は実行中のスクリプトをキャンセルします。スクリプトを編集した場合、変更は更新トリガーが次に実行されるときに表示されます。

イベント駆動型トリガーは、メインセッションがアイドル状態の場合（例えば、コーディネーターがバックグラウンドサブエージェントを待機している場合）、静かになる可能性があります。アイドル期間中に時間ベースまたは外部ソースのセグメントを最新に保つには、*`refreshInterval`* (ページ 786) を設定して、固定タイマーでもコマンドを再実行します。

スクリプトが出力できるもの

- **複数行**：各 `echo` または `print` ステートメントは別の行として表示されます。**複数行の例** (ページ 804) を参照してください。
- **色**：`\033[32m` のような ANSI エスケープコード を使用して緑色を表示します（ターミナルがサポートしている必要があります）。*git* **ステータスの例** (ページ 800) を参照してください。
- **リンク**：OSC 8 エスケープシーケンス を使用してテキストをクリック可能にします（macOS では Cmd+クリック、Windows/Linux では Ctrl+クリック）。iTerm2、Kitty、WezTerm などのハイパーリンクをサポートするターミナルが必要です。**クリック可能なリンクの例** (ページ 807) を参照してください。

ターミナルに出力をサイズ調整する

Claude Code はスクリプトの出力をキャプチャするため、ターミナルに直接接続しません。そのため、`tput cols` と言語レベルの幅検出はスクリプト内からターミナルサイズを読み取ることができません。代わりに `COLUMNS` および `LINES` 環境変数を読み取ってください。Claude Code はスクリプトを実行する前に、これらを現在のターミナルサイズに設定します。

注釈:

ステータスラインはローカルで実行され、API トークンを消費しません。ヘルプメニューとパーミッションプロンプトを含む特定の UI 相互作用中は一時的に非表示になります。

30.4 利用可能なデータ

Claude Code は以下の JSON フィールドを stdin 経由でスクリプトに送信します：

フィールド	説明
`model.id`、`model.display_name`	現在のモデル識別子と表示名

次のページに続く

表 1 – 前のページからの続き

フィールド	説明
cwd、workspace.current_dir	現在の作業ディレクトリ。両方のフィールドに同じ値が含まれます。workspace.current_dir は workspace.project_dir との一貫性のために推奨されます。
workspace.project_dir	Claude Code が起動されたディレクトリ。セッション中に作業ディレクトリが変更された場合、cwd と異なる場合があります
workspace.added_dirs	/add-dir または --add-dir 経由で追加された追加ディレクトリ。追加されていない場合は空配列
workspace.git_worktree	現在のディレクトリが git worktree add で作成されたリンク worktree 内にある場合の git worktree 名。メイン作業ツリーでは不在。worktree.* が worktree セッション 中のみに存在するのとは異なり、任意の git worktree に対して入力されます
workspace.repo.host、workspace.repo.owner、workspace.repo.name	origin リモートから解析されたリポジトリ ID。例えば "github.com"、"anthropics"、"claude-code"。git リポジトリの外部または origin リモートが設定されていない場合は不在。gitlab.com プロジェクトがサブグループにネストされている場合、owner は "group/subgroup" のようなスラッシュ付きの完全な名前空間パスです。v2.1.260 より前は、これらのプロジェクトでは workspace.repo が不在でした
cost.total_cost_usd	USD でのセッションの推定コスト。クライアント側で計算されます。modelPricing テーブルが有効な場合を除き、定価で計算されます。実際の請求額と異なる場合があります。/clear で新しいセッションが開始されると $0 にリセットされます。v2.1.211 より前は、/clear の後も合計が引き継がれていました
cost.total_duration_ms	セッション開始からの総経過時間（ミリ秒）
cost.total_api_duration_ms	API レスポンスを待つのに費やされた総時間（ミリ秒）
cost.total_lines_added、cost.total_lines_removed	変更されたコード行
context_window.total_input_tokens、context_window.total_output_tokens	コンテキストウィンドウに現在あるトークン数。最新の API レスポンスから取得。入力にはキャッシュ読み取りと書き込みが含まれます
context_window.context_window_size	トークン単位の最大コンテキストウィンドウサイズ。デフォルトは 200000、拡張コンテキストを持つモデルの場合は 1000000
context_window.used_percentage	事前計算されたコンテキストウィンドウ使用割合
context_window.remaining_percentage	事前計算されたコンテキストウィンドウ残り割合

次のページに続く

表 1 – 前のページからの続き

フィールド	説明
context_window.current_usage	最後の API 呼び出しからのトークン数。**コンテキストウィンドウフィールド** (ページ 796) で説明されています
exceeds_200k_tokens	最新の API レスポンスからの総トークン数（入力、キャッシュ、出力トークンの組み合わせ）が 200k を超えるかどうか。これは実際のコンテキストウィンドウサイズに関係なく固定閾値です。
fast_mode	セッションで fast mode が有効になっているかどうか
effort.level	現在の推論努力レベル（low、medium、high、xhigh、または max）。ライブセッション値を反映しており、セッション中の /effort 変更を含みます。現在のモデルが effort パラメータをサポートしていない場合は不在
thinking.enabled	セッションで拡張思考が有効になっているかどうか
rate_limits.five_hour.used_percentage、rate_limits.seven_day.used_percentage	5 時間または 7 日のレート制限の消費割合（0〜100）
rate_limits.five_hour.resets_at、rate_limits.seven_day.resets_at	5 時間または 7 日のレート制限ウィンドウがリセットされる Unix エポック秒
rate_limits.spend_limit.used_percentage、rate_limits.spend_limit.resets_at	Claude apps gateway の背後にある場合、支出制限をどれだけ使用したか、およびその期間がいつリセットされるか。**支出制限フィールド** (ページ 797) を参照してください。Claude Code v2.1.251 以降が必要です
rate_limits.spend_limit.used_usd、rate_limits.spend_limit.limit_usd、rate_limits.spend_limit.period	米ドル建ての推定支出額と制限額、および制限の期間。これらのフィールドは不在の場合があります。**支出制限フィールド** (ページ 797) を参照してください。Claude Code とゲートウェイの両方で v2.1.284 以降が必要です
prompt_cache	メイン会話の prompt cache 統計情報：ヒット率、ミス数、キャッシュがウォーム状態かどうか。すべてのフィールドについては *prompt cache* **フィールド** (ページ 797) を参照してください。メイン会話の最初の API レスポンスまで不在。Claude Code v2.1.251 以降が必要です
session_id	一意のセッション識別子
session_name	セッション名。--name フラグまたは /rename で設定されたカスタム名が存在する場合はそれを使用し、そうでない場合は AI が生成したセッションタイトルを使用します。デフォルト表示名（my-app-3f など）はこのフィールドに入力されません。セッションにカスタム名も AI が生成したタイトルもない場合は不在

次のページに続く

表 1 – 前のページからの続き

フィールド	説明
prompt_id	現在処理中のユーザープロンプトを識別する UUID。OpenTelemetry イベントの *prompt.id* **属性** (ページ 962) と一致します。最初のユーザー入力まで不在。Claude Code v2.1.196 以降が必要です
transcript_path	会話トランスクリプトファイルへのパス
version	Claude Code バージョン
output_style.name	現在の出力スタイルの名前
vim.mode	*vim* **モード** (ページ 84) が有効な場合の現在の vim モード（NORMAL、INSERT、VISUAL、または VISUAL LINE）
agent.name	--agent フラグまたはエージェント設定が設定されている場合のエージェント名
pr.number、pr.url	現在のブランチのオープンプルリクエスト。フッターの PR バッジをミラーします。リポジトリに GitLab リモートがある場合、Claude Code はこれらのフィールドをブランチのオープン *merge request* (ページ 103) から代わりに入力するため、pr.number はマージリクエスト番号です。マージリクエストデータには Claude Code v2.1.234 以降が必要です。git リポジトリにない場合、プルリクエストまたはマージリクエストが見つかるまで、またはマージもしくはクローズされた後は不在
pr.review_state	オープン PR のレビューステータス：approved、pending、changes_requested、または draft。pr が存在する場合でも独立して不在の可能性があります
pr.kind	*GitLab merge request* (ページ 103) を説明する場合は mr。GitHub プルリクエストでは不在なため、このフィールドの前に書かれたスクリプトは引き続き機能します。マージリクエストの場合、Claude Code は GitLab がマージ可能と報告した場合は review_state を approved に、その他のオープン状態の場合は pending に、ドラフトの場合は draft に設定します。Claude Code v2.1.234 以降が必要です
worktree.name	アクティブな worktree の名前。worktree セッション 中のみ存在
worktree.path	worktree ディレクトリへの絶対パス
worktree.branch	worktree の git ブランチ名（例："worktree-my-feature"）。フックベースの worktree では不在
worktree.original_cwd	worktree に入る前に Claude がいたディレクトリ
worktree.original_branch	worktree に入る前にチェックアウトされた git ブランチ。フックベースの worktree では不在

完全な JSON スキーマ

ステータスラインコマンドは stdin 経由でこの JSON 構造を受け取ります：

```
{
  "cwd": "/current/working/directory",
  "session_id": "abc123...",
  "session_name": "my-session",
  "prompt_id": "550e8400-e29b-41d4-a716-446655440000",
  "transcript_path": "/path/to/transcript.jsonl",
  "model": {
    "id": "claude-opus-5-5",
    "display_name": "Opus"
  },
  "workspace": {
    "current_dir": "/current/working/directory",
    "project_dir": "/original/project/directory",
    "added_dirs": [],
    "git_worktree": "feature-xyz",
    "repo": {
      "host": "github.com",
      "owner": "anthropics",
      "name": "claude-code"
    }
  },
  "version": "2.1.90",
  "output_style": {
    "name": "default"
  },
  "cost": {
    "total_cost_usd": 0.01234,
    "total_duration_ms": 45000,
    "total_api_duration_ms": 2300,
    "total_lines_added": 156,
    "total_lines_removed": 23
  },
  "context_window": {
    "total_input_tokens": 15500,
    "total_output_tokens": 1200,
    "context_window_size": 200000,
    "used_percentage": 8,
    "remaining_percentage": 92,
    "current_usage": {
      "input_tokens": 8500,
      "output_tokens": 1200,
      "cache_creation_input_tokens": 5000,
      "cache_read_input_tokens": 2000
    }
```

(次のページに続く)

(前のページからの続き)


```
  },
  "exceeds_200k_tokens": false,
  "prompt_cache": {
    "warm": true,
    "caching_observed": true,
    "ttl": "1h",
    "expires_at": 1738429200,
    "requests": 14,
    "misses": 2,
    "expected_rebuilds": 1,
    "hit_ratio": 0.91,
    "cache_write_tokens": 352000,
    "miss_recache_tokens": 310200,
    "last_miss_at": 1738425230,
    "last_miss_cause": {
      "causes": ["tools_changed"],
      "tools_added": 2,
      "tools_removed": 0
    },
    "miss_causes": {
      "tools_changed": 2
    },
    "recache_tokens_if_cold": 45000
  },
  "fast_mode": false,
  "effort": {
    "level": "high"
  },
  "thinking": {
    "enabled": true
  },
  "rate_limits": {
    "five_hour": {
      "used_percentage": 23.5,
      "resets_at": 1738425600
    },
    "seven_day": {
      "used_percentage": 41.2,
      "resets_at": 1738857600
    },
    "spend_limit": {
      "used_percentage": 62.8,
      "resets_at": 1740787200,
      "used_usd": 314.12,
      "limit_usd": 500,
```


(次のページに続く)

(前のページからの続き)

```
    "period": "monthly"
  }
},
"vim": {
  "mode": "NORMAL"
},
"agent": {
  "name": "security-reviewer"
},
"pr": {
  "number": 1234,
  "url": "https://github.com/anthropics/claude-code/pull/1234",
  "review_state": "pending"
},
"worktree": {
  "name": "my-feature",
  "path": "/path/to/.claude/worktrees/my-feature",
  "branch": "worktree-my-feature",
  "original_cwd": "/path/to/project",
  "original_branch": "main"
}
}
```

不在の可能性があるフィールド（JSON に存在しない）：

- `session_name`：`--name` または `/rename` でカスタム名が設定されている場合、または AI が生成したセッションタイトルが存在する場合に表示されます。`my-app-3f` などのデフォルト表示名はこのフィールドに入力されません
- `prompt_id`：最初のユーザー入力の後のみ表示
- `workspace.git_worktree`：現在のディレクトリがリンク git worktree 内にある場合のみ表示
- `workspace.repo`：git リポジトリ内で `origin` リモートが設定されている場合のみ表示
- `effort`：現在のモデルが推論努力パラメータをサポートしている場合のみ表示
- `vim`：vim モードが有効な場合のみ表示
- `agent`：`--agent` フラグまたはエージェント設定が設定されている場合のみ表示
- `pr`：現在のブランチのオープン PR または GitLab マージリクエストが見つかった場合のみ表示。PR またはマージリクエストがマージまたはクローズされると削除されます。`pr.review_state` と `pr.kind` は独立して不在の可能性があります
- `worktree`：worktree セッション 中のみ表示。存在する場合、`branch` と `original_branch` もフックベースの worktree では不在の可能性があります
- `rate_limits`：Claude.ai Pro および Max サブスクライバー、または支出制限を設定する Claude apps gateway の背後にある場合のみ表示。セッションの最初の API レスポンスの後のみ表示。各

ウィンドウ（five_hour、seven_day、spend_limit）は独立して不在の可能性があり、Claude Code は resets_at 時刻が経過するとウィンドウを削除します。jq -r '.rate_limits.five_hour.used_percentage // empty' を使用して、不在を適切に処理します。

- prompt_cache：メイン会話の最初の API レスポンスの後に表示。*prompt cache* **フィールド** (ページ 797) を参照してください

null **の可能性があるフィールド**：

- context_window.current_usage：セッションの最初の API 呼び出しの前は null。また /compact の直後は次の API 呼び出しが再度入力されるまで null
- context_window.used_percentage、context_window.remaining_percentage：セッションの早期段階では null の可能性があります

スクリプトで条件付きアクセスと null 値のフォールバックデフォルトを使用して、不在のフィールドを処理します。

30.4.1 コンテキストウィンドウフィールド

context_window オブジェクトは、最新の API レスポンスからのライブコンテキストウィンドウを説明します。

- **結合合計**（total_input_tokens、total_output_tokens）：コンテキストウィンドウに現在あるトークン。total_input_tokens は input_tokens、cache_creation_input_tokens、および cache_read_input_tokens の合計です。total_output_tokens は最新レスポンスからの出力トークンです。両方とも最初の API レスポンスの前は 0 です。
- **コンポーネント別使用状況**（current_usage）：カテゴリ別に分類された同じトークン数。キャッシュヒットを新規入力から分離する必要がある場合に使用します。

current_usage オブジェクトには以下が含まれます：

- input_tokens：現在のコンテキストの入力トークン
- output_tokens：生成された出力トークン
- cache_creation_input_tokens：キャッシュに書き込まれたトークン
- cache_read_input_tokens：キャッシュから読み取られたトークン

キャッシュフィールドの意味とそれらがどのように請求されるかについては、キャッシュパフォーマンスの確認 を参照してください。

used_percentage フィールドは入力トークンのみから計算されます：input_tokens + cache_creation_input_tokens + cache_read_input_tokens。output_tokens は含まれません。

current_usage から手動でコンテキスト割合を計算する場合、used_percentage と一致させるために同じ入力のみの式を使用します。

current_usage オブジェクトはセッションの最初の API 呼び出しの前は null です。また /compact の直後は null であり、次の API 呼び出しが再度入力されるまで null のままです。

30.4.2 支出制限フィールド

支出制限が設定された Claude apps gateway の背後では、rate_limits.spend_limit オブジェクトが適用される支出制限を説明します。このオブジェクトはセッションの最初の API レスポンスの後に表示され、Claude Code v2.1.251 以降が必要です。スクリプトはそのフィールドを別々のタイミングで受け取ります：

- used_percentage と resets_at：すべてのレスポンスに含まれるため、spend_limit が存在する場合は常に存在します。used_percentage は 0〜100 の範囲、または制限を超えると 100 を超える値になり、resets_at は制限の期間がリセットされる Unix エポック秒です。
- used_usd、limit_usd、period：これまでの推定支出額と制限額（米ドル）、および制限の対象期間（daily、weekly、monthly のいずれか）。ゲートウェイは used_usd をトークン数から計算する ため、これは推定値であり、請求額ではありません。Claude Code は、リクエストを送信している間、約 5 分ごとに別のリクエストでゲートウェイからこれらを読み取ります。金額は used_percentage より約 5 分古い場合があるため、両者が一時的に一致しないことがあります。Claude Code とゲートウェイの両方で v2.1.284 以降が必要です。

spend_limit が存在する場合でも、used_usd、limit_usd、period はオプションとして扱ってください。スクリプトはこれらより先に割合を受け取ります。また、CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC を設定するとそのリクエストがオフになるため、これらは不在のままになります。スクリプトでは、jq -r '.rate_limits.spend_limit.used_usd // empty' のように、それぞれをフォールバック付きで読み取ってください。

30.4.3 Prompt cache フィールド

prompt_cache オブジェクトは、セッションのメイン会話が prompt cache をどのように使用しているかを要約します。Claude Code は API のレスポンスのキャッシュトークン数から計算するため、すべてのプロバイダーで機能します。

オブジェクトはメイン会話の最初の API レスポンスの後に表示されます。Claude Code はこれらの統計情報でサブエージェントリクエストをカウントしません。Claude Code v2.1.251 以降が必要です。

テーブルは各フィールドとその意味を示しています。タイムスタンプは Unix エポック秒で、rate_limits.*.resets_at と同じ単位です。短いステータスラインは通常、これらの 1 つまたは 2 つを表示します。warm と hit_ratio はキャッシュ状態を最も直接的に要約します。

フィールド	説明
warm	キャッシュされたプレフィックスがまだ TTL 内にあるかどうか。最後のレスポンスがキャッシュトークンを報告しなかった場合は false。caching_observed が true の場合でも
caching_observed	このセッションのレスポンスがキャッシュトークンを報告したかどうか。false は prompt caching がオフ、またはプロバイダーまたはゲートウェイがそれを報告しないことを意味します
ttl	現在のキャッシュされたプレフィックスの キャッシュライフタイム："5m" または "1h"
expires_at	キャッシュされたプレフィックスが TTL を離れてコールドになる時刻（エポック秒）。最後のレスポンスがキャッシュトークンを報告しなかった場合は null

次のページに続く

表 2 – 前のページからの続き

フィールド	説明
`requests`	このセッションのメイン会話で記録された API リクエスト
`misses`	キャッシュが既に保持していたコンテンツを再処理したリクエスト：キャッシュが読み取ることができた 5% 以上かつ少なくとも 2,000 トークン。コンパクションまたはツール結果のクリアで不足を説明できない場合
`expected_rebuilds`	コンパクションまたは古いツール結果のクリアに続いたキャッシュリビルド
`hit_ratio`	キャッシュ読み取りトークンをこのセッションのすべての入力トークンの分数として表したもの。0〜1 の範囲。分母はキャッシュ読み取り、キャッシュ書き込み、およびキャッシュされていない入力をカウントします。これらのカウントがすべてゼロの場合は `null`
`cache_write_tokens`	このセッション中にキャッシュに書き込まれたすべてのトークン。最初のリクエストの初期書き込みを含む
`miss_recache_tokens`	ミスとしてカウントされたリクエストによってキャッシュに書き込まれたトークン
`last_miss_at`	最後のミスが発生した時刻（エポック秒）。セッションにミスがない場合は `null`
`last_miss_cause`	Claude Code が最後のミスの可能な原因として特定したもの。**最後のミスの原因** (ページ 798) で説明されています。Claude Code v2.1.260 以降が必要です
`miss_causes`	このセッションの診断されたミスのうち、各原因を持つ数。`last_miss_cause` と同じ原因名でキー付けされています。Claude Code v2.1.260 以降が必要です
`recache_tokens_if_cold`	キャッシュがそれまでにコールドになった場合、次のリクエストが再キャッシュするトークン。コンパクションまたは古いツール結果のクリアの直後は `null`。次のリクエストが書き直された会話のサイズを記録するまで `null` のままです

Claude Code はターミナルで同じ統計情報を表示します。*/usage* **コマンドの** *Prompt cache (main)* **行** (ページ 912) を参照してください。

最後のミスの原因

`last_miss_cause` オブジェクトは、Claude Code が最新のミスの可能な原因として特定したものを報告します。その `causes` 配列は `tools_changed`、`system_prompt_changed`、`ttl_expired_5m`、または `likely_server_side` などの 1 つ以上の原因名を保持します。オブジェクトはセッションの最初のミスまで `null` であり、Claude Code が最新のミスの原因を特定できなかった場合は再び `null` になります。Claude Code v2.1.260 以降が必要です。

2 つの原因はオブジェクトにカウントを追加します：

- `tools_added` と `tools_removed`：`tools_changed` を使用して、リクエストに追加または削除されたツールの数

- `system_char_delta`：`system_prompt_changed` を使用して、システムプロンプトの長さの変化（文字数）

30.5 例

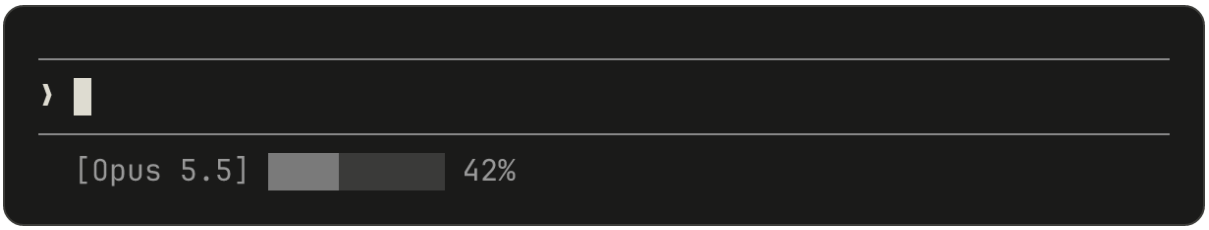
これらの例は一般的なステータスラインパターンを示しています。任意の例を使用するには：

1. スクリプトを `~/.claude/statusline.sh`（または `.py`/`.js`）などのファイルに保存します
2. 実行可能にします：`chmod +x ~/.claude/statusline.sh`
3. **設定** (ページ 786) にパスを追加します

Bash の例は `jq` を使用して JSON を解析します。Python と Node.js には組み込みの JSON 解析があります。

30.5.1 コンテキストウィンドウの使用状況

現在のモデルとコンテキストウィンドウの使用状況を視覚的なプログレスバーで表示します。各スクリプトは stdin から JSON を読み取り、`used_percentage` フィールドを抽出し、塗りつぶされたブロック（⛒）が使用状況を表す 10 文字のバーを構築します：



リスト 1: Bash

```bash
#!/bin/bash
# stdin 全体を変数に読み込む
input=$(cat)

# jq でフィールドを抽出します。"// 0" は null のフォールバックを提供します
MODEL=$(echo "$input" | jq -r '.model.display_name')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)

# プログレスバーを構築します：printf -v はスペースを作成し、
# ${var// /⛒} は各スペースをブロック文字に置き換えます
BAR_WIDTH=10
FILLED=$((PCT * BAR_WIDTH / 100))
EMPTY=$((BAR_WIDTH - FILLED))
BAR=""
[ "$FILLED" -gt 0 ] && printf -v FILL "%${FILLED}s" && BAR="${FILL// /⛒}"
[ "$EMPTY" -gt 0 ] && printf -v PAD "%${EMPTY}s" && BAR="${BAR}${PAD// /⛒}"

echo "[$MODEL] $BAR $PCT%"
```

リスト 2: Python

```python
#!/usr/bin/env python3
import json, sys

# json.load は stdin を 1 ステップで読み取り、解析します
data = json.load(sys.stdin)
model = data['model']['display_name']
# "or 0" は null 値を処理します
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)

# 文字列乗算がバーを構築します
filled = pct * 10 // 100
bar = '⛒' * filled + '⛶' * (10 - filled)

print(f"[{model}] {bar} {pct}%")
```

リスト 3: Node.js

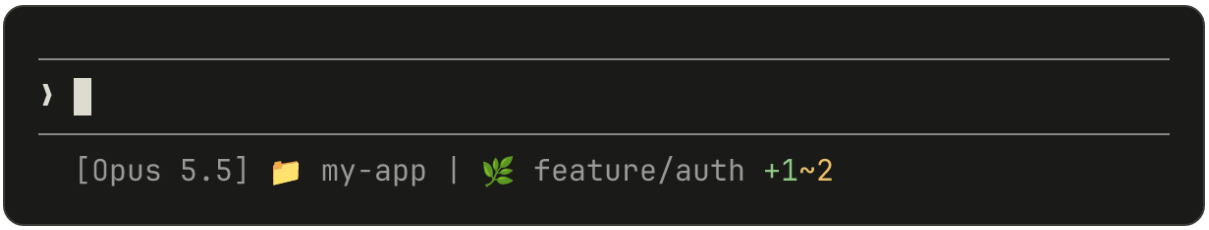
```javascript
#!/usr/bin/env node
// Node.js はイベントで stdin を非同期に読み取ります
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    // オプショナルチェーン (?.) は null フィールドを安全に処理します
    const pct = Math.floor(data.context_window?.used_percentage || 0);

    // String.repeat() がバーを構築します
    const filled = Math.floor(pct * 10 / 100);
    const bar = '⛒'.repeat(filled) + '⛶'.repeat(10 - filled);

    console.log(`[${model}] ${bar} ${pct}%`);
});
```

30.5.2 git ステータスと色

ステージングされたファイルと変更されたファイルのカラーコード化されたインジケーターを使用して git ブランチを表示します。このスクリプトはターミナルの色に ANSI エスケープコード を使用します：`\033[32m` は緑、`\033[33m` は黄、`\033[0m` はデフォルトにリセットします。



各スクリプトは現在のディレクトリが git リポジトリであるかどうかを確認し、ステージングされたファイルと変更されたファイルをカウントし、カラーコード化されたインジケーターを表示します：

リスト 4: Bash

```bash
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')

GREEN='\033[32m'
YELLOW='\033[33m'
RESET='\033[0m'

if git rev-parse --git-dir > /dev/null 2>&1; then
    BRANCH=$(git branch --show-current 2>/dev/null)
    STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
    MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')

    GIT_STATUS=""
    [ "$STAGED" -gt 0 ] && GIT_STATUS="${GREEN}+${STAGED}${RESET}"
    [ "$MODIFIED" -gt 0 ] && GIT_STATUS="${GIT_STATUS}${YELLOW}~${MODIFIED}${RESET}"

    echo -e "[$MODEL] ☒ ${DIR##*/} | ☒ $BRANCH $GIT_STATUS"
else
    echo "[$MODEL] ☒ ${DIR##*/}"
fi
```

リスト 5: Python

```python
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])

GREEN, YELLOW, RESET = '\033[32m', '\033[33m', '\033[0m'
```

(次のページに続く)

(前のページからの続き)

```
try:
    subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.
↪DEVNULL)
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=True).
↪strip()
    staged_output = subprocess.check_output(['git', 'diff', '--cached', '--numstat'],␣
↪text=True).strip()
    modified_output = subprocess.check_output(['git', 'diff', '--numstat'],␣
↪text=True).strip()
    staged = len(staged_output.split('\n')) if staged_output else 0
    modified = len(modified_output.split('\n')) if modified_output else 0

    git_status = f"{GREEN}+{staged}{RESET}" if staged else ""
    git_status += f"{YELLOW}~{modified}{RESET}" if modified else ""

    print(f"[{model}] ☒ {directory} | ☒ {branch} {git_status}")
except:
    print(f"[{model}] ☒ {directory}")
```

リスト 6: Node.js

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const dir = path.basename(data.workspace.current_dir);

    const GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RESET = '\x1b[0m';

    try {
        execSync('git rev-parse --git-dir', { stdio: 'ignore' });
        const branch = execSync('git branch --show-current', { encoding: 'utf8' }).
↪trim();
        const staged = execSync('git diff --cached --numstat', { encoding: 'utf8' }).
↪trim().split('\n').filter(Boolean).length;
        const modified = execSync('git diff --numstat', { encoding: 'utf8' }).trim().
↪split('\n').filter(Boolean).length;

        let gitStatus = staged ? `${GREEN}+${staged}${RESET}` : '';
        gitStatus += modified ? `${YELLOW}~${modified}${RESET}` : '';
```

(次のページに続く)

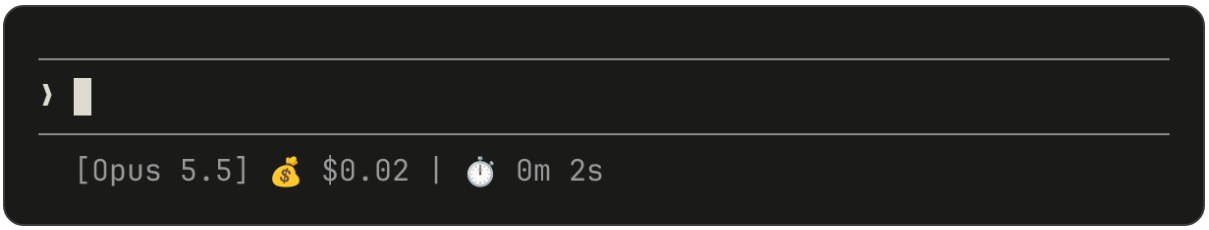
(前のページからの続き)

```
        console.log(`[${model}] ☒ ${dir} | ☒ ${branch} ${gitStatus}`);
    } catch {
        console.log(`[${model}] ☒ ${dir}`);
    }
});
```

30.5.3 コストと期間の追跡

セッションの API コストと経過時間を追跡します。cost.total_cost_usd フィールドは現在のセッションのすべての API 呼び出しの推定コストを累積します。cost.total_duration_ms フィールドはセッション開始からの総経過時間を測定し、cost.total_api_duration_ms は API レスポンスを待つのに費やされた時間のみを追跡します。

各スクリプトはコストを通貨としてフォーマットし、ミリ秒を分と秒に変換します：



リスト 7: Bash

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

COST_FMT=$(printf '$%.2f' "$COST")
DURATION_SEC=$((DURATION_MS / 1000))
MINS=$((DURATION_SEC / 60))
SECS=$((DURATION_SEC % 60))

echo "[$MODEL] ☒ $COST_FMT | ☒☒ ${MINS}m ${SECS}s"
```

リスト 8: Python

```
#!/usr/bin/env python3
import json, sys

data = json.load(sys.stdin)
model = data['model']['display_name']
```

(次のページに続く)

(前のページからの続き)

```
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

duration_sec = duration_ms // 1000
mins, secs = duration_sec // 60, duration_sec % 60

print(f"[{model}] ☒ ${cost:.2f} | ☒☒ {mins}m {secs}s")
```

リスト 9: Node.js

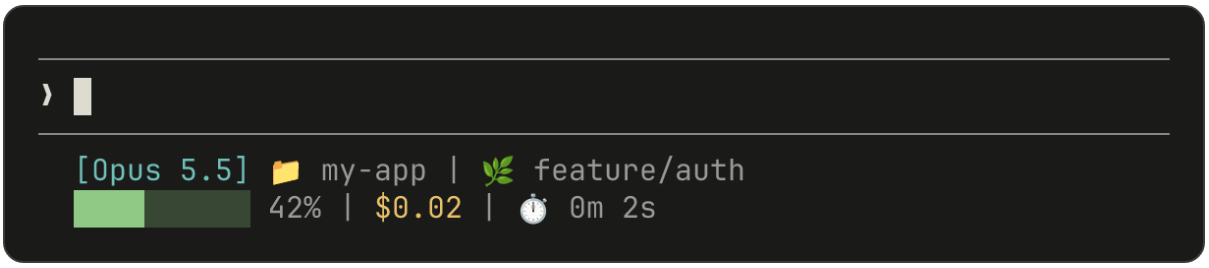
```
#!/usr/bin/env node
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const cost = data.cost?.total_cost_usd || 0;
    const durationMs = data.cost?.total_duration_ms || 0;

    const durationSec = Math.floor(durationMs / 1000);
    const mins = Math.floor(durationSec / 60);
    const secs = durationSec % 60;

    console.log(`[${model}] ☒ $${cost.toFixed(2)} | ☒☒ ${mins}m ${secs}s`);
});
```

30.5.4 複数行を表示する

スクリプトは複数の行を出力して、より豊かなディスプレイを作成できます。



この例は複数のテクニックを組み合わせています：閾値ベースの色（70% 未満は緑、70〜89% は黄、90% 以上は赤）、プログレスバー、git ブランチ情報。各 `print` または `echo` ステートメントは別の行を作成します：

リスト 10: Bash

```
#!/bin/bash
input=$(cat)
```

(次のページに続く)

(前のページからの続き)

```
MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
COST=$(echo "$input" | jq -r '.cost.total_cost_usd // 0')
PCT=$(echo "$input" | jq -r '.context_window.used_percentage // 0' | cut -d. -f1)
DURATION_MS=$(echo "$input" | jq -r '.cost.total_duration_ms // 0')

CYAN='\033[36m'; GREEN='\033[32m'; YELLOW='\033[33m'; RED='\033[31m'; RESET='\033[0m'

# コンテキスト使用状況に基づいてバーの色を選択します
if [ "$PCT" -ge 90 ]; then BAR_COLOR="$RED"
elif [ "$PCT" -ge 70 ]; then BAR_COLOR="$YELLOW"
else BAR_COLOR="$GREEN"; fi

FILLED=$((PCT / 10)); EMPTY=$((10 - FILLED))
printf -v FILL "%${FILLED}s"; printf -v PAD "%${EMPTY}s"
BAR="${FILL// /■}${PAD// /⛒}"

MINS=$((DURATION_MS / 60000)); SECS=$(((DURATION_MS % 60000) / 1000))

BRANCH=""
git rev-parse --git-dir > /dev/null 2>&1 && BRANCH=" | ⛒ $(git branch --show-current␣
↪2>/dev/null)"

echo -e "${CYAN}[$MODEL]${RESET} ⛒ ${DIR##*/}$BRANCH"
COST_FMT=$(printf '$%.2f' "$COST")
echo -e "${BAR_COLOR}${BAR}${RESET} ${PCT}% | ${YELLOW}${COST_FMT}${RESET} | ⛒⛒ $
↪{MINS}m ${SECS}s"
```

リスト 11: Python

```
#!/usr/bin/env python3
import json, sys, subprocess, os

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])
cost = data.get('cost', {}).get('total_cost_usd', 0) or 0
pct = int(data.get('context_window', {}).get('used_percentage', 0) or 0)
duration_ms = data.get('cost', {}).get('total_duration_ms', 0) or 0

CYAN, GREEN, YELLOW, RED, RESET = '\033[36m', '\033[32m', '\033[33m', '\033[31m', '\
↪033[0m'

bar_color = RED if pct >= 90 else YELLOW if pct >= 70 else GREEN
```

(次のページに続く)

(前のページからの続き)

```
filled = pct // 10
bar = '' * filled + '' * (10 - filled)

mins, secs = duration_ms // 60000, (duration_ms % 60000) // 1000

try:
    branch = subprocess.check_output(['git', 'branch', '--show-current'], text=True,␣
↪stderr=subprocess.DEVNULL).strip()
    branch = f" |  {branch}" if branch else ""
except:
    branch = ""

print(f"{CYAN}[{model}]{RESET}  {directory}{branch}")
print(f"{bar_color}{bar}{RESET} {pct}% | {YELLOW}${cost:.2f}{RESET} |  {mins}m
↪{secs}s")
```

リスト 12: Node.js

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const dir = path.basename(data.workspace.current_dir);
    const cost = data.cost?.total_cost_usd || 0;
    const pct = Math.floor(data.context_window?.used_percentage || 0);
    const durationMs = data.cost?.total_duration_ms || 0;

    const CYAN = '\x1b[36m', GREEN = '\x1b[32m', YELLOW = '\x1b[33m', RED = '\x1b[31m
↪', RESET = '\x1b[0m';

    const barColor = pct >= 90 ? RED : pct >= 70 ? YELLOW : GREEN;
    const filled = Math.floor(pct / 10);
    const bar = ''.repeat(filled) + ''.repeat(10 - filled);

    const mins = Math.floor(durationMs / 60000);
    const secs = Math.floor((durationMs % 60000) / 1000);

    let branch = '';
    try {
        branch = execSync('git branch --show-current', { encoding: 'utf8', stdio: [
```

(次のページに続く)

(前のページからの続き)

```
↪'pipe', 'pipe', 'ignore'] }).trim();
        branch = branch ? ` | ⌧ ${branch}` : '';
    } catch {}

    console.log(`${CYAN}[${model}]${RESET} ⌧ ${dir}${branch}`);
    console.log(`${barColor}${bar}${RESET} ${pct}% | ${YELLOW}$${cost.toFixed(2)}$
↪{RESET} | ⌧⌧ ${mins}m ${secs}s`);
});
```

30.5.5 クリック可能なリンク

この例は GitHub リポジトリへのクリック可能なリンクを作成します。Cmd（macOS）または Ctrl（Windows/Linux）を押しながらクリックして、ブラウザでリンクを開きます。

各スクリプトは git リモート URL を取得し、SSH 形式を HTTPS に変換し、リポジトリ名を OSC 8 エスケープコードでラップします。Bash バージョンは `printf '%b'` を使用します。これはバックスラッシュエスケープを異なるシェル間でより確実に解釈します：

リスト 13: Bash

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')

# git SSH URL を HTTPS に変換します
REMOTE=$(git remote get-url origin 2>/dev/null | sed 's/git@github.com:/https:\/\/
↪github.com\//' | sed 's/\.git$//')

if [ -n "$REMOTE" ]; then
    REPO_NAME=$(basename "$REMOTE")
    # OSC 8 形式：\e]8;;URL\a その後 TEXT その後 \e]8;;\a
    # printf %b はシェル間でエスケープシーケンスを確実に解釈します
    printf '%b' "[$MODEL] ⌧ \e]8;;${REMOTE}\a${REPO_NAME}\e]8;;\a\n"
else
    echo "[$MODEL]"
fi
```

リスト 14: Python

```python
#!/usr/bin/env python3
import json, sys, subprocess, re, os

data = json.load(sys.stdin)
model = data['model']['display_name']

# git リモート URL を取得します
try:
    remote = subprocess.check_output(
        ['git', 'remote', 'get-url', 'origin'],
        stderr=subprocess.DEVNULL, text=True
    ).strip()
    # SSH を HTTPS 形式に変換します
    remote = re.sub(r'^git@github\.com:', 'https://github.com/', remote)
    remote = re.sub(r'\.git$', '', remote)
    repo_name = os.path.basename(remote)
    # OSC 8 エスケープシーケンス
    link = f"\033]8;;{remote}\a{repo_name}\033]8;;\a"
    print(f"[{model}] 🔗 {link}")
except:
    print(f"[{model}]")
```

リスト 15: Node.js

```javascript
#!/usr/bin/env node
const { execSync } = require('child_process');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;

    try {
        let remote = execSync('git remote get-url origin', { encoding: 'utf8', stdio:␣
↪['pipe', 'pipe', 'ignore'] }).trim();
        // SSH を HTTPS 形式に変換します
        remote = remote.replace(/^git@github\.com:/, 'https://github.com/').replace(/\
↪.git$/, '');
        const repoName = path.basename(remote);
        // OSC 8 エスケープシーケンス
        const link = `\x1b]8;;${remote}\x07${repoName}\x1b]8;;\x07`;
```

(次のページに続く)

(前のページからの続き)

```
        console.log(`[${model}] ☒ ${link}`);
    } catch {
        console.log(`[${model}]`);
    }
});
```

30.5.6 レート制限の使用状況

claude.ai サブスクリプションのレート制限使用状況、または Claude アプリゲートウェイの支出制限に対する支出額をステータスラインに表示します。サブスクライバーの場合、rate_limits オブジェクトには、ローリング five_hour ウィンドウと週間 seven_day ウィンドウが含まれます。各ウィンドウは used_percentage（0〜100）とウィンドウがリセットされる Unix エポック秒の resets_at を提供します。ゲートウェイの背後では、支出制限フィールド (ページ 797) で説明されている spend_limit オブジェクトを読み取ります。

rate_limits オブジェクトは claude.ai Pro および Max サブスクライバー、または支出制限を持つ Claude アプリゲートウェイの背後にある場合のみ存在し、最初の API レスポンスの後のみです。各スクリプトは不在のフィールドを適切に処理し、ゲートウェイの背後では spend: $314.12 / $500 を出力します。ドル額のフィールドが不在の間は spend: 63% を出力します：

リスト 16: Bash

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
# "// empty" は rate_limits が不在の場合、出力を生成しません
FIVE_H=$(echo "$input" | jq -r '.rate_limits.five_hour.used_percentage // empty')
WEEK=$(echo "$input" | jq -r '.rate_limits.seven_day.used_percentage // empty')

LIMITS=""
[ -n "$FIVE_H" ] && LIMITS="5h: $(printf '%.0f' "$FIVE_H")%"
[ -n "$WEEK" ] && LIMITS="${LIMITS:+$LIMITS }7d: $(printf '%.0f' "$WEEK")%"

# Claude アプリゲートウェイの背後：ゲートウェイがドル額を報告する場合はドル額、それ以外はパーセンテージ
SPEND_PCT=$(echo "$input" | jq -r '.rate_limits.spend_limit.used_percentage // empty')
SPEND_USD=$(echo "$input" | jq -r '.rate_limits.spend_limit | select(.used_usd !=␣
↪null) | "$\(.used_usd) / $\(.limit_usd)"')
[ -n "$SPEND_PCT" ] && LIMITS="${LIMITS:+$LIMITS }spend: ${SPEND_USD:-$(printf '%.0f'
↪"$SPEND_PCT")%}"

[ -n "$LIMITS" ] && echo "[$MODEL] | $LIMITS" || echo "[$MODEL]"
```

リスト 17: Python

```python
#!/usr/bin/env python3
import json, sys

data = json.load(sys.stdin)
model = data['model']['display_name']

parts = []
rate = data.get('rate_limits', {})
five_h = rate.get('five_hour', {}).get('used_percentage')
week = rate.get('seven_day', {}).get('used_percentage')

if five_h is not None:
    parts.append(f"5h: {five_h:.0f}%")
if week is not None:
    parts.append(f"7d: {week:.0f}%")

# Claude アプリゲートウェイの背後：ゲートウェイがドル額を報告する場合はドル額、それ以外はパーセンテージ
spend = rate.get('spend_limit', {})
if spend.get('used_percentage') is not None:
    if spend.get('used_usd') is not None:
        parts.append(f"spend: ${spend['used_usd']} / ${spend['limit_usd']}")
    else:
        parts.append(f"spend: {spend['used_percentage']:.0f}%")

if parts:
    print(f"[{model}] | {' '.join(parts)}")
else:
    print(f"[{model}]")
```

リスト 18: Node.js

```javascript
#!/usr/bin/env node
let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;

    const parts = [];
    const fiveH = data.rate_limits?.five_hour?.used_percentage;
    const week = data.rate_limits?.seven_day?.used_percentage;
```

(次のページに続く)

(前のページからの続き)

```
    if (fiveH != null) parts.push(`5h: ${Math.round(fiveH)}%`);
    if (week != null) parts.push(`7d: ${Math.round(week)}%`);

    // Claude アプリゲートウェイの背後：ゲートウェイがドル額を報告する場合はドル額、それ以外
はパーセンテージ
    const spend = data.rate_limits?.spend_limit;
    if (spend?.used_percentage != null) {
        parts.push(spend.used_usd != null
            ? `spend: $${spend.used_usd} / $${spend.limit_usd}`
            : `spend: ${Math.round(spend.used_percentage)}%`);
    }

    console.log(parts.length ? `[${model}] | ${parts.join(' ')}` : `[${model}]`);
});
```

30.5.7 高コストな操作をキャッシュする

ステータスラインスクリプトはアクティブなセッション中に頻繁に実行されます。`git status` や `git diff` などのコマンドは、特に大規模なリポジトリでは遅い場合があります。この例は git 情報を一時ファイルにキャッシュし、5 秒ごとにのみ更新します。

キャッシュファイル名は、セッション内のステータスラインの呼び出し間で安定している必要がありますが、異なるリポジトリの同時セッションが互いのキャッシュされた git 状態を読み取らないように、セッション間で一意である必要があります。`$$`、`os.getpid()`、`process.pid` のようなプロセスベースの識別子は、呼び出しのたびに変わり、キャッシュを無効にします。代わりに JSON 入力から `session_id` を使用します：これはセッションの有効期間中は安定しており、セッションごとに一意です。

各スクリプトは git コマンドを実行する前に、キャッシュファイルが不在であるか 5 秒より古いかを確認します：

リスト 19: Bash

```
#!/bin/bash
input=$(cat)

MODEL=$(echo "$input" | jq -r '.model.display_name')
DIR=$(echo "$input" | jq -r '.workspace.current_dir')
SESSION_ID=$(echo "$input" | jq -r '.session_id')

CACHE_FILE="/tmp/statusline-git-cache-$SESSION_ID"
CACHE_MAX_AGE=5  # 秒

cache_is_stale() {
    [ ! -f "$CACHE_FILE" ] || \
    # stat -c %Y (Linux) または stat -f %m (macOS) はファイルの最終更新時刻を出力します。
    # Linux フォームは最初に実行する必要があります：Linux では、macOS フォームは stdout に
```

(次のページに続く)

(前のページからの続き)

```
ファイルシステムレポートを出力してから失敗し、その出力はコマンド置換によってキャプチャされ、算
術を破壊します。
    [ $(($(date +%s) - $(stat -c %Y "$CACHE_FILE" 2>/dev/null || stat -f %m "$CACHE_
↪FILE" 2>/dev/null || echo 0))) -gt $CACHE_MAX_AGE ]
}

if cache_is_stale; then
    if git rev-parse --git-dir > /dev/null 2>&1; then
        BRANCH=$(git branch --show-current 2>/dev/null)
        STAGED=$(git diff --cached --numstat 2>/dev/null | wc -l | tr -d ' ')
        MODIFIED=$(git diff --numstat 2>/dev/null | wc -l | tr -d ' ')
        echo "$BRANCH|$STAGED|$MODIFIED" > "$CACHE_FILE"
    else
        echo "||" > "$CACHE_FILE"
    fi
fi

IFS='|' read -r BRANCH STAGED MODIFIED < "$CACHE_FILE"

if [ -n "$BRANCH" ]; then
    echo "[$MODEL] ☒ ${DIR##*/} | ☒ $BRANCH +$STAGED ~$MODIFIED"
else
    echo "[$MODEL] ☒ ${DIR##*/}"
fi
```

リスト 20: Python

```
#!/usr/bin/env python3
import json, sys, subprocess, os, time

data = json.load(sys.stdin)
model = data['model']['display_name']
directory = os.path.basename(data['workspace']['current_dir'])
session_id = data['session_id']

CACHE_FILE = f"/tmp/statusline-git-cache-{session_id}"
CACHE_MAX_AGE = 5  # 秒

def cache_is_stale():
    if not os.path.exists(CACHE_FILE):
        return True
    return time.time() - os.path.getmtime(CACHE_FILE) > CACHE_MAX_AGE

if cache_is_stale():
    try:
```

(次のページに続く)

(前のページからの続き)

```
        subprocess.check_output(['git', 'rev-parse', '--git-dir'], stderr=subprocess.
↪DEVNULL)
        branch = subprocess.check_output(['git', 'branch', '--show-current'],␣
↪text=True).strip()
        staged = subprocess.check_output(['git', 'diff', '--cached', '--numstat'],␣
↪text=True).strip()
        modified = subprocess.check_output(['git', 'diff', '--numstat'], text=True).
↪strip()
        staged_count = len(staged.split('\n')) if staged else 0
        modified_count = len(modified.split('\n')) if modified else 0
        with open(CACHE_FILE, 'w') as f:
            f.write(f"{branch}|{staged_count}|{modified_count}")
    except:
        with open(CACHE_FILE, 'w') as f:
            f.write("||")

with open(CACHE_FILE) as f:
    branch, staged, modified = f.read().strip().split('|')

if branch:
    print(f"[{model}] ☒ {directory} | ☒ {branch} +{staged} ~{modified}")
else:
    print(f"[{model}] ☒ {directory}")
```

リスト 21: Node.js

```
#!/usr/bin/env node
const { execSync } = require('child_process');
const fs = require('fs');
const path = require('path');

let input = '';
process.stdin.on('data', chunk => input += chunk);
process.stdin.on('end', () => {
    const data = JSON.parse(input);
    const model = data.model.display_name;
    const dir = path.basename(data.workspace.current_dir);
    const sessionId = data.session_id;

    const CACHE_FILE = `/tmp/statusline-git-cache-${sessionId}`;
    const CACHE_MAX_AGE = 5; // 秒

    const cacheIsStale = () => {
        if (!fs.existsSync(CACHE_FILE)) return true;
        return (Date.now() / 1000) - fs.statSync(CACHE_FILE).mtimeMs / 1000 > CACHE_
```

(次のページに続く)

(前のページからの続き)

```
↪MAX_AGE;
    };

    if (cacheIsStale()) {
        try {
            execSync('git rev-parse --git-dir', { stdio: 'ignore' });
            const branch = execSync('git branch --show-current', { encoding: 'utf8' }
↪).trim();
            const staged = execSync('git diff --cached --numstat', { encoding: 'utf8'␣
↪}).trim().split('\n').filter(Boolean).length;
            const modified = execSync('git diff --numstat', { encoding: 'utf8' }).
↪trim().split('\n').filter(Boolean).length;
            fs.writeFileSync(CACHE_FILE, `${branch}|${staged}|${modified}`);
        } catch {
            fs.writeFileSync(CACHE_FILE, '||');
        }
    }

    const [branch, staged, modified] = fs.readFileSync(CACHE_FILE, 'utf8').trim().
↪split('|');

    if (branch) {
        console.log(`[${model}] ☒ ${dir} | ☒ ${branch} +${staged} ~${modified}`);
    } else {
        console.log(`[${model}] ☒ ${dir}`);
    }
});
```

30.5.8 Windows 設定

Windows では、Claude Code はステータスラインコマンドを Git Bash 経由で実行します。Git Bash がインストールされている場合、または Git Bash がない場合は PowerShell を通じて実行します。

Git Bash は引用符なしのバックスラッシュをエスケープ文字として扱うため、`C:\Users\username\script.mjs` のような Windows スタイルのパスはセパレーターが削除された状態でスクリプトランナーに到達し、目に見えるエラーなしでコマンドが失敗します。command 文字列のファイルパスを以下の例に示すようにフォワードスラッシュで記述します。~ 短縮形も機能し、Windows ホームディレクトリに展開されます。

PowerShell スクリプトをステータスラインとして実行するには、`powershell` 経由で呼び出します。これは Claude Code がコマンドを Git Bash または PowerShell を通じてルーティングするかどうかに関わらず機能します：

リスト 22: settings.json

```
{
  "statusLine": {
```

(次のページに続く)

(前のページからの続き)

```
    "type": "command",
    "command": "powershell -NoProfile -File C:/Users/username/.claude/statusline.ps1"
  }
}
```

リスト 23: statusline.ps1

```
$input_json = $input | Out-String | ConvertFrom-Json
$cwd = $input_json.cwd
$model = $input_json.model.display_name
$used = $input_json.context_window.used_percentage
$dirname = Split-Path $cwd -Leaf

if ($used) {
    Write-Host "$dirname [$model] ctx: $used%"
} else {
    Write-Host "$dirname [$model]"
}
```

または、Git Bash がインストールされている場合は、Bash スクリプトを直接実行します：

リスト 24: settings.json

```
{
  "statusLine": {
    "type": "command",
    "command": "~/.claude/statusline.sh"
  }
}
```

リスト 25: statusline.sh

```
#!/usr/bin/env bash
input=$(cat)
cwd=$(echo "$input" | grep -o '"cwd":"[^"]*"' | cut -d'"' -f4)
model=$(echo "$input" | grep -o '"display_name":"[^"]*"' | cut -d'"' -f4)
dirname="${cwd##*[/\\]}"
echo "$dirname [$model]"
```

30.6 サブエージェントステータスライン

subagentStatusLine 設定は、エージェントパネルに表示される各 **サブエージェント** (ページ 317) のカスタム行本体をレンダリングします。デフォルトの name ・ description ・ token count 行を独自のフォーマットに置き換えるために使用します。

```
{
  "subagentStatusLine": {
    "type": "command",
    "command": "~/.claude/subagent-statusline.sh"
  }
}
```

コマンドは、すべての表示されているサブエージェント行が stdin で単一の JSON オブジェクトとして渡される各リフレッシュティックで実行されます。入力には **基本フックフィールド** (ページ 466)、使用可能な行幅を示す `columns` フィールド、および `tasks` 配列が含まれます。各タスクには `id`、`name`、`type`、`status`、`description`、`label`、`startTime`、`model`、`effort`、`contextWindowSize`、`tokenCount`、`tokenSamples`、`cwd` があります。

タスクごとの `model` フィールドは、タスクが実行される解決済みモデル ID です。`contextWindowSize` はそのモデルのコンテキストウィンドウ（トークン単位）で、メインステータスラインの `context_window.context_window_size` と同じ方法で計算されるため、`tokenCount` から行ごとのパーセンテージをレンダリングできます。両方のフィールドには Claude Code v2.1.205 以降が必要で、モデルがまだ解決されていないタスクでは省略されます。

タスクごとの `effort` フィールドは、そのサブエージェントに設定された推論努力で、その **定義フロントマター** (ページ 326) または個別の呼び出しで設定されます。値は、努力レベル文字列 `low`、`medium`、`high`、`xhigh`、`max` のいずれか、またはトークン予算の数値です。フィールドは、記述されたとおりに設定された値を報告します。モデルがそのレベルをサポートしていない場合、Claude Code が実際に適用する努力は異なる可能性があります。フィールドには Claude Code v2.1.214 以降が必要で、サブエージェントがセッションの努力レベルを継承する場合は存在しません。

オーバーライドしたい各行に対して stdout に 1 つの JSON 行を書き込みます。形式は `{"id": "<task id>", "content": "<row body>"}` です。`content` 文字列はそのままレンダリングされます。ANSI 色と OSC 8 ハイパーリンクを含みます。タスクの `id` を省略して、その行のデフォルトレンダリングを保持します。空の `content` 文字列を出力して、その行を非表示にします。

`statusLine` に適用される同じトラストと `disableAllHooks` および `allowManagedHooksOnly` ゲートが `subagentStatusLine` に適用されます。プラグインは、`settings.json` でデフォルトの `subagentStatusLine` を配布できます。ただし、フックとは異なり、プラグインが管理設定で強制的に有効化されている場合でも、プラグイン値は `allowManagedHooksOnly` の下で実行されません。

30.7 ヒント

- **モック入力でテストする**：`echo '{"model":{"display_name":"Opus"},"workspace":{"current_dir":"/home/user/project"},"context_window":{"used_percentage":25},"session_id":"test-session-abc"}' | ./statusline.sh`
- **出力を短く保つ**：ステータスバーの幅は限られているため、長い出力は切り詰められたり、不適切にラップされたりする可能性があります
- **遅い操作をキャッシュする**：スクリプトはアクティブなセッション中に頻繁に実行されるため、`git status` などのコマンドは遅延を引き起こす可能性があります。これを処理する方法については、**キャッシング例** (ページ 811) を参照してください。

ccstatusline や starship-claude などのコミュニティプロジェクトは、テーマと追加機能を備えた事前構築設定を提供します。

30.8 トラブルシューティング

ステータスラインが表示されない

- スクリプトが実行可能であることを確認します：`chmod +x ~/.claude/statusline.sh`
- スクリプトが stdout に出力し、stderr に出力していないことを確認します
- スクリプトを手動で実行して、出力を生成することを確認します
- Windows で Git Bash がインストールされている場合、`command` パスのバックスラッシュはスクリプトが実行される前にエスケープ文字として消費される可能性があります。パスでは前方スラッシュを使用してください。*Windows* **設定** (ページ 814) を参照してください。
- **設定の優先順位** (ページ 466) が適用された後、`disableAllHooks` が管理設定外で `true` の場合、Claude Code は管理設定からの `statusLine` のみを実行し、管理 `statusLine` がない場合はステータスラインが無効になります。この設定を削除するか、それを設定するファイルで `false` に設定して、再度有効にします。`disableAllHooks` を参照してください。
- 組織が管理設定で `allowManagedHooksOnly` を設定している場合、カスタムステータスラインは警告なく消えます：これらの管理設定の `statusLine` 値からのみステータスラインを取得できます。allowManagedHooksOnly で実行される内容を参照して完全な動作を確認し、この設定があなたに適用されるかどうかを管理者に確認してください。
- `claude --debug` を実行して、セッションの最初のステータスラインの呼び出しからの終了コードと stderr をログに記録します
- Claude にスクリプトファイルを読み取り、`statusLine` コマンドを直接実行するよう依頼して、エラーを表示します

ステータスラインが -- または空の値を表示する

- フィールドは最初の API レスポンスが完了する前は `null` の可能性があります
- スクリプトで `// 0` のようなフォールバックを使用して null 値を処理します
- 複数のメッセージの後も値が空のままの場合は、Claude Code を再起動します

コンテキスト割合が予期しない値を表示する

- 最も単純で正確なコンテキスト状態には `used_percentage` を使用します
- ステータスラインは最後の API レスポンスからのカウントを報告し、`/context` はそのレスポンス以降に追加されたメッセージの推定値を追加するため、次のレスポンスまで `/context` はより高く読むことができます

OSC 8 リンクがクリック可能でない

- ターミナルが OSC 8 ハイパーリンクをサポートしていることを確認します（iTerm2、Kitty、WezTerm）
- Terminal.app はクリック可能なリンクをサポートしていません

- リンクテキストが表示されているがクリック可能でない場合、Claude Code がターミナルのハイパーリンクサポートを検出できていない可能性があります。Claude Code を起動する前に FORCE_HYPERLINK 環境変数を設定して、検出をオーバーライドします：

```
FORCE_HYPERLINK=1 claude
```

  PowerShell では、最初に現在のセッションで変数を設定します：

```
$env:FORCE_HYPERLINK = "1"; claude
```

- SSH と tmux セッションは設定に応じて OSC シーケンスをストリップする可能性があります
- エスケープシーケンスが \e]8;; のようなリテラルテキストとして表示される場合は、echo -e の代わりに printf '%b' を使用して、より確実なエスケープ処理を行います

エスケープシーケンスでの表示の不具合

- 複雑なエスケープシーケンス（ANSI 色、OSC 8 リンク）は、他の UI 更新と重複する場合、時々破損した出力を引き起こす可能性があります
- 破損したテキストが表示される場合は、スクリプトをプレーンテキスト出力に簡略化してみてください
- エスケープコード付きの複数行ステータスラインは、プレーンテキストの単一行よりもレンダリングの問題が発生しやすくなります

ワークスペーストラストが必要

- statusLine はシェルコマンドを実行するため、Claude Code は**設定ファイルのフックと同じワークスペーストラストルール** (ページ 1066) の下で実行します。フォルダのダイアログを受け入れるか、その信頼がそれに拡張される親ディレクトリのダイアログを受け入れるだけで十分です。
- それまでの間、ステータスラインは空白のままで、claude --debug は Status line command skipped: workspace trust not accepted をログに記録します。Claude Code を再起動し、トラストダイアログを受け入れて有効にします。

スクリプトエラーまたはハング

- ゼロ以外のコードで終了するか、出力を生成しないスクリプトは、ステータスラインを空白にします
- 遅いスクリプトは、完了するまでステータスラインの更新をブロックします。古い出力を避けるために、スクリプトを高速に保ちます。
- 遅いスクリプトの実行中に新しい更新がトリガーされた場合、実行中のスクリプトはキャンセルされます
- 設定する前に、モック入力を使用してスクリプトを独立してテストします

通知がステータスラインの行を共有する

フルスクリーンレンダリングの外では、Claude Code は通知をステータスラインと同じ行に表示します。フルスクリーンレンダリングでは、Claude Code は通知に独自の行を提供します。

- MCP サーバーエラーおよび自動更新などのシステム通知は、ステータスラインと同じ行の右側に表示されます。コンテキスト低警告などの一時的な通知もこの領域を循環します。

- 詳細モードを有効にすると、この領域にトークンカウンターが追加されます
- 狭いターミナルでは、これらの通知がステータスラインの出力を切り詰める可能性があります

第31章　出力スタイル

Concise や Explanatory などの組み込み出力スタイルを使用するか、カスタムスタイルを作成して、Claude Code のロール、トーン、応答形式を変更します。

出力スタイルは、セッション内のすべての応答に対して Claude のロール、トーン、応答形式を設定する一連の指示です。Claude Code には、デフォルトの他に 4 つの組み込みスタイルが含まれており、独自のスタイルを作成することもできます。

出力スタイルを使用すると、セッション全体で Claude がどのように応答し、あなたと協力するかを変更できるため、各プロンプトでリクエストを繰り返す必要がありません。たとえば、組み込みスタイルは応答をより短くしたり、各変更の説明を追加したり、Claude が定型的な質問をせずに作業を開始したりできます。カスタムスタイルは、Claude をソフトウェアエンジニア以外のもの（ライティングアシスタントやデータアナリストなど）に変えることもできます。

- 組み込みスタイルを使用するには、**組み込み出力スタイル** (ページ 821) から 1 つを選択して、**それに切り替えます** (ページ 824)。
- 独自の指示を作成するには、**カスタム出力スタイルを作成します** (ページ 824)。

注釈:

出力スタイルは Claude に従うべき指示を与えます。何かが常に起こることや決して起こらないことを保証するものではありません。一部のニーズは異なる機能に適しています。

- Claude がプロジェクトについて知っておくべきことについては、*CLAUDE.md* (ページ 997) を使用します。
- 各編集後のフォーマットやコマンドのブロックなど、毎回必ず起こる必要があることについては、**フック** (ページ 411) を使用します。
- スキル、サブエージェント、その他のオプションについては、**出力スタイルと他の機能の選択** (ページ 826) を参照してください。

31.1 組み込み出力スタイル

Claude Code は *Default* (ページ 822) スタイルで開始します。これはソフトウェアエンジニアリングタスクを完了するための標準的な指示です。他の 4 つの組み込みスタイルはこれらの指示を保持し、独自の指示を追加します。

このテーブルは各スタイルがセッションについて何を変更するか、そしていつそれが適切かを示しています。

スタイル	変更内容	使用する場合
Proactive (ページ 822)	Claude は即座に作業を開始し、日常的な決定について質問する代わりに合理的な仮定を立てます	Claude に日常的な決定を通じて作業を続けさせたい場合、仮定が間違っていれば方向を修正します
Concise (ページ 822)	レスポンスは結果で始まり、前置き、説明、および要約を省略します	デフォルトのレスポンスが想定より長い場合
Explanatory (ページ 823)	Claude は、記述したコードの背後にある選択肢を説明する短い `Insight` ブロックを追加します	コードベースを学習中、または変更と一緒に推論を望む場合
Learning (ページ 823)	Claude は選択肢を説明し、自分で記述するための小さなコード片を残します	タスクがまだ完了している間、実践的なコーディング練習を望む場合

31.1.1 Default

Default は出力スタイルが選択されていないことを意味します。Claude Code はスタイル指示を追加せず、Claude は Claude Code の標準システムプロンプトから動作します。これはソフトウェアエンジニアリングタスク用に記述されています。

`default` は他のスタイルと共に `/output-style` リストに表示されるため、**同じ方法で選択** (ページ 824) します。

31.1.2 Proactive

Proactive スタイルでは、Claude はタスクを送信するとすぐに実装を開始します。日常的な決定について合理的な仮定を立て、質問するために停止することはなく、計画を求めない限り計画モードに切り替わりません。任意の時点でそれをリダイレクトできます。

スタイルの指示は、データを削除するか、共有またはプロダクションシステムを変更するアクションの前に、Claude がカンバセーション内でユーザーに確認するよう指示します。その確認は Claude が従う指示であり、権限プロンプトとは別です。

Proactive スタイルに切り替えても、権限モードは変わりません。権限モードは依然としてユーザーに尋ねずに実行するツール呼び出しを決定するため、権限プロンプトは切り替え前と同じ方法で表示されます。

31.1.3 Concise

Concise スタイルでは、レスポンスの最初の文が何が起こったか、または答えが何かを述べています。Claude は導入部、ステップバイステップの説明、および終了時の要約を省略し、簡単な質問に 1 〜 3 文で答えます。Default スタイルと同じくらい徹底的にエンジニアリング作業を行います。Claude Code v2.1.237 以降が必要です。

Claude は以下の場合でも完全な長さで記述します。

- **ユーザーが求めるもの**: 説明または詳細情報を求めると、Claude は完全に答えます。
- **安全に行動するために必要なもの**: エラーレポート、失敗したテスト出力、セキュリティ警告、および破壊的なアクションの確認は完全な内容を保持します。

31.1.4 Explanatory

Explanatory スタイルでは、Claude は Default スタイルと同じ方法でタスクを実行し、立てた選択肢の理由についての短い説明を追加します。各説明はカンバセーションに表示され、それが関連するコードの前または後に、`Insight` というラベルの付いたブロックに表示されます。説明はファイルにコメントとして記述されません。

`Insight` ブロックは、コードベースまたは Claude が記述したコードについて 2 〜 3 つのポイントを含みます。例えば、API エンドポイントを追加した後のこのようなものです。

```
★ Insight ─────────────────────────────────────
- このリポジトリのすべてのルートは withAuth ラッパーを通過するため、新しいエンドポイントは独自のミドルウェアなしでセッションチェックを取得します。
- レート制限は limits.ts でルートごとに設定されるため、この変更はグローバルデフォルトではなく、そこにエントリを追加します。
─────────────────────────────────────────────────
```

31.1.5 Learning

Learning スタイルでは、Claude は *Explanatory* **スタイル** (ページ 823) と同じ `Insight` ブロックを追加し、ユーザーにコードの一部を記述するよう求めます。Claude は日常的な実装を自分で処理します。エラーハンドリング、データ構造、または複数の有効なアプローチを持つビジネスロジックなど、実際の設計決定を伴う部分に到達すると、ユーザーのために数行を残します。

Claude は `TODO(human)` コメントでファイル内の場所をマークし、既に構築されたもの、記述するもの、および検討すべきことを述べるリクエストを送信します。

```
● Learn by Doing

Context: アップロードフォームが配置され、ファイルを受け入れる前に validateFile() を呼び出します。サイズとタイプのチェックは画像に対して機能しますが、switch ステートメントにはドキュメントの処理がまだありません。

Your Task: upload.js で、validateFile() 内の case "document" ブランチを実装します。TODO(human) を探します。

Guidance: ドキュメントのサイズ制限を決定し、ファイル拡張子が MIME タイプと一致する必要があるかどうかを決定します。{valid: boolean, error?: string} を返します。
```

Claude はその後停止し、待機します。`TODO(human)` コメントでコードを記述し、完了したことを Claude に伝えます。Claude はユーザーのコードについて 1 つの `Insight` で応答し、タスクを続行します。

31.2 出力スタイルを変更する

コマンド、メニュー、または設定ファイルでスタイルを選択します。コマンドと両方のメニューは、選択内容を **ローカルプロジェクトレベル** (ページ 711) の `.claude/settings.local.json` に保存します。

- `/output-style` **コマンド**: `/output-style <style>` を実行してスタイルを切り替えます。例えば `/output-style concise` のように使用します。引数なしで実行すると、選択可能なスタイルが一覧表示され、現在のスタイルがマークされます。

  このコマンドは **非対話型モード** (ページ 695) と Agent SDK セッション、およびモバイルアプリまたは Web から リモートコントロール 経由でも機能します。この場合、**組み込みスタイル** (ページ 821) のみをリストして選択できます。Claude Code v2.1.269 以降が必要です。

- **Terminal メニュー**: `/config` を実行し、**Output style** を選択してメニューからスタイルを選択します。
- **VS Code extension**: **コマンドメニュー** (ページ 177) を `/` で開き、**Output styles** を選択してスタイルを選択します。カスタムスタイルも含まれます。Claude Code v2.1.257 以降が必要です。
- **Desktop app**: 設定ファイル（例えば `.claude/settings.local.json`、ターミナルメニューが書き込むファイル）の `outputStyle` フィールドを設定します。`/config` を実行すると、Claude Code は *Settings > Claude Code* (ページ 169) を開きます。メニューではなく設定画面が開きます。

メニューなしでスタイルを設定するには、設定ファイルの `outputStyle` フィールドを直接編集します。

```
{
  "outputStyle": "Explanatory"
}
```

値は大文字と小文字を区別するため、組み込み名は `Proactive`、`Concise`、`Explanatory`、`Learning` と記述してください。`explanatory` のようにスタイル名と完全に一致しない値を指定すると、デフォルトスタイルが適用されます。`/output-style` コマンドは大文字と小文字を区別しません。

プロジェクト全体でスタイルをデフォルトにするには、`~/.claude/settings.json` で `outputStyle` を設定します。プロジェクト独自の設定ファイルは **その値より優先されます** (ページ 719)。

セッション中にスタイルを切り替えると、Claude は次のメッセージから新しいスタイルを使用します。最初のメッセージのプロンプトキャッシングのコストについては、出力スタイルの変更 を参照してください。v2.1.251 より前は、新しいスタイルは `/clear` を実行するか新しいセッションを開始した後にのみ適用されました。

31.3 カスタム出力スタイルを作成する

カスタム出力スタイルは Markdown ファイルです。メタデータ用の frontmatter、その後に Claude の指示が続きます。

VS Code 拡張機能では、手書きではなく *Output styles* **メニュー** (ページ 177) からファイルを作成することもできます。これには Claude Code v2.1.261 以降が必要です。

1. **Markdown ファイルを作成する**

   3 つのレベルのいずれかに保存します。ファイル名がスタイル名になります。frontmatter で `name` を設定しない限り。

- ユーザー: `~/.claude/output-styles`
- プロジェクト: `.claude/output-styles`
- 管理ポリシー: 管理設定ディレクトリ 内の `.claude/output-styles`

プロジェクト出力スタイルは、作業ディレクトリとリポジトリルートの間のすべての `.claude/output-styles/` から読み込まれます。これらのネストされたディレクトリの複数が同じ名前のスタイルを定義する場合、Claude Code は作業ディレクトリに最も近いものを使用します。

2. **Frontmatter と指示を追加する**

   Claude Code のソフトウェアエンジニアリング指示を保持するかどうかを決定します。Claude がコミュニケーション方法を変更しているがまだ同じ方法でコーディングしたい場合は `keep-coding-instructions: true` を設定します。Claude がソフトウェアエンジニアリングを行わない場合は除外します。

   この例は Claude のコーディング動作を保持しながら、すべての説明を図で始めます。

```
---
name: Diagrams first
description: Lead every explanation with a diagram
keep-coding-instructions: true
---

When explaining code, architecture, or data flow, start with a Mermaid diagram␣
↪showing the structure, then explain in prose.

## Diagram conventions

Use `flowchart TD` for control flow and `sequenceDiagram` for request paths.␣
↪Keep diagrams under 15 nodes.
```

3. **スタイルに切り替える**

   ターミナルで `/output-style <style>` を実行するか、`/config` を実行して **Output style** でスタイルを選択します。Claude は次のメッセージから新しいスタイルを使用します。ターミナルでは、Claude Code はスタイルファイルを起動時に読み込むため、実行中のセッション中に作成または編集した場合は、Claude Code を再起動して変更を反映させてください。

プラグイン は `output-styles/` ディレクトリで出力スタイルを配布することもできます。

31.3.1 Frontmatter リファレンス

出力スタイルを YAML *frontmatter* で設定します。ファイルの上部の `---` マーカーの間に記述します。すべてのフィールドはオプションであり、フィールド名はハイフンで区切られた小文字の単語を使用します。スペルが間違ったフィールドはエラーなしで無視されます。YAML が解析できない場合、スタイルはファイル名で読み込まれ、フィールドは設定されません。`claude --debug` を実行して解析エラーを確認してください。

フィールド	必須	説明
`name`	いいえ	出力スタイルの名前。`/config` ピッカーに表示されます。デフォルト: ファイル名
`description`	いいえ	出力スタイルの説明。`/config` ピッカーに表示されます
`keep-coding-instructions`	いいえ	`true` に設定すると、Claude Code の組み込みソフトウェアエンジニアリング指示をスタイルと一緒に保持します。デフォルト: `false`
`force-for-plugin`	いいえ	プラグイン出力スタイルのみ。`true` に設定すると、プラグインが有効になるたびに、ユーザーが選択する必要なく、このスタイルを自動的に適用します。ユーザーの `outputStyle` 設定をオーバーライドします。複数の有効なプラグインがこれを設定する場合、Claude Code は最初に読み込まれたものを使用します。デフォルト: `false`

31.4 出力スタイルと他の機能の選択

出力スタイルはセッション内のすべての応答に適用されます。これは Claude が従う指示であるため、何も強制しません。必要なものがすべての応答より狭い範囲である場合、または確実に発生する必要がある場合は、別の機能がより適切です。

この表は、必要なものを、それを実現する機能に対応させています。

必要なもの	使用する機能	適切な理由
特定の声、長さ、または形式のすべての応答、または異なるロールの Claude	出力スタイル	セッション全体に適用され、1 つのコマンドでスタイルを切り替えることができます
Claude がプロジェクトの規約、コマンド、および構造を知っている	*CLAUDE.md* (ページ 997)	Claude が知るべきコードベースについての情報を保持し、選択したスタイルに関係なく読み込まれたままになります
リリースチェックリストやレビュー手順など、1 種類のタスクの指示	*skill* (ページ 275)	Claude はそれを呼び出すか、タスクが一致する場合にのみ読み込むため、関連のない応答を形成しません
各編集後のフォーマットやコマンドのブロックなど、例外なく毎回発生する必要があるもの	*hook* (ページ 411)	Claude Code はライフサイクルイベントでフック自体を実行するため、Claude が指示に従うことに依存しません
焦点を絞ったタスク用に独自の指示、モデル、およびツールを備えたヘルパー	*subagent* (ページ 317)	独自のシステムプロンプトを備えた別のコンテキストで実行され、会話に概要を返します
Claude Code を開始するときに渡す Claude の指示への追加	*`--append-system-prompt`* (ページ 72)	何も削除せずにシステムプロンプトに追加します

これらの機能は組み合わせることができます。たとえば、Claude が知るべきことに CLAUDE.md を使用し、応答方法に出力スタイルを使用し、保証する必要があるものに hook を使用できます。Claude Code を拡張すると、残りの拡張機能が比較されます。

31.5 出力スタイルの仕組み

出力スタイルは Claude Code が Claude に与える指示を変更します。

- Claude Code はすべてのリクエストで有効なスタイルの指示を送信します。
- カスタム出力スタイルは、`keep-coding-instructions` が `true` に設定されていない限り、変更のスコープ、コメントの書き方、作業の検証方法など、Claude Code の組み込みソフトウェアエンジニアリング指示を除外します。

出力スタイルはメイン会話と **フォーク** (ページ 354) に適用されます。フォークは親の完全な会話とシステムプロンプトを継承します。その他の **サブエージェントは独自のシステムプロンプトを実行** (ページ 350) するため、スタイルはそれらの応答方法を変更しません。

トークン使用量はスタイルによって異なります。スタイルの指示は入力トークンを追加しますが、プロンプトキャッシングはセッション内の最初のリクエスト後にこのコストを削減します。

組み込みの Explanatory および Learning スタイルは、設計上 Default よりも長い応答を生成し、これは出力トークンを増加させます。Concise スタイルはその逆で、Claude にデフォルトで応答を短く保つよう指示することで反対の効果を生じます。カスタムスタイルの場合、出力トークン使用量は、指示が Claude に何を生成するよう指示するかによって異なります。

31.6 関連リソース

- **設定** (ページ 711): `outputStyle` フィールドが存在する場所と設定の優先順位の仕組み
- 権限モード: Proactive スタイルがオートモードとどのように比較されるか
- プラグイン: スキル、フック、エージェントと一緒に出力スタイルをパッケージ化して配布する
- 設定をデバッグする: 出力スタイルが有効にならない理由を診断する

第32章　Claude Code 用にターミナルを設定する

Shift+Enter で改行を挿入する、Claude の処理完了時にターミナルベルを鳴らす、tmux を設定する、カラーテーマを合わせる、Claude Code CLI で Vim モードを有効にする方法を説明します。

Claude Code はターミナル設定なしで任意のターミナルで動作します。このページは、特定の動作が期待通りに機能していない場合のためのものです。以下から症状を見つけてください。すべてが既に正常に動作している場合は、このページは不要です。

- *Shift+Enter* **が改行を挿入する代わりに送信する** (ページ 829)
- *macOS* **でオプションキーのショートカットが機能しない** (ページ 830)
- *Claude* **の処理完了時に音またはアラートが出ない** (ページ 831)
- *Claude Code* **を** *tmux* **内で実行している** (ページ 832)
- *Windows* **でバックスペースが単語全体を削除する** (ページ 832)
- **表示がちらつくまたはスクロールバックがジャンプする** (ページ 837)
- **プロンプトで** *Vim* **キーを使いたい** (ページ 839)

このページは、ターミナルが Claude Code に正しい信号を送信するようにするためのものです。Claude Code 自体が応答するキーを変更するには、代わりに **キーバインディング** (ページ 765) を参照してください。

32.1 複数行のプロンプトを入力する

Enter キーを押すとメッセージが送信されます。送信せずに改行を追加するには、Ctrl+J を押すか、\ を入力してから Enter を押してください。どちらの方法もセットアップなしですべてのターミナルで機能します。

ほとんどのターミナルでは Shift+Enter を押すこともできますが、サポートはターミナルエミュレータによって異なります。

ターミナル	改行用の Shift+Enter
Ghostty、Kitty、iTerm2、WezTerm、Warp、Apple Terminal、Windows Terminal	セットアップなしで機能
kitty キーボードプロトコルをサポートする他のターミナル（foot や Alacritty 0.16 以降など）	セットアップなしで機能。Claude Code v2.1.269 以降が必要です
VS Code、Cursor、Devin Desktop、Alacritty 0.16 より前のバージョン、Zed	1 回 `/terminal-setup` を実行
gnome-terminal、PyCharm や Android Studio などの JetBrains IDE	利用不可。Ctrl+J または \ の後に Enter を使用

VS Code、Cursor、Devin Desktop、Alacritty 0.16 より前のバージョン、Zed の場合、`/terminal-setup` はターミナルの設定ファイルに Shift+Enter キーバインディングを書き込みます。初回実行時には `Installed VSCode terminal Shift+Enter key binding` などの確認メッセージが表示されます。既存のバインディングはそのまま保持されます。`VSCode terminal Shift+Enter key binding already configured` などのメッセージが表示される場合は、変更は加えられていません。`/terminal-setup` は tmux または screen の内部ではなく、ホストターミナル内で直接実行してください。ホストターミナルの設定に書き込む必要があるためです。

VS Code、Cursor、Devin Desktop では、`/terminal-setup` は 2 つのエディタ設定も更新します。統合ターミナルでのテキストの乱れを防ぐために `terminal.integrated.gpuAcceleration` を `"off"` に設定し、フルスクリーンモードでのスムーズなスクロールのために `terminal.integrated.mouseWheelScrollSensitivity` を設定します。GPU アクセラレーション変更を元に戻すには、`"auto"` に設定し直してエディタウィンドウをリロードしてください。

Zed では、`/terminal-setup` は `keymap.json` をその場で更新します。

- キーマップに既にバインディングがあり、そのいずれもターミナルの `shift-enter` ではない場合、Claude Code はまずそれを同じディレクトリ内のコピー（`keymap.json.1a2b3c4d.bak` など）にバックアップしてから、Shift+Enter バインディングをキーマップにマージし、他のキーバインディングとコメントを保持します
- Claude Code がキーマップを読み込めない、解析できない、バックアップできない、またはマージされた結果を検証できない場合、ファイルを変更せず、追加するキーバインディングブロックを出力します

tmux 内で実行している場合、外側のターミナルがサポートしている場合でも、Shift+Enter には以下の *tmux* **設定** (ページ 832) が必要です。

改行を別のキーにバインドするか、Enter が改行を挿入し Shift+Enter が送信するように動作を入れ替えるには、**キーバインディングファイル** (ページ 765) で `chat:newline` および `chat:submit` アクションをマップしてください。

32.2 macOS で Option キーショートカットを有効にする

Claude Code の一部のショートカットは Option キーを使用します。例えば、Option+Enter で改行したり、Option+P でモデルを切り替えたりします。macOS では、ほとんどのターミナルはデフォルトで Option を修飾キーとして送信しないため、これを有効にするまでこれらのショートカットは機能しません。ターミナル設定は通常「Use Option as Meta Key」というラベルが付いています。Meta は、現在 Option または Alt というラベルが付いているキーの歴史的な Unix 名です。

Apple Terminal

設定 → プロファイル → キーボードを開き、「Use Option as Meta Key」をチェックします。

Claude Code の初回実行時のターミナルセットアッププロンプトを受け入れた場合、これはすでに完了しています。そのプロンプトは `/terminal-setup` を実行し、Option を Meta として有効にし、Apple Terminal プロファイルで可聴ベルをオフにします。

スクリーンリーダーモードでは、`/terminal-setup` はベル設定を変更しないままにするため、ターミナルベルは可聴のままです。v2.1.211 より前では、`/terminal-setup` はスクリーンリーダーモードでもベルをオフにしていました。以前の実行でベルがオフになった場合は、設定 → プロファイル → 詳細 → 「可聴ベル」でオンに戻してください。

iTerm2
: 設定 → プロファイル → キー → 一般を開き、左 Option キーと右 Option キーを「Esc+」に設定します。

  iTerm2 で `/terminal-setup` を実行すると、設定 → 一般 → 選択の下で「Applications in terminal may access clipboard」が有効になり、`/copy` コマンドがシステムクリップボードに書き込むことができます。このコマンドは tmux 内から実行された場合でも iTerm2 を検出します。変更を有効にするために iTerm2 を再起動してください。

VS Code
: VS Code 設定に `"terminal.integrated.macOptionIsMeta": true` を追加します。

Ghostty、Kitty、およびその他のターミナルについては、ターミナルの設定ファイルで Option-as-Alt または Option-as-Meta 設定を探してください。

32.3 ターミナルベルまたは通知を取得する

Claude がタスクを完了するか権限プロンプトで一時停止し、ターミナルから離れているように見える場合、通知イベントが発火します。**各通知タイプが発火するタイミング** (ページ 518) を参照して、正確なタイミングを確認してください。これをターミナルベルまたはデスクトップ通知として表示することで、長いタスクが実行されている間に他の作業に切り替えることができます。

デフォルトでは Claude Code は Ghostty、Kitty、および iTerm2 でのみデスクトップ通知を送信します。他のターミナルでは、`preferredNotifChannel` を`"terminal_bell"`に設定してターミナルベルを鳴らすか、カスタム音またはコマンド用に**通知フック** (ページ 832) を設定してください。次の設定エントリはターミナルベルをオンにします：

リスト 1: ~/.claude/settings.json

```
{
  "preferredNotifChannel": "terminal_bell"
}
```

デスクトップ通知は SSH 経由でローカルマシンに到達するため、リモートセッションでもアラートを受け取ることができます。Ghostty と Kitty はさらなるセットアップなしで OS 通知センターに転送します。iTerm2 では転送を有効にする必要があります：

1. **Open iTerm2 notification settings**

   Go to Settings → Profiles → Terminal.

2. **Enable alerts**

   Check "Notification Center Alerts", then click "Filter Alerts" and enable "Send escape sequence-generated alerts".

通知がまだ表示されない場合は、ターミナルアプリケーションが OS 設定で通知権限を持っていることを確認し、tmux 内で実行している場合は**パススルーを有効にしてください** (ページ 832)。

32.3.1 通知フックで音を再生する

任意のターミナルで**通知フック** (ページ 413) を設定して、Claude があなたの注意が必要な場合に音を再生するか、カスタムコマンドを実行できます。フックは組み込み通知と並行して実行され、それを置き換えるのではなく、Warp や VS Code 統合ターミナルなどのデスクトップ通知を受け取らないターミナルは、フックを使用するか、代わりに `preferredNotifChannel` を`"terminal_bell"`に設定できます。

以下の例は macOS でシステム音を再生します。リンクされたガイドには、macOS、Linux、および Windows のデスクトップ通知コマンドがあります。

リスト 2: ~/.claude/settings.json

```
{
  "hooks": {
    "Notification": [
      {
        "hooks": [{ "type": "command", "command": "afplay /System/Library/Sounds/
↪Glass.aiff" }]
      }
    ]
  }
}
```

32.4 tmux を設定する

Claude Code が tmux 内で実行される場合、デフォルトでは Shift+Enter が改行を挿入する代わりに送信してしまい、デスクトップ通知とプログレスバーが外側のターミナルに到達しません。これらの行を `~/.tmux.conf` に追加してから、`tmux source-file ~/.tmux.conf` を実行して、実行中のサーバーに適用してください。

リスト 3: ~/.tmux.conf

```
set -g allow-passthrough on
set -s extended-keys on
set -as terminal-features 'xterm*:extkeys'
```

`allow-passthrough` 行により、通知とプログレス更新が tmux に飲み込まれるのではなく、外側のターミナルに到達できるようになります。`extended-keys` 行により、tmux が Shift+Enter と通常の Enter を区別できるようになり、改行ショートカットが機能するようになります。

32.5 Windows で Backspace がワード全体を削除する問題を修正する

Windows では、Claude Code は Backspace として到着した `^H` を Ctrl+Backspace として読み込み、**前のワードを削除します** (ページ 80)。ただし、`TERM_PROGRAM` が `mintty` または `TERM` が `cygwin` の場合は除きます。macOS と Linux では、Claude Code はこれをプレーン Backspace として読み込みます。

Backspace を押すたびにワード全体が削除される場合、ターミナルはプレーン Backspace に対して `^H` を送信しています。`CLAUDE_CODE_BS_AS_CTRL_BACKSPACE=0` を設定してください。その後、Backspace と

Ctrl+H は各々 1 文字を削除します。macOS または Linux で Ctrl+Backspace が 1 文字だけを削除する場合、ターミナルが ^H を送信しているため、代わりに変数を 1 に設定してください。

32.6 カラーテーマを合わせる

/theme コマンドを使用するか、/config のテーマピッカーを使用して、ターミナルに合わせた Claude Code テーマを選択します。auto オプションを選択すると、ターミナルの明るい背景または暗い背景が検出されるため、ターミナルが OS の外観の変更に従うたびにテーマが従います。Claude Code はターミナルアプリケーションによって設定されるターミナル自体のカラースキームを制御しません。

インターフェースの下部に表示される内容をカスタマイズするには、現在のモデル、作業ディレクトリ、git ブランチ、またはその他のコンテキストを表示する**カスタムステータスライン** (ページ 785) を設定します。

32.6.1 カスタムテーマを作成する

組み込みプリセットに加えて、/theme には定義したカスタムテーマと、インストール済みのプラグインによって提供されたテーマが一覧表示されます。リストの最後にある**新しいカスタムテーマ…**を選択して、対話的に作成します。テーマに名前を付けてから、個別のカラートークンを選択してオーバーライドします。カスタムテーマがハイライトされている間に Ctrl+E を押して編集します。

各カスタムテーマは ~/.claude/themes/ 内の JSON ファイルです。.json 拡張子を除いたファイル名がテーマのスラッグであり、テーマを選択すると custom:<slug> がテーマの設定として保存されます。ファイルには 3 つのオプションフィールドがあります。

フィールド	タイプ	説明
name	string	/theme に表示されるラベル。デフォルトはファイル名スラッグ
base	string	テーマの開始元となる組み込みプリセット：dark、light、dark-daltonized、light-daltonized、dark-ansi、または light-ansi。デフォルトは dark
overrides	object	カラートークン名をカラー値にマップします。ここにリストされていないトークンはベースプリセットにフォールスルーします

カラー値は #rrggbb、#rgb、rgb(r,g,b)、ansi256(n)、または ansi:<name> を受け入れます。ここで <name> は red や cyanBright などの 16 個の標準 ANSI カラー名の 1 つです。不明なトークンと無効なカラー値は無視されるため、タイプミスはレンダリングを破壊することはできません。

次の例は、暗いプリセットを保持しながら、プロンプトアクセント、エラーテキスト、成功テキストを再色付けするテーマを定義しています。

リスト 4: ~/.claude/themes/dracula.json

```
{
  "name": "Dracula",
  "base": "dark",
```

(次のページに続く)

(前のページからの続き)

```
  "overrides": {
    "claude": "#bd93f9",
    "error": "#ff5555",
    "success": "#50fa7b"
  }
}
```

Claude Code は ~/.claude/themes/ を監視し、ファイルが追加または変更されるとリロードするため、エディターで行われた編集は再起動なしで実行中のセッションに適用されます。Claude Code の起動時に ~/.claude/themes/ フォルダ自体が存在しなかった場合は、最初のテーマファイルを作成した後に 1 回再起動します。その後、変更は再起動なしで適用されます。

以下のリファレンスは、overrides で設定できるトークンをカバーしています。/theme の対話的エディターは、同じトークンをライブプレビューと共に表示し、さらにオンボーディング画面のカラーなどの単一目的のアクセントをいくつか表示します（ここでは省略されています）。

カラートークンリファレンス

次の例は、以下のいくつかのグループからトークンを組み合わせています。ブランドアクセント、プランモードボーダー、diff 背景、およびメッセージ背景です。

リスト 5: ~/.claude/themes/midnight.json

```
{
  "name": "Midnight",
  "base": "dark",
  "overrides": {
    "claude": "#a78bfa",
    "planMode": "#38bdf8",
    "diffAdded": "#14532d",
    "diffRemoved": "#7f1d1d",
    "userMessageBackground": "#1e1b4b"
  }
}
```

テキストとアクセントカラー

プライマリブランドアクセントと、インターフェース全体で使用されるフォアグラウンドテキストの色合いを制御します。

トークン	制御対象
claude	プライマリブランドアクセント、スピナーとアシスタントラベルに使用
text	デフォルトのフォアグラウンドテキスト
inverseText	ステータスバッジなどの色付き背景の上に描画されるテキスト
inactive	ヒント、タイムスタンプ、無効な項目などのセカンダリテキスト
subtle	薄いボーダーと強調されていないセカンダリテキスト

次のページに続く

表 3 – 前のページからの続き

トークン	制御対象
suggestion	オートコンプリート候補とピッカーの選択ハイライト
permission	パーミッションプロンプトとピッカーを含むダイアログボーダー
remember	メモリと CLAUDE.md インジケーター

ステータスカラー

メッセージとインジケーター全体で成功、失敗、警告状態を通知します。

トークン	制御対象
success	成功メッセージと合格チェック
error	エラーメッセージと失敗
warning	警告、注意メッセージ、および auto モードインジケーター
merged	マージされたプルリクエストステータス

入力ボックスとモードインジケーター

入力ボックスのボーダーカラーと、パーミッションモードまたはインジケーターがアクティブな間に表示されるアクセントを設定します。

トークン	制御対象
promptBorder	入力ボックスボーダー
planMode	Plan モードアクセント、プランメッセージ、および Plan モードダイアログ
autoAccept	Accept-edits モードアクセント
bashBorder	! シェルコマンドを入力するときの入力ボックスボーダー
ide	IDE 接続インジケーター
fastMode	Fast モードインジケーター
effortUltra	*ultracode* (ページ 747) がオンの間、入力ボックスボーダーの ultracode タグ。このカラーのオーバーライドは Claude Code v2.1.239 以降で有効になります

Diff レンダリング

ファイル編集とレビューで追加および削除されたコードを色付けします。

トークン	制御対象
diffAdded	追加された行の背景
diffRemoved	削除された行の背景

次のページに続く

表 6 – 前のページからの続き

トークン	制御対象
diffAddedDimmed	編集を拒否した後に表示される薄い diff の追加行の背景
diffRemovedDimmed	編集を拒否した後に表示される薄い diff の削除行の背景
diffAddedWord	追加された行内のワードレベルハイライト
diffRemovedWord	削除された行内のワードレベルハイライト

フルスクリーンモード

Claude Code は、デフォルトとフルスクリーンレンダラーの両方で userMessageBackground、bash MessageBackgroundColor、および memoryBackgroundColor を描画します。フルスクリーンレンダリングモードでのみ userMessageBackgroundHover と selectionBg を使用します。

トークン	制御対象
userMessageBackground	トランスクリプト内のメッセージの背後の背景
userMessageBackgroundHover	ホバーまたは展開されている間のメッセージの背後の背景
bashMessageBackgroundColor	トランスクリプト内の ! シェルコマンドエントリの背後の背景
memoryBackgroundColor	トランスクリプト内の # メモリエントリの背後の背景
selectionBg	マウスで選択されたテキストの背景

使用量メーターとスピーカーラベル

/usage ビューに表示されるバーと、メッセージを Claude のメッセージと区別するラベルを調整します。

トークン	制御対象
rate_limit_fill	使用量メーターの塗りつぶされた部分
rate_limit_empty	使用量メーターの塗りつぶされていない部分
briefLabelYou	メッセージの You ラベルのカラー
briefLabelClaude	アシスタントメッセージの Claude ラベルのカラー

シマーバリアントとサブエージェントカラー

いくつかのトークンには、スピナーのアニメーション化されたグラデーションで使用される明るいカラーを提供するペアのシマーバリアントがあります。アニメーションが一致しないように見える場合は、ベーストークンと一緒にシマーをオーバーライドします。

- claude と claudeShimmer
- warning と warningShimmer
- permission と permissionShimmer

- promptBorder と promptBorderShimmer
- inactive と inactiveShimmer
- fastMode と fastModeShimmer

各**サブエージェント** (ページ 317) と並列タスクは、トランスクリプト内で区別できるように 8 つの名前付きカラーの 1 つで表示されます。トークン名は <color>_FOR_SUBAGENTS_ONLY パターンに従います。ここで <color> は red、blue、green、yellow、purple、orange、pink、または cyan です。これらをオーバーライドして、各名前付きカラーの外観を変更します。たとえば、定義に color: blue を持つサブエージェントは、blue_FOR_SUBAGENTS_ONLY 値を使用して描画されます。

Claude Code は、プロンプト入力で *ultrathink* (ページ 750) キーワードを 7 色の虹グラデーションでレンダリングします。トークン名は rainbow_<color> と rainbow_<color>_shimmer パターンに従います。ここで <color> は red、orange、yellow、green、blue、indigo、または violet です。

32.7 フルスクリーンレンダリングに切り替える

スクリーンリーダーモードでは、このセクションは適用されません。Claude Code は、添付されたバックグラウンドセッションを除き、常にプレーンなスクロールテキストとしてレンダリングされます。また、他のセッションで /tui fullscreen を実行した場合、Claude Code はレンダラーを切り替える代わりに説明を出力します。

ディスプレイがちらつくか、Claude が作業中にスクロール位置がジャンプする場合は、フルスクリーンレンダリングモードに切り替えてください。このモードでは、ターミナルのネイティブスクロールバックではなく、マウスまたは PageUp を使用して Claude Code 内でスクロールします。検索とコピーの方法については、フルスクリーンページを参照してください。

ちらつきが唯一の問題で、ターミナルが同期出力をサポートしているが自動検出されていない場合 (Emacs eat など)、CLAUDE_CODE_FORCE_SYNC_OUTPUT=1 を設定して、レンダラーを変更せずにちらつきを停止します。

/tui fullscreen を実行して切り替え、設定を保存します。会話はそのまま再起動され、フルスクリーン開始に失敗しない限り、今後のセッションはフルスクリーンで開始されます。Claude Code を起動する前に、CLAUDE_CODE_NO_FLICKER 環境変数を設定することもできます。

リスト 6: Bash and Zsh

```
CLAUDE_CODE_NO_FLICKER=1 claude
```

リスト 7: PowerShell

```
$env:CLAUDE_CODE_NO_FLICKER = "1"; claude
```

リスト 8: ~/.claude/settings.json

```
{
  "env": {
    "CLAUDE_CODE_NO_FLICKER": "1"
```

(次のページに続く)

(前のページからの続き)

```
  }
}
```

32.8 幅広いターミナルでレスポンス幅を制限する

幅広いターミナルでは、Claude のレスポンス内の各行のテキストがウィンドウの全幅に渡ります。代わりに設定した列数でテキストを折り返すには、設定で `maxProseWidth` を設定してください。

32.9 大量のコンテンツを貼り付ける

800 文字以上または 3 行以上をプロンプトに貼り付けると、Claude Code は入力を `[Pasted text #1 +120 lines]` などのプレースホルダーに折りたたんで、入力ボックスが使用可能な状態を保ちます。送信時に完全なコンテンツを送信します。ファイル全体やログが長いなど、非常に大きな入力の場合は、コンテンツをファイルに書き込んで、貼り付けの代わりに Claude にそれを読むよう依頼してください。会話トランスクリプトが読みやすくなり、Claude が後のターンでパスによってファイルを参照できます。VS Code 統合ターミナルは、非常に大きな貼り付けから Claude Code に到達する前に文字をドロップできるため、そこではファイルを使用してください。

貼り付けに **目に見えない** *Unicode* **文字** (ページ 96) が含まれている場合、Claude Code は Enter キーを押すときにそれらを削除し、クリーンアップされたプロンプトを入力ボックスに戻して、別の Enter で送信できるようにします。

32.9.1 Claude が貼り付けたテキストをどのように処理するか

送信すると、Claude は各 `[Pasted text #N]` プレースホルダーの背後にあるコンテンツを、他の場所から貼り付けたテキストとして表示され、入力したテキストではなく表示されます。Claude には、貼り付けには入力していない指示が含まれる可能性があり、入力したメッセージがそれを要求する場所でのみ指示に従うよう指示されます。機能フラグを取得 しないセッションでは、貼り付けはマークされません。

32.9.2 折りたたまれた貼り付けを削除して復元する

`Ctrl+W` や `Ctrl+K` などの単語または行ショートカットで削除するか、`df]` などの `f/t` モーションを使用した vim 削除で削除し、削除範囲が `[Pasted text #N]` プレースホルダー内に達する場合、Claude Code はプレースホルダー全体を削除します。復元するには、単語または行ショートカットの後に *`Ctrl+Y`* (ページ 80) で削除を貼り付け直すか、vim 削除の後に *p NORMAL* **モード** (ページ 85) で復元してください。

32.9.3 貼り付けたテキストを含むプロンプトを呼び出す

Claude Code は折りたたまれたコンテンツを各 `[Pasted text #N]` プレースホルダーの背後に `~/.claude/paste-cache/` に保持するため、**コマンド履歴** (ページ 87) からプロンプトを呼び出して再送信すると、完全な貼り付けコンテンツが再度送信され、後のセッションでも含まれます。

保持期間スイープルール に従い、`cleanupPeriodDays` より古いキャッシュファイルは削除されるため、呼び出されたプロンプトは存在しなくなった貼り付けテキストを参照できます。そのようなプロンプトを送

信すると、Claude Code はリテラル `[Pasted text #N]` 文字列を送信することはなく、不足している貼り付けの名前を示す通知を表示します。

- テキストが残っているプレーンプロンプトでは、Claude Code はプレースホルダーを削除して残りのテキストを送信します。
- **シェルモード** (ページ 89) コマンドまたは / コマンドでは、削除が実行内容を変更する可能性があり、プロンプトの削除が空のままになる場合、Claude Code は送信をキャンセルし、元のテキストを入力に保持します。プレースホルダーはまだ含まれています。プレースホルダーを削除するか、コマンドを編集してから再送信してください。

32.10 Vim キーバインディングでプロンプトを編集する

Claude Code には、プロンプト入力用の Vim スタイルの編集モードが含まれています。`/config` → Editor mode から有効にするか、`~/.claude/settings.json` で `editorMode` を `"vim"` に設定することで有効にできます。Editor mode を `normal` に戻すと、オフになります。

Vim モードは、NORMAL モードと VISUAL モードのモーション演算子のサブセットをサポートしており、`hjkl` ナビゲーション、`v`/`V` 選択、テキストオブジェクトを使用した `d`/`c`/`y` などが含まれます。完全なキーテーブルについては、*Vim* **エディタモードリファレンス** (ページ 84) を参照してください。

Vim モーションはキーバインディングファイルを通じて再マップできません。`jj` を Escape にマップするなどの 2 キー INSERT モードシーケンスをマップするには、ユーザー設定で *`vimInsertModeRemaps`* (ページ 84) を設定してください。

INSERT モードで Enter キーを押すと、標準的な Vim とは異なり、プロンプトが送信されます。代わりに NORMAL モードで `o` または `O` を使用するか、Ctrl+J を使用して改行を挿入してください。

32.11 関連リソース

- **インタラクティブモード** (ページ 75)：完全なキーボードショートカットリファレンスと Vim キーテーブル
- **キーバインディング** (ページ 765)：Enter と Shift+Enter を含む任意の Claude Code ショートカットを再マップ
- フルスクリーンレンダリング：フルスクリーンモードでのスクロール、検索、コピーの詳細
- **フック ガイド** (ページ 411)：Linux と Windows の詳細な通知フック例
- **トラブルシューティング** (ページ 37)：ターミナル設定外の問題の修正

第33章　その他の LLM gateway

組織が既に実行している LLM gateway を通じて Claude Code をルーティングします。Claude Code をゲートウェイに接続する方法、組織向けのロールアウト、Claude Code がゲートウェイに送信する内容について説明します。

このセクションでは、Claude apps gateway ではなく、組織が既に実行しているゲートウェイ製品を使用する方法について説明します。ゲートウェイとは何か、Claude Code とプロバイダー間にどのように位置するか、Claude apps gateway と別の製品のどちらを選択するかについては、ゲートウェイの概要を参照してください。

注釈:

- 既存のゲートウェイに接続する開発者の場合：Claude Code をゲートウェイに接続
- 組織向けのゲートウェイをロールアウトする管理者の場合：ゲートウェイをデプロイして配布
- ゲートウェイ製品を設定している場合：ゲートウェイプロトコルリファレンス

サポートされている API 形式を公開するゲートウェイはすべて機能します。Anthropic は、サードパーティゲートウェイ製品を推奨、保守、または監査していません。また、任意のゲートウェイを通じて Claude Code を非 Claude モデルにルーティングすることはサポートしていません。ゲートウェイを独自のドキュメントに従ってデプロイしてから、以下の**ロールアウト手順** (ページ 842) で Claude Code 側を完了してください。

33.1 ゲートウェイが提供するもの

ゲートウェイは、組織が以下を管理する 1 つの場所を提供します：

- **認証情報**：プロバイダーキーはサーバー側に留まり、開発者はゲートウェイ認証情報を保持します
- **使用状況追跡**：リクエストを処理するプロバイダーに関係なく、開発者またはチームごとに使用状況を属性付けします
- **コスト管理**：予算とレート制限を 1 つの場所で実施します
- **監査ログ**：コンプライアンスのためにすべてのモデルリクエストをログに記録します
- **プロバイダー切り替え**：開発者マシンに触れることなく、ゲートウェイ設定でプロバイダーを変更します

プロバイダー切り替え以外のすべてが、アップストリームが Anthropic の API であるか**クラウドプロバイダー** (ページ 847) であるかに関わらず適用されます。プロバイダー切り替えが開発者マシンの再設定なしで機能するには、アップストリームに関わらず、ゲートウェイが単一の Anthropic 形式エンドポイントを

公開する必要があります。プロバイダー独自の形式を公開するゲートウェイは、クライアント設定をそのプロバイダーに結び付け、Claude Code が送信する内容と適用するデフォルトが変わります。

トレードオフとして、ゲートウェイは組織が運用するインフラストラクチャになります。Claude Code は各リリースで機能を追加し、ゲートウェイがそれらを転送しない場合、対応する機能が破損するため、Claude Code の進化に合わせてゲートウェイ製品を最新に保つ必要があります。ゲートウェイプロトコルリファレンスでは、何を転送するかについて説明しています。

33.2 ゲートウェイをロールアウトする

組織に LLM gateway をロールアウトする準備ができたら、選択するゲートウェイ製品に関わらず、シーケンスは同じです：

1. ゲートウェイをデプロイし、転送するリクエストを認証できるようにプロバイダー認証情報を提供します。
2. 各開発者にゲートウェイ認証情報を発行し、使用状況が開発者に属性付けられ、オフボーディングが 1 つの認証情報を取り消すようにします。
3. 管理設定ファイルとシークレットツーリングを通じて設定を配布し、すべてのマシンがベース URL と認証情報を受け取るようにします。両方が配布されると、開発者は何も設定しません。設定配布が整っていない場合、開発者は接続ページに従って変数を自分で設定します。
4. 各開発者に Claude Code で設定を確認させ、配布の問題がゲートウェイに依存する前に表面化するようにします。

組織向けの LLM gateway をロールアウトでは、各ステップを説明し、各ステップで配布する設定ファイルを示しています。ゲートウェイは組織セットアップの 1 つの部分です。ポリシー実施、使用状況の可視性、データ処理の決定については、組織向けに Claude Code をセットアップを参照してください。

`ANTHROPIC_BASE_URL` を通じて到達するゲートウェイを、管理対象マシンが使用できる唯一の宛先にするには、同じ管理設定ファイルで `allowedProviders` を `["customEndpoint"]` に設定し、ゲートウェイの `ANTHROPIC_BASE_URL` をそのファイルの `env` ブロックに配置します。Claude Code はその後、Anthropic に直接接続する場合や開発者独自のプロキシを含む他の場所を指すセッションを拒否し、`ANTHROPIC_BASE_URL` をそこに設定した値でのみ受け入れます。`ANTHROPIC_BEDROCK_BASE_URL` などのプロバイダー固有のエンドポイント変数を通じて到達するゲートウェイの場合、`allowedProviders` エントリはどの変数をピン留めするかを指定します。Claude Code v2.1.285 以降が必要です。

33.3 サブスクリプションとゲートウェイ

ゲートウェイ認証情報変数または `apiKeyHelper` がアクティブな場合、リクエストは開発者の claude.ai サブスクリプションログインの代わりにその認証情報を使用し、サブスクリプションの使用制限は適用されません。Claude Code はマシンに保存された claude.ai ログインを保持していますが、それらのリクエストでは送信しません。そのトラフィックは、ゲートウェイが転送する認証情報の所有者（組織の Anthropic Console アカウント、またはゲートウェイがそこにルーティングする場合の Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry アカウント）にトークンごとに請求されます。

`ANTHROPIC_BASE_URL` は Claude Code をゲートウェイに指し示す変数です。ゲートウェイ認証情報なしでその変数のみを設定しても、サブスクリプションは置き換わりません。リクエストはゲートウェイ経由でルーティングされますが、保存された claude.ai ログインはアクティブな認証情報のままなので、その使用

制限と請求が適用されます。このトラフィックを Anthropic に渡すゲートウェイは、`anthropic-beta` の OAuth 機能を転送する必要があります。リクエストヘッダーリファレンスを参照してください。

33.4 関連ページ

- ゲートウェイの概要：ゲートウェイの仕組みと Claude apps gateway と別の製品のどちらを選択するかについて
- Claude apps gateway：SSO サインインと OTLP テレメトリを備えた Anthropic の自己ホスト型ゲートウェイ
- Claude Code を LLM gateway に接続：自分のマシンでベース URL と認証情報を設定し、サーフェスごとの設定とトラブルシューティングテーブルを含みます
- 組織向けの LLM gateway をロールアウト：ゲートウェイをデプロイし、開発者認証情報を発行し、管理設定を配布するための管理者チェックリスト
- ゲートウェイプロトコルリファレンス：Claude Code がゲートウェイに送信するもの、ゲートウェイを設定する運用者向け、エンドポイント、転送するヘッダー、機能パススルーをカバーしています

第VI部

エンタープライズ

第34章　エンタープライズデプロイメント概要

Claude Code が様々なサードパーティサービスとインフラストラクチャと統合して、エンタープライズデプロイメント要件を満たす方法について学びます。

組織は Anthropic を通じて直接、またはクラウドプロバイダーを通じて Claude Code をデプロイできます。このページは、適切な構成を選択するのに役立ちます。

34.1 デプロイメントオプションの比較

ほとんどの組織では、Claude for Teams または Claude for Enterprise が最適なエクスペリエンスを提供します。チームメンバーは、単一のサブスクリプション、一元化された請求、インフラストラクチャセットアップが不要で、Claude Code と Web 上の Claude の両方にアクセスできます。

Claude for Teams はセルフサービスで、コラボレーション機能、管理ツール、SSO、請求管理、およびサーバー管理設定（組織全体の Claude Code 構成用）が含まれています。迅速に開始する必要がある小規模なチームに最適です。

Claude for Enterprise はドメインキャプチャ、ロールベースの権限、およびコンプライアンス API アクセスを追加します。セキュリティとコンプライアンス要件がある大規模な組織に最適です。

Team プランと Enterprise プランの詳細をご覧ください。

以下で比較されるデプロイメントオプションは、モデル推論が実行される場所をカバーしています。組織が運用するコンピュート上で *Claude Code on the web* (ページ 223) セッションを実行するには、セルフホスト環境を参照してください。

組織に特定のインフラストラクチャ要件がある場合は、以下のオプションを比較してください。

機能	Claude for Teams/Enterprise	Anthropic Console	Amazon Bedrock	Claude Platform on AWS	Google Cloud の Agent Platform（旧 Vertex AI）	Microsoft Foundry
最適な用途	ほとんどの組織（推奨）	個別開発者	AWS ネイティブデプロイメント	Claude API 機能を備えた AWS Marketplace 請求	GCP ネイティブデプロイメント	Azure ネイティブデプロイメント
請求	**Teams:** シートあたりのサブスクリプション（PAYG 利用可能）、価格を参照 **Enterprise:** 営業に連絡	PAYG	AWS 経由の PAYG	AWS Marketplace 経由の PAYG	GCP 経由の PAYG	Azure 経由の PAYG
リージョン	サポート対象国	サポート対象国	複数の AWS リージョン	複数の AWS リージョン	複数の GCP リージョン	複数の Azure リージョン
Prompt caching	デフォルトで有効	デフォルトで有効	デフォルトで有効	デフォルトで有効	デフォルトで有効	デフォルトで有効

次のページに続く

表 1 – 前のページからの続き

機能	Claude for Teams/Enterprise	Anthropic Console	Amazon Bedrock	Claude Platform on AWS	Google Cloud の Agent Platform（旧 Vertex AI）	Microsoft Foundry
認証	claude.ai SSO またはメール	API キーまたは *API キーなしのコンソールサインイン* (ページ 853)	API キーまたは AWS 認証情報	API キーまたは AWS 認証情報	GCP 認証情報	API キーまたは Microsoft Entra ID
コスト追跡	使用状況ダッシュボード	使用状況ダッシュボード	AWS Cost Explorer	AWS Cost Explorer	GCP Billing	Azure Cost Management
Web 上の Claude を含む	はい	いいえ	いいえ	いいえ	いいえ	いいえ
エンタープライズ機能	チーム管理、SSO、使用状況監視	なし	IAM ポリシー、CloudTrail	IAM ポリシー、CloudTrail	IAM ロール、Cloud Audit Logs	RBAC ポリシー、Azure Monitor

各オプションで利用可能な機能の詳細な比較については、機能の可用性を参照してください。

デプロイメントオプションを選択してセットアップ手順を表示します。

- *Claude for Teams* **または** *Enterprise* (ページ 852)
- *Anthropic Console* (ページ 853)
- Claude apps gateway、Amazon Bedrock、Claude Platform on AWS、Google Cloud の Agent Platform、Microsoft Foundry、または Anthropic API の前に IdP サインインを追加するセルフホスト型ゲートウェイ
- *Amazon Bedrock* (ページ 863)
- Claude Platform on AWS
- *Google Cloud* **の** *Agent Platform* (ページ 881)
- *Microsoft Foundry* (ページ 891)

Amazon Bedrock と Google Vertex AI の場合、ログインプロンプトで `claude` を実行して **3rd-party platform** を選択し、インタラクティブセットアップウィザードを起動することもできます。

34.2 プロキシとゲートウェイの構成

ほとんどの組織は、追加の構成なしでクラウドプロバイダーを直接使用できます。ただし、組織に特定のネットワークまたは管理要件がある場合は、企業プロキシまたは LLM ゲートウェイを構成する必要がある場合があります。これらは一緒に使用できる異なる構成です。

- **企業プロキシ**: HTTP/HTTPS プロキシを通じてトラフィックをルーティングします。組織がセキュリティ監視、コンプライアンス、またはネットワークポリシー実装のためにすべての送信トラフィックをプロキシサーバーを通じて渡す必要がある場合に使用します。`HTTPS_PROXY` または `HTTP_PROXY` 環境変数で構成します。**エンタープライズネットワーク構成** (ページ 897) で詳細をご覧ください。
- **LLM ゲートウェイ**: Claude Code とクラウドプロバイダーの間に位置して、認証とルーティングを処理するサービスです。チーム全体の一元化された使用状況追跡、カスタムレート制限または予算、または一元化された認証管理が必要な場合に使用します。`ANTHROPIC_BASE_URL`、`ANTHROPIC_BEDROCK_BASE_URL`、`ANTHROPIC_AWS_BASE_URL`、`ANTHROPIC_VERTEX_BASE_URL`、または `ANTHROPIC_FOUNDRY_BASE_URL` 環境変数で構成します。*LLM* **ゲートウェイ** (ページ 841) で詳細をご覧ください。

プロバイダーごとの環境変数で Amazon Bedrock、Microsoft Foundry、または Google Cloud の Agent Platform を LLM ゲートウェイを通じてルーティングする場合は、ゲートウェイを通じてクラウドプロバイダーにルーティングを参照してください。Claude Code で `/status` を実行して、セッションが使用しているプロバイダー、ベース URL、およびプロキシを確認します。

組織が顧客管理暗号化キー（CMEK）を使用し、Claude Code を LLM ゲートウェイまたはカスタム `ANTHROPIC_BASE_URL` を通じてルーティングする場合、CMEK はそれらのセッションの Claude Code の運用テレメトリには適用されません。すべての開発者のテレメトリをオフにするには、組織のテレメトリをオフにするに示されているように、管理設定を通じて `DISABLE_TELEMETRY` を配信します。

34.3 組織のベストプラクティス

34.3.1 ドキュメントとメモリに投資する

Claude Code がコードベースを理解できるようにドキュメントに投資することを強くお勧めします。組織は複数のレベルで CLAUDE.md ファイルをデプロイできます。**_CLAUDE.md_ ファイルをどこに配置できるか** (ページ 998) と **組織全体の _CLAUDE.md_ をデプロイする方法** (ページ 1006) をご覧ください。

34.3.2 デプロイメントを簡素化する

カスタム開発環境がある場合は、Claude Code をインストールする「ワンクリック」の方法を作成することが、組織全体での採用を促進するための鍵となることがわかっています。

34.3.3 ガイド付き使用から始める

新しいユーザーに Claude Code をコードベースの Q&A、または小さなバグ修正または機能リクエストで試すことをお勧めします。Claude Code にプランを作成するよう依頼します。Claude の提案を確認し、軌道を外れている場合はフィードバックを提供します。時間が経つにつれて、ユーザーがこの新しいパラダイムをより理解するようになると、Claude Code をより積極的に実行させるのに効果的になります。

34.3.4 クラウドプロバイダーのモデルバージョンをピン留めする

Amazon Bedrock (ページ 863)、*Google Cloud の Agent Platform* (ページ 881)、*Microsoft Foundry* (ページ 891)、または Claude Platform on AWS を通じてデプロイする場合は、`ANTHROPIC_DEFAULT_FABLE_MODEL`、`ANTHROPIC_DEFAULT_OPUS_MODEL`、`ANTHROPIC_DEFAULT_SONNET_MODEL`、および `ANTHROPIC_DEFAULT_HAIKU_MODEL` を使用して特定のモデルバージョンをピン留めします。ピン留めしない場合、モデルエイリアスは Claude Code のそのプロバイダーの組み込みデフォルトに解決され、最新リリースより遅れる可能性があり、アカウントでまだ有効になっていない可能性があります。ピン留めすることで、ユーザーが新しいモデルに移行するタイミングを制御できます。各プロバイダーがデフォルトが利用できない場合に何を行うかについては、**モデル構成** (ページ 758) を参照してください。

34.3.5 セキュリティポリシーを構成する

セキュリティチームは、Claude Code が実行できることと実行できないことに対する管理権限を構成できます。これはローカル構成によって上書きされません。**詳細をご覧ください** (ページ 1033)。

これらのデプロイメントオプションのうち、管理対象マシンが使用できるものを制限するには、管理設定で `allowedProviders` を設定します。たとえば、`["bedrock"]` は Amazon Bedrock のみを許可します。Mantle エンドポイントも有効にする Bedrock フリートは `"mantle"` も一覧に含めます。このエントリは、管理対象の env ピン留めも必要なエンドポイント変数を示します。Claude Code v2.1.285 以降が必要です。

34.3.6 統合に MCP を活用する

MCP は Claude Code にチケット管理システムやエラーログへの接続など、より多くの情報を提供する優れた方法です。1 つの中央チームが MCP サーバーを構成し、`.mcp.json` 構成をコードベースにチェックインして、すべてのユーザーが利益を得られるようにすることをお勧めします。**詳細をご覧ください** (ページ 363)。

34.4 次のステップ

デプロイメントオプションを選択し、チームのアクセスを構成したら、以下を実行します。

1. **チームにロールアウトする**: インストール手順を共有し、チームメンバーに *Claude Code* **をインストール** (ページ 19) して認証情報で認証するよう依頼します。
2. **共有構成をセットアップする**: リポジトリに *CLAUDE.md* **ファイル** (ページ 997) を作成して、Claude Code がコードベースとコーディング標準を理解するのに役立てます。
3. **権限を構成する**: **セキュリティ設定** (ページ 1033) を確認して、環境内で Claude Code が実行できることと実行できないことを定義します。

第35章　認証

Claude Code にログインし、個人、チーム、組織向けの認証を設定します。

Claude Code は、セットアップに応じて複数の認証方法をサポートしています。個人ユーザーは claude.ai アカウントでログインでき、チームは Claude for Teams または Enterprise、Claude Console、または Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry などのクラウドプロバイダーを使用できます。

35.1 Claude Code にログインする

Claude Code **をインストール** (ページ 20) した後、ターミナルで `claude` を実行します。初回起動時に、Claude Code はログインするためのブラウザウィンドウを開きます。`ANTHROPIC_API_KEY` 環境変数を設定している場合、Claude Code はログインプロンプトをスキップし、代わりにキーを承認するよう求めます。

ブラウザが自動的に開かない場合は、`c` を押してログイン URL をクリップボードにコピーし、ブラウザに貼り付けます。

ブラウザがサインイン後にリダイレクトされずにログインコードを表示する場合は、`Paste code here if prompted` プロンプトでそれをターミナルに貼り付けます。これは、ブラウザが Claude Code のローカルコールバックサーバーに到達できない場合に発生します。これは WSL2、SSH セッション、およびコンテナで一般的です。

ログインが完了すると、ターミナルに `Login successful` と表示され、`Enter` キーを押して続行するよう求められます。

以下のいずれかのアカウントタイプで認証できます。

- **Claude Pro または Max サブスクリプション**: claude.ai アカウントでログインします。claude.com/pricing で購読してください。
- **Claude for Teams または Enterprise**: チーム管理者が招待した claude.ai アカウントでログインします。
- **Claude Console**: Console 認証情報でログインします。管理者が事前に**招待** (ページ 853) している必要があります。*API* **キーを作成せずにサインイン** (ページ 853) することも、API キーを作成することもできます。
- **クラウドプロバイダー**: 組織が *Amazon Bedrock* (ページ 863)、*Google Cloud* **の** *Agent Platform* (ページ 881)、または *Microsoft Foundry* (ページ 891) を使用している場合は、`claude` を実行する前に必要な環境変数を設定するか、ログインプロンプトで **3rd-party platform** を選択してください。これにより、Bedrock と Vertex AI 向けのインタラクティブセットアップウィザードが起動します。ブラウザログインは不要です。
- **クラウドゲートウェイ**: 組織がセルフホストされた Claude apps gateway を実行している場合は、`/login` を通じて企業 SSO でサインインします。ゲートウェイが発行したトークンはセッションの唯一の認証情報です。

管理者は、どのログイン方法を開発者が使用するかを指定し、claude.ai ログインが特定の組織に属することを要求できます。**ログインを組織に制限する** (ページ 855) を参照してください。

ログアウトして再認証するには、Claude Code プロンプトで `/logout` と入力します。ログアウトすると、初回起動セットアップ状態もリセットされるため、次回 `claude` を実行するときはログインとセットアップを再度実行します。

ログインに問題がある場合は、認証のトラブルシューティングを参照してください。

35.1.1 複数のアカウントでログインする

複数のアカウント（仕事用と個人用など）に同時にログインしたままにするには、各アカウントに独自の設定ディレクトリを指定します。`claude` を起動するときに、`CLAUDE_CONFIG_DIR` 環境変数を使用するアカウントのディレクトリに設定します。各ディレクトリには、独自の設定、セッション履歴、および claude.ai ログインまたは API キーがあります。たとえば、Bash または Zsh では、`~/.bashrc` または `~/.zshrc` に次のエイリアスを追加して、`claude-work` が仕事用アカウントを使用し、`claude` が個人用アカウントを保持するようにできます。

```
alias claude-work='CLAUDE_CONFIG_DIR=~/.claude-work claude'
```

新しいターミナルを開いて初めて `claude-work` を実行した後、Claude Code は新しいディレクトリのログインとセットアップを実行します。別のディレクトリは、Claude Code がそのような種類のサインインを設定ディレクトリの外に保存するため、2 つの Claude Console サインイン *API* **キーなし** (ページ 853) を区別しません。

35.2 チーム認証を設定する

チームと組織の場合、Claude Code アクセスを以下のいずれかの方法で設定できます。

- *Claude for Teams* **または** *Enterprise* (ページ 852)（ほとんどのチームに推奨）
- *Claude Console* (ページ 853)
- Claude apps gateway（開発者を IdP でサインインさせ、設定したクラウドプロバイダーに推論をルーティングする自己ホスト型ゲートウェイ）
- *Amazon Bedrock* (ページ 863)
- *Google Cloud* **の** *Agent Platform* (ページ 881)
- *Microsoft Foundry* (ページ 891)

35.2.1 Claude for Teams または Enterprise

Claude for Teams と Claude for Enterprise は、Claude Code を使用する組織に最適なエクスペリエンスを提供します。チームメンバーは Claude Code と Web 上の Claude の両方にアクセスでき、一元化された請求とチーム管理が可能です。

- **Claude for Teams**: コラボレーション機能、管理ツール、SSO、請求管理、および組織全体の Claude Code 設定のための サーバー管理設定を備えたセルフサービスプラン。小規模なチームに最適です。

- **Claude for Enterprise**: ドメインキャプチャ、ロールベースの権限、およびコンプライアンス API を追加します。セキュリティとコンプライアンス要件を持つ大規模な組織に最適です。

1. **購読**

   Claude for Teams に購読するか、Claude for Enterprise の営業に連絡してください。

2. **チームメンバーを招待**

   管理ダッシュボードからチームメンバーを招待します。

3. **インストールしてログイン**

   チームメンバーは Claude Code をインストールし、claude.ai アカウントでログインします。

35.2.2 Claude Console 認証

API ベースの請求を希望する組織の場合、Claude Console を通じてアクセスを設定できます。

1. **Console アカウントを作成または使用**

   既存の Claude Console アカウントを使用するか、新しいアカウントを作成します。

2. **ユーザーを追加**

   以下のいずれかの方法でユーザーを追加できます。

   - Console 内からユーザーを一括招待します。Settings -> Members -> Invite
   - SSO を設定

3. **ロールを割り当て**

   ユーザーを招待する際に、以下のいずれかを割り当てます。

   - **Claude Code** ロール: ユーザーは Claude Code API キーのみを作成できます
   - **Developer** ロール: ユーザーはあらゆる種類の API キーを作成できます

4. **ユーザーがセットアップを完了**

   招待された各ユーザーは以下を実行する必要があります。

   - Console 招待を受け入れる
   - **システム要件を確認** (ページ 19)
   - *Claude Code* **をインストール** (ページ 20)
   - Console アカウント認証情報でログイン

API キーなしでサインイン

API キーを作成しなくても Console アカウントにサインインできます。組織が開発者による API キーの作成を許可していない場合でも可能です。`/login` プロンプトで Anthropic Console アカウントを選択すると、Claude Code はサインイン方法を尋ねます。Claude Code v2.1.242 以降が必要です。両方のルートは Console にブラウザーでサインインしますが、Claude Code がその後に保存する内容が異なります。

- **Console アカウントでサインイン**（（推奨）とラベル付け）: Claude Code はそのサインインから OAuth トークンを保持し、*Anthropic* **プロファイル** (ページ 859) として保存します。API キーは作成されません
- **API キーを作成**（（レガシー） とラベル付け）: Claude Code は Console API キーを作成し、他の認証情報と一緒に保存します

実際には、プロファイルは OAuth ログインを保存し、API キーは静的な認証情報です。Claude Code はプロファイルのログインを自動的に更新し、更新に失敗すると、再度サインインするまで Anthropic プロファイルログイン期限切れ でリクエストが失敗します。

すべてのマシンで選択肢が得られるわけではありません。Claude Code は以下の場合、確認なしに API キーを作成します。

- Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry などの **クラウドプロバイダーに対して実行** (ページ 847) するか、AWS 上の Claude Platform に対して実行する
- 任意の設定ファイルが *forceLoginOrgUUID* (ページ 855) を設定するか、`forceLoginMethod` を `"claudeai"` または `"console"` に設定する
- マシン上に管理設定ファイル、MDM プロファイル、キャッシュされたサーバー管理設定などの管理設定ソースが存在しますが、Claude Code が それを読み取ることができず、他の管理ソースがポリシーを提供していない

キーなしでサインインする前に `ANTHROPIC_API_KEY` を設定解除してください。

キーなしでサインインした後、保存された API キーの代わりにプロファイルが得られます。

- **書き込まれるプロファイル**: Claude Code は `ANTHROPIC_PROFILE` で指定されたプロファイル、またはアクティブなプロファイル、または `default` を書き込みます。そのプロファイルがフェデレーションプロファイルの場合、Claude Code はそれを上書きする代わりにサインインを拒否します
- **サインアウトする対象**: Claude Code はマシンに保存されている claude.ai ログインからサインアウトします
- **元に戻す方法**: `/logout` を実行します。これにより、このサインインが書き込んだ認証情報が削除および取り消されます

組織が サーバー管理設定を使用している場合、Claude Code v2.1.257 以降でこのサインインに適用されます。

プロファイルに関するその他すべてのことがこのサインインに適用されます。これには、他の認証情報に対するランク付け、`/status` で取得される `Profile` 行、および claude.ai ログインが必要な機能が含まれます。*Anthropic* **プロファイルとフェデレーション認証情報** (ページ 859) を参照してください。

35.2.3 クラウドプロバイダー認証

Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry を使用するチームの場合。

1. **プロバイダーセットアップに従う**

   Amazon Bedrock **ドキュメント** (ページ 863)、*Google Cloud* **の** *Agent Platform* **ドキュメント** (ページ 881)、または *Microsoft Foundry* **ドキュメント** (ページ 891) に従ってください。

2. **設定を配布**

環境変数とクラウド認証情報を生成するための手順をユーザーに配布します。**ここで設定を管理する方法** (ページ 711) についてさらに詳しく読んでください。

3. **Claude Code をインストール**

   ユーザーは *Claude Code* **をインストール** (ページ 20) できます。

35.2.4 ログインを組織に制限する

開発者の claude.ai ログインが特定の Anthropic 組織に属することを要求するには、管理設定で `forceLoginMethod` と `forceLoginOrgUUID` を設定します。`forceLoginOrgUUID` を組織 ID に設定します。組織 ID は Claude for Teams または Enterprise 組織の claude.ai 管理設定に表示されます。Claude Code は他の組織への claude.ai ログインについてエラーを報告し、使用中の claude.ai 認証情報がリストされていない組織に属している場合、起動時に終了します。

Claude Console ログインの場合、Claude Code は `forceLoginOrgUUID` を使用して、単一の Console 組織 ID に設定した場合、Console サインインページで組織を事前選択します。組織 ID は platform.claude.com/settings/organization に表示されます。ログイン時または起動時に、結果の Console 認証情報がどの組織に属しているかを確認しません。キーをデプロイする前に Console アカウントでログインした開発者はログインしたままになり、その保存されたキーはゲートウェイサインインも必要とするマシン上、またはクラウドプロバイダーを選択するセッション内でブロックされます。

任意の設定ファイルで `forceLoginOrgUUID` を設定した場合、Claude Code はそのファイルが適用されるセッションで **キーレス** *Console* **サインイン** (ページ 853) の提供を停止し、代わりに API キーを作成します。開発者を claude.ai サインインに向かわせるには、`forceLoginMethod` を `"claudeai"` に設定します。

Claude Code v2.1.212 以降では、ここにリストされているすべてのログインパスが `forceLoginMethod` を適用します。ターミナルのインタラクティブログイン画面（`/login` または初回オンボーディングで到達）では、Claude Code は `claudeai` または `console` メソッドを強制せずに事前選択するため、`forceLoginMethod` が `"claudeai"` に設定されている場合でも、開発者は Console ログインをそこで完了できます。

パスは `forceLoginOrgUUID` で異なります。

- **ターミナル、** *VS Code* **拡張機能** (ページ 173)**、および Agent SDK ログイン**: claude.ai アカウントログインの `forceLoginOrgUUID` を確認します
- `claude setup-token` **および** `/install-github-app`: `forceLoginMethod` のみを強制するため、別の組織でトークンを生成できます
- ゲートウェイ**サインイン**: `forceLoginMethod: "gateway"` によって選択され、それによって制限されず、Anthropic 組織に対して認証されないため、`forceLoginOrgUUID` は適用されません。ゲートウェイ ID プロバイダーを使用してアクセスを制限します

デバイス管理ツールを通じてキーをデプロイします。サーバー管理設定は、既に組織に認証されているアカウントにのみ到達するため、開発者の最初のログインをリダイレクトできません。組織がサーバー管理設定も配布する場合、両方の場所にキーを設定します。管理設定ソースは マージされず、キャッシュされたサーバー管理設定はデバイス管理ファイルを置き換えます。ただし、いくつかの キーごとの例外を除いて。`forceLoginOrgUUID` と `forceLoginMethod` の `"claudeai"` および `"console"` 値はこれらの例外に含まれていないため、両方の場所に保持します。

ゲートウェイデプロイメントでは、`forceLoginMethod` と `forceLoginOrgUUID` を ゲートウェイが提供する設定から除外したままにしてください。

キーはまた、ログイン認証情報を使用しないセッションが開始できるかどうかも決定します。設定リファレンスの forceLoginOrgUUID を参照して、完全な動作を確認してください。

- **ANTHROPIC_API_KEY、ANTHROPIC_AUTH_TOKEN、または apiKeyHelper**: 起動時にブロックされます。forceLoginOrgUUID では、環境認証情報の組織メンバーシップを確認できず、forceLoginMethod では認証情報が必要なサインインの代わりになります。管理設定が ゲートウェイサインインも必要とする場合、Claude Code は以前の Claude Console ログインによって保存された API キーを同じ方法でブロックします。管理者ポリシーがクラウドゲートウェイサインインを必要としますを参照してください
- **Amazon Bedrock などのクラウドプロバイダーセッション**: ANTHROPIC_API_KEY、ANTHROPIC_AUTH_TOKEN、または apiKeyHelper 認証情報、または以前の Claude Console ログインによって保存された API キーがマシン上にまだ存在する間のみブロックされます。それを削除するとセッションが開始します。これらのセッションはクラウドプロバイダーに対して認証され、クラウドプロバイダーのアクセスポリシーがそれらを管理します
- *Anthropic* **プロファイルまたはフェデレーション認証情報** (ページ 859): ANTHROPIC_API_KEY、ANTHROPIC_AUTH_TOKEN、または apiKeyHelper 認証情報、または以前の Claude Console ログインによって保存された API キーもマシン上に存在する場合を除き、ブロックされません。キーはプロファイルが属する組織を確認しません

35.2.5 マシンが使用できる API プロバイダーを制限する

管理設定の allowedProviders は、Anthropic API、Amazon Bedrock、LLM ゲートウェイなど、管理マシンが Claude に到達できるサービスをリストします。これは forceLoginMethod と forceLoginOrgUUID を補完します。これらは、セッションが Anthropic と通信するときに使用するアカウントを管理します。Claude Code v2.1.285 以降が必要です。

リスト 1: managed-settings.json

```
{
  "forceLoginMethod": "claudeai",
  "forceLoginOrgUUID": ["xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"],
  "allowedProviders": ["anthropic", "bedrock"]
}
```

このファイルを使用すると、claude.ai 組織にサインインした開発者または Amazon Bedrock 用に設定された開発者は正常に起動します。他のプロバイダー用に設定されたセッションは起動時に拒否され、実行中のセッションがそのセッションに切り替わると、次のリクエストで拒否されます。管理設定がこの API プロバイダーを許可していませんは各メッセージを表示します。

- **LLM ゲートウェイまたはプロキシを許可する**: "customEndpoint" をリストし、同じソースの管理 env ブロックでゲートウェイの URL を設定します。設定リファレンスはすべての値をリストし、どのエンドポイント変数が管理 env ピンを必要とするかを示します。
- **管理マシンにデプロイする**: リストをポリシーの残りを含む管理ソースに配置します。エントリの スコープ注記は、サーバー管理リストがそれとどのように組み合わされるかを示します。
- **サーバー管理設定のみ**: サーバー管理設定でのみ設定するリストは、組織の設定を取得するセッションにのみ到達するため、デバイス管理で到達できないマシンの利便性として扱い、強制として扱わな

いでください。プラットフォーム可用性はどのセッションがそれらを取得するかをリストします。

35.3 認証情報管理

Claude Code は認証情報を安全に管理します。

- **保存場所**:
  - macOS では、認証情報は暗号化された macOS Keychain に保存されます。Keychain が SSH セッションでロックされている場合など、書き込みを拒否する場合、Claude Code はログインを `~/.claude/.credentials.json` に保存し、ファイルモードは `0600` です。これは Linux で使用するのと同じストレージです。Keychain が書き込み可能になるまで、Console ログインで API キーを作成すると失敗します。ログインを Keychain に戻すには、復旧手順に従ってください。
  - Linux では、認証情報は `~/.claude/.credentials.json` に保存され、ファイルモードは `0600` です。
  - Windows では、認証情報は `%USERPROFILE%\.claude\.credentials.json` に保存され、ユーザープロファイルディレクトリのアクセス制御を継承します。これにより、ファイルはデフォルトでユーザーアカウントに制限されます。
  - `CLAUDE_CONFIG_DIR` 環境変数を設定している場合、Claude Code は `.credentials.json` ファイルをそのディレクトリの下に保存します。これには macOS フォールバックが書き込むファイルも含まれ、macOS Keychain エントリもそのディレクトリをキーとします。そのため、異なる `CLAUDE_CONFIG_DIR` を持つセッションは異なるエントリを読み込みます。
  - Claude Code は `/login` と `/logout` を通じて `.credentials.json` を管理します。リクエストをカスタム API エンドポイント経由でルーティングするには、代わりに `ANTHROPIC_BASE_URL` 環境変数を設定してください。
- **サポートされている認証タイプ**: claude.ai 認証情報、Claude API 認証情報、Microsoft Foundry Auth、Bedrock Auth、Vertex Auth、Anthropic プロファイルおよび Workload Identity Federation 認証情報、および Claude apps gateway セッショントークン。
- **カスタム認証情報スクリプト**: `apiKeyHelper` 設定を構成して、API キーを返すシェルスクリプトを実行します。
- **更新間隔**: `apiKeyHelper` の場合を参照してください。Claude Code はヘルパーを再実行する場合があります。
- **遅いヘルパー通知**: `apiKeyHelper` がキーを返すのに 10 秒以上かかる場合、Claude Code はプロンプトバーに経過時間を表示する警告通知を表示します。この通知が定期的に表示される場合は、認証情報スクリプトを最適化できるかどうかを確認してください。
- **ヘルパーの失敗**: スクリプトがエラーで終了したり、タイムアウトしたり、何も出力しない場合、リクエストは 3 回の試行内に `Your apiKeyHelper script is failing` で失敗します。

`apiKeyHelper`、`ANTHROPIC_API_KEY`、および `ANTHROPIC_AUTH_TOKEN` は CLI およびそれをラップするサーフェス（VS Code 拡張機能、Agent SDK、GitHub Actions を含む）に適用されます。Claude Desktop とクラウドセッションは `apiKeyHelper` を呼び出したり、これらの環境変数を読み込んだりしません。OAuth を使用します。ただし、サードパーティ推論設定を実行しているデスクトップセッションは、その設定の認証情報で認証します。

35.3.1 期限切れ間近のログインを更新する

`/login` で作成したログインが期限切れまで 3 日以内になると、Claude Code はスタートアップ時に警告を表示します。`Your login expires in 3 days · run /login to renew`。

`/login` を実行して更新します。警告は情報提供のみであり、リクエストをブロックすることはありません。ログインが実際に期限切れになるまで認証は機能し続けます。

保存されたログインが期限切れになり、更新できなくなると、再度サインインするまで、各モデルリクエストは `Login expired · Please run /login` で失敗します。

リクエストが失敗する前にこの状態を確認できます。`/status` は `Login` 行に `Expired — log in again` を表示し、期限切れのログインに保存されている組織とメールアドレスを表示します。この行は、保存されている claude.ai ログインがアクティブな認証情報である場合にのみ表示されます。この行には Claude Code v2.1.210 以降が必要です。

警告は claude.ai ログインがアクティブな認証情報である場合にのみ表示され、クラウドプロバイダー、`ANTHROPIC_API_KEY`、`ANTHROPIC_AUTH_TOKEN`、または `apiKeyHelper` が認証情報を提供する場合には表示されません。

更新を早期に行うことは、無人で実行されるセッションにとって最も重要です。agent view のバックグラウンドセッションまたは Remote Control セッションがログインより長く実行される場合、認証情報が期限切れになると進行が停止し、再度サインインするまで復旧できません。

35.3.2 認証の優先順位

複数の認証情報が存在する場合、Claude Code は以下の順序で 1 つを選択します。

1. `CLAUDE_CODE_USE_BEDROCK`、`CLAUDE_CODE_USE_VERTEX`、または `CLAUDE_CODE_USE_FOUNDRY` が設定されている場合のクラウドプロバイダー認証情報。セットアップについては、**サードパーティ統合** (ページ 847) を参照してください。

2. `ANTHROPIC_AUTH_TOKEN` 環境変数。`Authorization: Bearer` ヘッダーとして送信されます。Anthropic API キーではなくベアラートークンで認証する *LLM* **ゲートウェイまたはプロキシ** (ページ 841) を通じてルーティングする場合に使用します。

3. `ANTHROPIC_API_KEY` 環境変数。`X-Api-Key` ヘッダーとして送信されます。Claude Console からのキーを使用して Anthropic API に直接アクセスする場合に使用します。対話モードでは、キーを承認または拒否するよう 1 回プロンプトが表示され、選択が記憶されます。後で変更するには、`/config` の「Use custom API key」トグルを使用します。トグルは `ANTHROPIC_API_KEY` が環境に設定されている間のみ表示されます。非対話モード（`-p`）では、キーが存在する場合は常に使用されます。

4. `apiKeyHelper` スクリプト出力。動的または回転する認証情報（ボルトから取得した短期トークンなど）に使用します。

5. `CLAUDE_CODE_OAUTH_TOKEN` 環境変数。*`claude setup-token`* (ページ 860) によって生成された長期 OAuth トークン。ブラウザログインが利用できない CI パイプラインとスクリプトに使用します。`CLAUDE_CODE_OAUTH_TOKEN` が設定されている間に `/login` を実行すると、Claude Code は現在のセッションを新しいログインに切り替えますが、新しいセッションのたびに変数を再度読み込みます。シェルプロファイルまたは **設定ファイル** (ページ 711) の `env` ブロックから変数を削除するまで。

6. Anthropic プロファイルおよびフェデレーション認証情報。`ant` CLI および Workload Identity Federation が使用する認証情報。`ant auth login` が書き込んだプロファイルは、`ANTHROPIC_PROFILE` で名前を付けた場合にのみここにランク付けされます。そうでない場合は `/login` より下にランク付けされます。*Anthropic* **プロファイルおよびフェデレーション認証情報** (ページ 859) を参照してください。
7. `/login` からのサブスクリプション OAuth 認証情報。これは Claude Pro、Max、Team、および Enterprise ユーザーのデフォルトです。

署名済みの Claude apps gateway セッションはこのリストの外に位置します。これは Amazon Bedrock または Google Cloud の Agent Platform のようなプロバイダー選択であり、それらより優先されます。ゲートウェイセッションが存在する場合、CLI は `CLAUDE_CODE_USE_BEDROCK`、`CLAUDE_CODE_USE_VERTEX`、または `CLAUDE_CODE_USE_FOUNDRY` が設定されていても、ゲートウェイトークンで認証され、ベアラートークン、API キー、`apiKeyHelper`、およびプロファイルなどの上記の認証情報ソースは使用されません。

マシンの 管理設定が `forceLoginMethod` を `"gateway"` に設定するか、`forceLoginGatewayUrl` を設定し、`CLAUDE_CODE_USE_BEDROCK` または `CLAUDE_CODE_USE_VERTEX` などの変数を通じてクラウドプロバイダーを選択しない場合、セッションはゲートウェイサインインのみを使用します。Claude Code は他の認証情報ソースをスキップし、`/login` でサインインするよう求めます。残りの各認証情報で表示される内容については、Administrator policy requires a Cloud gateway sign-in を参照してください。Claude Code v2.1.261 以降、またはゲートウェイサインインのみを設定するマシンの v2.1.265 以降が必要です。

アクティブな Claude サブスクリプションがあり、環境に `ANTHROPIC_API_KEY` も設定されている場合、API キーは承認されると優先されます。キーが無効または期限切れの組織に属している場合、これは認証エラーを引き起こす可能性があります。

`unset ANTHROPIC_API_KEY` を実行してサブスクリプションにフォールバックし、`/status` をチェックしてどの方法がアクティブであるかを確認します。ログインと API キーの両方が設定されている場合、`/status` は使用中でない認証情報をマークします。

Cloud sessions (ページ 223) は常にサブスクリプション認証情報を使用します。クラウド環境で `ANTHROPIC_API_KEY` または `ANTHROPIC_AUTH_TOKEN` を設定しても、サブスクリプション認証情報はオーバーライドされません。

Anthropic プロファイルおよびフェデレーション認証情報

プロファイルは、Anthropic 設定ディレクトリ内の名前付き認証情報設定ファイルです。デフォルトでは macOS および Linux では `~/.config/anthropic`、Windows では `%APPDATA%\Anthropic` です。プロファイルの認証モードは、Workload Identity Federation（WIF）用に設定した場合は `oidc_federation`、または `ant auth login` が書き込んだ場合、または *API* **キーなしで** *Console* **アカウントにサインイン** (ページ 853) した場合は `user_oauth` です。

Claude Code は **ベアモード** (ページ 696)、Claude Desktop、またはクラウドセッションではプロファイルまたはフェデレーション変数を読み込みません。これらのセッションでは、`/status` は `Profile` 行を表示しません。

Claude Code は 3 つのソースをこの順序でチェックし、設定されている最初のソースで停止します。表は各ソースを設定するものと、`/login` 認証情報に対するランク付けを示しています。

ソース	設定者	/login に対するランク
名前付きプロファイル	ANTHROPIC_PROFILE	上、プロファイルが持つ認証モードに関係なく
フェデレーション変数	ANTHROPIC_FEDERATION_RULE_ID および ANTHROPIC_ORGANIZATION_ID、両方設定	上
アクティブプロファイル	設定ディレクトリ内の active_config ファイル、または default という名前のプロファイル	認証モードが oidc_federation の場合は上。認証モードが user_oauth の場合は、機能している /login 認証情報より下

フェデレーション変数の場合、Claude Code は ID トークンを交換する際に WIF リファレンスの ANTHROPIC_IDENTITY_TOKEN_FILE などの他の変数も読み込みます。プロファイルファイル形式については、WIF リファレンスを参照してください。

Claude Code が選択したソースを確認するには、/status を実行してください。Profile 行は Login method 行の代わりにソースを名前で表示し、プロファイルが使用中の認証情報である場合、Organization および Email 行はそのアカウントを表示します。

user_oauth プロファイルのログインが期限切れになり、Claude Code がそれを更新できない場合、リクエストは Anthropic profile login expired で失敗します。

claude.ai **コネクタ** (ページ 396) および /schedule などの claude.ai ログインが必要な機能は、これらのソースの 1 つが選択されている間は利用できません。Claude Code がソースを選択するのを停止するには、以下を実行してください。

- **名前付きプロファイルまたはフェデレーション変数**: ANTHROPIC_PROFILE をアンセットするか、フェデレーション変数のいずれかをアンセットします
- **アクティブプロファイル**: *API* **キーなしで** *Console* **アカウントにサインイン** (ページ 853) して現在の認証情報を書き込んだ user_oauth プロファイルの場合は /logout を実行し、ant auth login が現在の認証情報を書き込んだプロファイルの場合は ant auth logout を実行し、どちらの認証モードでも設定ディレクトリの configs/ からプロファイルのファイルを削除します

35.3.3 長期トークンを生成する

CI パイプライン、スクリプト、または対話的なブラウザログインが利用できない他の環境の場合、claude setup-token で 1 年間の OAuth トークンを生成します。

```
claude setup-token
```

このコマンドは /login と同じブラウザ認可フローを開き、ブラウザでアクセスを承認した後、トークンはターミナルに出力されます。トークンはどこにも保存されません。トークンをコピーして、認証したい場所で CLAUDE_CODE_OAUTH_TOKEN 環境変数として設定します。

macOS, Linux, WSL

```
export CLAUDE_CODE_OAUTH_TOKEN=your-token
```

Windows PowerShell

```
$env:CLAUDE_CODE_OAUTH_TOKEN = "your-token"
```

Windows CMD

```
set CLAUDE_CODE_OAUTH_TOKEN=your-token
```

このトークンは Claude サブスクリプションで認証され、Pro、Max、Team、または Enterprise プランが必要です。モデルリクエストのみを実行できるため、Remote Control セッションを確立したり、*claude.ai* **コネクタ** (ページ 396) をフェッチしたりすることはできません。ローカルで設定した MCP サーバーは引き続き機能します。

ベアモード (ページ 696) は `CLAUDE_CODE_OAUTH_TOKEN` を読み込みません。スクリプトが `--bare` を渡す場合は、`ANTHROPIC_API_KEY` または `apiKeyHelper` で認証します。

第36章　Amazon Bedrock 上の Claude Code

Amazon Bedrock を通じた Claude Code の設定方法（セットアップ、IAM 設定、トラブルシューティングを含む）について学習します。

36.1 前提条件

Claude Code を Amazon Bedrock で設定する前に、以下を確認してください。

- Amazon Bedrock アクセスが有効になっている AWS アカウント
- Amazon Bedrock で目的の Claude モデル（例：Claude Sonnet 4.6）へのアクセス
- AWS CLI がインストールされ、設定されていること（オプション - 認証情報を取得する別のメカニズムがない場合のみ必要）
- 適切な IAM 権限

Amazon Bedrock 認証情報を使用してサインインするには、以下の *Amazon Bedrock* **でサインイン** (ページ 863) に従ってください。チーム全体に Claude Code をデプロイするには、**手動でセットアップ** (ページ 864) の手順を使用し、ロールアウト前に**モデルバージョンをピン留め** (ページ 868) してください。

36.2 Bedrock でサインイン

AWS 認証情報を持っていて、Amazon Bedrock を通じて Claude Code の使用を開始したい場合、ログインウィザードがその手順を案内します。AWS 側の前提条件はアカウントごとに 1 回完了します。ウィザードが Claude Code 側を処理します。

1. **AWS アカウントで Anthropic モデルを有効にする**

   Amazon Bedrock コンソールで、モデルカタログを開き、Anthropic モデルを選択して、ユースケースフォームを送信します。送信直後にアクセスが許可されます。AWS Organizations については**ユースケースの詳細を送信** (ページ 864) を、権限については *IAM* **設定** (ページ 873) を参照してください。

2. **Claude Code を起動して Amazon Bedrock を選択する**

   `claude` を実行します。ログインプロンプトで、**3rd-party platform** を選択してから、**Amazon Bedrock** を選択します。既にサインインしていてチャットプロンプトが表示されている場合は、`/setup-bedrock` を実行してウィザードを開きます。`CLAUDE_CODE_USE_BEDROCK=1` が設定されるまで、Claude Code はコマンドメニューからコマンドを非表示にします。完全に入力してください。

3. **ウィザードプロンプトに従う**

   AWS への認証方法を選択します。`~/.aws` ディレクトリから検出された AWS プロファイル、Amazon Bedrock API キー、アクセスキーとシークレット、または環境に既に存在する認証情報です。ウィ

ザードはリージョンを要求し、アカウントが呼び出せる Claude モデルを確認し、それらをピン留めできます。結果は**ユーザー設定ファイル** (ページ 711) の env ブロックに保存されるため、環境変数を自分でエクスポートする必要はありません。

サインイン後、いつでも `/setup-bedrock` を実行してウィザードを再度開き、認証情報、リージョン、またはモデルピンを変更できます。モデルピンステップは、現在ピン留めされているモデルから開始します。ウィザードは `~/.claude/settings.json` に書き込むか、`CLAUDE_CONFIG_DIR` が設定されている場合は `$CLAUDE_CONFIG_DIR/settings.json` に書き込みます。

36.3 手動でセットアップする

ウィザードの代わりに環境変数を使用して Amazon Bedrock を設定する場合（例えば CI やスクリプト化されたエンタープライズロールアウト）、以下の手順に従ってください。

36.3.1 1. ユースケースの詳細を送信する

Anthropic モデルを初めて呼び出す前に、ユースケースの詳細を送信してください。これは AWS アカウントごとに 1 回行います。

1. 以下に説明する適切な IAM 権限があることを確認してください
2. Amazon Bedrock コンソールに移動します
3. **Model catalog** から Anthropic モデルを選択します
4. ユースケースフォームを完成させます。アクセスは送信直後に付与されます。

AWS Organizations を使用している場合、PutUseCaseForModelAccess API を使用して管理アカウントからフォームを 1 回送信できます。この呼び出しには `bedrock:PutUseCaseForModelAccess` IAM 権限が必要です。承認は自動的に子アカウントに拡張されます。

36.3.2 2. AWS 認証情報を設定する

Claude Code は AWS SDK のデフォルト認証情報チェーンを使用します。以下のいずれかの方法を使用して認証情報を設定してください。

オプション A: AWS CLI 設定

```
aws configure
```

オプション B: 環境変数（アクセスキー）

```
export AWS_ACCESS_KEY_ID=your-access-key-id
export AWS_SECRET_ACCESS_KEY=your-secret-access-key
export AWS_SESSION_TOKEN=your-session-token
```

オプション C: 環境変数（SSO プロファイル）

これらのコマンドを実行する前に、`your-profile-name` を AWS プロファイルの名前に置き換えてください。

```
aws sso login --profile=your-profile-name

export AWS_PROFILE=your-profile-name
```

Claude Code は、プロファイルの `sso_region` で指定された IAM Identity Center リージョンからロール認証情報をリクエストします。これは Amazon Bedrock を実行するリージョンと一致する必要はありません。v2.1.207 では、Amazon Bedrock リージョンが `sso_region` をオーバーライドしていたため、IAM Identity Center インスタンスが別のリージョンにあるプロファイルは `Session token not found or invalid` エラーで認証に失敗しました。

オプション D: AWS Management Console 認証情報

```
aws login
```

`aws login` について詳しく学ぶ。

オプション E: Amazon Bedrock API キー

```
export AWS_BEARER_TOKEN_BEDROCK=your-bedrock-api-key
```

Amazon Bedrock API キーは、完全な AWS 認証情報を必要としない、より簡単な認証方法を提供します。Amazon Bedrock API キーについて詳しく学ぶ。

認証情報キャッシングと解決タイムアウト

Claude Code は AWS デフォルト認証情報プロバイダーチェーンを 1 回解決し、解決された認証情報をメモリに保持します。認証情報の有効期限の 5 分前まで、または有効期限がない場合は 1 時間の間、それらを再利用するため、SSO でサポートされたプロファイルは認証情報の有効期間ごとに約 1 回 IAM Identity Center から認証情報をリクエストします。API からの認証情報エラーはキャッシュをクリアし、再試行は新しい認証情報を解決します。Claude Code v2.1.207 以降が必要です。

キャッシュは上記のすべての認証情報オプションをカバーしていますが、Amazon Bedrock API キーはプロバイダーチェーンを使用しないため除外されます。代わりにすべてのリクエストでチェーンを解決するには、`CLAUDE_CODE_SKIP_AWS_CRED_CACHE=1` を設定してください。

チェーンの各解決は 60 秒後にタイムアウトします。チェーン内のステップが停止した場合（例えば、受け取ることができない入力を待つ `credential_process` ヘルパー）、リクエストは `AWS default-chain credential resolve timed out` で失敗します。チェーンが正当に長い時間が必要なインタラクティブサインイン（`aws-vault` のようなラッパーを使用した MFA 付きブラウザベースの SSO など）を実行する場合、`CLAUDE_CODE_AWS_CHAIN_RESOLVE_TIMEOUT_MS` でミリ秒単位で制限を引き上げてください。v2.1.207 より前では、停止した認証情報解決はリクエストを無期限に待機させていました。

Amazon Bedrock API キーで認証する場合を除き、**セットアップウィザード** (ページ 863) は認証情報を検証する際に行う各 AWS 呼び出しに同じ制限を適用し、各モデルチェック前の認証情報ルックアップにも適用します。認証情報検証中に、制限を超えるチェックは `Timed out after 60s waiting for AWS` で失敗します。

高度な認証情報設定

Claude Code は AWS SSO および企業 ID プロバイダーの自動認証情報更新をサポートしています。これらの設定を Claude Code 設定ファイルに追加してください（ファイルの場所については *Settings* (ページ 711) を参照）。

これら 2 つの設定には異なるトリガー条件があります。

- `awsAuthRefresh`: Claude Code がタイムスタンプに基づいてローカルで AWS 認証情報が期限切れであることを検出した場合、または API が認証情報エラーを返した場合にのみ実行され、更新された認証情報でリクエストを再試行します。
- `awsCredentialExport`: セッション開始時および各認証情報リロード時に実行されます。AWS デフォルト認証情報プロバイダーチェーン内の認証情報がまだ有効な場合でも実行されます。Amazon Bedrock アカウントがデフォルトプロバイダーチェーンが解決するものと異なるクロスアカウント認証情報を必要とする場合に使用してください。

`awsAuthRefresh` コマンドを実行する前に、Claude Code は STS `GetCallerIdentity` 呼び出しを行って、認証情報が実際に期限切れであることを確認し、認証情報がまだ機能している場合はコマンドをスキップします。Claude Code はこのチェックを **プロキシ設定** (ページ 897) を通じて送信し、`HTTPS_PROXY` と `NO_PROXY` を尊重します。v2.1.239 より前では、Claude Code はこのチェックを直接送信し、プロキシを通じてのみエグレスを許可するネットワークでスタートアップでハングしていました。

設定例

```
{
  "awsAuthRefresh": "aws sso login --profile myprofile",
  "env": {
    "AWS_PROFILE": "myprofile"
  }
}
```

設定の説明

`awsAuthRefresh`: `.aws` ディレクトリを変更するコマンド（認証情報、SSO キャッシュ、または設定ファイルの更新など）に使用してください。コマンドの出力はユーザーに表示されますが、インタラクティブ入力はサポートされていません。これは CLI が URL またはコードを表示し、ブラウザで認証を完了するブラウザベースの SSO フローに適しています。

`awsCredentialExport`: `.aws` を変更できず、認証情報を直接返す必要がある場合にのみ使用してください。出力はサイレントにキャプチャされ、ユーザーに表示されません。コマンドは次の形式で JSON を出力する必要があります。

```
{
  "Credentials": {
    "AccessKeyId": "value",
    "SecretAccessKey": "value",
    "SessionToken": "value",
    "Expiration": "2026-01-01T00:00:00Z"
```

(次のページに続く)

(前のページからの続き)

```
  }
}
```

`aws configure export-credentials --format process` からのフラット出力も受け入れられます。同じキーが `Credentials` の下にネストされるのではなく、トップレベルにあります。

`Expiration` はオプションです。コマンドが有効な ISO 8601 `Expiration` を返すと、Claude Code はその時刻の 5 分前まで認証情報をキャッシュします。それがない場合、認証情報は 1 時間キャッシュされます。

`awsCredentialExport` を `awsAuthRefresh` なしで設定する場合、Claude Code はエクスポートされた認証情報を直接使用し、スタートアップで AWS デフォルト認証情報プロバイダーチェーンを再解決しません。Claude Code v2.1.206 以降が必要です。

36.3.3 3. Claude Code を設定する

Amazon Bedrock を有効にするには、以下の環境変数を設定してください。

```
# Bedrock 統合を有効にする
export CLAUDE_CODE_USE_BEDROCK=1
export AWS_REGION=us-east-1  # AWS プロファイルがすでにリージョンを設定している場合はオプション

# オプション: 小型/高速モデル（Bedrock と Mantle）の AWS リージョンをオーバーライドします。
# Bedrock では ANTHROPIC_DEFAULT_HAIKU_MODEL
# または非推奨の ANTHROPIC_SMALL_FAST_MODEL が設定されていない場合、効果がありません。
export ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION=us-west-2

# オプション: カスタムエンドポイントまたはゲートウェイ用に Bedrock エンドポイント URL をオーバーライドします
# export ANTHROPIC_BEDROCK_BASE_URL=https://bedrock-runtime.us-east-1.amazonaws.com
```

Claude Code で Amazon Bedrock を有効にする場合、以下の点に注意してください。

- `AWS_REGION` を設定する必要があるのは、AWS プロファイルのリージョンをオーバーライドする場合、またはプロファイルにリージョンがない場合のみです。Claude Code はこの順序でリージョンを解決します。
  - `AWS_REGION`
  - `AWS_DEFAULT_REGION`
  - AWS 共有認証情報ファイルから最初に読み込まれ、次に共有設定ファイルから読み込まれるアクティブな AWS プロファイルに設定された `region`（AWS SDK の優先順位と一致）
  - `us-east-1`

  これらのソースからの値がリージョン名のような形をしていない場合、Claude Code はそれを未設定として扱い、順序を続けます。例えば、Claude Code はスラッシュ、ドット、またはスペースを含む値を未設定として扱います。

アクティブなプロファイルは、設定されている場合は AWS_PROFILE、そうでない場合は default です。AWS_SHARED_CREDENTIALS_FILE または AWS_CONFIG_FILE を設定して、デフォルト以外のファイルパスを指定してください。

/status を実行して、解決されたリージョンを確認してください。リージョンが AWS 設定ファイルまたはデフォルトフォールバックから来た場合、Claude Code は /status 出力でソースも記載します。

- Amazon Bedrock を使用する場合、認証は AWS 認証情報を通じて処理されるため、/logout コマンドは利用できません。
- WebSearch ツールは Amazon Bedrock では利用できません。WebSearch ツールの動作を参照してください。
- AWS_PROFILE のような他のプロセスにリークしたくない環境変数に設定ファイルを使用できます。詳細については *Settings* (ページ 711) を参照してください。

36.3.4 4. モデルバージョンをピンする

警告:

複数のユーザーにデプロイする場合、特定のモデルバージョンをピンしてください。ピンしない場合、sonnet や opus などのモデルエイリアスは Claude Code の Amazon Bedrock 用の組み込みデフォルトに解決されます。これは最新リリースより遅れる可能性があり、アカウントでまだ利用できない場合があります。Claude Code はスタートアップ時にデフォルトが利用できない場合、**以前のモデルまたは下位層のモデルにフォールバック** (ページ 871) しますが、ピンすることでユーザーが新しいモデルに移行するタイミングを制御できます。

これらの環境変数を特定の Amazon Bedrock モデル ID に設定してください。

ANTHROPIC_DEFAULT_OPUS_MODEL がない場合、Amazon Bedrock の opus エイリアスは Opus 5.5 に解決され、ANTHROPIC_DEFAULT_SONNET_MODEL がない場合、sonnet エイリアスは Sonnet 4.5 に解決されます。この例では各エイリアスを特定のバージョンにピンしています。

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-8'
export ANTHROPIC_DEFAULT_SONNET_MODEL='us.anthropic.claude-sonnet-4-6'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'
```

これらの ID は us. クロスリージョン推論プロファイルプレフィックスを使用しています。別のリージョンプレフィックスまたはアプリケーション推論プロファイルを使用する場合は、それに応じて調整してください。AWS GovCloud リージョンでは、us-gov. プレフィックスを使用してください。

組み込みデフォルトモデルを保持し、優先プレフィックスのみを変更するには、ピンの代わりに *ANTHROPIC_BEDROCK_REGION_PREFIX* (ページ 872) を設定してください。opus エイリアスが解決する内容の違いを示します。

設定内容	opus エイリアスが解決する内容
ANTHROPIC_DEFAULT_OPUS_MODEL='us.anthropic.claude-opus-4-8'	us.anthropic.claude-opus-4-8、ピンした正確な ID

次のページに続く

表 1 – 前のページからの続き

設定内容	opus エイリアスが解決する内容
ANTHROPIC_BEDROCK_REGION_PREFIX=eu	eu.anthropic.claude-opus-5-5、優先プレフィックス付きの組み込みデフォルト

現在および従来のモデル ID については、Models overview を参照してください。ピン環境変数の完全なリストについては、*Model configuration* (ページ 758) を参照してください。

ピン変数が設定されていない場合、Claude Code はこれらのデフォルトモデルを使用します。

モデルタイプ	デフォルトモデル
プライマリモデル	Opus 5.5、例えば us-* リージョンの us.anthropic.claude-opus-5-5
小型/高速モデル	Sonnet 4.5、例えば us-* リージョンの us.anthropic.claude-sonnet-4-5-20250929-v1:0

セッションタイトル生成などのバックグラウンドタスクは、小型/高速モデル（通常は Haiku クラスモデル）を使用します。Amazon Bedrock では、すべてのアカウントまたはリージョンで Haiku が有効になっていない可能性があるため、Claude Code はバックグラウンドタスクにデフォルト Sonnet モデルを使用します。2 つの選択がどのモデルがそれらを実行するかを変更します。

- --model、ANTHROPIC_MODEL、または model 設定でプライマリモデルを選択する場合、バックグラウンドタスクはそのモデルを使用します。Claude Code が *ANTHROPIC_DEFAULT_MODEL* (ページ 731) で設定したモデルでセッションを開始する場合、バックグラウンドタスクもそのモデルを使用します。ANTHROPIC_DEFAULT_SONNET_MODEL なしで ANTHROPIC_DEFAULT_OPUS_MODEL を設定することも、組み込み Sonnet モデルが独自の Opus を操舵するアカウントで有効になっていない可能性があるため、選択としてカウントされます。
- バックグラウンドタスクに Haiku を使用するには、ANTHROPIC_DEFAULT_HAIKU_MODEL をアカウントで利用可能なモデル ID に設定してください。

警告:

Opus モデルは Sonnet モデルより高いトークンあたりの価格を持つため、プライマリモデルをピンしないデプロイメントは v2.1.207 以降に更新されると Opus レートで請求されます。Sonnet 4.5 をプライマリモデルとして保持するには、ANTHROPIC_MODEL をその完全なモデル ID に設定してください。ANTHROPIC_DEFAULT_SONNET_MODEL で操舵され、ANTHROPIC_DEFAULT_OPUS_MODEL を設定しないデプロイメントは、操舵された Sonnet モデルをデフォルトとして保持します。

v2.1.280 より前では、Amazon Bedrock のプライマリモデルは Opus 5 にデフォルト設定され、opus エイリアスは v2.1.219 から Opus 5 に解決されました。v2.1.207 から v2.1.218 では、Amazon Bedrock のプライマリモデルは Opus 4.8 にデフォルト設定され、opus エイリアスは Opus 4.8 に解決されました。v2.1.207 より前では、プライマリモデルは Sonnet 4.5 にデフォルト設定され、opus エイリアスは Opus 4.6 に解決され、バックグラウンドタスクは常にプライマリモデルを使用していました。

モデルをさらにカスタマイズするには、以下のいずれかの方法を使用してください。

```
# 推論プロファイル ID を使用する
export ANTHROPIC_MODEL='us.anthropic.claude-sonnet-4-6'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='us.anthropic.claude-haiku-4-5-20251001-v1:0'

# アプリケーション推論プロファイル ARN を使用する
export ANTHROPIC_MODEL='arn:aws:bedrock:us-east-2:your-account-id:application-
↪inference-profile/your-model-id'

# オプション: 必要に応じてプロンプトキャッシングを無効にする
# export DISABLE_PROMPT_CACHING=1

# オプション: デフォルトの 5 分の代わりに 1 時間のプロンプトキャッシュ TTL をリクエストする
# export ENABLE_PROMPT_CACHING_1H=1
```

1 時間のキャッシュ TTL はデフォルトの 5 分より高いレートで請求されます。キャッシュの有効期間を参照してください。メイン会話と Claude Code が外部で行うリクエストに異なる TTL を設定するには、TTL を自分で選択してください。

注釈:

プロンプトキャッシングはすべての Amazon Bedrock リージョンで利用できない場合があります。キャッシュトークンカウントがゼロのままの場合、Amazon Bedrock ドキュメントの サポートされているモデル、リージョン、および制限を確認してください。

各モデルバージョンを推論プロファイルにマップする

`ANTHROPIC_DEFAULT_*_MODEL` 環境変数は、モデルファミリーごとに 1 つの推論プロファイルを設定します。組織が同じファミリーの複数のバージョンを `/model` ピッカーで公開し、それぞれを独自のアプリケーション推論プロファイル ARN にルーティングする必要がある場合は、**設定ファイル** (ページ 711) の `modelOverrides` 設定を使用してください。

この例は 4 つの Opus バージョンを異なる ARN にマップするため、ユーザーは組織の推論プロファイルをバイパスすることなくそれらを切り替えることができます。

```
{
  "modelOverrides": {
    "claude-opus-4-7": "arn:aws:bedrock:us-east-2:123456789012:application-inference-
↪profile/opus-47-prod",
    "claude-opus-4-6": "arn:aws:bedrock:us-east-2:123456789012:application-inference-
↪profile/opus-46-prod",
    "claude-opus-4-5-20251101": "arn:aws:bedrock:us-east-2:123456789012:application-
↪inference-profile/opus-45-prod",
    "claude-opus-4-1-20250805": "arn:aws:bedrock:us-east-2:123456789012:application-
↪inference-profile/opus-41-prod"
  }
}
```

ユーザーが `/model` でこれらのバージョンのいずれかを選択すると、Claude Code はマップされた ARN で Amazon Bedrock を呼び出します。同じマッピングは `--model` または `ANTHROPIC_MODEL` を通じて Anthropic モデル ID を直接渡す場合に適用されます。オーバーライドのないバージョンは、組み込み Amazon Bedrock モデル ID またはスタートアップで検出された一致する推論プロファイルにフォールバックします。v2.1.200 より前では、`--model` と `ANTHROPIC_MODEL` の値はオーバーライドマップを通さずに Amazon Bedrock に到達していました。オーバーライドが `availableModels` および他のモデル設定とどのように相互作用するかについては、*Override model IDs per version* (ページ 761) を参照してください。

36.4 スタートアップモデルチェック

Claude Code が Amazon Bedrock で設定されて起動する場合、使用予定のモデルがアカウントでアクセス可能であることを確認します。

現在の Claude Code デフォルトより古いモデルバージョンをピン留めしており、アカウントが新しいバージョンを呼び出せる場合、Claude Code はピンを更新するよう促します。受け入れると、新しいモデル ID が**ユーザー設定ファイル** (ページ 711) に書き込まれ、Claude Code が再起動します。拒否すると、次のデフォルトバージョン変更まで記憶されます。**アプリケーション推論プロファイル** *ARN* (ページ 870) を指す pin は、管理者によって管理されるため、スキップされます。

モデルをピン留めしておらず、現在のデフォルトがアカウントで利用できない場合、Claude Code は現在のセッションにフォールバックし、通知を表示します。デフォルトモデルの以前のバージョンを最初に試し、デフォルトが Opus モデルで Opus バージョンが利用できない場合、デフォルト Sonnet モデルにフォールバックします。フォールバックは保持されません。Amazon Bedrock アカウントで新しいモデルを有効にするか、**バージョンをピン留めして** (ページ 868) 選択を永続的にしてください。

特定の Sonnet または Opus バージョンでセッションを開始する場合（例えば `--model`、`ANTHROPIC_MODEL`、または model 設定を使用）、そのバージョンは一致する `sonnet` または `opus` エイリアスのセッションのピン留めされたデフォルトとして機能します。Claude Code は、設定したモデルが置き換える組み込みデフォルトの可用性チェックをスキップし、設定したモデルで起動し、フォールバック通知はありません。

`opus` などのモデルエイリアスはピンとして機能せず、Claude Code が認識しないモデル ID（アプリケーション推論プロファイル ARN など）も同様です。

これらのチェックがアカウントが呼び出せないモデルを見つけた場合、Claude Code はこのマシンで最大 1 日間その拒否を記憶し、その時間中は Amazon Bedrock に再度問い合わせることなく記憶されたモデルをスキップして起動します。Claude Code は、現在のデフォルトモデルの記憶された拒否を、最後のチェック以降 10 分が経過すると起動時に再度チェックするため、管理者が再度有効にしたデフォルトが戻ります。メモリをオフにするには、`CLAUDE_CODE_SKIP_MODEL_ACCESS_MEMORY=1` を設定してください。

36.4.1 モデルがセッション中に無効化される場合

セッションが実行されているモデルへのアカウントアクセスが失われた場合（例えば、管理者が Amazon Bedrock アカウントでそれを無効化した場合）、Claude Code は各リクエストが失敗する代わりにセッションを別のモデルに切り替え、`Switched to <fallback> because <model> is not available` を表示します。スタートアップフォールバックと同じモデルを試します。同じティアの以前のバージョンを最初に試し、Opus セッションで Opus バージョンが利用できない場合、デフォルト Sonnet モデルを試します。

切り替えは、ピン留めしていないティアにのみ適用されます。これはスタートアップフォールバックと同じ条件です。選択した特定のバージョン、または**アプリケーション推論プロファイル** *ARN* (ページ 870) で

セッションを実行している場合、そのモデルを保持し、フォールバックモデルチェーンがないため、リクエストは失敗します。自動モードでは、Claude Code は Amazon Bedrock で自動モードがサポートするモデルにのみ切り替えます。それらのモデルも利用できない場合、リクエストは AWS 認証失敗で失敗し、モデルを有効にするためのヒントが表示されます。

設定した**フォールバックモデルチェーン** (ページ 743) はティア切り替えを置き換えます。これらの拒否では Claude Code は設定したフォールバックに切り替えます。拒否されたリクエストが切り替わるのではなく失敗するようにするには、`CLAUDE_CODE_DISABLE_MODEL_ACCESS_FALLBACK=1` を設定してください。設定したフォールバックチェーンはこれらの拒否で切り替わります。すべての拒否されたリクエストが失敗するようにしたい場合は、チェーンも削除してください。

36.5 クロスリージョン推論プロファイルプレフィックス

Amazon Bedrock の Invoke API では、Claude Code は組み込みのデフォルトモデルを クロスリージョン推論プロファイル ID に解決します。独自の推論プロファイルを通じてモデルバージョンをルーティングするには、**各モデルバージョンを推論プロファイルにマップする** (ページ 870) を参照してください。このテーブルは、解決された各 AWS リージョンに対して Claude Code が優先するプレフィックスを示しています。

AWS リージョン	プレフィックス
`us-gov-*` (AWS GovCloud)	`us-gov.`
`us-*`	`us.`
`eu-*`	`eu.`
`ap-*`	`apac.`
その他すべてのリージョン	`global.`

`ANTHROPIC_BEDROCK_REGION_PREFIX` を設定して、Claude Code が最初に試すプレフィックスを選択します。Claude Code がプロファイルの可用性を確認でき、モデルに一致するプロファイルが見つからない場合、以下の解決順序に従ってフォールバックします。有効な値は `us`、`eu`、`apac`、`jp`、`au`、および `global` です。たとえば、アカウントに `global.` プロファイルが有効になっているが、Claude Code が AWS リージョンから地理的に固有のプロファイルを導出する場合は、`global` に設定します。Claude Code v2.1.224 以降が必要です。

この例は、デフォルトモデルを `global.` プロファイルを通じてルーティングします。

```
export ANTHROPIC_BEDROCK_REGION_PREFIX=global
# us-* リージョンでは、プライマリモデルは
# us.anthropic.claude-opus-5-5 の代わりに
# global.anthropic.claude-opus-5-5 に解決されます
```

優先プレフィックスは保証ではなく、リージョンから来ているか変数から来ているかに関わらず、単なる優先度です。Claude Code がそれを適用する方法は、アカウント内のプロファイル可用性を確認できるかどうかによって異なります。

- Claude Code がアカウント内の **推論プロファイルをリストできる** (ページ 873) 場合、各モデルを次の順序で解決します。
    1. 優先プレフィックスを持つプロファイル。

2. そのプレフィックスを持つプロファイルがないモデルの場合、一致するプロファイル。
3. 一致するプロファイルがないモデルの場合、優先プレフィックスを持つ組み込みモデル ID。Claude Code はこのステップで可用性を確認せずにこの ID を適用します。**スタートアップモデルチェック** (ページ 871) はセッションのデフォルトモデルをカバーします。

- プロファイル検出が利用できない場合、Claude Code は可用性を確認せずにプレフィックスを適用します。アカウントにそのプレフィックスを持つ推論プロファイルが有効になっていない場合、リクエストは 400 エラーで失敗します。

Claude Code は、自分で設定した Amazon Bedrock 推論プロファイル ID または ARN、または *modelOverrides* (ページ 870) 値を書き直しません。Anthropic 形式のモデル ID は */model* **ピッカーと同じマッピング** (ページ 870) を通じて解決されます。Claude Code は 2 つのケースで変数を無視します。

- AWS GovCloud リージョンでは、Claude Code は常に `us-gov.` を使用します。これは GovCloud パーティション内でルーティングする唯一のプレフィックスです。
- 有効な値の 1 つではない値を設定した場合、Claude Code はリージョン導出の優先プレフィックスにフォールバックします。

36.6 IAM 設定

Claude Code に必要な権限を持つ IAM ポリシーを作成します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowModelAndInferenceProfileAccess",
      "Effect": "Allow",
      "Action": [
        "bedrock:InvokeModel",
        "bedrock:InvokeModelWithResponseStream",
        "bedrock:ListInferenceProfiles",
        "bedrock:GetInferenceProfile"
      ],
      "Resource": [
        "arn:aws:bedrock:*:*:inference-profile/*",
        "arn:aws:bedrock:*:*:application-inference-profile/*",
        "arn:aws:bedrock:*:*:foundation-model/*"
      ]
    },
    {
      "Sid": "AllowMarketplaceSubscription",
      "Effect": "Allow",
      "Action": [
        "aws-marketplace:ViewSubscriptions",
        "aws-marketplace:Subscribe"
```

(次のページに続く)

(前のページからの続き)

```
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "bedrock.amazonaws.com"
        }
      }
    }
  ]
}
```

より制限的な権限の場合は、リソースを特定の推論プロファイル ARN に制限できます。

`bedrock:GetInferenceProfile` により、Claude Code は**アプリケーション推論プロファイル** *ARN* (ページ 870) をそのバッキング基盤モデルに解決でき、そのモデルに対して正しいリクエスト形状を選択するために使用されます。

トークンにこの権限がない場合、Claude Code は代替形状で 1 回再試行することで自動的に復旧するため、リクエストは成功しますが、新しいモデルが追加されるたびに追加のラウンドトリップが発生します。権限を付与することで再試行を回避できます。これは `AWS_BEARER_TOKEN_BEDROCK` デプロイメントに最も頻繁に適用され、トークンのポリシーは通常、完全な IAM ロールよりも狭くなります。

詳細については、Amazon Bedrock IAM ドキュメントを参照してください。

注釈:

コスト追跡とアクセス制御を簡素化するために、Claude Code 用の専用 AWS アカウントを作成してください。

36.7 1M トークンコンテキストウィンドウ

Claude Sonnet 5、Opus 4.6 以降、および Sonnet 4.6 は、Amazon Bedrock で 1M トークンコンテキストウィンドウをサポートしています。Sonnet 5 は Invoke API と *Mantle* **エンドポイント** (ページ 875) の両方で常に 1M ウィンドウで実行され、選択する `[1m]` バリアントはありません。Invoke API 上の他のモデルについては、Claude Code は 1M モデルバリアントを選択すると、拡張コンテキストウィンドウを自動的に有効にします。

セットアップウィザード (ページ 863) は、モデルをピン留めするときに 1M コンテキストオプションを提供します。手動でピン留めされたモデルの代わりに有効にするには、モデル ID に `[1m]` を追加します。詳細については、**サードパーティデプロイメント用のモデルをピン留めする** (ページ 758) を参照してください。1M ウィンドウをピンを変更せずに使用する方法を含みます。

36.8 サービスティア

Amazon Bedrock サービスティアを使用すると、コストとレイテンシーのトレードオフを行うことができます。ANTHROPIC_BEDROCK_SERVICE_TIER を default、flex、または priority に設定します。

```
export ANTHROPIC_BEDROCK_SERVICE_TIER=priority
```

Claude Code は、各リクエストで X-Amzn-Bedrock-Service-Tier ヘッダーとしてこれを送信します。ティアの可用性はモデルとリージョンによって異なります。予約容量は、この設定の代わりにプロビジョニングされたスループット ARN をモデル ID として使用します。

36.9 AWS Guardrails

Amazon Bedrock Guardrails を使用すると、Claude Code のコンテンツフィルタリングを実装できます。Amazon Bedrock コンソールで Guardrail を作成し、バージョンを公開してから、Guardrail ヘッダーを *settings file* (ページ 711) に追加します。クロスリージョン推論プロファイルを使用している場合は、Guardrail でクロスリージョン推論を有効にしてください。

設定例：

```
{
  "env": {
    "ANTHROPIC_CUSTOM_HEADERS": "X-Amzn-Bedrock-GuardrailIdentifier: your-guardrail-
↪id\nX-Amzn-Bedrock-GuardrailVersion: 1"
  }
}
```

組織が Claude apps gateway ポリシーを通じて guardrail ヘッダーを配信する場合、それらは 承認が必要な設定としてカウントされます。

guardrail が応答を途中でブロックした場合、それまでにストリーミングされたテキストはそのまま残り、応答はブロックされた応答用に guardrail で設定されたメッセージで終了します。

36.10 Mantle エンドポイントを使用する

Mantle は、Bedrock Invoke API ではなく、ネイティブ Anthropic API シェイプを通じて Claude モデルを提供する Amazon Bedrock エンドポイントです。同じ *AWS* **認証情報** (ページ 864) と *awsAuthRefresh* **設定** (ページ 866) を使用します。

Mantle は bedrock-mantle: プレフィックスの下に独自の IAM アクションを持つため、*IAM* **設定** (ページ 873) の bedrock: アクションはこれをカバーしていません。推論用に bedrock-mantle:CreateInference と、トークンカウント用に bedrock-mantle:CountTokens を IAM アイデンティティに付与します。AWS ドキュメントの 推論リクエストの実行 と トークンのカウント、および サービス認可リファレンス を参照して、すべての Mantle アクションを確認してください。

36.10.1 Mantle を有効にする

AWS 認証情報が既に設定されている場合、CLAUDE_CODE_USE_MANTLE を設定して、リクエストを Mantle エンドポイントにルーティングします。

```
export CLAUDE_CODE_USE_MANTLE=1
export AWS_REGION=us-east-1
```

Claude Code は AWS リージョンからエンドポイント URL を構築します。リージョンは **上記の** *Amazon Bedrock* (ページ 867) と同じ優先順位で解決されます。カスタムエンドポイントまたはゲートウェイの URL をオーバーライドするには、ANTHROPIC_BEDROCK_MANTLE_BASE_URL を設定します。

Claude Code 内で /status を実行して確認します。Mantle がアクティブな場合、プロバイダー行は Amazon Bedrock (Mantle) を表示します。

36.10.2 Mantle モデルを選択する

Mantle は anthropic. で始まり、バージョンサフィックスのないモデル ID を使用します。例えば anthropic.claude-sonnet-5 または anthropic.claude-haiku-4-5 です。アカウントで利用可能なモデルは、組織に付与されたものに依存します。追加のモデル ID は AWS からのオンボーディング資料に記載されています。AWS アカウントチームに連絡して、許可リストされたモデルへのアクセスをリクエストしてください。

--model フラグまたは Claude Code 内の /model でモデルを設定します。

```
claude --model anthropic.claude-haiku-4-5
```

36.10.3 Mantle を Invoke API と並行して実行する

Mantle で利用可能なモデルは、今日使用するすべてのモデルを含まない場合があります。CLAUDE_CODE_USE_BEDROCK と CLAUDE_CODE_USE_MANTLE の両方を設定すると、Claude Code は同じセッションから両方のエンドポイントを呼び出せます。Mantle 形式に一致するモデル ID は Mantle にルーティングされ、他のすべてのモデル ID は Amazon Bedrock Invoke API に移動します。

```
export CLAUDE_CODE_USE_BEDROCK=1
export CLAUDE_CODE_USE_MANTLE=1
```

Mantle モデルを /model ピッカーに表示するには、*settings file* (ページ 711) の availableModels にその ID をリストします。この設定はピッカーをリストされたエントリに制限するため、保持したいバージョンのバージョンプレフィックスまたは完全な ID もリストします。Mantle ID と haiku エイリアスは同じモデルファミリーに解決されるため、マージはより具体的なエントリのみを保持します。*Merge behavior* (ページ 738) を参照してください。

```
{
  "availableModels": ["opus", "sonnet", "claude-haiku-4-5", "anthropic.claude-haiku-4-
↪5"]
}
```

anthropic. プレフィックス付きのエントリはカスタムピッカーオプションとして追加され、そのうち Mantle 形式に一致するものが Mantle にルーティングされます。anthropic.claude-haiku-4-5 をアカウントに付与されたモデル ID に置き換えます。availableModels が他のモデル設定とどのように相互作用するかについては、*Restrict model selection* (ページ 732) を参照してください。

両方のプロバイダーがアクティブな場合、/status は Amazon Bedrock + Amazon Bedrock (Mantle) を表示します。

36.10.4 Mantle をゲートウェイ経由でルーティングする

組織がモデルトラフィックを集中化された *LLM gateway* (ページ 841) を通じてルーティングし、AWS 認証情報をサーバー側に注入する場合、クライアント側認証を無効にして、Claude Code が SigV4 署名または x-api-key ヘッダーなしでリクエストを送信するようにします。

```
export CLAUDE_CODE_USE_MANTLE=1
export CLAUDE_CODE_SKIP_MANTLE_AUTH=1
export ANTHROPIC_BEDROCK_MANTLE_BASE_URL=https://your-gateway.example.com
```

36.10.5 Mantle 環境変数

これらの変数は Mantle エンドポイントに固有です。完全なリストについては、Environment variables を参照してください。

変数	目的
CLAUDE_CODE_USE_MANTLE	Mantle エンドポイントを有効にします。1 または true に設定します。
ANTHROPIC_BEDROCK_MANTLE_BASE_URL	デフォルト Mantle エンドポイント URL をオーバーライド
CLAUDE_CODE_SKIP_MANTLE_AUTH	プロキシセットアップのクライアント側認証をスキップ
ANTHROPIC_SMALL_FAST_MODEL_AWS_REGION	Haiku クラスモデルの AWS リージョンをオーバーライド（Bedrock と共有）

36.11 トラブルシューティング

36.11.1 SSO と企業プロキシでの認証ループ

AWS SSO を使用する場合にブラウザタブが繰り返し生成される場合は、*settings file* (ページ 711) から awsAuthRefresh 設定を削除してください。これは、企業 VPN または TLS 検査プロキシが SSO ブラウザフローを中断した場合に発生する可能性があります。Claude Code は中断された接続を認証失敗として扱い、awsAuthRefresh を再実行し、無限ループします。

ネットワーク環境が自動ブラウザベースの SSO フローに干渉する場合は、awsAuthRefresh に依存する代わりに、Claude Code を開始する前に手動で aws sso login を使用してください。

36.11.2 TLS 検査プロキシの背後での証明書エラー

Claude Code は、*CA certificate store* (ページ 898) 設定を AWS へのリクエストに適用します。これには以下が含まれます：

- モデル検出
- トークンカウント
- AWS 認証情報を解決する STS および SSO ロール認証情報呼び出し
- *setup wizard* (ページ 863) の認証情報検証とモデルチェック

これらのリクエストについては、OS トラストストアまたは `NODE_EXTRA_CA_CERTS` バンドル内の企業ルート証明書には、Amazon Bedrock 固有のセットアップは必要ありません。

v2.1.260 より前では、Claude Code は設定されたプロキシを通過するリクエストにのみ CA 設定を適用し、直接接続では実行時のデフォルト証明書ストアのみを信頼していました。

v2.1.261 より前では、**Use credentials already in my environment** オプションを使用した setup wizard のモデルチェックの背後での認証情報ルックアップは、実行時のデフォルト証明書ストアのみを信頼していました。ルート証明書が OS ストアにのみある TLS 検査プロキシの背後では、影響を受けるリクエストは `unable to get local issuer certificate` で失敗するか、ウィザードはモデルを `unreachable` として表示していましたが、推論リクエストは成功していました。v2.1.261 以降に更新してください。

36.11.3 リージョンの問題

リージョンの問題が発生した場合：

- モデルの可用性を確認：`aws bedrock list-inference-profiles --region your-region`
- サポートされているリージョンに切り替え：`export AWS_REGION=us-east-1`
- クロスリージョンアクセスに推論プロファイルの使用を検討

「on-demand throughput isn't supported」エラーが表示される場合：

- モデルを inference profile ID として指定します

Claude Code は Amazon Bedrock Invoke API を使用し、Converse API はサポートしていません。

36.11.4 ゲートウェイまたはプロキシの背後でのストリーミングエラー

Amazon Bedrock は `InvokeModelWithResponseStream` レスポンスをバイナリイベントストリーム形式でストリーミングし、ヘッダー `Content-Type: application/vnd.amazon.eventstream` を含みます。Claude Code と Amazon Bedrock の間のゲートウェイまたはプロキシは、Amazon Bedrock が送信したレスポンスボディとそのヘッダー（`Content-Type` を含む）を変更されずに転送する必要があります。

ゲートウェイが `Content-Type` を別の値に書き換える場合、Claude Code は `Bedrock streaming response has content-type` で始まるエラーでレスポンスを拒否し、受け取った値を名前付けます。一般的な書き換えは `text/event-stream` で、ストリームをサーバー送信イベントとして再発行する統合からのものです。

ゲートウェイがヘッダーをドロップまたは空白にする代わりに、Claude Code は本体が Amazon Bedrock のイベントストリームであると仮定してデコードするため、ゲートウェイが変更されずに通した本体はストリーミングを続けます。

ヘッダーをドロップするゲートウェイがストリームをサーバー送信イベントとして再発行する場合、Claude Code は本体をデコードできず、すべてのターンで遅いノンストリーミングパスにフォールバックします。各レスポンスはストリーミングされるのではなく、完全になった後にのみ表示されます。その場合、`CLAUDE_CODE_DISABLE_BEDROCK_CONTENT_TYPE_DEFAULT=1` を設定して、Claude Code が本体をサーバー送信イベントとして読み取るようにしてください。

エラーまたはフォールバックを修正するには、ゲートウェイを設定して `InvokeModelWithResponseStream` レスポンスボディとその `Content-Type` ヘッダーを変更されずに転送するようにしてください。

ストリームをサーバー送信イベントに変換するゲートウェイは、Amazon Bedrock API を提供していません。Anthropic Messages API リクエストも受け入れる場合は、`CLAUDE_CODE_USE_BEDROCK` の代わりに `ANTHROPIC_BASE_URL` を使用して LLM gateway として接続してください。

36.11.5 /context でのゼロトークンカウント

`/context` コマンドは、ツールスキーマを Amazon Bedrock count-tokens API に送信することで、各ツールグループのトークンをカウントします。Claude Code v2.1.196 より前のバージョンでは、スキーマが count-tokens API が受け入れないフィールドを含んでいたため、Amazon Bedrock がそのリクエストを拒否し、すべてのツールグループが 0 トークンを表示していました。メッセージやメモリファイルなど、内訳の他の行は影響を受けません。

v2.1.196 以降に更新してください。

36.11.6 Mantle エンドポイントエラー

`CLAUDE_CODE_USE_MANTLE` を設定した後、`/status` が `Amazon Bedrock (Mantle)` を表示しない場合、変数がプロセスに到達していません。Claude Code を起動したシェルでエクスポートされているか、*settings file* (ページ 711) の `env` ブロックで設定されていることを確認してください。

Mantle エンドポイントからの `403` が何を意味するかは、エラーが IAM アクションを名前付けるかどうかによって異なります：

- エラーが `bedrock-mantle:` アクションを名前付ける場合は、IAM アイデンティティにそのアクションを付与してください。
- エラーがアクションを名前付けず、認証情報が有効な場合は、AWS アカウントがリクエストしたモデルへのアクセスを許可されていません。AWS アカウントチームに連絡してアクセスをリクエストしてください。

モデル ID を名前付ける `400` は、そのモデルが Mantle で提供されていないことを意味します。Mantle は標準 Amazon Bedrock カタログとは別の独自のモデルラインアップを持っているため、`us.anthropic.claude-sonnet-4-6` などの推論プロファイル ID は機能しません。Mantle 形式の ID を使用するか、**両方のエンドポイントを有効にして** (ページ 876)、Claude Code が各リクエストをモデルが利用可能なエンドポイントにルーティングするようにしてください。

36.12 追加リソース

- Amazon Bedrock ドキュメント
- Amazon Bedrock 料金
- Amazon Bedrock 推論プロファイル
- Amazon Bedrock トークンバーンダウンとクォータ
- Claude Code on Amazon Bedrock: Quick Setup Guide
- Claude Code Monitoring Implementation（Amazon Bedrock）

第37章　Google Cloud の Agent Platform 上の Claude Code

Google Cloud の Agent Platform（旧 Vertex AI）を通じた Claude Code の設定方法について学びます。セットアップ、IAM 設定、トラブルシューティングを含みます。

37.1 前提条件

Claude Code を Google Cloud の Agent Platform（旧 Vertex AI）で設定する前に、以下を確認してください。

- 請求が有効になっている Google Cloud Platform（GCP）アカウント
- Google Cloud の Agent Platform API が有効になっている GCP プロジェクト
- 目的の Claude モデルへのアクセス（例：Claude Sonnet 4.6）
- Google Cloud SDK（`gcloud`）がインストールされ、設定されていること
- 目的の GCP リージョンに割り当てられたクォータ

Google Cloud の Agent Platform 認証情報でサインインするには、以下の *Agent Platform* **でサインイン** (ページ 881) に従ってください。チーム全体に Claude Code をデプロイするには、**手動セットアップ** (ページ 882) の手順を使用し、ロールアウト前に**モデルバージョンをピン留めして** (ページ 885) ください。

37.2 Agent Platform でサインイン

Google Cloud 認証情報を持っていて、Google Cloud の Agent Platform を通じて Claude Code の使用を開始したい場合、ログインウィザードがそれをガイドします。GCP 側の前提条件はプロジェクトごとに 1 回完了します。ウィザードが Claude Code 側を処理します。

1. **GCP プロジェクトで Claude モデルを有効にする**

   プロジェクトの *Google Cloud* **の** *Agent Platform API* **を有効にして** (ページ 882)、Google Cloud の Agent Platform Model Garden で必要な Claude モデルへのアクセスをリクエストしてください。アカウントに必要な権限については、*IAM* **設定** (ページ 887) を参照してください。

2. **Claude Code を起動して Google Cloud の Agent Platform を選択する**

   `claude` を実行します。ログインプロンプトで、**3rd-party platform**、次に **Google Vertex AI** を選択します。これはログインプロンプトが Google Cloud の Agent Platform に対して使用しているラベルです。既にサインインしている場合は、`/login` を実行して同じメニューを開きます。

3. **ウィザードプロンプトに従う**

Google Cloud への認証方法を選択します。gcloud からの Application Default Credentials、サービスアカウントキーファイル、または環境内に既にある認証情報です。ウィザードはプロジェクトとリージョンを検出し、プロジェクトが呼び出せる Claude モデルを確認し、それらをピン留めできます。結果は**ユーザー設定ファイル** (ページ 711) の env ブロックに保存されるため、環境変数を自分でエクスポートする必要はありません。

サインイン後、いつでも /setup-vertex を実行してウィザードを再度開き、認証情報、プロジェクト、リージョン、またはモデルピンを変更できます。モデルピンステップは、現在ピン留めされているモデルから開始します。ウィザードは ~/.claude/settings.json に書き込むか、CLAUDE_CONFIG_DIR が設定されている場合は $CLAUDE_CONFIG_DIR/settings.json に書き込みます。

37.3 リージョン設定

Claude Code は Google Cloud の Agent Platform のグローバル、マルチリージョン、および地域別エンドポイントをサポートしています。CLOUD_ML_REGION を global、eu または us などのマルチリージョンロケーション、または us-east5 などの特定のリージョンに設定します。Claude Code は各フォームの正しい Google Cloud の Agent Platform ホスト名を選択します。これには、マルチリージョンロケーション用の aiplatform.eu.rep.googleapis.com および aiplatform.us.rep.googleapis.com ホストが含まれます。

注釈:

Google Cloud の Agent Platform は、すべてのエンドポイントタイプで Claude Code デフォルトモデルをサポートしていない場合があります。モデルの可用性は、特定のリージョン、マルチリージョンロケーション、およびグローバルエンドポイントによって異なります。サポートされているロケーションに切り替えるか、サポートされているモデルを指定する必要がある場合があります。

37.4 手動でセットアップする

ウィザードの代わりに環境変数を通じて Google Cloud の Agent Platform を設定するには、例えば CI またはスクリプト化されたエンタープライズロールアウトで、以下の手順に従ってください。

37.4.1 1. Agent Platform API を有効にする

GCP プロジェクトで Google Cloud の Agent Platform API を有効にします。YOUR-PROJECT-ID を GCP プロジェクト ID に置き換えてください。以下の設定手順でも同じように置き換えます。

```
# プロジェクト ID を設定
gcloud config set project YOUR-PROJECT-ID

# Agent Platform API を有効にする
gcloud services enable aiplatform.googleapis.com
```

37.4.2 2. モデルアクセスをリクエストする

Google Cloud の Agent Platform で Claude モデルへのアクセスをリクエストします。

1. Google Cloud の Agent Platform Model Garden に移動します
2. 「Claude」モデルを検索します
3. 目的の Claude モデルへのアクセスをリクエストします（例：Claude Sonnet 4.6）
4. 承認を待ちます（24 〜 48 時間かかる場合があります）

37.4.3 3) GCP 認証情報を設定する

Claude Code は標準的な Google Cloud 認証を使用します。

詳細については、Google Cloud 認証ドキュメントを参照してください。

Claude Code は、同じ Application Default Credentials チェーンを通じて X.509 証明書ベースのワークロード ID フェデレーションをサポートしています。`GOOGLE_APPLICATION_CREDENTIALS` を認証情報設定ファイルのパスに設定します。

注釈:

Claude Code は Google Cloud の Agent Platform リクエストを `ANTHROPIC_VERTEX_PROJECT_ID` のプロジェクトにアドレス指定します。`GCLOUD_PROJECT`、`GOOGLE_CLOUD_PROJECT`、または `GOOGLE_APPLICATION_CREDENTIALS` で参照される認証情報ファイルが異なるプロジェクトを持つ場合でも同じです。

高度な認証情報設定

Claude Code は `gcpAuthRefresh` 設定を通じて GCP の自動認証情報更新をサポートしています。これを Claude Code の **設定ファイル** (ページ 711)（例：`~/.claude/settings.json`）に追加します。Claude Code が GCP 認証情報の有効期限が切れているか読み込めないことを検出すると、リクエストを再試行する前に新しい認証情報を取得するために設定されたコマンドを実行します。

```
{
  "gcpAuthRefresh": "gcloud auth application-default login",
  "env": {
    "ANTHROPIC_VERTEX_PROJECT_ID": "your-project-id"
  }
}
```

コマンドを実行する前に、Claude Code は現在の認証情報でアクセストークンをリクエストして、実際に有効期限が切れていることを確認し、まだ機能している場合はコマンドをスキップします。

チェックが 5 秒以内に完了しない場合、Claude Code もコマンドをスキップし、リクエストが認証情報エラーで失敗した後にのみ実行します。v2.1.261 より前は、タイムアウトしたチェックは有効期限が切れた認証情報としてカウントされたため、認証情報がまだ有効であってもコマンドがスタートアップ時にブラウザを開く可能性がありました。

Claude Code はコマンドの出力を表示しますが、コマンドに対話的な入力を送信することはできません。これは、CLI が URL を表示し、ブラウザで認証を完了するブラウザベースの認証フローに適しています。認証が完了しない場合、更新コマンドは 3 分後にタイムアウトします。`.claude/settings.json` などのプロジェクト設定で `gcpAuthRefresh` を設定した場合、Claude Code は設定ファイルの **ワークスペース信頼ルールと同じルール** (ページ 1066) の下でそれを実行します。これには、信頼したことのないフォルダ内の `-p` セッションが含まれます。

37.4.4 4. Claude Code を設定する

次の環境変数を設定します。

```
# Agent Platform 統合を有効にする
export CLAUDE_CODE_USE_VERTEX=1
export CLOUD_ML_REGION=global
export ANTHROPIC_VERTEX_PROJECT_ID=YOUR-PROJECT-ID

# オプション：カスタムエンドポイントまたはゲートウェイ用に Agent Platform エンドポイント␣
↪URL をオーバーライドする
# export ANTHROPIC_VERTEX_BASE_URL=https://aiplatform.googleapis.com

# CLOUD_ML_REGION=global の場合、グローバルエンドポイントをサポートしていないモデルのリージョンをオーバーライドする
export VERTEX_REGION_CLAUDE_HAIKU_4_5=us-east5
export VERTEX_REGION_CLAUDE_4_6_SONNET=europe-west1
```

ほとんどのモデルバージョンには、対応する `VERTEX_REGION_CLAUDE_*` 変数があります。完全なリストについては、環境変数リファレンスを参照してください。どのモデルがグローバルエンドポイントをサポートしているか、または地域別のみをサポートしているかを確認するには、Google Cloud の Agent Platform Model Garden を確認してください。

リージョン値がリージョンまたはロケーション名のような形状でない場合、Claude Code はそれを未設定として扱います。例えば、Claude Code はスラッシュ、ドット、またはスペースを含む値を未設定として扱います。Claude Code は各変数に対して異なるソースにフォールバックします。

- `VERTEX_REGION_CLAUDE_*`：Claude Code は `CLOUD_ML_REGION` にフォールバックします。
- `CLOUD_ML_REGION`：Claude Code は `us-east5` にフォールバックします。

prompt caching は自動的に有効になります。これを無効にするには、`DISABLE_PROMPT_CACHING=1` を設定します。デフォルトの 5 分ではなく 1 時間のキャッシュ TTL をリクエストするには、`ENABLE_PROMPT_CACHING_1H=1` を設定します。1 時間の TTL でのキャッシュ書き込みはより高いレートで課金されます。メインの会話と Claude Code が外部で行うリクエストに異なる TTL を設定するには、TTL を自分で選択してください。

レート制限を高くするには、Google Cloud サポートに連絡してください。Google Cloud の Agent Platform を使用する場合、認証が Google Cloud 認証情報を通じて処理されるため、`/logout` コマンドは利用できません。

Claude Code は *MCP tool search* (ページ 406) と事前ロードをモデル生成によって決定します。

- **Claude Opus 4.5、Sonnet 4.5、Haiku 4.5、およびそれ以降**：Claude Code はデフォルトでツール検索を有効にします。
- **Claude 3.x モデルを含む以前のモデル**：Claude Code は MCP ツール定義を事前にロードします。これは、Agent Platform サービングスタックが必要なベータヘッダーを拒否するためです。`ENABLE_TOOL_SEARCH=true` を設定してもこれをオーバーライドしません。

すべてのモデルでツール検索を無効にするには、`ENABLE_TOOL_SEARCH=false` を設定します。v2.1.221 より前は、Claude Code は `ENABLE_TOOL_SEARCH=true` を設定しない限り、Google Cloud の Agent Platform 上のすべてのモデルでツール検索を無効にしていました。

37.4.5 5. モデルバージョンをピン留めする

警告:

複数のユーザーにデプロイする場合は、特定のモデルバージョンをピン留めしてください。ピン留めなしでは、`sonnet` および `opus` などのモデルエイリアスは Claude Code の Google Cloud の Agent Platform 用の組み込みデフォルトに解決され、最新リリースより遅れる可能性があり、プロジェクトでまだ有効になっていない可能性があります。Claude Code は、デフォルトが利用できない場合、**スタートアップ時** (ページ 886) に前のバージョンまたは下位層のモデルにフォールバックしますが、ピン留めすることで、ユーザーが新しいモデルに移行するタイミングを制御できます。

これらの環境変数を特定の Google Cloud の Agent Platform モデル ID に設定します。

`ANTHROPIC_DEFAULT_OPUS_MODEL` がない場合、Google Cloud の Agent Platform 上の `opus` エイリアスは Opus 5.5 に解決され、`ANTHROPIC_DEFAULT_SONNET_MODEL` がない場合、`sonnet` エイリアスは Sonnet 4.5 に解決されます。この例では、各エイリアスを特定のバージョンにピン留めします。

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-8'
export ANTHROPIC_DEFAULT_SONNET_MODEL='claude-sonnet-5'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='claude-haiku-4-5@20251001'
```

現在および従来のモデル ID については、モデル概要を参照してください。環境変数の完全なリストについては、**モデル設定** (ページ 758) を参照してください。

Claude Code は、ピン留め変数が設定されていない場合、これらのデフォルトモデルを使用します。

モデルタイプ	デフォルト値
プライマリモデル	`claude-opus-5-5`
小型/高速モデル	`claude-sonnet-4-5@20250929`

セッションタイトル生成などのバックグラウンドタスクは、小型/高速モデル（通常は Haiku クラスモデル）を使用します。Google Cloud の Agent Platform では、Haiku がすべてのプロジェクトまたはリージョンで有効になっていない可能性があるため、Claude Code はバックグラウンドタスクにデフォルトの Sonnet モデルを使用します。2 つの選択がどのモデルがバックグラウンドタスクを実行するかを変更します。

- `--model`、`ANTHROPIC_MODEL`、または `model` 設定でプライマリモデルを選択すると、バックグラウ

ンドタスクはそのモデルを使用します。*ANTHROPIC_DEFAULT_MODEL* (ページ 731) で設定したモデルで Claude Code がセッションを開始する場合、バックグラウンドタスクもそのモデルを使用します。ANTHROPIC_DEFAULT_SONNET_MODEL なしで ANTHROPIC_DEFAULT_OPUS_MODEL を設定することも、組み込み Sonnet モデルがプロジェクトで有効になっていない可能性があるため、選択としてカウントされます。

- バックグラウンドタスクに Haiku を使用するには、ANTHROPIC_DEFAULT_HAIKU_MODEL をプロジェクトで利用可能なモデル ID に設定します。

警告:

Opus モデルは Sonnet モデルより高いトークンあたりの価格を持つため、プライマリモデルをピン留めしないデプロイメントは v2.1.207 以降に更新されると Opus レートで課金されます。Sonnet 4.5 をプライマリモデルとして保つには、ANTHROPIC_MODEL をその完全なモデル ID に設定します。ANTHROPIC_DEFAULT_SONNET_MODEL でデフォルトを制御し、ANTHROPIC_DEFAULT_OPUS_MODEL を設定しないデプロイメントは、制御された Sonnet モデルをデフォルトとして保ちます。

v2.1.280 より前は、Google Cloud の Agent Platform 上のプライマリモデルは Opus 5 にデフォルト設定され、opus エイリアスは v2.1.219 から Opus 5 に解決されました。v2.1.207 〜 v2.1.218 では、Google Cloud の Agent Platform 上のプライマリモデルは Opus 4.8 にデフォルト設定され、opus エイリアスは Opus 4.8 に解決されました。v2.1.207 より前は、プライマリモデルは Sonnet 4.5 にデフォルト設定され、opus エイリアスは Opus 4.6 に解決され、バックグラウンドタスクは常にプライマリモデルを使用していました。

モデルをさらにカスタマイズするには、以下を実行します。

```
export ANTHROPIC_MODEL='claude-opus-4-8'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='claude-haiku-4-5@20251001'
```

37.4.6 6. 設定を確認する

Claude Code を起動して /status を実行し、セットアップを確認します。API provider 行は Google Vertex AI を表示し、GCP project、Default region、および Model 行はプロジェクト ID、リージョン、および解決されたモデルを表示します。プロバイダー行が見つからない場合、環境変数がプロセスに到達していません。claude を起動したシェルでエクスポートされているか、**設定ファイル** (ページ 711) の env ブロックで設定されていることを確認してください。

37.5 起動時のモデルチェック

Claude Code が Google Cloud の Agent Platform で設定されて起動すると、使用するモデルがプロジェクトでアクセス可能であることを確認します。

Claude Code デフォルトより古いモデルバージョンをピン留めしていて、プロジェクトが新しいバージョンを呼び出せる場合、Claude Code はピンを更新するよう促します。受け入れると、新しいモデル ID が**ユーザー設定ファイル** (ページ 711) に書き込まれ、Claude Code が再起動されます。拒否すると、次のデフォルトバージョン変更まで記憶されます。

モデルをピン留めしていなくて、現在のデフォルトがプロジェクトで利用できない場合、Claude Code は現在のセッション用にフォールバックし、通知を表示します。デフォルトモデルの以前のバージョンを最初に試し、デフォルトが Opus モデルで Opus バージョンが利用できない場合は、デフォルト Sonnet モデル

にフォールバックします。フォールバックは永続化されません。Model Garden で新しいモデルを有効にするか、**バージョンをピン留めして** (ページ 885) 選択を永続化してください。

特定の Sonnet または Opus バージョンでセッションを開始する場合（例えば `--model`、`ANTHROPIC_MODEL`、または model 設定を使用）、そのバージョンは一致する `sonnet` または `opus` エイリアスのセッションのピン留めされたデフォルトとして機能します。Claude Code は、設定したモデルが置き換える組み込みデフォルトの可用性チェックをスキップし、設定したモデルで起動します。フォールバック通知はありません。

`opus` などのモデルエイリアスはピンとして機能せず、Claude Code が認識しないモデル ID も同様です。

これらのチェックがプロジェクトが呼び出せないモデルを見つけた場合、Claude Code はこのマシン上でその拒否を最大 1 日間記憶し、その間の起動時に記憶されたモデルをスキップして Agent Platform に再度問い合わせません。Claude Code は、現在のデフォルトモデルの記憶された拒否を、最後のチェック以降 10 分が経過した後に起動時に再度チェックするため、管理者が再度有効にしたデフォルトが戻ります。メモリをオフにするには、`CLAUDE_CODE_SKIP_MODEL_ACCESS_MEMORY=1` を設定してください。

37.5.1 セッション中にモデルが無効化された場合

プロジェクトがセッションで実行しているモデルへのアクセスを失った場合（例えば、管理者が Model Garden で無効化した場合）、Claude Code は各リクエストが失敗する代わりにセッションを別のモデルに切り替え、`Switched to <fallback> because <model> is not available` を表示します。起動時フォールバックと同じモデルを試します。同じティアの以前のバージョンを最初に試し、Opus セッションで Opus バージョンが利用できない場合は、デフォルト Sonnet モデルを試します。

切り替えは、ピン留めしていないティアにのみ適用されます。これは起動時フォールバックと同じ条件です。選択した特定のバージョンでのセッションはそのモデルを保持し、フォールバックモデルチェーンがない場合、リクエストは失敗します。auto モードでは、Claude Code は Agent Platform で auto モードがサポートするモデルにのみ切り替えます。それらのモデルも利用できない場合、リクエストは失敗します。

設定した**フォールバックモデルチェーン** (ページ 743) はティア切り替えを置き換えます。これらの拒否時に Claude Code は設定したフォールバックに切り替えます。拒否されたリクエストが切り替わらずに失敗するようにするには、`CLAUDE_CODE_DISABLE_MODEL_ACCESS_FALLBACK=1` を設定してください。設定したフォールバックチェーンはこれらの拒否時に切り替わります。すべての拒否されたリクエストが失敗するようにしたい場合は、チェーンも削除してください。

37.6 IAM 設定

`roles/aiplatform.user` ロールを割り当てます。このロールには必要な権限が含まれています。

- `aiplatform.endpoints.predict` - モデル呼び出しとトークンカウントに必要

より制限的な権限については、上記の権限のみを持つカスタムロールを作成してください。

詳細については、Google Cloud の Agent Platform IAM ドキュメントを参照してください。

注釈:

Claude Code 用に専用の GCP プロジェクトを作成して、コスト追跡とアクセス制御を簡素化してください。

37.7 100 万トークンコンテキストウィンドウ

Claude Sonnet 5、Opus 4.6 以降、および Sonnet 4.6 は、Google Cloud の Agent Platform で 100 万トークンコンテキストウィンドウをサポートしています。Sonnet 5 は常に 100 万ウィンドウで実行され、選択する `[1m]` バリアントはありません。その他のモデルについては、Claude Code は 100 万トークンモデルバリアントを選択すると、拡張コンテキストウィンドウを自動的に有効にします。

セットアップウィザード (ページ 881) は、モデルをピン留めするときに 100 万トークンコンテキストオプションを提供します。手動でピン留めされたモデルの代わりに有効にするには、モデル ID に `[1m]` を追加します。詳細については、**サードパーティデプロイメント用のモデルをピン留めする** (ページ 758) を参照してください。1M ウィンドウを使用する際にピン留めを変更しない方法を含みます。

37.8 トラブルシューティング

「デフォルト認証情報を読み込めません」エラーが発生した場合：

- `gcloud auth application-default login` を実行して Application Default Credentials をセットアップしてください
- `GOOGLE_APPLICATION_CREDENTIALS` をサービスアカウントキーファイルパスに設定してください
- すべてのオプションについては、*GCP* **認証情報の設定** (ページ 883) を参照してください

クォータの問題が発生した場合：

- Cloud Console を通じて現在のクォータを確認するか、クォータ増加をリクエストしてください

「モデルが見つかりません」404 エラーが発生した場合：

- Model Garden でモデルが有効になっていることを確認してください
- 指定したロケーションでモデルが利用可能であることを確認してください。一部のモデルは `global` またはマルチリージョンロケーション（`eu` および `us` など）でのみ提供され、特定のリージョンでは提供されていません
- `CLOUD_ML_REGION=global` を使用している場合、Model Garden の「サポートされている機能」でモデルがグローバルエンドポイントをサポートしていることを確認してください。グローバルエンドポイントをサポートしていないモデルの場合は、以下のいずれかを実行してください：
  - `ANTHROPIC_MODEL` または `ANTHROPIC_DEFAULT_HAIKU_MODEL` を通じてサポートされているモデルを指定するか、
  - `VERTEX_REGION_<MODEL_NAME>` 環境変数を使用してリージョンまたはマルチリージョンロケーションを設定してください

429 エラーが発生した場合：

- 地域別エンドポイントの場合、プライマリモデルと小型/高速モデルが選択したリージョンでサポートされていることを確認してください
- より良い可用性のために `CLOUD_ML_REGION=global` に切り替えることを検討してください

37.9 追加リソース

- Google Cloud の Agent Platform ドキュメント
- Google Cloud の Agent Platform 価格
- Google Cloud の Agent Platform クォータと制限

第38章　Claude Code on Microsoft Foundry

Microsoft Foundry を通じて Claude Code を構成する方法について学びます。セットアップ、構成、トラブルシューティングを含みます。

38.1 前提条件

Microsoft Foundry で Claude Code を構成する前に、以下を確認してください：

- Microsoft Foundry へのアクセス権を持つ Azure サブスクリプション
- Microsoft Foundry リソースとデプロイメントを作成するための RBAC 権限
- Azure CLI がインストールされ、構成されている（オプション - 認証情報を取得する別のメカニズムがない場合のみ必要）

注釈:

Claude Code を複数のユーザーにデプロイする場合は、**モデルバージョンをピン留めして** (ページ 893)、ロールアウト前に実施してください。

38.2 セットアップ

38.2.1 1. Microsoft Foundry リソースをプロビジョニングする

まず、Azure で Claude リソースを作成します：

1. Microsoft Foundry ポータルに移動します
2. 新しいリソースを作成し、リソース名をメモします
3. Claude モデルのデプロイを作成します。各デプロイに付与する名前をメモしてください。ステップ 4 でこれらの名前をモデル変数として設定します：
   - Claude Opus
   - Claude Sonnet
   - Claude Haiku

   デプロイを設定する際に、ホスティングオプションも選択します。これにより、推論が Azure で実行されるか、Anthropic インフラストラクチャで実行されるかが決まります。

38.2.2 2) Azure 認証情報を構成する

Claude Code は Microsoft Foundry の 3 つの認証方法をサポートしています。セキュリティ要件に最適な方法を選択してください。

オプション A：API キー認証

1. Microsoft Foundry ポータルでリソースに移動します
2. **エンドポイントとキー**セクションに移動します
3. **API キー**をコピーします
4. 環境変数を設定します。`your-azure-api-key` をコピーしたキーに置き換えます：

```
export ANTHROPIC_FOUNDRY_API_KEY=your-azure-api-key
```

オプション B：Microsoft Entra ID 認証

`ANTHROPIC_FOUNDRY_API_KEY` も `ANTHROPIC_FOUNDRY_AUTH_TOKEN` も設定されていない場合、Claude Code は Azure SDK デフォルト認証情報チェーンを自動的に使用します。これは、ローカルおよびリモートワークロードを認証するためのさまざまな方法をサポートしています。

ローカル環境では、一般的に Azure CLI を使用できます：

```
az login
```

オプション C：ベアラートークン認証

Claude Code は、すべてのリクエストで `ANTHROPIC_FOUNDRY_AUTH_TOKEN` の値を `Authorization: Bearer` ヘッダーとして送信します。ホストアプリケーションやサインインスクリプトなど、別のプロセスがすでにアクセストークンを取得している場合に、このオプションを使用します。Claude Code v2.1.203 以降が必要です。

変数を、Microsoft Entra ID がリソース用に発行したベアラートークンに設定します：

```
export ANTHROPIC_FOUNDRY_AUTH_TOKEN=your-entra-access-token
```

`ANTHROPIC_FOUNDRY_AUTH_TOKEN` は `ANTHROPIC_FOUNDRY_API_KEY` およびデフォルト認証情報チェーンより優先されます。

注釈:

Microsoft Foundry を使用する場合、認証が Azure 認証情報を通じて処理されるため、`/logout` コマンドは利用できません。

38.2.3 3. Claude Code を構成する

Microsoft Foundry を有効にするには、以下の環境変数を設定します：

```
# Microsoft Foundry 統合を有効にする
export CLAUDE_CODE_USE_FOUNDRY=1

# Azure リソース名（{resource} をリソース名に置き換えます）
export ANTHROPIC_FOUNDRY_RESOURCE={resource}
# または完全なベース URL を提供します：
# export ANTHROPIC_FOUNDRY_BASE_URL=https://{resource}.services.ai.azure.com/anthropic
```

ANTHROPIC_FOUNDRY_RESOURCE には、my-resource のようにリソース名のみを設定します。URL やホスト名を設定した場合、メッセージを送信した時点で Claude Code はそれを拒否します。

38.2.4 4. モデルバージョンをピン留めする

警告:

すべてのデプロイに対して特定のモデルバージョンをピン留めしてください。ピン留めなしでは、sonnet や opus などのモデルエイリアスが Claude Code の Microsoft Foundry 用の組み込みデフォルトに解決されます。これは最新リリースより遅れている可能性があり、アカウントでまだ利用できない場合があります。Microsoft Foundry にはスタートアップモデルチェックがないため、デフォルトが利用できない場合、リクエストは失敗します。Azure デプロイを作成するときは、「最新に自動更新」ではなく、特定のモデルバージョンを選択してください。

モデル変数をステップ 1 で作成したデプロイ名と一致するように設定します。

ANTHROPIC_DEFAULT_OPUS_MODEL がない場合、Microsoft Foundry の opus エイリアスは Opus 4.6 に解決されます。より新しい Opus モデル（Opus 4.8 など）の ID に設定します：

```
export ANTHROPIC_DEFAULT_OPUS_MODEL='claude-opus-4-8'
export ANTHROPIC_DEFAULT_SONNET_MODEL='claude-sonnet-5'
export ANTHROPIC_DEFAULT_HAIKU_MODEL='claude-haiku-4-5'
```

セッションタイトル生成などのバックグラウンドタスクは、通常は Haiku クラスモデルである小型/高速モデルを使用します。Microsoft Foundry では、すべてのアカウントが Haiku デプロイを持っているわけではないため、Claude Code はこれをプライマリモデルにデフォルト設定します。バックグラウンドタスクに Haiku を使用するには、上記のように、アカウントで利用可能な Haiku デプロイに ANTHROPIC_DEFAULT_HAIKU_MODEL を設定します。

現在および従来のモデル ID については、モデル概要を参照してください。環境変数の完全なリストについては、**モデル設定** (ページ 758) を参照してください。

プロンプトキャッシュは自動的に有効になります。デフォルトの 5 分ではなく 1 時間のキャッシュ TTL をリクエストするには、以下の変数を設定します。1 時間の TTL でのキャッシュ書き込みはより高いレートで課金されます：

```
export ENABLE_PROMPT_CACHING_1H=1
```

メインの会話と、Claude Code がその外部で実行するリクエストとで異なる TTL を設定するには、TTL を自分で選択してください。

38.2.5 5. Claude Code を実行する

環境変数を設定したら、プロジェクトディレクトリから Claude Code を起動します：

```
claude
```

Claude Code は環境から `CLAUDE_CODE_USE_FOUNDRY` およびその他の Microsoft Foundry 変数を読み込み、最初のプロンプトで Azure リソースに接続します。Amazon Bedrock および Google Cloud の Agent Platform とは異なり、Microsoft Foundry には対話型セットアップウィザードがないため、ステップ 3 およびステップ 4 の環境変数が唯一の設定方法です。

セットアップを確認するには、Claude Code 内で `/status` を実行します。API プロバイダー行に `Microsoft Foundry` が表示され、設定したリソース名またはベース URL が表示されます。

38.3 Azure RBAC 構成

`Azure AI User` および `Cognitive Services User` デフォルトロールには、Claude モデルを呼び出すために必要なすべての権限が含まれています。

より制限的な権限の場合は、以下を含むカスタムロールを作成します：

```
{
  "permissions": [
    {
      "dataActions": [
        "Microsoft.CognitiveServices/accounts/providers/*"
      ]
    }
  ]
}
```

詳細については、Microsoft Foundry RBAC ドキュメントを参照してください。

38.4 トラブルシューティング

「Failed to get token from azureADTokenProvider: ChainedTokenCredential authentication failed」というエラーが表示される場合：

- 環境で Entra ID を構成するか、`ANTHROPIC_FOUNDRY_API_KEY` を設定してください。

最初のプロンプトで接続エラーが繰り返される場合、リクエストが失敗する場合：

- `ANTHROPIC_FOUNDRY_RESOURCE` がプレースホルダーではなく、実際のリソース名に設定されていることを確認してください。Claude Code はこの値からエンドポイント URL を構築するため、不正な名前は存在しないホストを指します。

38.5 その他のリソース

- Microsoft Foundry ドキュメント
- Microsoft Foundry モデル
- Microsoft Foundry 価格

第39章　エンタープライズネットワーク設定

プロキシサーバー、カスタム認証局（CA）、相互 Transport Layer Security（mTLS）認証を使用して、エンタープライズ環境向けに Claude Code を設定します。

Claude Code は、環境変数を通じてさまざまなエンタープライズネットワークおよびセキュリティ設定をサポートしています。これには、企業プロキシサーバーを通じたトラフィックのルーティング、カスタム認証局（CA）の信頼、および強化されたセキュリティのための相互 Transport Layer Security（mTLS）証明書による認証が含まれます。

Claude Code を起動する前に、これらの環境変数を設定してください。シェルでエクスポートされた変数は起動時に 1 回読み込まれるため、実行中のセッションはシェル環境への後の変更を取得しません。

注釈:

このページに表示されているすべての環境変数は、*settings.json* (ページ 711) でも設定できます。

39.1 プロキシ設定

39.1.1 環境変数

Claude Code は標準的なプロキシ環境変数に対応しています。Claude Desktop セッションでアプリがプロバイダー接続を管理する場合、Claude Code はマネージド設定と `~/.claude/settings.json` からのみこれらを読み込みます。スコープルールについては *mTLS* **認証** (ページ 899) を参照してください。

```
# HTTPS プロキシ（推奨）
export HTTPS_PROXY=https://proxy.example.com:8080

# HTTP プロキシ（HTTPS が利用できない場合）
export HTTP_PROXY=http://proxy.example.com:8080

# 特定のリクエストのプロキシをバイパス - スペース区切り形式
export NO_PROXY="localhost 192.168.1.1 example.com .example.com"
# 特定のリクエストのプロキシをバイパス - カンマ区切り形式
export NO_PROXY="localhost,192.168.1.1,example.com,.example.com"
# すべてのリクエストのプロキシをバイパス
export NO_PROXY="*"
```

小文字のバリアントも機能し、Claude Code は `https_proxy`、`HTTPS_PROXY`、`http_proxy`、`HTTP_PROXY` の順序で設定されている最初のものを使用します。

Claude Code は `localhost`、`::1`、または `127.0.0.0/8` への WebSocket 接続をプロキシ経由で送信しないため、`NO_PROXY` でこれらのループバックエントリは不要です。

注釈:

Claude Code は SOCKS プロキシをサポートしていません。

39.1.2 基本認証

プロキシが基本認証を必要とする場合は、プロキシ URL に認証情報を含めます。

```
export HTTPS_PROXY=http://username:password@proxy.example.com:8080
```

警告:

スクリプトにパスワードをハードコーディングすることは避けてください。代わりに環境変数またはセキュアな認証情報ストレージを使用してください。

Tip:

高度な認証（NTLM、Kerberos など）が必要なプロキシの場合は、認証方法をサポートする LLM Gateway サービスの使用を検討してください。

39.2 CA 証明書ストア

デフォルトでは、Claude Code はバンドルされた Mozilla CA 証明書とオペレーティングシステムの証明書ストアの両方を信頼しています。OS ストアを読み取るには、`tls.getCACertificates` を備えたランタイムが必要です。ネイティブインストーラーは常にこれを備えており、npm インストールは Node 22.15 以降が必要です。古い Node バージョンでは、バンドルされたセットと `NODE_EXTRA_CA_CERTS` のみが適用されます。エンタープライズ TLS インスペクションプロキシは、ルート証明書が OS 信頼ストアにインストールされており、ランタイムがそれを読み取ることができる場合、追加の設定なしで動作します。

`CLAUDE_CODE_CERT_STORE` はカンマ区切りのソースリストを受け入れます。認識される値は、Claude Code に付属する Mozilla CA セットの場合は `bundled`、オペレーティングシステムの信頼ストアの場合は `system` です。デフォルトは `bundled,system` です。

バンドルされた Mozilla CA セットのみを信頼するには：

```
export CLAUDE_CODE_CERT_STORE=bundled
```

OS 証明書ストアのみを信頼するには：

```
export CLAUDE_CODE_CERT_STORE=system
```

注釈:

CLAUDE_CODE_CERT_STORE には、専用の settings.json スキーマキーがありません。~/.claude/settings.json の env ブロック、またはプロセス環境で直接設定してください。

39.3 カスタム CA 証明書

エンタープライズ環境でカスタム CA を使用している場合は、Claude Code をそれを直接信頼するように設定します。

```
export NODE_EXTRA_CA_CERTS=/path/to/ca-cert.pem
```

39.4 mTLS 認証

エンタープライズ環境でクライアント証明書認証が必要な場合：

```
# 認証用のクライアント証明書
export CLAUDE_CODE_CLIENT_CERT=/path/to/client-cert.pem

# クライアント秘密鍵
export CLAUDE_CODE_CLIENT_KEY=/path/to/client-key.pem

# オプション：暗号化された秘密鍵のパスフレーズ
export CLAUDE_CODE_CLIENT_KEY_PASSPHRASE="your-passphrase"
```

Claude Code は起動時に証明書とキーファイルを読み込み、管理設定の env ブロックを組織が変更する場合など、設定を適用するたびに再度読み込みます。

証明書とキーをローテーションするには、同じパスのファイルを置き換えます。Claude Code は実行中のセッションで再起動なしに置き換えを検出します。接続リセットや TLS ハンドシェイクエラーなどの接続レベルのエラーで API リクエストが失敗した場合、両方のファイルを再度読み込み、新しいペアでリクエストを再試行します。v2.1.232 より前では、Claude Code は接続エラー時に再度読み込まず、次に設定を適用するか再起動するまで既に読み込まれたペアを保持していました。

Claude Code はファイルの変更を監視するのではなく、失敗したリクエストに応答してファイルを再度読み込みます：

- **タイミング**：Claude Code はファイルを置き換えた時点では何もしません。適格な失敗後の再試行時、または設定を適用した後の次のリクエスト時のいずれか早い方で新しいペアを提示します。
- **ゲートウェイの拒否**：Claude Code は、ゲートウェイが古いペアの受け入れを停止した後に接続をリセットするか TLS ハンドシェイクを拒否した場合に再度読み込みます。ゲートウェイがハンドシェイクを完了して HTTP エラーで応答した場合は再度読み込みません。その場合、Claude Code は次に設定を適用するか再起動するときに新しいペアを読み込みます。

- **半分書き込まれたローテーション**：Claude Code がローテーション中に再度読み込む場合（証明書とキーが互いに一致しないなど）、前のペアを保持し、次の失敗時に再度読み込みます。
- **OTLP テレメトリエクスポーター**：Claude Code は**エクスポーター** (ページ 941) が最初に使用したときに読み込んだ証明書を保持するため、ローテーションされた証明書がテレメトリコレクターに到達するには Claude Code を再起動してください。
- **リロードをオフにする**：`CLAUDE_CODE_DISABLE_MTLS_RELOAD_ON_STALE_CONNECTION=1` を設定して接続エラー時の再度読み込みをオフにします。Claude Code はその後、次に設定を適用するか次の起動時にのみローテーションされたファイルを検出します。

Claude Code がローテーションを検出したことを確認するには、**デバッグログで セッションを開始** (ページ 900) し、ログで `Stale connection — reloaded rotated mTLS client material` を探してください。Claude Code は設定を適用する際にローテーションを検出した場合、このラインをログに記録しないため、ラインが見つからないだけではローテーションが失敗したことを意味しません。

Claude Code が次の起動時に既に期限切れのペアを読み込まないように、現在のペアが期限切れになる前にファイルを置き換えてください。

クラウドセッション (ページ 223) では、ホスティング環境が API への接続を管理するため、Claude Code は設定ファイルの env ブロックから来た場合、以下の変数を無視します：

- `CLAUDE_CODE_CLIENT_CERT`
- `CLAUDE_CODE_CLIENT_KEY`
- `CLAUDE_CODE_CLIENT_KEY_PASSPHRASE`
- `NODE_EXTRA_CA_CERTS`
- `NODE_TLS_REJECT_UNAUTHORIZED`
- `CLAUDE_CODE_OAUTH_SCOPES`

Claude Code はセッションのデバッグログで無視された各キーを記録します。

アプリがプロバイダー接続を管理する *Claude Desktop* (ページ 137) セッション（**サードパーティプロバイダー** (ページ 847) の Code タブや Cowork セッションなど）では、Claude Code はこれらの変数とプロキシ変数 `HTTP_PROXY`、`HTTPS_PROXY`、および `NO_PROXY` を 管理設定 と `~/.claude/settings.json` からのみ読み込みます。リポジトリ独自の設定ファイルではこれらを無視するため、チェックアウトされたリポジトリはアプリから認証情報が来るセッションの TLS またはプロキシパスをリダイレクトできません。claude.ai を通じてサインインしたローカル、SSH、または WSL Code タブセッションでは、アプリは接続を管理せず、Claude Code はすべての設定スコープからこれらの変数を読み込みます（任意のターミナルセッションと同様）。**クラウドセッション** (ページ 223) はどこから開始しても上記のクラウドセッションルールに従います。v2.1.217 より前では、アプリが接続を管理していた場合、Claude Code はすべての設定ファイルでこれらの変数を無視していました。

39.5 設定を確認する

通常、プロキシアドレスが間違っているか、証明書パスが不正かは、後のリクエストで 接続エラーまたは証明書エラー が発生することで判明します。Claude Code はこれらの設定のほとんどを読み込む際に検証しないためです。起動時にチェックされる唯一の設定はプロキシ URL です。`http://` スキームが欠落し

ているなど、値を解析できない場合、Claude Code は修正する必要がある変数を名前に含むエラーで起動を停止します。

リクエストを送信する前に設定が読み込まれたことを確認するには、デバッグログを有効にして Claude Code を起動します。

```
claude --debug
```

デバッグ出力はターミナルではなく `~/.claude/debug/<session-id>.txt` に、または `--debug-file <path>` で設定したパスに出力されます。ログで、各ファイルが読み込まれたことを確認する行を探します。

```
CA certs: Appended extra certificates from NODE_EXTRA_CA_CERTS (/etc/ssl/certs/corp-
↪ca.pem)
mTLS: Loaded client certificate from CLAUDE_CODE_CLIENT_CERT
mTLS: Loaded client key from CLAUDE_CODE_CLIENT_KEY
```

Claude Code がこれらのファイルの 1 つを読み込めない場合、ログには代わりに `Failed to read` または `Failed to load` の行と理由が表示されます。

対話型セッションで `/status` を実行して、これらの行を確認することもできます。

- **Proxy**: アクティブなプロキシ URL を表示し、解析できない値を無効で無視とマークします。
- **mTLS client cert** および **mTLS client key**: ファイルが読み込まれた場合にのみ表示されるため、行がない場合は読み込みが失敗し、デバッグログに理由が記載されています。
- **Additional CA cert(s)**: ファイルが読み込まれたことを確認せずに `NODE_EXTRA_CA_CERTS` パスを表示するため、デバッグログでこれを確認します。

39.6 バックグラウンドエージェントにネットワーク設定を適用する

バックグラウンドエージェントは、それらをディスパッチしたターミナルの内部では実行されません。ユーザーごとのスーパーバイザープロセスがオンデマンドで起動し、シェルより長く存続し、すべての `claude agents`、`--bg`、および `/background` セッションをホストします。バックグラウンドセッションがどのようにホストされるかを参照してください。これにより、このページの設定がこれらのセッションに到達する方法が変わります。

39.6.1 シェルではなく設定でネットワーク変数を設定する

スーパーバイザーはすべてのターミナルで共有される 1 つのプロセスです。これは最初にそれを起動するシェルの環境を継承し、OS がインストールしたスーパーバイザーはシェル環境をまったく受け取りません。プロキシ、CA パス、または mTLS 変数をシェルにのみエクスポートする場合、そのシェルがスーパーバイザーをコールドスタートしたときはバックグラウンドエージェントに到達しますが、別のシェルが起動した場合は静かに到達しません。

代わりに、`~/.claude/settings.json` の `env` ブロックまたはマネージド設定 (ページ 711) に同じ変数を配置してください。このページのすべての変数をそこで設定でき、設定はすべてのマシンのすべてのバックグラウンドセッションに到達する唯一の設定です。

39.6.2 企業ランチャーを設定として構成する

一部の組織では、すべての Claude Code プロセスが、サンドボックス化、ネットワーク制御、または認証情報インジェクションを適用する企業ランチャーを通じて起動することが必要です。スーパーバイザーとそのワーカーは、PATH で claude を検索するのではなく、固定パスから Claude Code を起動するため、すべてのバックグラウンドエージェントは PATH の前に配置したラッパーをバイパスします。

processWrapper 設定を設定して、スーパーバイザー、そのワーカー、およびランチャーがカバーするものの下にリストされている他のバックグラウンドプロセスをランチャーでプレフィックスします。同等の CLAUDE_CODE_PROCESS_WRAPPER 環境変数は、両方が設定されている場合に優先され、同じルールの対象となります。マネージド設定または ~/.claude/settings.json を通じて配信し、シェルエクスポートではありません。企業ランチャーの背後で Claude Code を実行するは、ランチャーが満たす必要があるコントラクト、それが到達するもの、到達しないもの、およびロールアウト方法をカバーしています。

注釈:

既に実行中のスーパーバイザーは、起動時に開始した起動設定を保持します。ランチャー設定をデプロイした後、claude daemon stop --any を実行して、次の claude agents または --bg がそれを尊重するスーパーバイザーを起動するようにします。インストール済みサービスは --any なしで claude daemon stop を実行します。

39.7 ストリーミングアイドルウォッチドッグ

Claude Code は 4 つの独立したタイマーを実行し、ストリーミングモデルレスポンスが静止状態になるとそれを中止するため、接続が切れた場合はハングするのではなく失敗して再試行されます。最初のバイト期限はレスポンスヘッダーの到着を待つ間をカバーし、レスポンスが到着する前です。他の 3 つのタイマーはそれぞれ、ライブレスポンスの異なるシグナルを監視します。

タイマー	中止する条件	実行対象	デフォルトタイムアウト
最初のバイト期限	Claude Code がリクエストを送信した後、レスポンスヘッダーが到着しない	直接 Anthropic API および Claude Platform on AWS（HTTPS プロキシを含む）。ただし、ANTHROPIC_BASE_URL または ANTHROPIC_AWS_BASE_URL が gateway を経由する場合は除外。Amazon Bedrock では CLAUDE_ENABLE_BYTE_WATCHDOG_BEDROCK=1 でオプトイン。Google Cloud の Agent Platform または Microsoft Foundry では実行されない	直接 Anthropic API では 180 秒、その他では 300 秒、加えてリクエストボディの 32KB ごとに 1 秒

次のページに続く

表 1 – 前のページからの続き

タイマー	中止する条件	実行対象	デフォルトタイムアウト
イベントレベルウォッチドッグ	レスポンスイベントが解析されない。バイトレベルウォッチドッグが Amazon Bedrock 以外の接続で実行される場合、キープアライブピングを含む到着バイトもこのウォッチドッグをリセットし、解析されたイベントがない状態で約 5 分間まで実行される	すべてのプロバイダー	300 秒
バイトレベルウォッチドッグ	ワイヤー上にバイトが到着しない（SSE キープアライブピングを含む）	直接 Anthropic API、Claude Platform on AWS、および gateway 接続（カスタム `ANTHROPIC_BASE_URL` を含む）。Amazon Bedrock の `vnd.amazon.eventstream` レスポンスでは `CLAUDE_ENABLE_BYTE_WATCHDOG_BEDROCK=1` でオプトイン。Google Cloud の Agent Platform または Microsoft Foundry では実行されない	直接 Anthropic API では 180 秒、その他では 300 秒
ボディアイドルタイムアウト	5 分間バイトが到着しない	直接 Anthropic API、Claude Platform on AWS、および `CLAUDE_ENABLE_BYTE_WATCHDOG_BEDROCK=1` が設定された Amazon Bedrock 以外のプロバイダー。ただし、`API_FORCE_IDLE_TIMEOUT` がこれを変更する場合は除外	5 分

`CLAUDE_ENABLE_BYTE_WATCHDOG_BEDROCK=1` を設定した場合、バイトレベルウォッチドッグは Bedrock のボディアイドルタイムアウトを置き換え、並行して実行されるのではなく実行されます。その後、`CLAUDE_STREAM_IDLE_TIMEOUT_MS` は Bedrock ストリームが接続を切断されたものとして扱われるまで静止状態を保つことができる期間も制御し、以下にリストされた制限内です。到着バイトは Bedrock のイベントレベルウォッチドッグをリセットしません。デバッグログが有効な場合、各 Bedrock ストリームは `wire-heartbeat: _chunkTimes absent` で始まるデバッグメッセージをログに記録します。

これらの変数を使用してタイマーを設定します。各変数の詳細は 環境変数リファレンス に記載されてい

ます。

- CLAUDE_ENABLE_STREAM_WATCHDOG および CLAUDE_ENABLE_BYTE_WATCHDOG は、テーブルにリストされている接続内で、対応するウォッチドッグを 1 でオンまたは 0 でオフに強制します。どちらの変数もウォッチドッグをカバーしていない接続タイプに拡張しません。CLAUDE_ENABLE_BYTE_WATCHDOG を 0 に設定すると、最初のバイト期限もオフになります。
- CLAUDE_STREAM_IDLE_TIMEOUT_MS は両方のウォッチドッグのタイムアウトを設定します。Claude Code は 5 分未満の値を 5 分に引き上げ、バイトレベルウォッチドッグの値を 30 分でキャップします。
- CLAUDE_BYTE_STREAM_IDLE_TIMEOUT_MS はイベントレベルウォッチドッグを変更せずにバイトレベルウォッチドッグのタイムアウトを設定し、10 秒から 30 分の間にクランプされ、そのウォッチドッグについて CLAUDE_STREAM_IDLE_TIMEOUT_MS より優先されます。
- CLAUDE_STREAM_FIRST_BYTE_TIMEOUT_MS は最初のバイト期限を直接設定します。設定しないままにすると、Claude Code はバイトレベルウォッチドッグのタイムアウトを使用するため、CLAUDE_STREAM_IDLE_TIMEOUT_MS および CLAUDE_BYTE_STREAM_IDLE_TIMEOUT_MS も期限を変更します。クランプ、アップロード許容量、API_TIMEOUT_MS キャップ、および no-response 中止後の再試行待機時間については、API からのレスポンスなし を参照してください。
- API_FORCE_IDLE_TIMEOUT を 0 に設定するとボディアイドルタイムアウトがオフになり、1 に設定するとすべてのプロバイダーでオンになります。ウォッチドッグはそれとは独立して実行されるため、ストリームをそれらのしきい値より長く一時停止させるには、それらも引き上げるか無効にしてください。

ウォッチドッグが停止したストリームを中止すると、Claude Code は中止を mid-stream 失敗として扱い、表示される内容はレスポンスがどこまで進んだかによって異なります。Claude Code はリクエストを再試行するか、エラーで ターンを終了し、完了した出力を保持して 不完全なレスポンス通知 を表示するか、ターンを正常に終了します。自動再試行 は各結果がどこに適用されるかを説明しています。

非対話型セッション (ページ 695) では、およびいかなるセッションでもサブエージェントのレスポンスについて、Claude Code は最初に Claude にカットオフレスポンスを続行するよう促す場合があります。その通知のエントリ は、それがいつ行われるか、およびいつ通知が表示されるかを説明しています。

最初のバイト期限が発火すると、レスポンスが開始されていないため、保持する部分的な出力はありません。Claude Code がリクエストを再送信する方法と、ターンが代わりに終了する場合については、API からのレスポンスなし を参照してください。

39.8 ネットワークアクセス要件

Claude Code は以下の URL へのアクセスが必要です。プロキシ設定とファイアウォールルールでこれらをホワイトリストに登録してください。特にコンテナ化された環境や制限されたネットワーク環境では重要です。初回実行時のセットアップ接続確認は、api.anthropic.com または platform.claude.com に到達できない場合、ここを指します。確認メッセージと復旧手順については、Anthropic サービスに接続できないを参照してください。

URL	必要な用途
api.anthropic.com	Claude API リクエスト（WebFetch の**ドメイン安全性チェック** (ページ 1111)、機能フラグ取得、テレメトリイベントログを含む）
claude.ai	claude.ai アカウント認証
claude.com	claude.ai アカウントサインインがブラウザで claude.com ページを開き、claude.ai にリダイレクトされます。事前承認された WebFetch ドキュメント検索も CLI からこのホストに到達します
platform.claude.com	Anthropic Console アカウント認証。OAuth トークン交換、更新、失効も claude.ai アカウントではこのホストに送信されるため、Console と claude.ai の両方のサインインに必要です
mcp-proxy.anthropic.com	*claude.ai* **からの** *MCP* **コネクタ** (ページ 396)（組織管理者が設定するコネクタを含む）。コネクタトラフィックはこのプロキシを経由してルーティングされます。コネクタは claude.ai 認証ユーザーではデフォルトで有効です。Claude Code がそれらを取得するのを停止するには、ENABLE_CLAUDEAI_MCP_SERVERS=false または disableClaudeAiConnectors 設定を設定してください
downloads.claude.ai	プラグイン実行ファイルダウンロード、ネイティブインストーラー、ネイティブ自動更新プログラム、更新バージョンチェック
storage.googleapis.com	/plugin に表示されるプラグインインストール数とメタデータ
storage.googleapis.com	2.1.116 より前のバージョンのネイティブインストーラーとネイティブ自動更新プログラム
registry.npmjs.org	プラグインインストール（npm ソースプラグインパッケージの取得とプラグインの Node.js パッケージ依存関係のインストール）、npx で起動された MCP サーバー、および Claude Code 自体の npm と bun インストール用パッケージレジストリ
bridge.claudeusercontent.com	*Chrome* **の** *Claude* (ページ 213) 拡張機能 WebSocket ブリッジ
*.frame.claudeusercontent.com	Artifact コンテンツ読み取り。CLI は Claude がアーティファクトを開いたときにこのホストからアーティファクトのファイルを取得し、Artifact ツールがアカウントで利用可能な場合のみです。ツールをオフにしてこの要件を削除するには、"enableArtifact": false または CLAUDE_CODE_DISABLE_ARTIFACT=1 を設定してください
github.com	GitHub ホスト型プラグインマーケットプレイスとプラグインのクローン（公式 Anthropic マーケットプレイスを含む）。HTTPS または SSH 経由。GitHub owner/repo ソースを HTTPS のみでクローンするには、CLAUDE_CODE_PLUGIN_PREFER_HTTPS=1 を設定してください

次のページに続く

表 2 – 前のページからの続き

URL	必要な用途
raw.githubusercontent.com	/release-notes のチェンジログフィード。対話型セッションでは、Claude Code はキャッシュされたチェンジログがまだ実行中のバージョンをカバーしていない場合（更新後の初回起動など）、スタートアップ時にバックグラウンドでそれを取得します。非対話型およびクラウドセッションは決してそれを取得しません
*-review.googlesource.com	googlesource.com チェックアウトでの Gerrit 変更検索。Claude Desktop Code タブセッションが **信頼済み** (ページ 1064) チェックアウトで開始または再開され、その origin が googlesource.com ホストである場合、Claude Code はそのホストの -review サーバーに匿名で、HEAD の Change-Id に一致するオープン変更を 1 回（開始または再開ごと）要求します。他のセッションタイプはこの検索をスキップし、他の Gerrit ホストには接続されません。オプション：CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC で無効化
http-intake.logs.us5.datadoghq.com	運用テレメトリイベント。CLI が Anthropic API を直接使用する場合にのみ送信され、Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry では送信されません。オプション：*DISABLE_TELEMETRY* (ページ 1109) または DO_NOT_TRACK で無効化
browser-intake-us5-datadoghq.com	運用エラーレポート。CLI が Anthropic API を直接使用し、サーバー側のロールアウトゲートがそれらを有効にする場合に送信されます。オプション：DISABLE_ERROR_REPORTING または DISABLE_TELEMETRY で無効化。**テレメトリサービス** (ページ 1109) を参照してください
formulae.brew.sh	Homebrew インストールでのバージョンチェック更新。他のインストール方法はこのホストに接続しません
code.claude.com	組み込みの claude-code-guide エージェントによる Claude Code ドキュメント検索と事前承認された WebFetch リクエスト。このホストをブロックするとドキュメント検索のみに影響します

npm または独自のバイナリ配布を通じて Claude Code をインストールする場合、エンドユーザーはネイティブインストーラーと自動更新プログラムの downloads.claude.ai の使用は不要ですが、npm と bun インストールは組織がミラーしない限り、パッケージレジストリ registry.npmjs.org が必要です。テーブル内の他の用途はインストール方法に関係なく適用されます。

2 つの Datadog インテークホストはオプションの運用テレメトリのみを実行し、CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC を設定するとその両方が無効になります。サードパーティプロバイダーのセッションは、プラットフォームが CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST を設定し、テレメトリメトリクスがデフォルトでオンになっている場合でも、これらのホストに送信されません。Claude Code が送信するすべてのもの、およびホワイトリストを最終化する前にそれを無効にする方法については、**テレメトリサービス** (ページ 1109) を参照してください。

Amazon Bedrock (ページ 863)、*Google Cloud の Agent Platform* (ページ 881)、*Microsoft Foundry* (ページ 891)、

またはサインイン済みの Claude apps gateway セッションを使用する場合、モデルトラフィックと認証は `api.anthropic.com`、`claude.ai`、または `platform.claude.com` ではなく、プロバイダーまたはゲートウェイに送信されます。WebFetch ツールは、**設定** (ページ 711) で `skipWebFetchPreflight: true` を設定しない限り、**ドメイン安全性チェック** (ページ 1111) のために `api.anthropic.com` を呼び出します。

`ANTHROPIC_BASE_URL` を使用して *LLM* **ゲートウェイ** (ページ 841) を経由してルーティングする場合、高速モードの可用性チェックはゲートウェイベース URL ではなく `api.anthropic.com` を呼び出します。チェックは設定された HTTP プロキシを尊重するため、ネットワークブロックが原因の場合、プロキシ内の `api.anthropic.com` のホワイトリストエントリが修正です。ネットワークブロックはホストがプロキシを通じても到達不可能な場合にのみチェックに失敗し、高速モードは接続エラーを報告します。チェックがゲートウェイ発行の認証情報を提示し、Anthropic がそれを拒否する場合も同じ接続エラーが表示されます。ホワイトリストはそこでは役に立ちません。何もブロックされていないためです。プロキシと LLM ゲートウェイの背後で高速モードを使用するを参照して、それを復元する変数を確認してください。

39.8.1 組織 IP ホワイトリストとプロキシ出力

組織で Claude に対して IP ホワイトリストが有効になっている場合、`bridge.claudeusercontent.com` を `claude.ai` および `api.anthropic.com` と同じプロキシ出力を経由してルーティングしてください。たとえば、同じ Zscaler アプリセグメントまたは Netskope ステアリングポリシーに配置することで実現できます。そのようにルーティングできない場合は、プロキシがそのホストに使用する出力アドレスを組織の IP ホワイトリストに追加してください。ただし、そのアドレスが組織専用の場合のみです。共有プロキシ出力範囲は、プロキシベンダーの他の顧客も許可します。

Anthropic は、到着元のアドレスを使用して、組織の IP ホワイトリストに対して `bridge.claudeusercontent.com` への接続をチェックします。プロキシがそのホストのトラフィックを、そのホワイトリストにないアドレスを通じて送信する場合、Claude Code の残りの部分は機能していても、*Chrome* **の** *Claude* (ページ 213) 拡張機能に接続できません。

39.8.2 GitHub ホワイトリストとファイアウォール

Anthropic ホスト環境での *Web* **上の** *Claude Code* (ページ 223) と Code Review は Anthropic 管理インフラストラクチャからリポジトリに接続します。自己ホスト環境のセッションはネットワーク内から接続します。ただし、ランナーが Anthropic git プロキシにオプトインする場合は除きます。これは Anthropic 側から取得します。

GitHub Enterprise Cloud 組織が IP アドレスでアクセスを制限している場合、インストール済み GitHub Apps の IP ホワイトリスト継承を有効にし、Anthropic のアウトバウンド IP アドレスのホワイトリストエントリを追加してください。継承は Claude GitHub App がインストールとして行うリクエストのみをカバーし、ユーザーの代わりに行うリクエストはカバーしません。他のファイアウォールについては、Anthropic API IP アドレスを参照してください。

ファイアウォールの背後にある自己ホスト GitHub Enterprise Server インスタンスの場合、Anthropic のアウトバウンド IP アドレスをホワイトリストに登録して、Anthropic インフラストラクチャが GHES ホストに到達してリポジトリをクローンし、レビューコメントを投稿できるようにしてください。自己ホスト環境のセッションはネットワーク内から GHES ホストに到達するため、その露出は Anthropic ホストセッション、リポジトリピッカーなどのホスト前セッションフロー、および Anthropic git プロキシにオプトインする自己ホストランナーにのみ適用されます。これは Anthropic 側から取得します。SCM コネクタは利用できないため、ホスト前セッションフローはネットワーク内でのみルーティング可能な GHES ホストに到達できません。

39.8.3 デスクトップと claude.ai

前述のテーブルはスタンドアロン CLI をカバーしています。Claude Desktop アプリと browser の claude.ai は、アプリケーションコードとユーザーコンテンツを追加の Anthropic CDN ホストから読み込みます。これには `assets-proxy.anthropic.com` と、これらのアプリで artifacts を提供する他の `*.claudeusercontent.com` オリジンが含まれます。`claude.ai` を許可しながらこれらのホストをブロックすると、エラーではなく空白ページが表示されます。Desktop ページの **ネットワークアクセス要件** (ページ 164) を参照してください。

Claude Desktop と claude.ai は、一部のコネクタが提供する MCP Apps など、会話内の一部のツール結果をインタラクティブなウィジェットとしてレンダリングすることもあります。これらのウィジェットは `claudemcpcontent.com` の生成されたサブドメインから読み込まれるため、ワイルドカードをそのまま残して `*.claudemcpcontent.com` を許可してください。これをブロックしても、アプリの残りの部分は引き続き機能しますが、これらのウィジェットは読み込まれません。

アーティファクトのフォントとライブラリ用のサードパーティホスト

Google Fonts からタイプフェイスを読み込む artifact は、`fonts.googleapis.com` と `fonts.gstatic.com` もリクエストします。両方のホストはオプションです。それらをブロックすると、artifacts はフォールバックタイプフェイスでレンダリングされます。フォントリクエストが即座に失敗するように、高速拒否でブロックしてください。ページの最初のレンダリングを遅延させるのではなく。

Artifacts は React やチャートパッケージなどの JavaScript ライブラリを `cdnjs.cloudflare.com`、`cdn.jsdelivr.net`、`cdn.tailwindcss.com`、`code.jquery.com`、および `unpkg.com` から読み込むことができ、他の外部ホストからは読み込めません。これらのホストをブロックすると、ライブラリに依存する artifact の部分は機能しません。ブロックされたフォントとは異なり、ブロックされたライブラリにはフォールバックがありません。ここでも高速拒否でブロックしてください。ブロックされたライブラリリクエストが即座に失敗するように、タイムアウトするまでハングするのではなく。

39.9 追加リソース

- **設定ファイルと優先順位** (ページ 711)
- 環境変数リファレンス
- **トラブルシューティングガイド** (ページ 37)

第VII部

運用

第40章　コストを効果的に管理する

トークン使用量を追跡し、チームの支出制限を設定し、コンテキスト管理、モデル選択、拡張思考設定、前処理フックを使用して Claude Code のコストを削減します。

Claude Code は API トークン消費によって課金されます。サブスクリプションプラン価格（Pro、Max、Team、Enterprise）については、claude.com/pricing を参照してください。開発者あたりのコストは、モデル選択、コードベースサイズ、複数インスタンスの実行や自動化などの使用パターンに基づいて大きく異なります。

エンタープライズ展開全体では、平均コストは開発者 1 人あたり 1 日約 13 ドル、開発者 1 人あたり月額 150〜250 ドルで、90% のユーザーの 1 日あたりのコストは 30 ドル以下です。自分のチームの支出を見積もるには、小規模なパイロットグループから始めて、以下の追跡ツールを使用してベースラインを確立してから、より広い展開を行ってください。

このページでは、**コストを追跡する方法** (ページ 911)、**チームのコストを管理する方法** (ページ 915)、および **トークン使用量を削減する方法** (ページ 919) について説明します。

40.1 コストを追跡する

40.1.1 /usage コマンドを使用する

注釈:

/usage のセッションブロックは API トークン使用量を表示し、API ユーザーを対象としています。Claude Max および Pro サブスクライバーはサブスクリプションに使用量が含まれているため、セッションコスト数値は請求目的では関連がありません。サブスクライバーは同じ画面でプラン使用量バー、アクティビティ統計、および使用量の内訳を表示します。

/usage の上部のセッションブロックは、現在のセッションの詳細なトークン使用統計を表示します。Claude Code はトークン数からドル数値をローカルで計算します。ただし、`modelPricing` テーブルが有効な場合は除きます。管理者は組織の管理設定でテーブルを設定し、数値が契約レートを使用するようにします。テーブルが有効な場合、`Total cost` 行には `at your organization's configured rates` という注記が付きます。この数値は推定値であるため、権限のある請求については Claude Console の使用量ページを参照してください。

```
Total cost:            $0.55
Total duration (API):  6m 20s
Total duration (wall): 6h 33m 10s
Total code changes:    0 lines added, 0 lines removed
Usage by model:
   claude-sonnet-4-6:  1.2k input, 5.3k output, 940.0k cache read, 50.0k cache write (
↪$0.55)
```

これらの合計は `/clear` が新しいセッションを開始するとリセットされるため、次のセッションの合計コストは $0 から始まります。v2.1.211 より前では、`/clear` 全体で累積し続け、Claude Code プロセスの存続期間中に蓄積されていました。

1.1 × データレジデンシーレート で請求される Claude API からの応答の場合、Claude Code はその応答のトークンのリスト価格に 1.1 を乗じてセッションコスト数値に反映します。Claude Code は **ステータス行のコストフィールド** (ページ 803) に同じ合計を報告し、*`--max-budget-usd`* (ページ 51) と比較します。v2.1.239 より前では、Claude Code はこれらの応答に 1.1 × を適用しなかったため、セッションコスト数値は請求額より低くなっていました。

プロンプトキャッシュ統計

メイン会話の最初の API 応答の後、Claude Code はセッションブロックに `Prompt cache (main)` 行も追加します。これはセッションの プロンプトキャッシュ 使用量をまとめたものです。リクエスト数、キャッシュから提供された入力トークンのシェア、キャッシュミス、およびキャッシュが現在ウォーム状態かどうかです。Claude Code v2.1.251 以降が必要です。

```
Prompt cache (main):  14 requests ·  91% of input tokens from cache ·  2 misses␣
↪(last 6m 10s ago, 310.2k tokens re-cached) ·  1 expected rebuild (compaction or␣
↪tool-result clearing) ·  warm (1h TTL, last activity 40s ago)
```

行のミス、予想される再構築、およびウォーム状態またはコールド状態の部分は以下を意味します。

- **ミス**: キャッシュが既に保持していたコンテンツを再処理したリクエスト。最後のミスの時刻と、それらのリクエストがキャッシュに書き戻したトークン数が表示されます。キャッシュを無効化するアクション は通常の原因をリストしています。Claude Code が最後のミスの可能性のある原因を特定できる場合、行はそれも名前を付けます。例えば `likely cause: tool definitions changed` のようにです。可能性のある原因テキストには Claude Code v2.1.260 以降が必要です。
- **予想される再構築**: Claude Code が会話を再度書き直した場合。圧縮 またはコンテキストから古いツール結果をクリアすることで、同じ種類のミスを予想される再構築としてカウントします。この部分は、少なくとも 1 つの予想される再構築が発生した後にのみ表示されます。
- **ウォーム状態またはコールド状態**: キャッシュされたプレフィックスが キャッシュ有効期間 内にあるかどうか。有効な TTL が表示されます。キャッシュがコールド状態の場合、行はセッションがアイドル状態だった期間を表示します。API がキャッシュトークンを報告していない場合、行は代わりに `no prompt caching reported by the API` で終わります。

カウントは API の応答のキャッシュトークンフィールドから取得されるため、行はすべてのプロバイダーとゲートウェイで機能します。メイン会話のみをカバーし、サブエージェントはカバーしません。`/clear` はセッションブロックの残りとともにリセットします。

ステータス行スクリプトは *prompt_cache* **オブジェクト** (ページ 797) から同じ数値を読み取ることができます。

プラン使用量の内訳

Pro、Max、Team、または Enterprise プランでは、`/usage` はプラン制限に対してカウントされるものの内訳も表示します。

- **属性**: スキル、サブエージェント、プラグイン、および個別の MCP サーバーに属性付けされた最近の使用量。それぞれが合計のパーセンテージとして表示されます。 MCP サーバーのシェアは、そのツール結果の 1 つを消費したリクエストのみをカウントします。v2.1.222 より前では、MCP サーバーへの 1 回の呼び出しの後、Claude Code はその後のすべてのリクエストをそのサーバーに属性付けし、そのシェアを過大評価していました。
- **動作フラグ**: 長いコンテキストやキャッシュミスなどの動作。最近の使用量の 10% 以上を占める場合にフラグが付けられます。
- **ループ**: 最近実行された最も負荷の高い /loop またはその他のスケジュール済みタスク の行。合計トークン数で順序付けられ、残りのカウントが表示されます。Claude Code は各タスクの実行頻度、実行回数、合計トークンと実行ごとのトークン、および最後の実行時刻を報告します。Claude Code はタスクのプロンプトで行をキーにするため、停止して再作成したループは 1 つの行のままです。Claude Code v2.1.242 以降が必要です。

`d` または `w` を押して、過去 24 時間と過去 7 日間を切り替えます。数値は概算であり、このマシン上のローカルセッション履歴から計算されるため、他のデバイスまたは claude.ai からの使用量は含まれていません。

VS Code **拡張機能** (ページ 184) では、属性シェアと動作フラグが Account & usage ダイアログに Day および Week トグルとともに表示されます。ループ行は含まれません。

使用量クレジット支出を確認する

`/usage` は **使用量クレジット** (ページ 914) がオンの間、使用量クレジット行も表示します。行に表示される内容はプランによって異なります。

- **Pro および Max**: 月間支出制限を設定している場合、その月間支出制限に対して測定された現在の月の支出。制限を設定していない場合、行は `Unlimited` を表示し、支出数値は表示されません。
- **Team および Enterprise**: 組織が設定した **制限** (ページ 916) が適用される場合、現在の月の自分の支出。組織全体をカバーする制限は行に表示されません。自分の制限がない場合、行は制限なしで支出を表示します。使用量クレジットがオフの場合、`/usage` は使用量クレジット行を表示しません。

支出制限がある場合、使用量クレジットがオンになるとすぐに行が表示され、最初に使用量クレジットを支出するまで 0% を表示します。v2.1.236 より前では、`/usage` は Pro および Max プランでのみ行を表示し、支出制限のある行は何かを支出するまで非表示のままでした。

使用量リクエストが失敗した場合

プラン制限のリクエストが失敗した場合（ほとんどの場合、使用量エンドポイントがレート制限されているため）、`/usage` は過去 60 分以内にこのマシンで読み込んだ最後の使用量バーを表示し、そのデータがいつ取得されたかを示す `Showing last-known usage` ノートが表示されます。`r` を押して再試行します。再試行が成功すると、最後に認識されたバーが新しいデータに置き換わります。過去 60 分以内のスナップショットがない場合、`/usage` は使用量エンドポイントがレート制限されていることを報告し、同じ再試行ショートカットを提供します。v2.1.208 より前では、使用量をまだ読み込んでいないセッションでレート制限されたリクエストは常にバーなしでエラーを表示していました。

40.1.2 使用パターンを分析する

`/insights` を実行して、使用したトークン数ではなく、作業方法に関するレポートを取得します。このマシン上の最近のセッションを分析し、作業内容、誤解されたリクエストやバグのあるコードなどの摩擦点、および Claude Code をより効果的に使用するための提案をカバーする HTML レポートを作成します。1 回の実行は、まだ見たことのない最大 200 セッションを分析し、非常に短いセッションはスキップします。セッションが除外される場合、レポートヘッダーは分析されたカウントを括弧内の合計とともに表示します。例えば `200 sessions (412 total)` のようにです。

auto mode がセッションで利用可能で、最近のセッションのほとんどがそれなしで実行された場合、レポートはそれらのセッション全体で auto mode が処理できたであろう権限プロンプトの数の推定値も含めることができます。

Claude Code は最新のレポートを `~/.claude/usage-data/report.html` に書き込み、各実行のタイムスタンプ付きコピーを同じディレクトリに保存するため、以前のレポートは上書きされません。Claude Code はセッションデータの残りと同じスケジュールでレポートを削除します。起動時に、`cleanupPeriodDays` より古いファイルを削除します。デフォルトは 30 日です。

任意のプランおよび任意のプロバイダーで `/insights` を実行できます。分析は通常のセッションと同じプロバイダーとアカウントを通じて実行され、トークンはプランまたは API 使用量に対してカウントされます。他のデバイスおよび claude.ai からのセッションは含まれていません。

40.1.3 サブスクリプションに使用量クレジットを追加する

使用量クレジット を使用すると、プランの使用量制限を超えて作業を続けることができます。これらを管理するには、`/login` を通じて claude.ai サブスクリプションにサインインした後、`/usage-credits` を実行します。このコマンドは API キー認証では利用できません。セルフサービス Enterprise 組織、Enterprise トライアル、および AWS Marketplace を通じて請求される Enterprise 組織では、コマンドには Claude Code v2.1.248 以降が必要です。以前のバージョンは `Unknown command: /usage-credits` で拒否します。開かれるものはロールによって異なります。

ロール	`/usage-credits` の動作
Pro または Max サブスクライバー	ブラウザで Settings > Usage を claude.ai で開きます。**Usage credits** セクションで、使用量クレジットをオンまたはオフにし、クレジット残高、今月の支出、および月間支出制限を確認できます
請求アクセス権を持つ Team または Enterprise メンバー	組織の使用量設定 Admin settings > Usage をブラウザで開きます
請求アクセス権を持たない Team または Enterprise メンバー	確認を求めてから、組織の管理者にリクエストを送信します。v2.1.211 より前では、Claude Code は確認ステップなしでリクエストを送信していました

請求アクセス権を持たない Team および Enterprise メンバーの場合、確認はインタラクティブセッションでのみ表示されます。`-p` フラグを使用した非インタラクティブモードおよび Remote Control からは、コマンドはリクエストを送信せず、インタラクティブセッションで実行するよう指示します。

以前のリクエストが管理者を待機している間に `/usage-credits` を再度実行した場合、Claude Code はリクエストが既に送信されたことを通知し、重複を送信しません。管理者がリクエストを却下した後、コマ

ンドを再度実行すると新しいリクエストが送信されます。v2.1.222 より前では、却下されたリクエストも新しいリクエストをブロックしていました。

Pro および Max プランでは、支出制限に達しても使用量クレジットがまだ利用可能な場合、Claude Code は制限を引き上げるか削除するよう促し、CLI を離れることなく続行できます。サーバーが変更を拒否した場合、Could not update your spend limit を参照してください。

40.2 組織のコストを管理する

Claude Code にアクセスする方法によって、利用可能なコントロールが異なります。Claude for Teams または Enterprise プラン、Claude Console、またはクラウドプロバイダーです。Teams および Enterprise プランでは、使用量は各メンバーのシート割り当てから引き出されます。Console およびクラウドプロバイダーでは、使用量はトークンごとに組織に請求されます。組織がサインイン方法を混在させている場合、各開発者は認証した方法に従ってメーター化されます。

次の表は、各セットアップを、支出を確認する場所、支出をキャップする場所、およびユーザーごとの数値を取得する方法にマップしています。個別の Pro または Max プランでは、管理する組織がないため、fast mode を含む **サブスクリプションに使用クレジットを追加** (ページ 914) の下で、独自の使用クレジット支出を追跡してください。

セットアップ	支出を確認	支出をキャップ	ユーザーごとのレポート
Claude for Teams **または** *Enterprise* (ページ 916)	org analytics の支出レポート	管理者設定の支出制限	支出レポート CSV、Enterprise の Enterprise Analytics API
Claude Console（*API*） (ページ 917)	Console 使用状況ページ	ワークスペース支出制限	Console ダッシュボード、Claude Code Analytics API
Amazon Bedrock、*Google Cloud* **の** *Agent Platform*、**または** *Microsoft Foundry* (ページ 918)	クラウド請求コンソール	クラウドの予算コントロール	*OpenTelemetry* (ページ 935) または *LLM gateway* (ページ 841)

OpenTelemetry **エクスポート** (ページ 935) はすべてのセットアップで機能し、ユーザーごとのトークンおよびコストメトリクスをほぼリアルタイムで独自の可観測性スタックにストリーミングする唯一のオプションです。

40.2.1 契約レートで支出をレポートする

デフォルトでは、Claude Code は開発者に表示するすべてのコスト数値を定価で計算するため、組織が契約レートを支払う場合、`/usage`、ステータス行、および OpenTelemetry の数値は請求書と一致しません。一致させるには、`modelPricing` マネージド設定をレートに設定してください。この設定は Claude Code が報告する内容を変更し、Anthropic が請求する内容ではありません。Claude Code v2.1.242 以降が必要です。

1. **契約からレートを取得する**

   契約からのトークンあたり百万単位のレートを入力してください。Claude Code は Claude Console から取得しないため、契約が変更されたときに設定を更新してください。

2. **設定を書き込む**

   定価からのフラット割引に対して `multiplier` を 1 未満に設定するか、マークアップに対して 1 を超えて設定し、`overrides` の下に各モデルの 4 つのトークンあたりレートをリストするか、またはその両方を実行してください。マークアップには Claude Code v2.1.271 以降が必要です。modelPricing エントリ には形状とペースト可能な例があります。

3. **マネージド設定を通じてデプロイする**

   マネージド設定 として配信してください。サーバー管理設定、MDM ポリシー、`managed-settings.json`、または ポリシーヘルパー。Claude Code はユーザー、プロジェクト、ローカル設定、および `--settings` のキーを無視します。

レートが有効であることを確認するには、マネージド設定を受け取った セッションで `/usage` を実行してください。Session ブロックの `Total cost` 行には `at your organization's configured rates` という注記が付きます。数値はまだ推定値であり、請求書ではありません。`/model` ピッカーのトークンあたり百万単位の価格は定価のままです。

40.2.2 Claude for Teams および Enterprise

Claude for Teams および Enterprise プランでは、各メンバーの Claude Code 使用量は、ローリング 5 時間ウィンドウと週間ウィンドウでリセットされるシート単位の割り当てから引き出されます。割り当ては Claude チャットおよび Cowork と共有され、そのサイズはメンバーの シート層（Standard または Premium）に依存します。コントロールは Claude Console ではなく claude.ai 管理コンソールにあります。

- **支出を確認**: org analytics の支出レポート は、ユーザーごとおよびモデルごとの推定支出を CSV エクスポート付きで表示し、毎日更新されます。レポートは使用クレジット支出をカバーし、使用クレジットがオンになると表示されます。シート割り当て内の使用量はドルでメーター化されません。
- **採用を確認**: analytics ダッシュボード は、日次アクティブユーザー、セッション、および貢献メトリクスを表示し、貢献データの CSV エクスポート付きです。*analytics* **でチーム使用状況を追跡** (ページ 927) を参照してください。
- **支出をキャップ**: シート割り当てはデフォルトの上限です。メンバーがそれを超えて続行できるようにするには、使用クレジット をオンにして、組織、グループ、または個別メンバーレベルで支出制限を設定します。
- **ユーザーごとの数値を取得**: Enterprise プランでは、Enterprise Analytics API は Claude Code を含む Claude サーフェス全体のユーザーごとの使用状況およびコストレポートを返します。Primary Owner は claude.ai/analytics/api-keys で `read:analytics` スコープを持つキーを作成します。Teams プランでは、支出レポート CSV をエクスポートします。これはユーザーごとおよびモデルごとのトークン使用量と推定支出をリストします。

Claude Enterprise 消費ガイド は管理者向けの計画リファレンスです。Claude チャット、Claude Code、および Cowork 全体で消費がどのように異なるかを説明し、予算編成のためのユーザーごとのドル開始点を提供します。コーディングシートのチャットシートより多くの予算を計上してください。各 Claude Code ターンはファイルコンテンツ、ツール呼び出し、および多段階推論を含むため、1 つのデバッグセッションはチャットの 1 日分以上を消費できます。

40.2.3 Claude Console

API 組織は ワークスペース を通じて Claude Code 支出を管理します。ワークスペース支出制限を設定 して Claude Code 支出の合計をキャップし、Console で コストと使用状況レポートを表示 できます。

注釈:

Claude Code を Claude Console アカウントで初めて認証すると、「Claude Code」というワークスペースが自動的に作成されます。このワークスペースは、組織内のすべての Claude Code 使用量の一元化されたコスト追跡と管理を提供します。このワークスペースの API キーを作成することはできません。これは Claude Code 認証と使用量専用です。
カスタムレート制限を持つ組織の場合、このワークスペースの Claude Code トラフィックは組織全体の API レート制限にカウントされます。Claude Console の Limits ページでこのワークスペースに ワークスペースレート制限 を設定して、Claude Code の共有をキャップし、他の本番ワークロードを保護できます。

ユーザーごとのレポートについては、Console ダッシュボード はメンバーごとの支出と受け入れられた行を表示し、Claude Code Analytics API は Admin API キー を使用してプログラムで同じ日次ユーザーごとのメトリクスを返します。*API* **カスタマー向けの** *analytics* (ページ 932) を参照してください。

レート制限の推奨事項

チーム向けに Claude Code を設定する場合、組織のサイズに基づいて、これらのユーザーあたりのトークン/分（TPM）およびリクエスト/分（RPM）の推奨事項を検討してください。

チームサイズ	ユーザーあたり TPM	ユーザーあたり RPM
1〜5 ユーザー	200k〜300k	5〜7
5〜20 ユーザー	100k〜150k	2.5〜3.5
20〜50 ユーザー	50k〜75k	1.25〜1.75
50〜100 ユーザー	25k〜35k	0.62〜0.87
100〜500 ユーザー	15k〜20k	0.37〜0.47
500 ユーザー以上	10k〜15k	0.25〜0.35

たとえば、200 ユーザーがいる場合、各ユーザーに 20k TPM をリクエストするか、合計 400 万 TPM（200*20,000 = 400 万）をリクエストできます。

チームサイズが大きくなるにつれて、ユーザーあたりの TPM は減少します。これは、より大きな組織では Claude Code を同時に使用するユーザーが少ない傾向があるためです。これらのレート制限は個別ユーザーレベルではなく組織レベルで適用されます。つまり、他のユーザーが積極的にサービスを使用していない場合、個別ユーザーは計算された共有量を一時的に超えて消費できます。

注釈:

大規模グループとのライブトレーニングセッションなど、異常に高い同時使用シナリオが予想される場合は、ユーザーあたりのより高い TPM 割り当てが必要になる場合があります。

40.2.4 クラウドプロバイダー

Amazon Bedrock、Google Cloud の Agent Platform、および Microsoft Foundry では、Claude Code はトークンごとにクラウドアカウントに請求され、支出コントロールはクラウドプロバイダーの請求コンソールにあります。Claude Code はクラウドからメトリクスを Anthropic に送信しないため、*analytics* **ダッシュボード** (ページ 927) および Claude Code Analytics API はこの使用量をカバーしません。

ユーザーごとのコスト帰属については、3 つのオプションがあります。

- **OpenTelemetry**: 各開発者のマシンから **メトリクスをエクスポート** (ページ 935) して、独自の可観測性スタックに送信します。これにより、プロバイダーに関係なく、ユーザーごとのトークンカウント、コスト、およびツールアクティビティが得られます。
- **Claude apps gateway**: セルフホストされた Claude apps gateway は、ユーザーごとの使用状況帰属、トークンカウント付きの OTLP メトリクス、およびこれらのプロバイダーの ユーザーごとの支出制限 を提供します。
- **LLM gateway**: すべての Claude Code トラフィックをキーごとに支出を追跡するプロキシを通じてルーティングします。複数の大規模企業は *LiteLLM* (ページ 841) を使用していると報告しており、これはオープンソースツールで キーごとに支出を追跡 します。このプロジェクトは Anthropic と提携していないため、セキュリティについて監査されていません。

40.2.5 開発者が制限について質問する場合

開発者は通常、制限に関する質問を管理者に持ち込むため、どの上限に達したかを知ることが役立ちます。これらの状況は異なることを意味します。

- **「セッション制限に達しました」または「週間制限に達しました」**: サブスクリプションプランのシートベースの使用ウィンドウ。これらのウィンドウはすべてのモデル全体で共有されるため、開発者は `/model` でモデルを切り替えてアクセスを復元することはできません。メッセージはウィンドウがリセットされるときを表示します。モデル固有の「Opus 制限に達しました」または「Sonnet 制限に達しました」メッセージの後、`/model` でそのファミリー外のモデルに切り替えると、開発者は作業を続けることができます。使用制限エラー を参照してください。開発者がその間にできることは以下の通りです。
  - 使用クレジット がオンになっている場合、`/usage-credits` を実行して割り当てを超えた使用をリクエストしてください。
  - Claude Code v2.1.234 以降では、**リセット後に中断されたタスクを自動的に待機して続行** (ページ 100) してください。そのセクションでは、Claude Code がいつ自動的に待機を開始するか、および開発者が `/rate-limit-options` からそれを選択するときを一覧表示しています。フリート全体で Claude Code が自動的に待機を開始するかどうかを制御するには、**マネージド設定** (ページ 719) で `autoContinueAtUsageLimit` を設定してください。
- **「個別支出制限に達しました」、「組織の月間支出制限」、または「チームの共有予算」**: 開発者のリクエストは使用クレジットに請求され、それらのクレジットは設定した支出制限に達しています。開発者が続行できるようにするには、管理者設定 > 使用状況 に移動して、メッセージが名前を付ける制限を増やしてください。メッセージがプラン リセット時間も名前を付ける場合、開発者は代わりにそれまで待つことができます。各バリアントについては、エラー リファレンス を参照してください。

- Claude apps gateway **からの支出制限メッセージ**: 開発者はセルフホストされたゲートウェイに設定した支出上限を超過し、ゲートウェイは期間がリセットされるか上限が引き上げられるまでリクエストをブロックします。ゲートウェイ支出制限 で上限、リセットスケジュール、および開発者が見るメッセージを参照してください。
- **コンテキストまたは auto-compact 警告**: 使用制限ではありません。会話がセッションの *auto-compact* **ウィンドウ** (ページ 754) に近づいており、Claude Code が古い履歴を要約して領域を解放するしきい値です。開発者を **トークン使用量を削減** (ページ 919) に指してください。
- **API またはクラウドプロバイダープランで予期しない高い支出**: 通常、クリアされたことのない長いセッション、または Opus がデフォルトモデルとして残されていることに遡ります。共有する最も影響の大きい習慣は、関連のないタスク間でクリアすることとジョブにモデルを一致させることの両方で、**トークン使用量を削減** (ページ 919) でカバーされています。

40.2.6 エージェントチームのトークンコスト

エージェントチーム は複数の Claude Code インスタンスを生成し、各インスタンスは独自のコンテキストウィンドウを持ちます。トークン使用量はアクティブなチームメイトの数と各チームメイトが実行される期間に応じてスケーリングされます。

エージェントチームのコストを管理可能に保つには、以下を実行してください。

- チームメイトに Sonnet を使用します。これは調整タスクの機能とコストのバランスを取ります。
- チームを小さく保ちます。各チームメイトは独自のコンテキストウィンドウを実行するため、トークン使用量はおおよそチームサイズに比例します。
- スポーンプロンプトを焦点を絞ったものにします。チームメイトは CLAUDE.md、MCP サーバー、およびスキルを自動的に読み込みますが、スポーンプロンプト内のすべてが最初からコンテキストに追加されます。
- 作業が完了したらチームをシャットダウンします。アクティブなチームメイトはアイドル状態でもトークンを消費し続けます。
- エージェントチームはデフォルトで無効になっています。*settings.json* (ページ 711) または環境で `CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1` を設定して有効にします。エージェントチームを有効にする を参照してください。

40.3 トークン使用量を削減する

トークンコストはコンテキストサイズに応じてスケールします。Claude が処理するコンテキストが大きいほど、より多くのトークンを使用します。Claude Code は プロンプトキャッシング を通じてコストを自動的に最適化します。プロンプトキャッシングはシステムプロンプトなどの繰り返されるコンテンツのコストを削減し、auto-compact はコンテキスト制限に近づいたときに会話履歴を要約します。

以下の戦略は、コンテキストを小さく保ち、メッセージあたりのコストを削減するのに役立ちます。

40.3.1 コンテキストを積極的に管理する

/usage を使用して現在のトークン使用量を確認するか、**ステータスラインを設定** (ページ 799) して継続的に表示します。

- **タスク間でクリアする**: 関連のない作業に切り替えるときは /clear を使用して新しく開始します。古いコンテキストは後続のすべてのメッセージでトークンを無駄にします。クリアする前に /rename を使用してセッションに名前を付けると、後で簡単に見つけることができます。その後 /resume を使用して復帰します。
- **カスタムコンパクション指示を追加する**: /compact Focus on code samples and API usage は、要約中に保持する内容を Claude に指示します。

プロジェクトのルートにある CLAUDE.md ファイルでコンパクション動作をカスタマイズすることもできます。

```
# Compact instructions

When you are using compact, please focus on test output and code changes
```

40.3.2 適切なモデルを選択する

Sonnet はほとんどのコーディングタスクをうまく処理でき、Opus よりもコストが低くなります。複雑なアーキテクチャの決定や多段階の推論には Opus を予約します。/model を使用してセッション中にモデルを切り替えるか、/config でデフォルトを設定します。Opus への切り替えは **セッションのモデルを継承するサブエージェント** (ページ 728) にも適用されます。シンプルなサブエージェントタスクの場合は、**サブエージェント設定** (ページ 329) で model: haiku を指定します。

40.3.3 MCP サーバーのオーバーヘッドを削減する

MCP ツール定義は **デフォルトで遅延** (ページ 406) されるため、Claude が特定のツールを使用するまで、ツール名とサーバー指示のみがコンテキストに入ります。/context を実行して、何がスペースを消費しているかを確認します。

- **利用可能な場合は CLI ツールを優先する**: gh、aws、gcloud、sentry-cli などのツールは、ツールごとのリストを追加しないため、MCP サーバーよりもコンテキスト効率が高いです。Claude は CLI コマンドを直接実行できます。
- **未使用のサーバーを無効にする**: /mcp を実行して設定されたサーバーを確認し、積極的に使用していないサーバーを無効にします。

40.3.4 型付き言語用のコード インテリジェンスプラグインをインストールする

コード インテリジェンスプラグイン は、Claude にテキストベースの検索の代わりに正確なシンボルナビゲーションを提供し、不慣れなコードを探索するときの不要なファイル読み取りを削減します。単一の「定義に移動」呼び出しは、複数の候補ファイルを読み取る前に grep を実行する必要があったものを置き換えます。インストールされた言語サーバーは、編集後に型エラーを自動的に報告するため、Claude はコンパイラを実行せずにエラーをキャッチします。

40.3.5 hooks と skills に処理をオフロードする

カスタム *hooks* (ページ 443) は Claude がそれを見る前にデータを前処理できます。Claude が 10,000 行のログファイルを読んでエラーを見つける代わりに、hook は ERROR に対して grep を実行し、一致する行のみを返すことができます。これにより、コンテキストを数万トークンから数百に削減します。

skill (ページ 275) は Claude にドメイン知識を与えることができるため、探索する必要がありません。たとえば、「codebase-overview」skill はプロジェクトのアーキテクチャ、主要なディレクトリ、命名規則を説明できます。Claude が skill を呼び出すと、構造を理解するために複数のファイルを読むトークンを費やす代わりに、このコンテキストが即座に得られます。

たとえば、この PreToolUse hook はテスト出力をフィルタリングして失敗のみを表示します。

settings.json

これを *settings.json* (ページ 711) に追加して、すべての Bash コマンドの前に hook を実行します。

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "hooks": [
          {
            "type": "command",
            "command": "~/.claude/hooks/filter-test-output.sh"
          }
        ]
      }
    ]
  }
}
```

filter-test-output.sh

hook はこのスクリプトを呼び出します。`mkdir -p ~/.claude/hooks` でフォルダを作成し、以下のスクリプトを `~/.claude/hooks/filter-test-output.sh` として保存し、`chmod +x ~/.claude/hooks/filter-test-output.sh` で実行可能にします。コマンドがテストランナーであるかどうかをチェックし、失敗のみを表示するように変更します。

```
#!/bin/bash
input=$(cat)
cmd=$(echo "$input" | jq -r '.tool_input.command')

# If running tests, filter to show only failures
if [[ "$cmd" =~ ^(npm test|pytest|go test) ]]; then
  filtered_cmd="$cmd 2>&1 | grep -A 5 -E '(FAIL|ERROR|error:)' | head -100"
  echo "$input" | jq --arg filtered "$filtered_cmd" \
    '{hookSpecificOutput: {hookEventName: "PreToolUse", permissionDecision:
↪"allow", updatedInput: (.tool_input + {command: $filtered})}}'
```

(次のページに続く)

(前のページからの続き)

```
else
  echo "{}"
fi
```

セットアップを確認するには、/hooks を実行して、hook が PreToolUse の下に表示されることを確認します。claude --debug-file ./claude-debug.txt で Claude Code を起動し、Claude に npm test を実行するよう依頼することもできます。hook がコマンドを書き直すと、そのログファイルには command と他の Bash 入力フィールドをリストする modified tool input keys 行が含まれます。

40.3.6 CLAUDE.md から skills に指示を移動する

CLAUDE.md (ページ 997) ファイルはセッション開始時にコンテキストに読み込まれます。PR レビューやデータベースマイグレーションなどの特定のワークフロー用の詳細な指示が含まれている場合、関連のない作業を行っているときでもそれらのトークンが存在します。*Skills* (ページ 275) はオンデマンドでのみ呼び出されたときに読み込まれるため、特殊な指示を skills に移動することで、ベースコンテキストを小さく保ちます。必須項目のみを含めることで、CLAUDE.md を 200 行以下に保つことを目指します。

40.3.7 拡張思考を調整する

拡張思考はデフォルトで有効になっています。複雑な計画と推論タスクのパフォーマンスを大幅に向上させるためです。思考トークンは出力トークンとして課金され、デフォルトの予算はモデルに応じてリクエストあたり数万トークンになる可能性があります。

深い推論が不要なシンプルなタスクの場合は、/effort または /model で **努力レベル** (ページ 747) を下げるか、/config で思考を無効にすることでコストを削減できます。Opus 5.5、Sonnet 5.5、または Fable モデルでは思考をオフにすることはできません。これらは常に拡張思考を使用します。

固定思考予算 (ページ 751) を持つモデルでは、MAX_THINKING_TOKENS 環境変数 を設定することで予算を下げることもできます。たとえば MAX_THINKING_TOKENS=8000 です。適応推論モデルはゼロ以外の予算を無視するため、代わりに努力レベルを使用します。

40.3.8 冗長な操作をサブエージェントに委譲する

テストの実行、ドキュメントの取得、またはログファイルの処理は、かなりのコンテキストを消費する可能性があります。これらを **サブエージェント** (ページ 348) に委譲して、冗長な出力がサブエージェントのコンテキストに留まり、要約のみがメインの会話に返されるようにします。

サブエージェント自身のリクエストはまだあなたの使用量を引き出します。それらにより少なく費やすには、**サブエージェント用にはより小さいモデルを選択** (ページ 329) するか、**すべてのサブエージェントを *1* つのモデルで実行** (ページ 331) します。

40.3.9 エージェントチームのコストを管理する

エージェントチームは、チームメイトが plan mode で実行されるときに標準セッションよりも約 7 倍多くのトークンを使用します。各チームメイトが独自のコンテキストウィンドウを維持し、別の Claude インスタンスとして実行されるためです。チームタスクを小さく自己完結させて、チームメイトあたりのトークン使用量を制限します。詳細は エージェントチーム を参照してください。

40.3.10 具体的なプロンプトを作成する

「このコードベースを改善する」のような曖昧なリクエストは広範なスキャンをトリガーします。「auth.ts のログイン関数に入力検証を追加する」のような具体的なリクエストは、Claude が最小限のファイル読み取りで効率的に作業できるようにします。

40.3.11 複雑なタスクで効率的に作業する

より長いまたはより複雑な作業の場合、これらの習慣は間違った方向に進むことからのトークン浪費を回避するのに役立ちます。

- **複雑なタスクに plan mode を使用する**: Shift+Tab を押して実装前に plan mode に切り替えます。Claude はコードベースを探索し、承認のためのアプローチを提案し、初期の方向が間違っている場合の高額な再作業を防ぎます。
- **早期に方向を修正する**: Claude が間違った方向に向かい始めたら、Escape を押してすぐに停止します。`/rewind` を使用するか、Escape をダブルタップして、会話とコードを以前のチェックポイントに復元します。
- **検証ターゲットを指定する**: テストケースを含める、スクリーンショットを貼り付ける、またはプロンプトで予想される出力を定義します。Claude が独自の作業を検証できる場合、修正をリクエストする必要がある前に問題をキャッチします。
- **段階的にテストする**: 1 つのファイルを作成し、テストしてから続行します。これにより、修正が安価なときに早期に問題をキャッチします。

40.4 バックグラウンドトークン使用量

Claude Code はアイドル状態でも、バックグラウンド機能にトークンを使用します。

- **会話要約**: `claude --resume` 機能の前の会話を要約するバックグラウンドジョブ
- **コマンド処理**: `/usage` などの一部のコマンドは、ステータスを確認するためにリクエストを生成する場合があります

これらのバックグラウンドプロセスは、アクティブなインタラクションがなくても、少量のトークン（通常はセッションあたり $0.04 未満）を消費します。

プロンプト提案がオンの場合、Claude Code は Claude が応答した後、セッションが使用しているモデルに短いリクエストを送信して、**次のプロンプトを提案** (ページ 92) します。そのリクエストは会話のプロンプトキャッシュを再利用するため、ほぼキャッシュ読み取りと少数の出力トークンです。Claude Code は**アカウントが使用量制限に近い、または達している場合、提案をスキップ** (ページ 92) します。これらのリクエストを停止するには、**プロンプト提案をオフにしてください** (ページ 93)。

40.5 長いセッションで使用量が増加する理由

数時間開いているセッションは、アクティビティが示唆するよりもはるかに多くのプラン制限を使用する可能性があります。通常、以下のいずれかの理由によります。

- **長いコンテキスト**: Claude Code はすべてのリクエストで完全な会話を送信し、Claude がツールを使用するたびに、そのツール結果のバッチを含む別のリクエストを送信します。プロンプトキャッシングを使用すると、Claude Code はその履歴をキャッシュされたトークンレートで再度読み込むため、一日中開いているセッションの 1 行の質問でも、会話全体の使用量が発生します。コンテキストを小さく保つ方法については、**コンテキストを積極的に管理する** (ページ 920) を参照してください。
- **キャッシュミス**: キャッシュライフタイムより長い休止後の最初のメッセージはキャッシュをミスし、完全なコンテキストを再処理します。ライフタイムはサブスクリプションで 1 時間で、使用クレジットを使用している場合は 5 分に短縮されます。API キーまたはクラウドプロバイダーでは、デフォルトで 5 分です。使用クレジットを使用しながら 1 時間のライフタイムを保つには、TTL を自分で選択してください。Pro および Max プランでは、長い休止後に大規模なセッションを再開する場合、Claude Code はサマリーから再開することを提案し、後のリクエストが完全な履歴を含まないようにします。
- **スケジュール済みタスク**: スケジュール済みタスクはセッションがアイドル状態でも間隔で実行され、毎回完全なコンテキストを送信します。
- **クロスセッションメッセージ**: Claude Code は別のセッションからのメッセージをこのセッションがアイドル状態のときに新しいターンとして配信し、毎回完全なコンテキストを送信します。受信メッセージを配信する代わりに保持するには、`crossSessionInbound` を `hold` に設定してください。
- **ゴールチェックイン**: バックグラウンド作業がアクティブなゴールを待機させている間、Claude Code はセッションがアイドル状態でもその作業をチェックするよう Claude に要求し、完全なコンテキストを送信する新しいターンを開始します。Claude Code はプロンプト間で最大 3 つのアイドルチェックインをゴールごとに開始します。v2.1.246 より前は、アイドルチェックインは無制限でした。チェックインをオフにするには、`CLAUDE_CODE_GOAL_CHECKIN_MINUTES` を `0` に設定してください。アイドルチェックインには Claude Code v2.1.236 以降が必要です。
- **サブエージェントとワークフロー**: すべてのサブエージェント、および動的ワークフローが生成するすべてのエージェントは、メイン会話の上に独自のリクエストを送信します。**属性の内訳** (ページ 913) はサブエージェントのシェアを表示します。
- **エージェントチームメイト**: アクティブな**チームメイト** (ページ 919) ごとに、終了するまでトークンを消費し続けます。
- **コンパクション**: `/compact` は要約するコンテキストを読み込むため、大規模なコンテキストをコンパクトにすること自体が大規模なリクエストです。継続性ではなく新しいスタートが必要な場合、`/clear` はコストがかかりません。

Pro、Max、Team、または Enterprise プランでは、`/usage` の内訳は長いコンテキストやキャッシュミスなど、最近の使用量の 10% 以上を占める動作にフラグを立て、それぞれ削減するためのヒントを提供します。

40.6 Claude Code の動作の変更を理解する

Claude Code は定期的に更新を受け取り、機能の動作方法（コスト報告を含む）が変わる可能性があります。`claude --version` を実行して、現在のバージョンを確認してください。

アカウント固有の請求に関する質問については、製品内メッセンジャーを通じて Anthropic サポートにお問い合わせください。

- **サブスクリプションプラン**（Pro、Max、Team、Enterprise）：claude.ai にサインインし、左下のイニシャルをクリックして、**Get help** を選択してください
- **Console（API）請求**：platform.claude.com にサインインし、イニシャルをクリックして、**Get help** を選択してください

各プランで人間のエージェントに連絡できるユーザーを含む完全なフローについては、How to get support を参照してください。

第 41 章　チームの使用状況を分析で追跡する

> Claude Code の使用メトリクスを表示し、採用状況を追跡し、分析ダッシュボードでエンジニアリング速度を測定します。

Claude Code は、組織が開発者の使用パターンを理解し、貢献メトリクスを追跡し、Claude Code がエンジニアリング速度にどのような影響を与えるかを測定するのに役立つ分析ダッシュボードを提供します。お客様のプランに応じたダッシュボードにアクセスしてください。

プラン	ダッシュボード URL	含まれる内容	詳細
Claude for Teams / Enterprise	claude.ai/analytics/claude-code	使用メトリクス、GitHub 統合による貢献メトリクス、リーダーボード、データエクスポート	**詳細** (ページ 927)
API（Claude Console）	platform.claude.com/claude-code	使用メトリクス、支出追跡、チームインサイト	**詳細** (ページ 932)

41.1 Team と Enterprise の分析にアクセスする

claude.ai/analytics/claude-code に移動してください。管理者とオーナーがダッシュボードを表示できます。

Team と Enterprise ダッシュボードには以下が含まれます。

- **使用メトリクス**：受け入れられたコード行数、提案受け入れ率、日次アクティブユーザー数とセッション数
- **貢献メトリクス**：*GitHub* **統合** (ページ 928) を使用した Claude Code 支援による PR とシップされたコード行数
- **リーダーボード**：Claude Code 使用量でランク付けされたトップコントリビューター
- **データエクスポート**：カスタムレポート用に貢献データを CSV としてダウンロード

ユーザーごとのトークン数とコスト推定については、*OpenTelemetry* **エクスポート** (ページ 935) を構成するか、組織の分析設定から支出レポートをエクスポートしてください。これはユーザーごと、モデルごとのトークン使用量と推定使用クレジット支出を一覧表示します。

41.1.1 貢献メトリクスを有効にする

注釈:

貢献メトリクスはパブリックベータ版であり、Claude for Teams と Claude for Enterprise プランで利用可能です。これらのメトリクスは、claude.ai 組織内のユーザーのみをカバーしています。Claude Console API またはサードパーティ統合を通じた使用は含まれていません。

使用状況と採用データは、すべての Claude for Teams と Claude for Enterprise アカウントで利用可能です。貢献メトリクスには、GitHub 組織を接続するための追加セットアップが必要です。

分析設定を構成するには、オーナーロールが必要です。GitHub 管理者が GitHub アプリをインストールする必要があります。

警告:

Zero Data Retention が有効になっている組織では、貢献メトリクスは利用できません。分析ダッシュボードは使用メトリクスのみを表示します。

1. **GitHub アプリをインストールする**

   GitHub 管理者が、github.com/apps/claude で組織の GitHub アカウントに Claude GitHub アプリをインストールします。

2. **Claude Code 分析を有効にする**

   Claude オーナーが claude.ai/admin-settings/claude-code に移動し、Claude Code 分析機能を有効にします。

3. **GitHub 分析を有効にする**

   同じページで、'GitHub analytics' トグルを有効にします。

4. **GitHub で認証する**

   GitHub 認証フローを完了し、分析に含める GitHub 組織を選択します。

データは通常、有効化後 24 時間以内に表示され、毎日更新されます。データが表示されない場合は、以下のいずれかのメッセージが表示される可能性があります。

- **「GitHub app required」**：貢献メトリクスを表示するには GitHub アプリをインストールしてください
- **「Data processing in progress」**：数日後に確認し、データが表示されない場合は GitHub アプリがインストールされていることを確認してください

貢献メトリクスは GitHub Cloud と GitHub Enterprise Server をサポートしています。

41.1.2 サマリーメトリクスを確認する

注釈:

これらのメトリクスは意図的に保守的であり、Claude Code の実際の影響の過小評価を表しています。Claude Code の関与に高い信頼度がある行と PR のみがカウントされます。

ダッシュボードの上部に以下のサマリーメトリクスが表示されます。

- **PRs with CC**：Claude Code で記述された少なくとも 1 行のコードを含むマージされたプルリクエストの総数
- **Lines of code with CC**：Claude Code 支援で記述されたすべてのマージされた PR 全体のコード行数。「有効な行」のみがカウントされます。正規化後に 3 文字以上の行で、空行と括弧またはトリビアルな句読点のみの行を除きます。
- **PRs with Claude Code（%）**：Claude Code 支援コードを含むすべてのマージされた PR のパーセンテージ
- **Suggestion accept rate**：ユーザーが Claude Code のコード編集提案を受け入れる回数のパーセンテージ。Edit、Write、NotebookEdit ツール使用を含みます。
- **Lines of code accepted**：ユーザーがセッション内で受け入れた Claude Code で記述されたコード行の総数。これは拒否された提案を除外し、その後の削除を追跡しません。

41.1.3 チャートを探索する

ダッシュボードには、時系列でトレンドを視覚化するためのいくつかのチャートが含まれています。

採用を追跡する

採用チャートは日次使用トレンドを表示します。

- **users**：日次アクティブユーザー
- **sessions**：1 日あたりのアクティブな Claude Code セッション数

ユーザーあたりの PR を測定する

このチャートは、時系列での個々の開発者アクティビティを表示します。

- **PRs per user**：1 日にマージされた PR の総数を日次アクティブユーザーで割った値
- **users**：日次アクティブユーザー

これを使用して、Claude Code の採用が増加するにつれて個々の生産性がどのように変化するかを理解します。

プルリクエストの内訳を表示する

プルリクエストチャートは、マージされた PR の日次内訳を表示します。

- **PRs with CC**：Claude Code 支援コードを含むプルリクエスト
- **PRs without CC**：Claude Code 支援コードを含まないプルリクエスト

Lines of code ビューに切り替えて、PR 数ではなくコード行数による同じ内訳を表示します。

トップコントリビューターを見つける

リーダーボードは、貢献量でランク付けされたトップ 10 ユーザーを表示します。以下を切り替えます。

- **Pull requests**：各ユーザーの Claude Code を使用した PR とすべての PR を表示
- **Lines of code**：各ユーザーの Claude Code を使用した行とすべての行を表示

Export all users をクリックして、すべてのユーザーの完全な貢献データを CSV ファイルとしてダウンロードします。エクスポートには、表示されているトップ 10 だけでなく、すべてのユーザーが含まれます。

41.1.4 PR 属性

貢献メトリクスが有効になっている場合、Claude Code はマージされたプルリクエストを分析して、Claude Code 支援で記述されたコードを判定します。これは、Claude Code セッションアクティビティを各 PR のコードと照合することで行われます。

属性プロセス

プルリクエストがマージされるとき：

1. 追加された行が PR diff から抽出されます
2. 時間ウィンドウ内で一致するファイルを編集した Claude Code セッションが識別されます
3. PR 行が複数の戦略を使用して Claude Code 出力と照合されます
4. AI 支援行と総行数のメトリクスが計算されます

比較前に、行は正規化されます。空白がトリミングされ、複数のスペースが折りたたまれ、引用符が標準化され、テキストが小文字に変換されます。

Claude Code 支援行を含むマージされたプルリクエストは、GitHub で `claude-code-assisted` としてラベル付けされます。

時間ウィンドウ

PR マージ日の 21 日前から 2 日後のセッションが属性マッチングの対象と見なされます。

除外されたファイル

特定のファイルは自動生成されるため、分析から自動的に除外されます。

- ロックファイル：package-lock.json、yarn.lock、Cargo.lock など
- 生成されたコード：Protobuf 出力、ビルドアーティファクト、縮小ファイル
- ビルドディレクトリ：dist/、build/、node_modules/、target/
- テストフィクスチャ：スナップショット、カセット、モックデータ
- 1,000 文字を超える行（縮小または生成されている可能性が高い）

属性に関する注記

属性データを解釈する際は、以下の追加の詳細に注意してください。

- 開発者によって大幅に書き直されたコード（20% 以上の差がある場合）は Claude Code に属性されません
- 21 日ウィンドウ外のセッションは考慮されません
- アルゴリズムは属性を実行するときに PR ソースまたは宛先ブランチを考慮しません

41.1.5 分析から最大限の価値を得る

貢献メトリクスを使用して ROI を実証し、採用パターンを特定し、他のユーザーが開始するのを支援できるチームメンバーを見つけます。

採用を監視する

採用チャートとユーザー数を追跡して、以下を特定します。

- ベストプラクティスを共有できるアクティブユーザー
- 組織全体の全体的な採用トレンド
- 摩擦または問題を示す可能性のある使用の低下

ROI を測定する

貢献メトリクスは、独自のコードベースからのデータを使用して「このツールは投資する価値があるか？」という質問に答えるのに役立ちます。

- 採用が増加するにつれて、時系列でユーザーあたりの PR の変化を追跡します
- Claude Code を使用した場合と使用しない場合の PR とシップされたコード行を比較します
- DORA メトリクス、スプリント速度、または他のエンジニアリング KPI と一緒に使用して、Claude Code の採用による変化を理解します

パワーユーザーを特定する

リーダーボードは、高い Claude Code 採用を持つチームメンバーを見つけるのに役立ちます。彼らは以下を行うことができます。

- プロンプティング技術とワークフローをチームと共有する
- 何がうまく機能しているかについてのフィードバックを提供する
- 新しいユーザーのオンボーディングを支援する

プログラムでデータにアクセスする

Enterprise プランでは、Claude Enterprise Analytics API は、Claude Code を含む Claude サーフェス全体で、組織のユーザーごとのエンゲージメント、使用状況、コストレポートを返します。プライマリオーナーが claude.ai/analytics/api-keys で `read:analytics` スコープを持つキーを作成します。API は Teams プランでは利用できません。

GitHub を通じてこのデータをクエリするには、`claude-code-assisted` でラベル付けされた PR を検索します。

41.2 API カスタマー向けの分析にアクセスする

Claude Console を使用している API カスタマーは、platform.claude.com/claude-code で分析にアクセスできます。ダッシュボードにアクセスするには UsageView 権限が必要です。これは Developer、Billing、Admin、Owner、Primary Owner ロールに付与されます。同じ日次ユーザーごとのメトリクスをプログラムで取得するには、Admin API キーを使用して Claude Code Analytics API を使用してください。

注釈:

GitHub 統合による貢献メトリクスは、現在 API カスタマーには利用できません。Console ダッシュボードは使用メトリクスと支出メトリクスのみを表示します。

Console ダッシュボードは以下を表示します。

- **Lines of code accepted**：ユーザーがセッション内で受け入れた Claude Code で記述されたコード行の総数。これは拒否された提案を除外し、その後の削除を追跡しません。
- **Suggestion accept rate**：ユーザーがコード編集ツール使用を受け入れる回数のパーセンテージ。Edit、Write、NotebookEdit ツールを含みます。
- **Activity**：チャートに表示される日次アクティブユーザーとセッション。
- **Spend**：ユーザー数と並んでドルでの日次 API コスト。

41.2.1 チームインサイトを表示する

チームインサイトテーブルはユーザーごとのメトリクスを表示します。

- **Members**：Claude Code に認証されたすべてのユーザー。API キーユーザーはキー識別子で表示され、OAuth ユーザーはメールアドレスで表示されます。
- **Spend this month**：ユーザーごとの現在の月の API コストの合計。
- **Lines this month**：ユーザーごとの現在の月の受け入れられたコード行の合計。

注釈:

Console ダッシュボードの支出数値は分析目的の推定値です。実際のコストについては、請求ページを参照してください。

41.3 関連リソース

- *OpenTelemetry* **での監視** (ページ 935)：リアルタイムメトリクスとイベントを可観測性スタックにエクスポート
- **コストを効果的に管理する** (ページ 911)：支出制限を設定し、トークン使用を最適化
- **権限** (ページ 1039)：ロールと権限を構成

第42章　監視

Claude Code の OpenTelemetry を有効にして設定する方法を学びます。

OpenTelemetry (OTel) を通じてテレメトリデータをエクスポートすることで、組織全体で Claude Code の使用状況、コスト、ツールアクティビティを追跡します。Claude Code はメトリクスを標準メトリクスプロトコル経由で時系列データとしてエクスポートし、イベントをログ/イベントプロトコル経由でエクスポートし、オプションで **トレースプロトコル** (ページ 943) 経由で分散トレースをエクスポートします。

42.1 クイックスタート

環境変数を使用して OpenTelemetry を設定します:

```
# 1. テレメトリを有効にする
export CLAUDE_CODE_ENABLE_TELEMETRY=1

# 2. エクスポーターを選択する (両方はオプション - 必要なものだけを設定してください)
export OTEL_METRICS_EXPORTER=otlp       # オプション: otlp、prometheus、console、none
export OTEL_LOGS_EXPORTER=otlp          # オプション: otlp、console、none

# 3. OTLP エンドポイントを設定する (OTLP エクスポーター用)
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317

# 4. 認証を設定する (必要な場合)
export OTEL_EXPORTER_OTLP_HEADERS="Authorization=Bearer your-token"

# 5. デバッグ用: エクスポート間隔を短縮し、本番環境での使用に向けてリセットしてください
export OTEL_METRIC_EXPORT_INTERVAL=10000  # 10 秒 (デフォルト: 60000ms)
export OTEL_LOGS_EXPORT_INTERVAL=5000     # 5 秒 (デフォルト: 5000ms)

# 6. Claude Code を実行する
claude
```

メトリクスをエクスポートするセットアップを検証するには、バックエンドで `claude_code.session.count` メトリクスを確認してください。Claude Code はセッション開始時にこのメトリクスを出力します。ログのみのセットアップを検証するには、プロンプトを送信して `claude_code.user_prompt` イベントを確認してください。

何も到着しない場合は、`claude --debug-file <path>` を使用して Claude Code を起動し、そのパスに書き込まれるログを確認してください。Claude Code は、設定したエクスポーターからの失敗を `[3P telemetry]`

エラーとして報告します。ここで 3P はサードパーティを意味します。`[Anthropic telemetry]` で始まる行は、*Anthropic* **の個別の運用テレメトリ** (ページ 1109) について説明しており、セットアップの問題を示していません。

完全な設定オプションについては、OpenTelemetry 仕様を参照してください。

42.2 管理者設定

管理者は、管理設定ファイルを通じてすべてのユーザーの OpenTelemetry 設定を設定できます。設定がどのように適用されるかについては、**設定の優先順位** (ページ 719) を参照してください。

管理設定の設定例：

```
{
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_METRICS_EXPORTER": "otlp",
    "OTEL_LOGS_EXPORTER": "otlp",
    "OTEL_EXPORTER_OTLP_PROTOCOL": "grpc",
    "OTEL_EXPORTER_OTLP_ENDPOINT": "http://collector.example.com:4317",
    "OTEL_EXPORTER_OTLP_HEADERS": "Authorization=Bearer example-token"
  }
}
```

Claude Desktop アプリでは、Code タブセッションは**各種** *Desktop* **セッションに到達する** (ページ 161) ソースからこれらの管理設定を読み込みます。管理コンソールのデータとプライバシー設定の**監視**下にある Cowork の OpenTelemetry フォームは Cowork セッションのみに適用されるため、ターミナル CLI も Code タブも、そこで設定したコレクターにはエクスポートしません。

Claude Code は、リポジトリの `.claude/settings.json` と `.claude/settings.local.json` の OpenTelemetry エクスポーター変数を無視するため、リポジトリはそれらを使用してテレメトリをオンにしたり、送信先を選択したり、コンテンツをキャプチャしたりすることはできません。管理設定で設定するか、各開発者がシェルまたは `~/.claude/settings.json` で設定してください。リポジトリは、`OTEL_LOGS_EXPORTER` などのエクスポーターセレクターを none に設定することでシグナルをオフにすることはできますが、管理設定、`--settings` ファイル、または Claude Code を起動する環境がその変数を設定している場合を除きます。

Claude Code は、Bash ツール、フック、MCP サーバー、言語サーバーを含む、生成するサブプロセスに `OTEL_*` 環境変数を渡しません。OpenTelemetry でインストルメント化されたアプリケーションを Bash ツール経由で実行する場合、Claude Code のエクスポーターエンドポイントまたはヘッダーを継承しないため、そのアプリケーションが独自のテレメトリをエクスポートする必要がある場合は、コマンド内でこれらの変数を直接設定してください。

42.2.1 管理設定が OTLP 宛先をロックする方法

管理設定で `OTEL_EXPORTER_OTLP_*` 変数を設定すると、Claude Code は起動時に競合する開発者設定の変数を削除し、デバッグログに警告をログに記録します。削除される内容は、設定する変数によって異なります：

- **エンドポイント**：`OTEL_EXPORTER_OTLP_ENDPOINT` を設定すると、Claude Code はすべての開発者設定のシグナル別エンドポイントを削除します。開発者は 1 つのシグナルを別のコレクターにポイントできないため、管理設定でシグナル別エンドポイント変数も設定する必要はありません。
- **プロトコル**：`OTEL_EXPORTER_OTLP_PROTOCOL` を設定すると、Claude Code はすべての開発者設定のシグナル別プロトコルを削除します。
- **認証情報**：`OTEL_EXPORTER_OTLP_HEADERS`、`OTEL_EXPORTER_OTLP_CLIENT_KEY`、または `OTEL_EXPORTER_OTLP_CLIENT_CERTIFICATE` を設定すると、Claude Code はその変数の開発者設定のシグナル別バージョンと、すべての開発者設定のエンドポイント変数（汎用またはシグナル別）を削除します。これらの認証情報が管理設定で選択されていないコレクターに到達するのを防ぐためです。
- **エクスポーターセレクター**：`OTEL_METRICS_EXPORTER`、`OTEL_LOGS_EXPORTER`、およびベータ版の `OTEL_TRACES_EXPORTER` は通常のキーごとの優先順位に従います。開発者の設定はシグナルを無効にするか、コンソールエクスポーターに切り替えることができるため、ロックが必要な場合は管理設定でセレクターも設定してください。管理ソース全体で、`OTEL_LOGS_EXPORTER` はテレメトリユニットに従い、他の 2 つのセレクターはキーごとにマージされます。Claude Code v2.1.223 以降が必要です。
- **ベータ版トレーシングエンドポイント**：**詳細ベータ版トレーシング** (ページ 943) がアクティブな場合、Claude Code はログとトレースをログおよびトレースエクスポーターを通じてではなく `BETA_TRACING_ENDPOINT` にエクスポートします。したがって、Claude Code は以下の管理設定のいずれかがシグナルの宛先を決定するたびに、開発者設定の `BETA_TRACING_ENDPOINT` を削除します：
  - 汎用またはログ/トレースエンドポイントまたは認証情報
  - `otelHeadersHelper`
  - `none`、`console`、または空に設定されたログまたはトレースエクスポーターセレクター。これらの値はシグナルをコレクターから外します
  - `CLAUDE_CODE_ENABLE_TELEMETRY` がオフ

  メトリクスのみのエンドポイントまたは認証情報は削除しません。v2.1.251 より前では、開発者設定の `BETA_TRACING_ENDPOINT` は、管理設定がコレクターをピン留めしている場合でも、詳細ベータ版トレーシングがエクスポートするログとトレースをリダイレクトしていました。

Claude Code は管理設定自体で設定したシグナル別変数を削除しないため、その変数をそこに設定することで 1 つのシグナルを別のコレクターにルーティングできます。*SIEM* **の例** (ページ 992) がこれを行っています。そこでシグナル別認証情報を設定する場合、Claude Code はそのシグナルの開発者設定エンドポイントを削除します。

この削除動作は、テレメトリが配信される場所を変更するもので、Claude Code が収集する内容ではありません。

v2.1.217 より前では、すべての変数は独立してキーごとの設定優先順位に従っていたため、ユーザー設定またはシェルで設定されたシグナル固有のエンドポイントがそのシグナルを管理コレクターから離れた場所にリダイレクトしていました。

デスクトップアプリまたはセルフホスト環境ランナーが Claude Code を起動し、提供する環境で OTLP エンドポイントを指定する場合、Claude Code は同じ方法で宛先をピン留めします。ランナーのテレメトリ変数は、管理設定と同じように開発者設定の変数を削除します。Claude Code はランナー自体が設定した変数を削除しません。Claude Code v2.1.251 以降が必要です。

42.3 設定の詳細

42.3.1 共通の設定変数

これらの変数は、すべてのデプロイに共通するエクスポーター、エンドポイント、エクスポート動作を設定します。

OTEL_EXPORTER_OTLP_METRICS_ENDPOINT のようなシグナルごとのエンドポイント変数またはプロトコル変数を設定すると、Claude Code はそのシグナルについて汎用の変数の代わりにそれを使用します。OTEL_EXPORTER_OTLP_METRICS_HEADERS のようなシグナルごとのヘッダー変数を設定すると、Claude Code はそのシグナルについて汎用の OTEL_EXPORTER_OTLP_HEADERS とマージします。

管理設定があるマシンで Claude Code が何を削除するかについては、**管理設定が *OTLP* の送信先をロックする仕組み** (ページ 936) を参照してください。

環境変数	説明	値の例
CLAUDE_CODE_ENABLE_TELEMETRY	テレメトリ収集を有効にします (必須)	1
OTEL_METRICS_EXPORTER	メトリクスエクスポーターの種類 (カンマ区切り)。無効にするには none を使用します	console, otlp, prometheus, none
OTEL_LOGS_EXPORTER	ログ/イベントエクスポーターの種類 (カンマ区切り)。無効にするには none を使用します	console, otlp, none
OTEL_EXPORTER_OTLP_PROTOCOL	OTLP エクスポーターのプロトコル。すべてのシグナルに適用されます。Claude Code にはデフォルトのプロトコルがないため、有効にする otlp エクスポーターごとに、この変数またはシグナル固有のプロトコル変数を設定してください	grpc, http/json, http/protobuf
OTEL_EXPORTER_OTLP_ENDPOINT	すべてのシグナル向けの OTLP コレクターエンドポイント	http://localhost:4317
OTEL_EXPORTER_OTLP_METRICS_PROTOCOL	メトリクスのプロトコル。全般設定を上書きします	grpc, http/json, http/protobuf
OTEL_EXPORTER_OTLP_METRICS_ENDPOINT	OTLP メトリクスエンドポイント。全般設定を上書きします	http://localhost:4318/v1/metrics
OTEL_EXPORTER_OTLP_LOGS_PROTOCOL	ログのプロトコル。全般設定を上書きします	grpc, http/json, http/protobuf
OTEL_EXPORTER_OTLP_LOGS_ENDPOINT	OTLP ログエンドポイント。全般設定を上書きします	http://localhost:4318/v1/logs
OTEL_EXPORTER_OTLP_HEADERS	OTLP の認証ヘッダー	Authorization=Bearer token
OTEL_EXPORTER_OTLP_METRICS_HEADERS	メトリクスの認証ヘッダー。全般のヘッダーとマージされます	Authorization=Bearer token

次のページに続く

表 1 – 前のページからの続き

環境変数	説明	値の例
OTEL_EXPORTER_OTLP_LOGS_HEADERS	ログの認証ヘッダー。全般のヘッダーとマージされます	Authorization=Bearer token
OTEL_METRIC_EXPORT_INTERVAL	エクスポート間隔（ミリ秒、デフォルト: 60000）	5000, 60000
OTEL_LOGS_EXPORT_INTERVAL	ログのエクスポート間隔（ミリ秒、デフォルト: 5000）	1000, 10000
OTEL_LOG_USER_PROMPTS	ユーザープロンプトの内容のログ記録を有効にします（デフォルト: 無効）	有効にするには 1
OTEL_LOG_ASSISTANT_RESPONSES	assistant_response イベントでのアシスタント応答テキストのログ記録を有効にします（デフォルト: 無効）。未設定の場合は OTEL_LOG_USER_PROMPTS の値にフォールバックします。Claude Code v2.1.193 以降が必要です	有効にするには 1、編集済みのままにするには 0
OTEL_LOG_TOOL_DETAILS	ツールイベントとトレーススパン属性におけるツールパラメーターと入力引数のログ記録を有効にします。対象は Bash コマンド、MCP サーバー名とツール名、スキル名、ユーザーが作成したワークフロー名、ツール入力です。また、user_prompt イベントでのカスタムコマンド、プラグインコマンド、MCP コマンドの名前と、**コストおよびトークンのカウンター** (ページ 960) での実際のエージェント名、スキル名、プラグイン名、MCP サーバー名とツール名も有効にします（デフォルト: 無効）。Claude Desktop の組み込みサーバーについては、Claude Desktop が所有するセッションでは、このフラグがオフでも tool_decision/tool_result で mcp_server_name/mcp_tool_name が出力されます。この例外には Claude Code v2.1.214 以降が必要です	有効にするには 1

次のページに続く

表 1 – 前のページからの続き

環境変数	説明	値の例
OTEL_LOG_TOOL_CONTENT	*tool.output* **スパンイベント** (ページ 949) でのツールコンテンツのログ記録を有効にします（デフォルト: 無効）。スパン属性は**独自のゲート** (ページ 951) のもとでツールコンテンツを保持します。**トレース** (ページ 943) が必要です。コンテンツはコンテンツ上限（デフォルトで 60 KB）で切り詰められます	有効にするには 1
OTEL_LOG_MANAGED_SETTINGS	編集済みの管理設定と、編集前の設定の SHA-256 ダイジェストを**管理設定の解決** (ページ 985) イベントに追加します（デフォルト: 無効）。プロジェクト設定またはローカル設定の値ではオンになりません。Claude Code v2.1.274 以降が必要です	有効にするには 1
OTEL_LOG_RAW_API_BODIES	Anthropic Messages API のリクエストとレスポンスの完全な JSON を `api_request_body` / `api_response_body` ログイベントとして出力します（デフォルト: 無効）。ボディには会話履歴全体が含まれます。これを有効にすることは、OTEL_LOG_USER_PROMPTS、OTEL_LOG_TOOL_DETAILS、OTEL_LOG_TOOL_CONTENT によって公開されるすべての内容への同意を意味します	コンテンツ上限（デフォルトで 60 KB）で切り詰められたインラインのボディには 1、切り詰められていないボディをディスクに保存し、イベントに `body_ref` ポインターを含めるには `file:<dir>`

次のページに続く

表 1 – 前のページからの続き

環境変数	説明	値の例
CLAUDE_CODE_OTEL_CONTENT_MAX_LENGTH	コンテンツ上限: モデルの応答、ツールコンテンツ、システムプロンプト、生の API ボディなど、コンテンツを保持する属性の最大長（切り詰めマーカーを含む、UTF-16 コード単位、デフォルト: 61440、つまり 60 KB）。デフォルト値は属性値を 64 KB に制限するバックエンドに合わせたサイズです。バックエンドがより大きな値を受け付ける場合にのみ引き上げるか、テレメトリ量を削減するために引き下げてください。OpenTelemetry SDK の属性上限である OTEL_ATTRIBUTE_VALUE_LENGTH_LIMIT またはその logrecord 版や span 版のいずれかがより小さく設定されている場合、Claude Code はその小さい値で切り詰め、[TRUNCATED ...] マーカーが SDK の上限内に収まるようにします。Claude Code v2.1.214 以降が必要です	262144
OTEL_EXPORTER_OTLP_METRICS_TEMPORALITY_PREFERENCE	メトリクスのテンポラリティ設定（デフォルト: delta）。バックエンドが累積テンポラリティを想定している場合は cumulative に設定します	delta, cumulative
CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS	動的ヘッダーの更新間隔（デフォルト: 1740000ms / 29 分）	900000

http/protobuf および http/json プロトコルでは、Claude Code は各エクスポートリクエストを Content-Length ヘッダー付きで送信します。v2.1.212 より前は、v2.1.191 以降の Claude Code バージョンがこれらのリクエストをチャンク転送エンコーディングで送信していたため、Azure Monitor など長さの宣言を必要とするエンドポイントは 411 Length Required または 400 エラーでこれらを拒否していました。

42.3.2 mTLS 認証

OTLP エクスポーターのクライアント証明書の設定方法は、そのシグナルで使用される OTLP プロトコル（OTEL_EXPORTER_OTLP_PROTOCOL またはシグナルごとの上書きで設定）によって異なります。同じ設定がメトリクス、ログ、トレースに適用されます。

プロトコル	クライアント証明書の変数	コレクターの CA を信頼する方法
http/protobuf, http/json	CLAUDE_CODE_CLIENT_CERT、CLAUDE_CODE_CLIENT_KEY、および必要に応じて CLAUDE_CODE_CLIENT_KEY_PASSPHRASE。**ネットワーク設定** (ページ 899) を参照してください	NODE_EXTRA_CA_CERTS
grpc	OTEL_EXPORTER_OTLP_CLIENT_KEY と OTEL_EXPORTER_OTLP_CLIENT_CERTIFICATE、またはシグナルごとに異なる証明書を使用する場合は OTEL_EXPORTER_OTLP_METRICS_CLIENT_KEY などのシグナルごとのバリアント	OTEL_EXPORTER_OTLP_CERTIFICATE

grpc の場合、OpenTelemetry SDK が標準の OTLP 変数を直接読み取るため、シグナルごとのメトリクス変数を設定している既存の設定はそのまま動作します。管理設定があるマシンでは、Claude Code は起動時に**開発者が設定したシグナルごとの認証情報とエンドポイントを削除する場合があります** (ページ 936)。

42.3.3 メトリクスのカーディナリティ制御

以下の環境変数は、カーディナリティを管理するためにメトリクスに含める属性を制御します。

環境変数	説明	デフォルト値	無効にする例
OTEL_METRICS_INCLUDE_SESSION_ID	メトリクスに session.id 属性と、クラウドセッションでは ccr.session.id 属性を含めます	true	false
OTEL_METRICS_INCLUDE_VERSION	メトリクスに app.version 属性を含めます	false	true
OTEL_METRICS_INCLUDE_ACCOUNT_UUID	メトリクスに user.account_uuid 属性と user.account_id 属性を含めます	true	false
OTEL_METRICS_INCLUDE_ENTRYPOINT	メトリクスに app.entrypoint 属性を含めます	false	true
OTEL_METRICS_INCLUDE_RESOURCE_ATTRIBUTES	OTEL_RESOURCE_ATTRIBUTES のキーをメトリクスのデータポイントの属性として含めます	true	false
OTEL_METRICS_INCLUDE_REPOSITORY	メトリクスとイベントに vcs.* **リポジトリ識別属性** (ページ 958) を含めます。Claude Code v2.1.269 以降が必要です	false	true

一般に、カーディナリティが低いほどパフォーマンスが向上しストレージコストが下がりますが、分析用のデータの粒度は粗くなります。

42.3.4 トレース（ベータ）

分散トレースは、各ユーザープロンプトと、それによってトリガーされる API リクエストおよびツール実行を結び付けるスパンをエクスポートするため、トレースバックエンドで 1 つのリクエスト全体を単一のトレースとして確認できます。

トレースはデフォルトでオフです。有効にするには、`CLAUDE_CODE_ENABLE_TELEMETRY=1` と `CLAUDE_CODE_ENHANCED_TELEMETRY_BETA=1` の両方を設定し、`OTEL_TRACES_EXPORTER` を設定してスパンの送信先を選択します。トレースは、エンドポイント、プロトコル、ヘッダー、*mTLS* (ページ 941) について**共通の *OTLP* 設定** (ページ 938) を再利用します。管理設定があるマシンでは、Claude Code は起動時に**開発者が設定したシグナルごとの認証情報とエンドポイントを削除する場合があります** (ページ 936)。

環境変数	説明	値の例
`CLAUDE_CODE_ENHANCED_TELEMETRY_BETA`	スパントレースを有効にします（必須）。`ENABLE_ENHANCED_TELEMETRY_BETA` も受け付けられます	`1`
`OTEL_TRACES_EXPORTER`	トレースエクスポーターの種類（カンマ区切り）。無効にするには `none` を使用します	`console`, `otlp`, `none`
`OTEL_EXPORTER_OTLP_TRACES_PROTOCOL`	トレースのプロトコル。`OTEL_EXPORTER_OTLP_PROTOCOL` を上書きします	`grpc`, `http/json`, `http/protobuf`
`OTEL_EXPORTER_OTLP_TRACES_ENDPOINT`	OTLP トレースエンドポイント。`OTEL_EXPORTER_OTLP_ENDPOINT` を上書きします	`http://localhost:4318/v1/traces`
`OTEL_EXPORTER_OTLP_TRACES_HEADERS`	トレースの認証ヘッダー。`OTEL_EXPORTER_OTLP_HEADERS` とマージされます	`Authorization=Bearer token`
`OTEL_TRACES_EXPORT_INTERVAL`	スパンのバッチエクスポート間隔（ミリ秒、デフォルト: 5000）	`1000`, `10000`

スパンでは、ユーザープロンプトのテキスト、ツール入力の詳細、ツールコンテンツがデフォルトで編集（秘匿）されます。これらを含めるには、`OTEL_LOG_USER_PROMPTS=1`、`OTEL_LOG_TOOL_DETAILS=1`、`OTEL_LOG_TOOL_CONTENT=1` を設定します。

トレースが有効な場合、Bash と PowerShell のサブプロセスは、アクティブなツール実行スパンの W3C トレースコンテキストを含む `TRACEPARENT` 環境変数を自動的に継承します。これにより、`TRACEPARENT` を読み取るサブプロセスは自身のスパンを同じトレースの下に親付けでき、Claude が実行するスクリプトやコマンドを通じたエンドツーエンドの分散トレースが可能になります。

トレースが有効で、Claude Code が Anthropic API に直接接続されている場合、各モデルリクエストには `claude_code.llm_request` スパンのコンテキストに設定された W3C `traceparent` ヘッダーが付与され、API の `traceresponse` ヘッダーはスパンリンクとして記録されます。これらを組み合わせることで、準拠した中継サーバーを経由しても、Claude Code のクライアント側スパンがサーバー側のトレースに接続されます。送信される HTTP MCP リクエストにも同様に `traceparent` が付与されます。このヘッダーはサードパーティのプロバイダーには送信されません。

一部のプロキシは認識できないヘッダーを拒否するため、デフォルトでは、モデルリクエストと HTTP MCP

リクエストの traceparent ヘッダーは、ANTHROPIC_BASE_URL が未設定であるか Anthropic API を指している場合にのみ送信されます。一貫性のため、サブプロセスの TRACEPARENT 変数も同じスイッチで制御されます。カスタムの ANTHROPIC_BASE_URL プロキシを介して Claude Code を実行していて、トレースコンテキストを伝播させたい場合は、CLAUDE_CODE_PROPAGATE_TRACEPARENT=1 を設定してください。

Agent SDK および -p で開始した非対話型セッションでは、Claude Code は各インタラクションスパンの開始時に、自身の環境から TRACEPARENT と TRACESTATE も読み取ります。これにより、埋め込み側のプロセスがアクティブな W3C トレースコンテキストをサブプロセスに渡し、Claude Code のスパンを呼び出し元の分散トレースの子として表示させることができます。対話型セッションでは、CI やコンテナ環境の周囲の値を誤って継承しないよう、受信した TRACEPARENT は無視されます。

受信したトレースコンテキストは**イベント** (ページ 962) にも適用されます。TRACEPARENT が設定された Agent SDK および -p セッションでは、トレースエクスポーターが設定されていない場合でも、各 OTLP イベントログレコードにアプリケーションのトレースと結び付ける trace_id と span_id の値が付与されるため、ログバックエンドでイベントをトレースの他の部分と関連付けることができます。

インタラクションがアクティブな間に出力されたレコードには、Claude Code がスパンの非同期コンテキストの外で出力した場合（権限プロンプトのコールバック内や、起動中にバッファリングされて後でエクスポートされたレコードなど）でも、インタラクションスパンの ID が付与されます。アクティブなインタラクションスパンがない状態で出力されたレコードには、受信した TRACEPARENT の ID が直接付与されます。v2.1.214 より前は、スパンの非同期コンテキストの外で出力されたレコードには、スパンの ID ではなく受信した TRACEPARENT の ID が付与されていました。v2.1.212 より前は、アクティブなスパンの外で出力されたイベントレコードには trace_id も span_id も付与されていませんでした。

スパンの階層

各ユーザープロンプトは claude_code.interaction ルートスパンを開始します。API 呼び出し、ツール呼び出し、フックの実行は、その子として記録されます。ツールスパンには独自の子スパンが 2 つあります。1 つは権限の決定を待つ時間、もう 1 つは実行自体の時間です。Agent ツールまたはレガシーの Task ツールがサブエージェントを起動すると、サブエージェントの API スパンとツールスパンは親の claude_code.tool スパンの下にネストされます。

```
claude_code.interaction
├─── claude_code.llm_request
├─── claude_code.hook                    (requires detailed beta tracing)
└─── claude_code.tool
    ├─── claude_code.tool.blocked_on_user
    ├─── claude_code.tool.execution
    └─── (Agent tool) subagent claude_code.llm_request / claude_code.tool spans
```

Agent SDK および claude -p セッションでは、環境に TRACEPARENT が設定されている場合、claude_code.interaction 自体が呼び出し元のスパンの子になります。

PreToolUse フックが**ツール呼び出しを延期** (ページ 504) すると、Claude Code は延期したターンのトレースコンテキストを保存します。セッションを再開してツールが再実行されると、そのツールのスパンは以前のターンのトレースに、そのターンの claude_code.interaction スパンの子として加わります。

スパン属性

すべてのスパンには、**標準属性** (ページ 956) に加えて、スパン名と一致する `span.type` 属性が付与されます。以下の表は、各スパンに設定される追加の属性を示しています。`llm_request`、`tool.execution`、hook スパンは、失敗を記録した場合に OpenTelemetry のステータス `ERROR` を設定します。その他のスパンは常にステータス `UNSET` で終了します。

`claude_code.interaction`

属性	説明	ゲート
`user_prompt`	プロンプトのテキスト。ゲートが設定されていない限り、値は `<REDACTED>` です	`OTEL_LOG_USER_PROMPTS`
`user_prompt_length`	プロンプトの長さ（文字数）	
`interaction.sequence`	インタラクションの 1 から始まるカウンター。*`event.sequence`* (ページ 962) の説明のとおり、セッションごとではなく Claude Code プロセスごとにカウントされます	
`parent.source`	スパンがトレースの親をどのように取得したか。受信した `TRACEPARENT` の下に親付けされた場合は `env`、独自のトレースを開始した場合は `none` です。Claude Code v2.1.268 以降が必要です	
`interaction.duration_ms`	ターンの実時間	

`claude_code.llm_request`

属性	説明	ゲート
`model`	モデル識別子	
`gen_ai.system`	常に `anthropic`。OpenTelemetry GenAI セマンティック規約	
`gen_ai.request.model`	`model` と同じ値。OpenTelemetry GenAI セマンティック規約	
`query_source`	リクエストを発行したサブシステム（`repl_main_thread` やサブエージェント名など）	`ENABLE_BETA_TRACING_DETAILED`

次のページに続く

表 6 – 前のページからの続き

属性	説明	ゲート
query_source_safe	query_source の値域を限定した形式。詳細ベータトレースが有効かどうかにかかわらず出力され、repl_main_thread や agent.builtin.general-purpose などの値を取ります。: は . になり、ユーザーが名前を付けたエージェントは agent.custom として表示されます。Claude Code v2.1.268 以降が必要です	
agent_id	リクエストを発行したサブエージェントまたはチームメイトの識別子。メインセッションでは付与されません	
parent_agent_id	このエージェントを起動したエージェントの識別子。メインセッションと、メインセッションから直接起動されたエージェントでは付与されません	
workflow.run_id	このエージェントを起動した Workflow ツール実行の実行識別子（接頭辞 wf_）。ワークフローによって起動されていないエージェントでは付与されません	
workflow.name	このエージェントを起動したワークフローの名前。ゲートが設定されていない限り、ユーザーが作成した名前は custom に置き換えられます	OTEL_LOG_TOOL_DETAILS
speed	fast または normal	
effort	リクエストに適用された *effort* **レベル** (ページ 747): low、medium、high、xhigh、または max。Claude Code が effort レベルを送信しない場合（effort をサポートしていないモデルの場合など）は付与されません。Claude Code v2.1.274 以降が必要です	
llm_request.context	親スパンに応じて interaction、tool、または standalone	
duration_ms	再試行を含む実時間	
ttft_ms	最初のトークンまでの時間（ミリ秒）	
first_content_ms	リクエスト開始から、成功した試行の最初のコンテンツブロックまでの時間（ミリ秒）。非ストリーミングパスにフォールバックしたリクエストでは付与されません。Claude Code v2.1.268 以降が必要です	

次のページに続く

表 6 – 前のページからの続き

属性	説明	ゲート
input_tokens	API の使用量ブロックから取得した入力トークン数。プロンプトキャッシュから読み取られたトークンや書き込まれたトークンは含まれず、それらは cache_read_tokens と cache_creation_tokens で報告されます	
output_tokens	出力トークン数	
cache_read_tokens	プロンプトキャッシュから読み取られたトークン	
cache_creation_tokens	プロンプトキャッシュに書き込まれたトークン	
request_id	API リクエスト ID。request_id **イベント相関属性** (ページ 962) と同じ値です	
gen_ai.response.id	request_id と同じ値。OpenTelemetry GenAI セマンティック規約	
client_request_id	最終試行のクライアント生成 x-client-request-id	
attempt	このリクエストで行われた試行の合計回数	
success	true または false	
status_code	リクエストが失敗した場合の HTTP ステータスコード	
error	リクエストが失敗した場合のエラーメッセージ	
error_class	リクエストが失敗した場合の短いエラークラストークン（api_timeout や server_overload など）。Claude Code v2.1.268 以降が必要です	
response.has_tool_call	レスポンスにツール使用ブロックが含まれていた場合は true	
stop_reason	API レスポンスの stop_reason（end_turn、tool_use、max_tokens、stop_sequence、pause_turn、refusal など）	
gen_ai.response.finish_reasons	stop_reason と同じ値を文字列配列でラップしたもの。OpenTelemetry GenAI セマンティック規約	

各再試行は、attempt 属性と client_request_id 属性を持つ gen_ai.request.attempt スパンイベントとしても記録されます。

claude_code.tool

属性	説明	ゲート
tool_name	ツール名	
tool_name_safe	ユーザーが選んだ名前を含まない形式の tool_name。組み込みツール名はそのまま渡されます。MCP ツール名は mcp_other として表示されますが、browser_* という名前の playwright ツールなど、いくつかの固定された形式に一致するツール名はそのまま渡されます。Claude Code v2.1.268 以降が必要です	
bash_command_class	Bash ツールの場合: 固定リストに基づくコマンドの最初のプログラムのカテゴリ（vcs や package_manager など）。リスト外のプログラムの場合は other、行を解析できない場合は unparsed です。Claude Code v2.1.268 以降が必要です	
bash_argv0	Bash ツールの場合: コマンドの最初のプログラムが同じ固定リストに含まれている場合はそのプログラム（git や npm など）。リスト外のプログラムの場合は other です。Claude Code v2.1.268 以降が必要です	
duration_ms	権限の待機と実行を含む実時間	
result_tokens	ツール結果のおおよそのトークンサイズ	
agent_id	ツールを実行したサブエージェントまたはチームメイトの識別子。メインセッションでは付与されません	
parent_agent_id	このエージェントを起動したエージェントの識別子。メインセッションと、メインセッションから直接起動されたエージェントでは付与されません	
workflow.run_id	このエージェントを起動した Workflow ツール実行の実行識別子 (接頭辞 wf_)。ワークフローによって起動されていないエージェントでは付与されません	
workflow.name	このエージェントを起動したワークフローの名前。ゲートが設定されていない限り、ユーザーが作成した名前は custom に置き換えられます	OTEL_LOG_TOOL_DETAILS
tool_use_id	この呼び出しに対するモデルの tool_use ブロック ID。*tool_result* (ページ 965) イベントと *tool_decision* (ページ 971) イベント、およびフックのペイロードにある tool_use_id と一致するため、スパンをそれらのレコードと結合できます	
gen_ai.tool.call.id	tool_use_id と同じ値。OpenTelemetry GenAI セマンティック規約	

次のページに続く

表 7 – 前のページからの続き

属性	説明	ゲート
file_path	Read、Edit、Write ツールの対象ファイルパス	OTEL_LOG_TOOL_DETAILS
full_command	Bash ツールのコマンド文字列	OTEL_LOG_TOOL_DETAILS
skill_name	Skill ツールのスキル名	OTEL_LOG_TOOL_DETAILS
subagent_type	Agent ツールまたはレガシーの Task ツールのサブエージェントタイプ	OTEL_LOG_TOOL_DETAILS

claude_code.tool の tool.output スパンイベント

OTEL_LOG_TOOL_CONTENT=1 を設定すると、Read と Bash の呼び出しで claude_code.tool スパンに tool.output スパンイベントを記録できます。Edit と Write の呼び出しでは、OTEL_LOG_TOOL_DETAILS=1 も設定した場合にのみ記録されます。この変数はこれら 2 つのツールに限定されないため、他の箇所で追加される引数については**設定テーブルの該当行** (ページ 938) を確認してください。

MCP ツール、WebFetch、WebSearch も、Claude Code v2.1.283 以降ではこのイベントを記録します。

Claude Code はツール呼び出しが正常に戻ったときにこのイベントを書き込むため、エラーを発生させた呼び出しは、ツールにかかわらず何も記録しません。正常に戻った呼び出しのうち、次の場合は tool.output イベントを記録しません。

- Read、Edit、Write、Bash、WebFetch、WebSearch、MCP ツール以外のツールの呼び出し
- ファイルのテキスト以外を返す Read（画像、PDF、内容が変更されていないファイルの再読み込みなど）
- OTEL_LOG_TOOL_DETAILS=1 も設定していない場合の Edit または Write の呼び出し
- 待機中のメッセージが Claude に届くよう、実行中に Claude Code がバックグラウンドに移した WebFetch または WebSearch の呼び出し。後から届く結果も記録されません。Claude Code が呼び出しを移すタイミングについては、ターミナルの場合は**キューに入れた内容を *Claude Code* が送信するタイミング** (ページ 90) を、Agent SDK セッションの場合は priority フィールドを参照してください

このイベントには以下の属性が付与され、それぞれコンテンツ上限（デフォルトで 60 KB）で切り詰められます。ゲートは OTEL_LOG_TOOL_CONTENT=1 に加えて属性に必要な変数を示します。Edit と Write については、その変数は属性ではなくイベント自体のゲートとなります。

属性	説明	ゲート
content	Read ツールが返したテキスト、または Write の呼び出しで書き込みを指示されたテキスト	Write ツールの場合は OTEL_LOG_TOOL_DETAILS
output	Bash ツールの場合は、stderr が stdout に混在したコマンドの結合出力。MCP ツール、WebFetch、WebSearch の場合は、ツールが返した結果（テキストブロックを改行で連結し、画像やドキュメントは [image] などのプレースホルダーに置き換えたもの）	
diff	Edit ツールが適用した構造化パッチ	OTEL_LOG_TOOL_DETAILS

次のページに続く

表 8 – 前のページからの続き

属性	説明	ゲート
file_path	Read、Edit、Write ツールの対象ファイルパス。同名のスパン属性を繰り返したもの	OTEL_LOG_TOOL_DETAILS
bash_command	Bash ツールのコマンド文字列	OTEL_LOG_TOOL_DETAILS

親スパンの tool_name 属性で、イベントがどのツールから来たかがわかります。コンテンツ上限で切り詰められた属性には、<attribute>_truncated と <attribute>_original_length が付随します。

claude_code.tool.blocked_on_user

属性	説明	ゲート
duration_ms	権限の決定を待っていた時間	
decision	accept または reject	
source	決定のソース。**ツール決定イベント** (ページ 971) と一致します	

claude_code.tool.execution

属性	説明	ゲート
duration_ms	ツール本体の実行に費やした時間	
tool_use_id	親の claude_code.tool スパンと同じ値	
gen_ai.tool.call.id	tool_use_id と同じ値。OpenTelemetry GenAI セマンティック規約	
success	true または false	
error	実行が失敗した場合のエラーカテゴリ文字列 (Error:ENOENT や ShellError など)。ゲートが設定されている場合は、代わりに完全なエラーメッセージが含まれます	OTEL_LOG_TOOL_DETAILS
error_class	識別子形式のエラーカテゴリ。英字、数字、アンダースコア以外の文字は _ に置き換えられます (Error_ENOENT や ShellError など)。error に完全なメッセージが含まれる場合でもカテゴリを保持します。Claude Code v2.1.268 以降が必要です	

claude_code.hook

このスパンは詳細ベータトレースが有効な場合にのみ表示されます。詳細ベータトレースには ENABLE_BETA_TRACING_DETAILED=1 と BETA_TRACING_ENDPOINT が必要で、この組み合わせはログとトレースの送信先も変更します。この組み合わせは、シェル、ユーザー設定、または管理設定で設定してください。どちらの変数もプロジェクト設定とローカル設定では無視されます。CLAUDE_CODE_ENHANCED_TELEMETRY_BETA だけではこのスパンは生成されません。

対話型 CLI セッションでは、詳細ベータトレースには、組織がこの機能の許可リストに登録されているこ

とも必要です。Agent SDK および非対話型の -p セッションでは、許可リストへの登録は不要です。

属性	説明	ゲート
hook_event	フックイベントの種類（PreToolUse など）	
hook_name	完全なフック名（PreToolUse:Write など）	
num_hooks	実行された一致するフックコマンドの数	
hook_definitions	JSON シリアライズされたフック設定	OTEL_LOG_TOOL_DETAILS
duration_ms	一致するすべてのフックの実時間	
num_success	正常に完了したフックの数	
num_blocking	ブロックの決定を返したフックの数	
num_non_blocking_error	ブロックせずに失敗したフックの数	
num_cancelled	完了前にキャンセルされたフックの数	

注釈:

new_context、system_prompt_preview、user_system_prompt、tool_input、response.model_output など、コンテンツを保持する追加の属性は、詳細ベータトレースが有効な場合にのみ出力されます。これらは安定したスパンスキーマには含まれません。

new_context のゲートは、それを保持するスパンによって異なり、各コピーはコンテンツ上限（デフォルトで 60 KB）で切り詰められます。claude_code.tool スパンでは、ツールにかかわらずそのツール呼び出しの結果を保持し、OTEL_LOG_TOOL_CONTENT=1 が必要です。claude_code.interaction スパンではユーザープロンプトを、claude_code.llm_request スパンではそのリクエストの新しいユーザーメッセージとツール結果を保持します。これらはどちらも OTEL_LOG_USER_PROMPTS=1 が必要です。

user_system_prompt にはさらに OTEL_LOG_USER_PROMPTS=1 が必要です。これは、systemPrompt SDK オプションまたは --system-prompt フラグと --append-system-prompt フラグで指定したシステムプロンプトのテキストのみを保持し、コンテンツ上限（デフォルトで 60 KB）で切り詰められ、リクエストごとではなくセッションごとに 1 回出力されます。

42.3.5 動的ヘッダー

動的な認証を必要とするエンタープライズ環境では、ヘッダーを動的に生成するスクリプトを設定できます。動的ヘッダーは http/protobuf および http/json プロトコルにのみ適用されます。grpc プロトコルでは、Claude Code は静的なヘッダー変数である OTEL_EXPORTER_OTLP_HEADERS とそのシグナルごとのバリアントのみを使用します。

設定

.claude/settings.json に以下を追加し、パスを独自のスクリプトに置き換えます。

```
{
  "otelHeadersHelper": "/path/to/generate-otel-headers.sh"
}
```

値には、実行可能ファイルへのパス（スペースを含むパスも可）、または引数付きのシェルコマンドラインを指定できます。Windows では値は常にシェルを介して実行されるため、スペースを含むパスは JSON 値内で引用符で囲んでください。

スクリプトの要件

スクリプトは、HTTP ヘッダーを表す文字列のキーと値のペアを含む有効な JSON を出力する必要があります。

```
#!/bin/bash
# Example: Multiple headers
echo "{\"Authorization\": \"Bearer $(get-token.sh)\", \"X-API-Key\": \"$(get-api-key.
↪sh)\"}"
```

ヘルパーが失敗するか、これらの要件を満たさない出力を表示した場合、エクスポートは失敗し、ヘルパーが再び動作するまでテレメトリバックエンドはそのセッションから何も受信しません。Claude Code は次の場所で失敗を報告します。

- 対話型セッションでの警告通知 `otelHeadersHelper failed; telemetry is not being exported`。ヘルパーが最初に失敗したときに、セッションごとに 1 回表示されます
- `/status` の出力
- *`--debug`* (ページ 51) 付きで実行した場合、またはセッション内で `/debug` を実行した後のデバッグログ
- `-p` で開始した非対話型セッションでの stderr

更新の動作

ヘッダーヘルパースクリプトは、トークンの更新をサポートするために、起動時とその後定期的に実行されます。デフォルトでは、スクリプトは 29 分ごとに実行されます。間隔は `CLAUDE_CODE_OTEL_HEADERS_HELPER_DEBOUNCE_MS` 環境変数でカスタマイズできます。

42.3.6 複数チームの組織のサポート

複数のチームや部門を持つ組織は、`OTEL_RESOURCE_ATTRIBUTES` 環境変数を使用してカスタム属性を追加し、異なるグループを区別できます。

```
# Add custom attributes for team identification
export OTEL_RESOURCE_ATTRIBUTES="department=engineering,team.id=platform,cost_
↪center=eng-123"
```

これらのカスタム属性はすべてのメトリクスとイベントに含まれるため、次のことが可能になります。

- チームまたは部門でメトリクスをフィルタリングする
- コストセンターごとにコストを追跡する
- チーム固有のダッシュボードを作成する

- 特定のチーム向けのアラートを設定する

Claude Code は、これらの値を OTLP リソースブロックで送信するだけでなく、すべてのメトリクスのデータポイントとイベントレコードに属性として付与します。ほとんどのメトリクスバックエンドはデータポイントの属性をクエリ可能なラベルとして公開するため、カスタムキーでメトリクスを直接グループ化およびフィルタリングできます。`vcs.*` **リポジトリ属性** (ページ 958) を除き、カスタムキーが `user.id` や `session.id` などの**標準属性** (ページ 956) を上書きすることはありません。キーが衝突した場合、Claude Code は組み込みの値を保持します。

各カスタムキーはすべてのメトリクス系列のラベルになるため、カーディナリティの高い値はメトリクスバックエンドのストレージコストを増加させます。カスタム属性をリソースブロックでのみ送信し、データポイントのラベルから除外するには、`OTEL_METRICS_INCLUDE_RESOURCE_ATTRIBUTES=false` を設定します。**メトリクスのカーディナリティ制御** (ページ 942) を参照してください。

警告:

`OTEL_RESOURCE_ATTRIBUTES` 環境変数は、厳密なフォーマット要件を持つカンマ区切りの key=value ペアを使用します。

- **スペース不可**: 値にスペースを含めることはできません。たとえば、`user.organizationName=My Company` は無効です
- **フォーマット**: カンマ区切りの key=value ペアである必要があります: `key1=value1,key2=value2`
- **使用可能な文字**: 制御文字、空白、二重引用符、カンマ、セミコロン、バックスラッシュを除く US-ASCII 文字のみ
- **特殊文字**: 使用可能な範囲外の文字はパーセントエンコードする必要があります

スペースが必要な値には、代わりにアンダースコアまたはキャメルケースを使用してください。以下の例では、それぞれの形式で `org.name` を設定しています。

```
export OTEL_RESOURCE_ATTRIBUTES="org.name=Johns_Organization"
export OTEL_RESOURCE_ATTRIBUTES="org.name=JohnsOrganization"
```

除外された文字に限らず、任意の文字をパーセントエンコードできます。この例では、スペースとアポストロフィの両方をエンコードしています。

```
export OTEL_RESOURCE_ATTRIBUTES="org.name=John%27s%20Organization"
```

値を引用符で囲んでもスペースはエスケープされません。たとえば、`org.name="My Company"` は `My Company` ではなく、引用符を含むリテラル値 `"My Company"` になります。

42.3.7 設定例

`claude` を実行する前に、これらの環境変数を設定してください。以下の各シナリオは完全な設定を示しており、各変数は**共通の設定変数** (ページ 938) で説明しています。設定が反映されたことを確認するには、セッションを開始した後にバックエンドで `claude_code.session.count` メトリクスを確認してください。ログのみの検証方法と、何も届かない場合の確認事項については**クイックスタート** (ページ 935) で説明しています。

エクスポート間隔 1 秒でのコンソールデバッグの場合:

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console
export OTEL_METRIC_EXPORT_INTERVAL=1000
```

gRPC 経由の OTLP の場合:

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

`http://localhost:9464/metrics` からスクレイプする Prometheus の場合:

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=prometheus
```

セルフホスト環境では、セッションがポート 9464 をバインドするのは、ランナーのキャパシティがデフォルトの 1 の場合のみです。キャパシティがそれより大きい場合、ランナーは代わりにセッションのカウンターとゲージを自身の `/metrics` エンドポイントで再公開します。

メトリクスを複数のエクスポーターに送信するには:

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=console,otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=http/json
```

メトリクスとログを異なるエンドポイントまたはバックエンドに送信するには:

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_LOGS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_METRICS_PROTOCOL=http/protobuf
export OTEL_EXPORTER_OTLP_METRICS_ENDPOINT=http://metrics.example.com:4318
export OTEL_EXPORTER_OTLP_LOGS_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_LOGS_ENDPOINT=http://logs.example.com:4317
```

イベントやログなしでメトリクスのみをエクスポートするには:

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_METRICS_EXPORTER=otlp
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

メトリクスなしでイベントとログのみをエクスポートするには:

```
export CLAUDE_CODE_ENABLE_TELEMETRY=1
export OTEL_LOGS_EXPORTER=otlp
```

(次のページに続く)

(前のページからの続き)

```
export OTEL_EXPORTER_OTLP_PROTOCOL=grpc
export OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4317
```

42.4 クラウドセッションと Claude Tag からのテレメトリ

クラウドセッション (ページ 223)（Claude Tag チャネルセッションを含む）は、ユーザーのデバイス上ではなく クラウド環境 で実行されるため、これらのデバイス上の管理設定ファイルまたはシェルプロファイルではテレメトリを設定できません。Anthropic ホスト環境のセッションの場合、このセクションではテレメトリ変数を設定する場所、コレクターを環境から到達可能にする方法、およびエクスポートされたデータでクラウドセッションと Claude Tag セッションを区別する方法について説明します。

これらのセッションからテレメトリをエクスポートするには、**管理者設定** (ページ 936) の例と同じキーを使用して、`CLAUDE_CODE_ENABLE_TELEMETRY` と `OTEL_*` 変数を次の 2 つの場所のいずれかに設定します。

- **サーバー管理設定**: 組織の サーバー管理設定 の env ブロックに追加します。Claude Code は **サーバー管理設定が適用される** (ページ 735) 場所（ユーザーのマシンと Claude Tag チャネルセッション以外のクラウドセッションを含む）で起動時にこれらの設定を取得します。Claude Tag セッションはサーバー管理設定を受け取らないため、このルートではそれらを設定できません。
- **環境の変数**: クラウド環境の 環境変数 に追加して、その環境で実行されるセッションのみを設定します。これは Claude Tag セッションに到達するルートです。

環境を使用する誰もがその変数を読み取ることができるため、`OTEL_EXPORTER_OTLP_HEADERS` のコレクタートークンなどの認証情報をそこに配置しないでください。環境の API 認証情報 も役に立ちません。Claude Code 独自のテレメトリエクスポートは、認証情報を取得しないリクエスト の 1 つだからです。コレクターが認証情報を必要とする場合は、代わりにサーバー管理設定を通じてエクスポート全体を設定してください。認証情報をそこに設定すると、*Claude Code* **は管理設定外で設定されたエンドポイント変数を削除します** (ページ 936)。

クラウドセッションのテレメトリを設定する際は、これらの制約を念頭に置いてください。

- **セッションがコレクターに到達できるようにする**: Claude Code はセッションのネットワークを通じてエクスポートを送信するため、`OTEL_EXPORTER_OTLP_ENDPOINT` のホストに到達できるかどうかは、環境の ネットワークアクセスレベル によって異なります。セッションが選択したレベルでコレクターのドメインに到達できない場合は、ドメインを環境のアローリストに追加してください。サーバー管理設定はドメインを環境のネットワークアローリストに追加しないためです。
- **Claude Tag チャネルは組織レベルの環境を使用します**: チャネルセッションはメンバーの個人環境ではなく組織レベルの環境で実行されるため、共有環境 でアローリストと環境変数の変更を行い、組織のデフォルトとして設定するか、チャネルにピン留めしてください。
- **Cowork は個別に設定されます**: **サーフェスカバレッジテーブル** (ページ 735) に示されているように、Cowork セッションはサーバー管理設定を受け取らないため、サーバー管理 env ブロックはそれらのテレメトリを設定しません。

42.4.1 クラウドセッションにテレメトリを属性付けする

デフォルトでは、クラウドセッションからのメトリクスとイベントは、session.id、ccr.session.id、organization.id を含む **標準属性** (ページ 956) を含むため、追加の設定なしでセッションまたは組織でフィルタリングできます。ccr.session.id の値はセッションの CLAUDE_CODE_REMOTE_SESSION_ID です。これをセッションのトランスクリプト URL に変換するには、出力をセッションにリンク戻す を参照してください。

テレメトリをより詳細に属性付けするには、これらのオプションを使用します。

- **Claude Tag セッションを識別する**: **メトリクスカーディナリティ制御** (ページ 942) で説明されているように、OTEL_METRICS_INCLUDE_ENTRYPOINT=true を設定します。メトリクスは app.entrypoint を含むようになり、Claude Tag セッションの値は claude-in-slack です。
- **カスタム属性を追加する**: これらのセッションの他の OTEL_* 変数を設定する場所と同じ場所に *OTEL_RESOURCE_ATTRIBUTES* (ページ 952) を設定します。代わりに環境の セットアップスクリプト でそれを export する場合、値は Claude Code に到達しません。セットアップスクリプトは Claude Code が起動する前に実行される別の Bash スクリプトであり、それがエクスポートする変数はそれで終わります。

Claude Tag チャネルセッションでは、Claude はメンバーではなく組織の 共有 ID として機能するため、user.* 属性に依存して Claude にタグを付けたユーザーを識別しないでください。

42.5 利用可能なメトリクスとイベント

42.5.1 標準属性

すべてのメトリクスとイベントは、以下の標準属性を共有します。

属性	説明	制御方法
session.id	一意のセッション識別子	OTEL_METRICS_INCLUDE_SESSION_ID（デフォルト: true）
ccr.session.id	クラウドセッション識別子。クラウド環境で実行されるセッションにおける CLAUDE_CODE_REMOTE_SESSION_ID の値	OTEL_METRICS_INCLUDE_SESSION_ID（デフォルト: true）
app.version	現在の Claude Code のバージョン	OTEL_METRICS_INCLUDE_VERSION（デフォルト: false）
app.entrypoint	セッションの起動方法。cli、sdk-cli、sdk-ts、sdk-py、claude-vscode、または Claude Tag セッションの場合は claude-in-slack など	OTEL_METRICS_INCLUDE_ENTRYPOINT（デフォルト: false）
organization.id	組織 UUID（認証時）	利用可能な場合は常に含まれる
user.account_uuid	アカウント UUID（認証時）	OTEL_METRICS_INCLUDE_ACCOUNT_UUID（デフォルト: true）

次のページに続く

表 12 – 前のページからの続き

属性	説明	制御方法
user.account_id	Anthropic の管理 API と一致するタグ付き形式のアカウント ID（認証時）。例: user_01BWBeN28...	OTEL_METRICS_INCLUDE_ACCOUNT_UUID（デフォルト: true）
user.id	初回実行時に生成され ~/.claude.json に保存されるランダムな匿名識別子。個人情報は含まれず、Claude アカウントから派生したものでもありません。このファイルを削除すると、次回実行時に無関係な新しい値が生成されます。	常に含まれる
user.email	ユーザーのメールアドレス。サインイン情報から取得されるか、**クラウドセッション** (ページ 223) ではセッション自体の認証情報から取得されます	利用可能な場合は常に含まれる
terminal.type	ターミナルの種類。iTerm.app、vscode、cursor、tmux など	検出された場合は常に含まれる
OTEL_RESOURCE_ATTRIBUTES のキー	設定したカスタム属性（department や team.id など）。**マルチチーム組織のサポート** (ページ 952) を参照してください	OTEL_METRICS_INCLUDE_RESOURCE_ATTRIBUTES（デフォルト: true）
vcs.repository.url.full、vcs.owner.name、vcs.repository.name、vcs.provider.name	origin リモートから導出された、セッションのリポジトリの識別情報。**リポジトリ属性** (ページ 958) を参照してください	OTEL_METRICS_INCLUDE_REPOSITORY（デフォルト: false）。Claude Code v2.1.269 以降が必要

/login を通じて Claude apps gateway にサインインしたセッションでは、CLI は認証済みの ID をエクスポートに付与します。user.id は IdP のサブジェクト、user.email はサインインしたメールアドレスで、user.groups は IdP のグループメンバーシップをカンマ区切りの文字列として保持します。各エクスポートには identity.source: gateway-oidc も付与されます。ゲートウェイの ID は最後に適用されるため、これらのセッションでは OTEL_RESOURCE_ATTRIBUTES で設定した user.* および identity.* キーは無視されます。

ゲートウェイ経由で接続する Claude Desktop および Cowork セッションの ID 属性については、ゲートウェイの telemetry リファレンスを参照してください。

イベントには、さらに以下の属性が含まれます。これらはカーディナリティが無制限に増大する原因となるため、メトリクスには付与されません。

- prompt.id: ユーザーのプロンプトと、次のプロンプトまでに発生する後続のすべてのイベントを関連付ける UUID。**イベント相関属性** (ページ 962) を参照してください。
- workspace.host_paths: デスクトップアプリで選択されたホストのワークスペースディレクトリ（文字列配列）

- `workflow.run_id`: `wf_` をプレフィックスとする実行識別子。Workflow ツールの実行に属するエージェントが発行する API イベントとツールイベントに付与されます。1 つの `workflow.run_id` でイベントをフィルタリングすると、その実行の API リクエストとツール結果を再構成できます。この識別子は、ワークフロースクリプトが起動するエージェントと、それらがさらに起動するエージェント（スキルの呼び出しなど）を対象とします。Workflow ツールの結果で報告される実行識別子と一致します。その他のすべてのイベントには含まれません。Claude Code v2.1.202 以降が必要です
- `workflow.name`: ワークフローの名前（スクリプトの `meta.name`）。`workflow.run_id` と一緒に出力されます。組み込みワークフローの名前は、未変更の組み込みスクリプトが実行された場合にそのまま表示されます。ユーザーが作成した名前（組み込みスクリプトを編集したコピーを含む）は、`OTEL_LOG_TOOL_DETAILS=1` が設定されていない限り `custom` に置き換えられます。Claude Code v2.1.202 以降が必要です

リポジトリ属性

`OTEL_METRICS_INCLUDE_REPOSITORY=true` を設定すると、メトリクスとイベントにセッションのリポジトリの識別情報がタグ付けされ、共有コレクターで使用量をリポジトリごとに集計できるようになります。Claude Code v2.1.269 以降が必要です。

Claude Code は、これらの属性をセッションごとに 1 回、リポジトリの `origin` リモートから導出します。GitHub、GitLab、Bitbucket Cloud のように、リポジトリの HTTPS リモートと SSH リモートが同じホストと同じパスを指している場合、どちらからも同一の値が生成されます。

属性	値
`vcs.repository.url.full`	`.git` を除いたリポジトリのブラウザ URL。例: `https://github.com/example-org/example-repo`
`vcs.owner.name`	オーナーまたはグループのパス。例: `example-org`。リモートのパスが単一のセグメントの場合は省略されます
`vcs.repository.name`	リポジトリ名のみ。例: `example-repo`
`vcs.provider.name`	Claude Code がリモートのホストまたは URL の形式を `github`、`gitlab`、`bitbucket`、`gitea` のいずれかのプロバイダーとして認識した場合のその値。それ以外の場合は省略されます

値は小文字に変換され、リモート URL の認証情報、クエリ文字列、フラグメントが含まれることはありません。セッションに `origin` リモートがない場合、リモートが URL 形式でない場合、または唯一の親リポジトリがホームディレクトリである場合、これらの属性は省略されます。

クラウドセッション (ページ 223) でこれらの属性を取得するには、`OTEL_METRICS_INCLUDE_REPOSITORY` を含むテレメトリ変数を、そのセッションのクラウド環境に設定します。また、環境のネットワークアクセスでコレクターのドメインを許可してください。

`OTEL_RESOURCE_ATTRIBUTES` (ページ 952) で宣言した `vcs.*` キーは、そのキーの導出値を置き換えます。`vcs.repository.url.full` を宣言した場合、Claude Code はリモートを読み取らず、宣言したキーのみを報告します。

1 つのリポジトリの HTTPS クローンと SSH クローンが異なる値を報告する場合（たとえば、HTTPS のクローン URL に SSH の URL にはないパスプレフィックスが含まれるセルフホスト環境など）、`OTEL_RESOURCE_ATTRIBUTES` で `vcs.repository.url.full` を、報告させたい他のすべての `vcs.*` キーととも

に宣言してください。これにより、すべてのクローンが宣言した識別情報を報告するようになります。

これらの属性は独自のエクスポーターにのみ送信されます。Anthropic のテレメトリはすべての `vcs.*` キーを破棄します。

42.5.2 メトリクス

Claude Code は以下のメトリクスをエクスポートします。「単位」列には各メトリクスに付与される OpenTelemetry の単位文字列を示しています。カウント系のメトリクスには単位はありません。

メトリクス名	説明	単位
`claude_code.session.count`	開始された CLI セッションの数	なし
`claude_code.lines_of_code.count`	変更されたコードの行数	なし
`claude_code.pull_request.count`	作成されたプルリクエストの数	なし
`claude_code.commit.count`	作成された Git コミットの数	なし
`claude_code.cost.usage`	Claude Code セッションのコスト	USD
`claude_code.token.usage`	使用されたトークン数	tokens
`claude_code.code_edit_tool.decision`	コード編集ツールの権限決定の数	なし
`claude_code.active_time.total`	合計アクティブ時間	s

`OTEL_METRICS_EXPORTER` に指定されたエクスポーターが `prometheus` のみの場合、スクレイプ結果が有効な Prometheus テキスト形式となるよう、Claude Code はエクスポートするメトリクスから `USD`、`tokens`、`s` の単位を省略します。メトリクス名は変わりません。また、`otlp,prometheus` のようにエクスポーターを組み合わせた設定では単位が維持されます。v2.1.216 より前は、Prometheus のスクレイプ結果に OpenMetrics 専用の `# UNIT` 行が含まれており、一部のスクレイパーで拒否されていました。

42.5.3 メトリクスの詳細

各メトリクスには上記の標準属性が含まれます。追加のコンテキスト固有の属性を持つメトリクスについては、以下に記載しています。

セッションカウンター

各セッションの開始時にインクリメントされます。

属性:

- すべての**標準属性** (ページ 956)
- `start_type`: セッションの開始方法。`"fresh"`、`"resume"`、`"continue"`、`"agents_view"` のいずれかです。`"agents_view"` の値は `claude agents` ダッシュボードプロセスを示します。これは会話セッションではなく、ユーザーが起動するローカル UI です。ダッシュボードで UI プロセスの起動と会話セッションを区別するには、この値でフィルタリングしてください。

コード行数カウンター

コードが追加または削除されたときにインクリメントされます。

属性:

- すべての**標準属性** (ページ 956)
- `type`: （`"added"`、`"removed"`）
- `model`: 変更を行ったモデルのモデル識別子（例: "claude-sonnet-5"）

プルリクエストカウンター

Claude Code がシェルコマンドまたは MCP ツールを通じてプルリクエストまたはマージリクエストを作成したときにインクリメントされます。

属性:

- すべての**標準属性** (ページ 956)

コミットカウンター

Claude Code 経由で Git コミットを作成したときにインクリメントされます。

属性:

- すべての**標準属性** (ページ 956)

コストカウンター

各 API リクエストの後にインクリメントされます。

`agent.name`、`skill.name`、`plugin.name`、`mcp_server.name`、`mcp_tool.name` の各属性は、デフォルトで一部の名前を `"custom"` または `"third-party"` というプレースホルダーに置き換えて秘匿化します。`OTEL_LOG_TOOL_DETAILS=1` を設定すると、代わりに実際の名前が含まれます。v2.1.273 より前は、`OTEL_LOG_TOOL_DETAILS=1` を設定していても、コストカウンターとトークンカウンター、および `api_request`、`api_error`、`api_refusal` イベントには秘匿化された値が含まれていました。

属性:

- すべての**標準属性** (ページ 956)
- `model`: モデル識別子（例: "claude-sonnet-5"）
- `query_source`: リクエストを発行したサブシステムのカテゴリ。`"main"`、`"subagent"`、`"auxiliary"` のいずれか
- `speed`: リクエストが fast mode を使用した場合は `"fast"`。それ以外の場合は含まれません
- `effort`: リクエストに適用された *effort* **レベル** (ページ 747)。`"low"`、`"medium"`、`"high"`、`"xhigh"`、`"max"` のいずれか。Claude Code が effort レベルを送信しない場合（effort をサポートしていないモデルなど）は含まれません。

- `agent.name`: リクエストを発行したサブエージェントの種類。組み込みエージェント名と公式マーケットプレイスのプラグインのエージェントはそのまま表示されます。その他のユーザー定義のエージェント名は `"custom"` に置き換えられます。名前付きのサブエージェントの種類によって発行されたリクエストでない場合は含まれません。
- `skill.name`: リクエストでアクティブなスキル。Skill ツールまたは / コマンドによって設定されるか、起動されたサブエージェントに継承されます。組み込み、バンドル、ユーザー定義、および公式マーケットプレイスのプラグインのスキル名はそのまま表示されます。サードパーティのプラグインのスキル名は `"third-party"` に置き換えられます。アクティブなスキルがない場合は含まれません。
- `plugin.name`: アクティブなスキルまたはサブエージェントがプラグインによって提供されている場合の、所有元のプラグイン。公式マーケットプレイスのプラグイン名はそのまま表示されます。サードパーティのプラグイン名は `"third-party"` に置き換えられます。スキルとサブエージェントのどちらにも所有元のプラグインがない場合は含まれません。
- `marketplace.name`: 所有元のプラグインのインストール元のマーケットプレイス。`OTEL_LOG_TOOL_DETAILS=1` が設定されている場合でも、公式マーケットプレイスのプラグインに対してのみ出力されます。それ以外の場合は含まれません。
- `mcp_server.name`: このリクエストが消費したツール結果を返した MCP サーバー。組み込み、claude.ai 経由のプロキシ、および公式レジストリのサーバー名はそのまま表示されます。ユーザーが設定したサーバー名は `"custom"` に置き換えられます。リクエストが MCP ツールの結果を消費しなかった場合は含まれません。v2.1.222 より前は、Claude Code はツール結果を消費したリクエストだけでなく、MCP ツール呼び出し後のすべてのリクエストにこの属性を設定していたため、この属性を集計するダッシュボードではアップグレード後に値が減少します。
- `mcp_tool.name`: このリクエストが結果を消費した MCP ツール。秘匿化とバージョンの挙動は `mcp_server.name` と同じです。リクエストが MCP ツールの結果を消費しなかった場合は含まれません。

トークンカウンター

各 API リクエストの後にインクリメントされます。

属性:

- すべての**標準属性** (ページ 956)
- `type`: （`"input"`、`"output"`、`"cacheRead"`、`"cacheCreation"`）。`"input"` タイプには、プロンプトキャッシュから読み取られた、またはプロンプトキャッシュに書き込まれたトークンは含まれません。これらは `"cacheRead"` と `"cacheCreation"` でカウントされます
- `model`: モデル識別子（例: "claude-sonnet-5"）
- `query_source`: リクエストを発行したサブシステムのカテゴリ。`"main"`、`"subagent"`、`"auxiliary"` のいずれか
- `speed`: リクエストが fast mode を使用した場合は `"fast"`。それ以外の場合は含まれません
- `effort`: リクエストに適用された *effort* **レベル** (ページ 747)。詳細は**コストカウンター** (ページ 960) を参照してください。

- agent.name、skill.name、plugin.name、marketplace.name、mcp_server.name、mcp_tool.name: リクエストのスキル、プラグイン、エージェント、MCP の帰属情報。定義と秘匿化の挙動については**コストカウンター** (ページ 960) を参照してください。

コード編集ツール決定カウンター

ユーザーが Edit、Write、NotebookEdit ツールの使用を承認または拒否したときにインクリメントされます。

属性:

- すべての**標準属性** (ページ 956)
- tool_name: ツール名（"Edit"、"Write"、"NotebookEdit"）
- decision: ユーザーの決定（"accept"、"reject"）
- source: 決定の出どころ。"config"、"hook"、"user_permanent"、"user_temporary"、"user_abort"、"user_reject" のいずれかです。各値の意味については**ツール決定イベント** (ページ 971) を参照してください。
- language: 編集されたファイルのプログラミング言語。"TypeScript"、"Python"、"JavaScript"、"Markdown" など。認識されないファイル拡張子の場合は "unknown" を返します。

アクティブ時間カウンター

アイドル時間を除き、Claude Code をアクティブに使用した実際の時間を記録します。このメトリクスは、ユーザーの操作中（入力や応答の閲覧など）と、CLI の処理中（ツールの実行や AI の応答生成など）にインクリメントされます。

属性:

- すべての**標準属性** (ページ 956)
- type: キーボード操作の場合は "user"、ツールの実行と AI の応答の場合は "cli"

42.5.4 イベント

Claude Code は、OpenTelemetry のログ/イベントを通じて以下のイベントをエクスポートします（OTEL_LOGS_EXPORTER が設定されている場合）。

イベント相関属性

ユーザーがプロンプトを送信すると、Claude Code は複数の API 呼び出しを行い、いくつかのツールを実行することがあります。prompt.id 属性を使用すると、それらのイベントすべてを、それらをトリガーした単一のプロンプトに結び付けることができます。

属性	説明
prompt.id	単一のユーザープロンプトの処理中に生成されたすべてのイベントを関連付ける UUID v4 識別子

次のページに続く

表 15 – 前のページからの続き

属性	説明
event.sequence	イベントを順序付けるための 0 始まりのカウンター。セッションごとではなく Claude Code プロセスごとにカウントされます
message.uuid	セッションのトランスクリプト（~/.claude/projects/*/*.jsonl ファイル）に保存されたメッセージの UUID。assistant_response、api_response_body、および user_prompt に含まれます。ただし、0 個または複数のメッセージを生成する可能性があるコマンドのディスパッチは除きます。assistant_response と api_response_body では、これはレスポンスの最後のトランスクリプトエントリであり、次のターンの parentUuid がこれにチェーンされます。Claude Code v2.1.214 以降が必要です。api_response_body では v2.1.274 以降が必要です
request_id	サーバーが割り当てた API リクエストの ID。request-id レスポンスヘッダーから読み取られます（例: req_011...）。*Amazon Bedrock* (ページ 863) のように request-id ヘッダーのないレスポンスでは、代わりに x-amzn-requestid ヘッダーから値を取得します。レスポンスにいずれかのヘッダーが含まれる場合、api_request、api_error、api_refusal、assistant_response、api_response_body に含まれます。llm_request トレーススパンの同名の属性と一致します。x-amzn-requestid からの取得には Claude Code v2.1.282 以降が必要です
client_request_id	x-client-request-id リクエストヘッダーとして送信される、クライアントが生成した UUID。ファーストパーティの API 接続では api_request と api_error に含まれます。サードパーティのプロバイダーのバックエンドや、非ストリーミングのフォールバックを通じてリクエストが再試行された場合には含まれません。リクエストとそのレスポンスを対応付けるもので、タイムアウトなどサーバーの request_id が生成されなかった失敗の場合にも利用できます。llm_request トレーススパンの同名の属性と一致します。Claude Code v2.1.214 以降が必要です

単一のプロンプトによってトリガーされたすべてのアクティビティを追跡するには、特定の prompt.id の値でイベントをフィルタリングします。これにより、そのプロンプトの処理中に発生した user_prompt イベント、api_request イベント、tool_result イベントが返されます。

event.sequence は Claude Code プロセスが開始されるたびに 0 から始まり、そのプロセスが存続する間カウントアップされます。新しい session.id が割り当てられる /clear の後もカウントは継続します。フォークせずにセッションを再開した場合、セッションは session.id を維持しますが、event.sequence の値は再開したプロセスのものになります。そのため、1 つのセッション内で後のイベントが前のイベントより小さい値を持つことや、同じ値が重複することがあります。セッションのイベントを順序付けるには、event.timestamp で並べ替え、同じタイムスタンプを持つイベントの順序付けに event.sequence を使用してください。

メッセージ単位で再構成するために、各イベントクラスはセッションのトランスクリプトのフィールドと一致するキーを保持しています。トランスクリプトのエントリ形式は Claude Code の内部仕様であり、バージョン間で変更されるため、これらのフィールドで結合するパイプラインはどのリリースでも壊れる可能

性があります。結合は安定した契約ではなく、バージョン固有のものとして扱ってください。

- user_prompt、assistant_response、api_response_body の message.uuid
- API イベントの request_id。トランスクリプトのアシスタントエントリには requestId として保存されます
- tool_result および tool_decision イベントの tool_use_id

ユーザープロンプトイベント

プロンプトが送信されたときにログに記録されます。Claude Code が自ら開始するターンも含まれます。

イベント名: claude_code.user_prompt

属性:

- すべての**標準属性** (ページ 956)
- event.name: "user_prompt"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- prompt_length: プロンプトの長さ
- prompt: プロンプトの内容。デフォルトでは秘匿化されます。含めるには OTEL_LOG_USER_PROMPTS=1 を設定してください
- message.uuid: 結果として生成されたユーザーメッセージの UUID。保存されたトランスクリプトのエントリと一致します。0 個または複数のメッセージを生成する可能性があるコマンドのディスパッチには含まれません。Claude Code v2.1.214 以降が必要です
- command_name: プロンプトがコマンドを呼び出す場合のコマンド名。compact や debug などの組み込みおよびバンドルのコマンド名はそのまま出力されます。reset などのエイリアスは正規の名前ではなく入力されたとおりに出力されます。カスタム、プラグイン、MCP のコマンド名は、OTEL_LOG_TOOL_DETAILS=1 が設定されていない限り custom または mcp にまとめられます
- command_source: コマンドが存在する場合のコマンドの出どころ。builtin、custom、mcp のいずれかです。プラグインが提供するコマンドは custom として報告されます

アシスタント応答イベント

モデルからテキストコンテンツを返す各 API リクエストの後にログに記録されます。応答のテキストブロックのみが含まれ、思考ブロックとツール使用ブロックは除外されます。Claude Code v2.1.193 以降が必要です。

イベント名: claude_code.assistant_response

属性:

- すべての**標準属性** (ページ 956)
- event.name: "assistant_response"

- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `response_length`: 応答テキストの長さ（文字数）
- `response`: 応答テキスト。コンテンツの上限（デフォルトは 60 KB）で切り詰められます。デフォルトでは `<REDACTED>` に秘匿化されます。含めるには `OTEL_LOG_ASSISTANT_RESPONSES=1` を設定してください。`OTEL_LOG_ASSISTANT_RESPONSES` が未設定の場合は、代わりに `OTEL_LOG_USER_PROMPTS` によって制御されます。そのため、プロンプトのログ記録を有効にしたまま応答を秘匿化しておくには `OTEL_LOG_ASSISTANT_RESPONSES=0` を設定してください
- `model`: モデル識別子（例: "claude-sonnet-5"）
- `request_id`: API リクエスト ID。**イベント相関属性** (ページ 962) で説明しています
- `message.uuid`: 応答の最後のトランスクリプトエントリの UUID。API レスポンスはコンテンツブロックごとに 1 つのトランスクリプトエントリとして保存されます。これはその最後のエントリであり、次のターンの `parentUuid` がこれにチェーンされます。Claude Code v2.1.214 以降が必要です
- `query_source`: リクエストを発行したサブシステム。`"repl_main_thread"`、`"compact"`、またはサブエージェント名など

ツール結果イベント

ツールの実行が完了したときにログに記録されます。ツール呼び出しが拒否された場合は出力されません。拒否については**ツール決定イベント** (ページ 971) を参照してください。

イベント名: `claude_code.tool_result`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"tool_result"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `tool_name`: ツールの名前
- `tool_use_id`: このツール呼び出しの一意の識別子。フックに渡される `tool_use_id` と一致するため、OTel イベントとフックで取得したデータを関連付けることができます。
- `success`: `"true"` または `"false"`
- `duration_ms`: 実行時間（ミリ秒）
- `error_type`: ツールが失敗した場合のエラーカテゴリの文字列。`"Error:ENOENT"` や `"ShellError"` など
- `error`（`OTEL_LOG_TOOL_DETAILS=1` の場合）：ツールが失敗した場合の完全なエラーメッセージ

- decision_type: 常に "accept"。このイベントはツールの実行後にのみ出力されるためです。拒否された呼び出しはツール結果を生成しません
- decision_source: 権限決定の出どころ。"config"、"hook"、"user_permanent"、"user_temporary" のいずれかです。各値の意味については**ツール決定イベント** (ページ 971) を参照してください。拒否専用のソースである "user_abort" と "user_reject" がこのイベントに現れることはありません。
- tool_input_size_bytes: JSON シリアライズされたツール入力のサイズ（バイト）
- tool_result_size_bytes: ツール結果のサイズ（バイト）
- mcp_server_scope: MCP サーバーのスコープ識別子（MCP ツールの場合）
- vcs.ref.head.revision、vcs.ref.head.name、vcs.ref.head.type（OTEL_LOG_TOOL_DETAILS=1 の場合）: Bash または PowerShell ツールによって実行され成功した git commit のコミットの識別情報。vcs.ref.head.revision はコミット SHA、vcs.ref.head.name はコミットされたブランチ、vcs.ref.head.type は branch です。detached HEAD 上でコミットされた場合、名前と種類は省略されます。Claude Code v2.1.269 以降が必要です
- tool_parameters（OTEL_LOG_TOOL_DETAILS=1 の場合）: ツール固有のパラメーターを含む JSON 文字列。Claude Desktop の組み込みサーバーについては、Claude Desktop が所有するセッションでは、フラグがオフでも mcp_server_name/mcp_tool_name のペアが含まれます。これは**ツール決定イベント** (ページ 971) と同じ、ホストが作成した名前に対する例外であり、Claude Code v2.1.214 以降が必要です。パラメーターはツールによって異なります。
  - Bash ツールの場合: bash_command、full_command、timeout、description、dangerouslyDisableSandbox を含みます。さらに git commit コマンドが成功した場合は git_commit_id と git_branch を含みます。git_commit_id は、コミットがセッションの作業ディレクトリの HEAD である場合は完全なコミット SHA、それ以外の場合は Git の短縮 SHA です。git_branch はコミットされたブランチで、detached HEAD の場合は省略されます
  - デスクトップアプリのワークスペース Bash ツール（tool_name も Bash として報告されます）の場合: bash_command、full_command、timeout のみを含みます
  - MCP ツールの場合: mcp_server_name、mcp_tool_name を含みます
  - Skill ツールの場合: skill_name を含みます
  - Agent ツールまたは従来の Task ツールの場合: subagent_type を含みます
- tool_input（OTEL_LOG_TOOL_DETAILS=1 の場合）: JSON シリアライズされたツールの引数。512 文字を超える個々の値は切り詰められ、ペイロード全体は約 4 K 文字に制限されます。MCP ツールを含むすべてのツールに適用されます。

API リクエストイベント

Claude への各 API リクエストについてログに記録されます。

イベント名: claude_code.api_request

属性:

- すべての**標準属性** (ページ 956)

- `event.name`: `"api_request"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `model`: 使用されたモデル（例: "claude-sonnet-5"）
- `cost_usd`: 推定コスト（USD）
- `cost_usd_micros`: 推定コスト（100 万分の 1 米ドル単位）。整数として出力されます
- `duration_ms`: リクエストの所要時間（ミリ秒）
- `input_tokens`: 入力トークン数。プロンプトキャッシュから読み取られた、またはプロンプトキャッシュに書き込まれたトークンは含みません
- `output_tokens`: 出力トークン数
- `cache_read_tokens`: キャッシュから読み取られたトークン数
- `cache_creation_tokens`: キャッシュの作成に使用されたトークン数
- `request_id`: API リクエスト ID（例: `"req_011..."`）。**イベント相関属性** (ページ 962) で説明しています。
- `client_request_id`: `x-client-request-id` リクエストヘッダーとして送信される、クライアントが生成した UUID。含まれる条件については**イベント相関属性** (ページ 962) の表を参照してください。Claude Code v2.1.214 以降が必要です
- `speed`: `"fast"` または `"normal"`。fast mode が有効だったかどうかを示します
- `query_source`: リクエストを発行したサブシステム。`"repl_main_thread"`、`"compact"`、またはサブエージェント名など
- `effort`: リクエストに適用された *effort* **レベル** (ページ 747)。`"low"`、`"medium"`、`"high"`、`"xhigh"`、`"max"` のいずれか。Claude Code が effort レベルを送信しない場合（effort をサポートしていないモデルなど）は含まれません。
- `agent.name`、`skill.name`、`plugin.name`、`marketplace.name`、`mcp_server.name`、`mcp_tool.name`: リクエストのスキル、プラグイン、エージェント、MCP の帰属情報。定義と秘匿化の挙動については**コストカウンター** (ページ 960) を参照してください。

API エラーイベント

Claude への API リクエストが失敗したときにログに記録されます。

イベント名: `claude_code.api_error`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"api_error"`
- `event.timestamp`: ISO 8601 タイムスタンプ

- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `model`: 使用されたモデル（例: "claude-sonnet-5"）
- `error`: エラーメッセージ
- `status_code`: 数値としての HTTP ステータスコード。接続の失敗など、HTTP 以外のエラーの場合は含まれません。
- `duration_ms`: リクエストの所要時間（ミリ秒）
- `attempt`: 最初のリクエストを含む試行の合計回数（1 は再試行が発生しなかったことを意味します）
- `request_id`: API リクエスト ID（例: `"req_011..."`）。**イベント相関属性** (ページ 962) で説明しています。
- `client_request_id`: `x-client-request-id` リクエストヘッダーとして送信される、クライアントが生成した UUID。タイムアウトや接続エラーなどの失敗によってサーバーの `request_id` が生成されなかった場合でも利用できます。含まれる条件については**イベント相関属性** (ページ 962) の表を参照してください。Claude Code v2.1.214 以降が必要です
- `speed`: `"fast"` または `"normal"`。fast mode が有効だったかどうかを示します
- `query_source`: リクエストを発行したサブシステム。`"repl_main_thread"`、`"compact"`、またはサブエージェント名など
- `effort`: リクエストに適用された *effort* **レベル** (ページ 747)。Claude Code が effort レベルを送信しない場合（effort をサポートしていないモデルなど）は含まれません。
- `agent.name`、`skill.name`、`plugin.name`、`marketplace.name`、`mcp_server.name`、`mcp_tool.name`: リクエストのスキル、プラグイン、エージェント、MCP の帰属情報。定義と秘匿化の挙動については**コストカウンター** (ページ 960) を参照してください。

API 拒否イベント

API リクエストが `stop_reason: "refusal"` を返したときにログに記録されます。拒否は HTTP エラーとしてではなく成功したレスポンスストリーム上で届くため、`api_error` イベントは発生しません。このイベントを使用すると、拒否の頻度を追跡し、`api_request` や `api_error` と同じ属性で拒否をグループ化できます。

イベント名: `claude_code.api_refusal`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"api_refusal"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `model`: リクエストのモデル識別子

- request_id: API リクエスト ID（例: "req_011..."）。**イベント相関属性** (ページ 962) で説明しています。
- query_source: リクエストを発行したサブシステム。"repl_main_thread"、"compact"、またはサブエージェント名など。定義については *api_request* (ページ 966) を参照してください。
- speed: Fast mode が有効な場合は "fast"、それ以外は "normal"
- attempt: 再試行の試行番号。最初の試行は 1 です。
- effort: リクエストに適用された *effort* **レベル** (ページ 747)。Claude Code が effort レベルを送信しない場合（effort をサポートしていないモデルなど）は含まれません。
- server_fallback_hop: API のサーバー側のモデルフォールバックがこの拒否をすでに別のモデルで再試行したため、ユーザーにはこの拒否が表示されなかった場合は true。リクエストが拒否で終了した場合は false。フォールバック先のモデルも拒否した場合、1 つのターンで true のホップイベントと、その後の false の最終イベントの両方が出力されることがあります。
- has_category: API レスポンスに "cyber"、"bio"、"frontier_llm"、"reasoning_extraction" のいずれかの stop_details.category が含まれていた場合は true。レスポンスにカテゴリが含まれていなかった場合、またはその集合以外の値だった場合は false。ホップのブロックには stop_details が含まれないため、server_fallback_hop が true の場合は含まれません。
- has_explanation: API レスポンスに stop_details.explanation が含まれていた場合は true、それ以外は false。server_fallback_hop が true の場合は含まれません。
- category: API レスポンスの stop_details.category の値。"cyber"、"bio"、"frontier_llm"、"reasoning_extraction" のいずれかです。OTEL_LOG_TOOL_DETAILS=1 が設定され、かつ has_category が true の場合にのみ含まれます。
- agent.name、skill.name、plugin.name、marketplace.name、mcp_server.name、mcp_tool.name: リクエストのスキル、プラグイン、エージェント、MCP の帰属情報。定義と秘匿化の挙動については**コストカウンター** (ページ 960) を参照してください。

API リクエストボディイベント

OTEL_LOG_RAW_API_BODIES が設定されている場合、各 API リクエストの試行についてログに記録されます。試行ごとに 1 つのイベントが出力されるため、パラメーターを調整した再試行はそれぞれ独自のイベントを生成します。

イベント名: claude_code.api_request_body

属性:

- すべての**標準属性** (ページ 956)
- event.name: "api_request_body"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- body: システムプロンプト、メッセージ、ツールなどを含む、JSON シリアライズされた Messages API のリクエストパラメーター。コンテンツの上限（デフォルトは 60 KB）で切り詰められます。過

去のアシスタントのターンに含まれる拡張思考のコンテンツは秘匿化されます。インラインモード（OTEL_LOG_RAW_API_BODIES=1）でのみ出力されます。

- body_ref: 切り詰められていないボディを含む <dir>/<uuid>.request.json ファイルへの絶対パス。ファイルモード（OTEL_LOG_RAW_API_BODIES=file:<dir>）でのみ出力されます。
- body_length: 切り詰め前のボディの長さ。OTEL_LOG_RAW_API_BODIES=file:<dir> の場合は UTF-8 バイト、=1 の場合は UTF-16 コード単位です
- body_truncated: インラインでの切り詰めが発生した場合は "true"。ファイルモードの場合、および切り詰めが発生しなかった場合は含まれません。
- model: リクエストパラメーターのモデル識別子
- query_source: リクエストを発行したサブシステム（例: "compact"）
- request_body_id: この試行のリクエストボディを識別する UUID。成功した試行の *api_response_body* **イベント** (ページ 970) にも同じ値が含まれるため、レスポンスとそれを生成した正確なリクエストを対応付けることができます。Claude Code v2.1.274 以降が必要です

API レスポンスボディイベント

OTEL_LOG_RAW_API_BODIES が設定されている場合、成功した各 API レスポンスについてログに記録されます。

ファイルモード（OTEL_LOG_RAW_API_BODIES=file:<dir>）では、Claude Code は成功したレスポンスごとに <dir>/index.jsonl にも 1 行の JSON を追記します。この行には timestamp、session_id、query_source、model、request_id、message_id、message_uuid、request_file、response_file のフィールドが含まれます。これを読むと、テレメトリバックエンドにクエリを実行することなく、特定のトランスクリプトメッセージの背後にあるリクエストファイルとレスポンスファイルを見つけることができます。インデックスファイルには Claude Code v2.1.274 以降が必要です。

イベント名: claude_code.api_response_body

属性:

- すべての**標準属性** (ページ 956)
- event.name: "api_response_body"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- body: ID、コンテンツブロック、使用量、停止理由を含む、JSON シリアライズされた Messages API のレスポンス。コンテンツの上限（デフォルトは 60 KB）で切り詰められます。拡張思考のコンテンツは秘匿化されます。インラインモード（OTEL_LOG_RAW_API_BODIES=1）でのみ出力されます。
- body_ref: 切り詰められていないボディを含む <dir>/<request_id>.response.json ファイルへの絶対パス。ファイルモード（OTEL_LOG_RAW_API_BODIES=file:<dir>）でのみ出力されます。
- body_length: 切り詰め前のボディの長さ。OTEL_LOG_RAW_API_BODIES=file:<dir> の場合は UTF-8 バイト、=1 の場合は UTF-16 コード単位です

- `body_truncated`: インラインでの切り詰めが発生した場合は `"true"`。ファイルモードの場合、および切り詰めが発生しなかった場合は含まれません。
- `model`: モデル識別子
- `query_source`: リクエストを発行したサブシステム
- `request_id`: API リクエスト ID（例: `"req_011..."`）。**イベント相関属性** (ページ 962) で説明しています。
- `request_body_id`: このレスポンスが応答する *api_request_body* **イベント** (ページ 969) の `request_body_id`。Claude Code v2.1.274 以降が必要です
- `message.id`: API がレスポンスに割り当てたメッセージ ID（レスポンスボディの `id` フィールド）。Claude Code v2.1.274 以降が必要です
- `message.uuid`: レスポンスの最後のトランスクリプトエントリの UUID。`request_body_id` と組み合わせることで、トランスクリプトのメッセージをその背後にあるリクエストボディとレスポンスボディに結び付けます。Claude Code v2.1.274 以降が必要です

ツール決定イベント

ツールの権限決定（承認/拒否）が行われたときにログに記録されます。

イベント名: `claude_code.tool_decision`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"tool_decision"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `tool_name`: ツールの名前（例: "Read"、"Edit"、"Write"、"NotebookEdit"）
- `tool_use_id`: このツール呼び出しの一意の識別子。フックに渡される `tool_use_id` と一致するため、OTel イベントとフックで取得したデータを関連付けることができます。
- `decision`: `"accept"` または `"reject"`
- `tool_source`: 常に含まれます。ツールの出自を示す、CLI が定める閉じた集合の値です。Claude Code v2.1.214 以降が必要です
  - `"builtin"`: CLI 自身のツール
  - `"mcp"`: 一般的な MCP サーバー
  - `"sdk_host_builtin_mcp"`: Claude Desktop が所有するセッションにおける、Claude Desktop 自体に組み込まれたインプロセスサーバー。Claude Desktop は、自身のエントリポイント（`claude-desktop`、`claude-desktop-3p`、`local-agent`）のいずれかから開始したセッションが、ネストされた子セッションでない場合にそのセッションを所有します。Claude Code 自体が起動す

るセッションを含むネストされたセッションでは、これらのサーバーは "mcp" として報告されます

- source: 決定の出どころ:
  - "config": プロンプトを表示せずに自動的に決定されたもの。プロジェクト設定、ユーザーの個人設定の許可ルールまたは拒否ルール、エンタープライズの管理ポリシー、--allowedTools または --disallowedTools フラグ、アクティブな権限モード、同じ対話型 CLI セッション内の以前のプロンプトで付与されたセッションスコープの権限、またはツールが本質的に安全であることに基づきます。イベントには、これらのうちどのソースが一致したかは示されません。Claude Code は、権限プロンプトのリクエスト自体が失敗した場合にも "config" を報告します。たとえば、Agent SDK の canUseTool コールバックや *--permission-prompt-tool* (ページ 51) ツールが無効な結果を返した場合や、リクエストの保留中に入力ストリームが閉じられた場合です。v2.1.216 より前は、Claude Code はこれらの失敗を "user_reject" として報告していました。
  - "hook": PreToolUse または PermissionRequest フックが決定を返したもの。
  - "user_permanent": ユーザーが権限プロンプトで「Yes, and don't ask again for ...」を選択したときに出力されます。この選択により、ユーザーの個人設定に許可ルールが保存されます。対話型 CLI では、その選択自体に対してのみ出力され、保存されたルールに一致する以降の呼び出しでは代わりに "config" が出力されます。Agent SDK または非対話型の -p セッションでは、最初の選択と以降のルール一致の両方で "user_permanent" が出力されます。承認として扱われます。
  - "user_temporary": ユーザーが権限プロンプトで 1 回限りの承認として「Yes」を選択した場合、またはファイルの編集や読み取りのプロンプトでセッションの残りの間アクセスを許可するオプションを選択した場合に出力されます。対話型 CLI では選択自体に対してのみ出力され、そのセッションスコープの権限によって許可された以降の呼び出しでは代わりに "config" が出力されます。Agent SDK または非対話型の -p セッションでは、選択と以降の一致の両方で "user_temporary" が出力されます。承認として扱われます。
  - "user_abort": ユーザーが応答せずに権限プロンプトを閉じたときに出力されます。Agent SDK および非対話型の -p セッションでは、canUseTool または --permission-prompt-tool の権限リクエストが保留中にターンを中断した場合も含まれます。v2.1.216 より前は、Claude Code はその中断を "user_reject" として報告していました。拒否として扱われます。
  - "user_reject": プロンプトが表示されたときにユーザーが「No」を選択したときに出力されます。対話型 CLI では、その選択自体に対してのみ出力され、ユーザーの個人設定の拒否ルールに一致する呼び出しでは代わりに "config" が出力されます。Agent SDK または非対話型の -p セッションでは、個人設定の拒否ルールに一致する呼び出しで "user_reject" が出力されます。拒否として扱われます。
- tool_parameters（OTEL_LOG_TOOL_DETAILS=1 の場合）: ツール固有のパラメーターを含む JSON 文字列。**ツール結果イベント** (ページ 965) と同じ形式ですが、git_commit_id などの実行後のフィールドは含まれません。権限決定が updatedInput を介してツール入力を書き換えた場合、承認された呼び出しでは値が tool_result と異なることがあります。decision が "reject" の場合にどのコマンドが拒否されたかを確認するには、この属性を使用してください。
  - "sdk_host_builtin_mcp" ツールの場合: ホストアプリケーションがこれらの名前を定義しているため、OTEL_LOG_TOOL_DETAILS がオフの場合でも mcp_server_name と mcp_tool_name

が含まれます。これらがなければ、これらの組み込みサーバーへの拒否された呼び出しをデフォルトのストリームで特定できなくなります。ユーザーが設定した MCP サーバーの場合、イベントの `tool_name` は常にリテラルの `"mcp_tool"` であり、サーバー名とツール名はフラグがオンの場合にのみ `tool_parameters` に表示されます。引数の内容はどの場合もフラグが必要です。Claude Code v2.1.214 以降が必要です

- Bash ツールの場合: `bash_command`、`full_command`、`timeout`、`description`、`dangerouslyDisableSandbox` を含みます。デスクトップアプリのワークスペース Bash ツールも `tool_name` を `Bash` として報告しますが、`bash_command`、`full_command`、`timeout` のみを含みます
- MCP ツールの場合: `mcp_server_name`、`mcp_tool_name` を含みます
- Skill ツールの場合: `skill_name` を含みます
- Agent ツールまたは従来の Task ツールの場合: `subagent_type` を含みます

権限モード変更イベント

権限モードが変更されたときにログに記録されます。たとえば、`Shift+Tab` による切り替え、plan モードの終了、auto モードのゲートチェックなどです。

イベント名: `claude_code.permission_mode_changed`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"permission_mode_changed"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `from_mode`: 以前の権限モード。例: `"default"`、`"plan"`、`"acceptEdits"`、`"auto"`、`"bypassPermissions"`
- `to_mode`: 新しい権限モード
- `trigger`: 変更の原因。`"shift_tab"`、`"exit_plan_mode"`、`"auto_gate_denied"`、`"auto_opt_in"` のいずれかです。遷移が SDK またはブリッジから発生した場合は含まれません

認証イベント

`/login` または `/logout` が完了したときにログに記録されます。

イベント名: `claude_code.auth`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"auth"`
- `event.timestamp`: ISO 8601 タイムスタンプ

- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- action: "login" または "logout"
- success: "true" または "false"
- auth_method: 認証方法。"oauth" など
- error_category: アクションが失敗した場合のエラーの種類のカテゴリ。生のエラーメッセージが含まれることはありません
- status_code: アクションが HTTP エラーで失敗した場合の、文字列としての HTTP ステータスコード

MCP サーバー接続イベント

MCP サーバーが接続、切断、または接続に失敗したときにログに記録されます。

イベント名: claude_code.mcp_server_connection

属性:

- すべての**標準属性** (ページ 956)
- event.name: "mcp_server_connection"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- status: "connected"、"failed"、"disconnected" のいずれか
- transport_type: サーバーのトランスポート。"stdio"、"sse"、"http" など
- server_scope: サーバーが設定されているスコープ。"user"、"project"、"local" など
- duration_ms: 接続試行の所要時間（ミリ秒）
- error_code: 接続が失敗した場合のエラーコード
- is_plugin: サーバーがプラグインによって提供されている場合は true、それ以外は false
- plugin_id_hash（is_plugin が true の場合）: プラグイン名とマーケットプレイスの安定したハッシュ。名前を公開せずにプラグインごとにイベントをグループ化するために使用します。Claude Code は**プラグイン読み込みイベント** (ページ 976) で説明している方法でこれを計算します
- plugin.name（is_plugin が true の場合）: サーバーを提供するプラグインの名前。サードパーティのプラグインの場合、OTEL_LOG_TOOL_DETAILS=1 でない限り、これはリテラル文字列 "third-party" になります。これにより、デフォルトではサードパーティのプラグイン名がログに表示されないよう保護されます。Anthropic の公式ソースのプラグインは常に名前で識別されます。plugin_id_hash と plugin.name の属性は独自の監視バックエンドに送信され、Anthropic には送信されません
- server_name（OTEL_LOG_TOOL_DETAILS=1 の場合）: 設定されたサーバー名
- error（OTEL_LOG_TOOL_DETAILS=1 の場合）: 接続が失敗した場合の完全なエラーメッセージ

内部エラーイベント

Claude Code が予期しない内部エラーを捕捉したときにログに記録されます。エラークラス名と errno 形式のコードのみが記録されます。エラーメッセージとスタックトレースが含まれることはありません。このイベントは、Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry に対して実行している場合、または `DISABLE_ERROR_REPORTING` が設定されている場合は出力されません。

イベント名: `claude_code.internal_error`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"internal_error"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `error_name`: エラークラス名。`"TypeError"` や `"SyntaxError"` など
- `error_code`: エラーに存在する場合の Node.js の errno コード（`"ENOENT"` など）

プラグインインストールイベント

プラグインのインストールが完了したときにログに記録されます。`claude plugin install` CLI コマンドと対話型の `/plugin` UI の両方が対象です。

イベント名: `claude_code.plugin_installed`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"plugin_installed"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `marketplace.is_official`: マーケットプレイスが Anthropic の公式マーケットプレイスの場合は `"true"`、それ以外は `"false"`
- `install.trigger`: `"cli"` または `"ui"`
- `plugin.name`: インストールされたプラグインの名前。サードパーティのマーケットプレイスの場合は、`OTEL_LOG_TOOL_DETAILS=1` の場合にのみ含まれます
- `plugin.version`: マーケットプレイスのエントリで宣言されている場合のプラグインのバージョン。サードパーティのマーケットプレイスの場合は、`OTEL_LOG_TOOL_DETAILS=1` の場合にのみ含まれます
- `marketplace.name`: プラグインのインストール元のマーケットプレイス。サードパーティのマーケットプレイスの場合は、`OTEL_LOG_TOOL_DETAILS=1` の場合にのみ含まれます

プラグイン読み込みイベント

セッションの開始時に、有効化されているプラグインごとに 1 回ログに記録されます。インストール操作自体を記録する `plugin_installed` を補完するものとして、このイベントを使用してフリート全体でどのプラグインがアクティブかを把握できます。

イベント名: `claude_code.plugin_loaded`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"plugin_loaded"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `plugin.name`: プラグインの名前。公式マーケットプレイスおよび組み込みバンドル以外のプラグインの場合、`OTEL_LOG_TOOL_DETAILS=1` でない限り値は `"third-party"` になります
- `marketplace.name`: 判明している場合の、プラグインのインストール元のマーケットプレイス。`plugin.name` と同じ条件で `"third-party"` に秘匿化されます
- `plugin.version`: プラグインマニフェストのバージョン。名前が秘匿化されておらず、マニフェストでバージョンが宣言されている場合にのみ含まれます
- `plugin.scope`: プラグインの出自カテゴリ。`"official"`、`"community"`、`"org"`、`"user-local"`、`"default-bundle"` のいずれか
- `enabled_via`: プラグインが有効化された経緯。`"default-enable"`、`"org-policy"`、`"admin-install"`、`"seed-mount"`、`"user-install"` のいずれかです。`"admin-install"` の値は、Organization settings > Plugins & skills でプラグインが組織に対して必須または自動インストールに設定されていることを意味します。v2.1.246 より前は、Claude Code はこれらのプラグインを `"user-install"` または `"seed-mount"` として報告していました
- `plugin_id_hash`: プラグイン名とマーケットプレイスの決定論的なハッシュ。設定したエクスポーターにのみ送信されます。名前を記録せずに、フリート全体で読み込まれた個別のサードパーティプラグインの数をカウントできます。claude.ai から同期されたプラグインの場合、Claude Code は、claude.ai がそのプラグインについて報告するマーケットプレイス名、それがない場合は `synced` とプラグイン名を組み合わせてハッシュします。v2.1.246 より前は、Claude Code は claude.ai が報告するマーケットプレイス名をハッシュに使用していませんでした
- `has_hooks`: プラグインがフックを提供するかどうか
- `has_mcp`: プラグインが MCP サーバーを提供するかどうか
- `host_owned_mcp`: SDK ホストがこのプラグインの MCP 接続を管理しており、Claude Code がプラグインの MCP サーバー設定の読み取りをスキップした場合は `true`、それ以外は `false`。Claude Code v2.1.172 以降が必要です
- `skill_path_count`: プラグインが宣言するスキルディレクトリの数
- `command_path_count`: プラグインが宣言するコマンドディレクトリの数

- agent_path_count: プラグインが宣言するエージェントディレクトリの数
- safe_mode: セッションが *--safe-mode* (ページ 45) で開始された場合は "true"、それ以外は "false"。safe モードでは、このイベントは設定されたインベントリのみを報告し、プラグインのコマンド、スキル、フック、MCP サーバーは読み込まれません。Claude Code v2.1.169 以降が必要です

スキル有効化イベント

スキルが呼び出されたときにログに記録されます。Claude が Skill ツールを通じて呼び出した場合と、ユーザーが / コマンドとして実行した場合の両方が対象です。

イベント名: claude_code.skill_activated

属性:

- すべての**標準属性** (ページ 956)
- event.name: "skill_activated"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- skill.name: スキルの名前。ユーザー定義およびサードパーティのプラグインのスキルの場合、OTEL_LOG_TOOL_DETAILS=1 でない限り値はプレースホルダーの "custom_skill" になります
- invocation_trigger: スキルがトリガーされた方法（"user-slash"、"claude-proactive"、"nested-skill"）
- skill.source: スキルの読み込み元（例: "bundled"、"userSettings"、"projectSettings"、"plugin"）
- skill.kind: スキルがワークフロースキルの場合は "workflow"。それ以外の場合は含まれません
- plugin.name（OTEL_LOG_TOOL_DETAILS=1 の場合、またはプラグインが公式マーケットプレイスのものの場合）: スキルがプラグインによって提供されている場合の、所有元のプラグインの名前
- marketplace.name（OTEL_LOG_TOOL_DETAILS=1 の場合、またはプラグインが公式マーケットプレイスのものの場合）: スキルがプラグインによって提供されている場合の、所有元のプラグインのインストール元のマーケットプレイス

@ メンションイベント

Claude Code がプロンプト内の @ メンションを解決したときにログに記録されます。すべてのメンションでイベントが出力されるわけではありません。権限の拒否、サイズが大きすぎるファイル、PDF の参照添付、ディレクトリ一覧の取得失敗などの早期終了パスでは、ログに記録せずに戻ります。

イベント名: claude_code.at_mention

属性:

- すべての**標準属性** (ページ 956)
- event.name: "at_mention"

- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- mention_type: メンションの種類（"file"、"directory"、"agent"、"mcp_resource"、"peer"）。"peer" の値は、ユーザーの他の Claude Code セッションのいずれかをメンションしたことを意味します。Claude Code v2.1.232 以降が必要です
- success: メンションが正常に解決されたかどうか（"true" または "false"）

API 再試行上限到達イベント

API リクエストが複数回の試行の後に失敗したときに 1 回ログに記録されます。最後の api_error イベントと一緒に出力されます。

イベント名: claude_code.api_retries_exhausted

属性:

- すべての**標準属性** (ページ 956)
- event.name: "api_retries_exhausted"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- model: 使用されたモデル
- error: 最終的なエラーメッセージ
- status_code: 数値としての HTTP ステータスコード。HTTP 以外のエラーの場合は含まれません。
- total_attempts: 試行の合計回数
- total_retry_duration_ms: すべての試行にわたる合計の実経過時間
- speed: "fast" または "normal"

フック登録イベント

セッションの開始時に、設定されたフックごとに 1 回ログに記録されます。実行ごとの hook_execution_start および hook_execution_complete イベントを補完するものとして、このイベントを使用してフリート全体でどのフックがアクティブかを把握できます。

イベント名: claude_code.hook_registered

属性:

- すべての**標準属性** (ページ 956)
- event.name: "hook_registered"
- event.timestamp: ISO 8601 タイムスタンプ

- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- hook_event: フックイベントの種類。"PreToolUse" や "PostToolUse" など
- hook_type: フックの実装の種類。"command"、"prompt"、"mcp_tool"、"http"、"agent" のいずれか
- hook_source: フックが定義されている場所。"userSettings"、"projectSettings"、"localSettings"、"flagSettings"、"policySettings"、"pluginHook" のいずれか
- safe_mode: セッションが *--safe-mode* (ページ 45) で開始された場合は "true"、それ以外は "false"。Claude Code v2.1.169 以降が必要です
- hook_matcher（OTEL_LOG_TOOL_DETAILS=1 の場合）: フック設定で matcher が設定されている場合の matcher 文字列
- plugin.name（hook_source が "pluginHook" の場合）: フックを提供するプラグインの名前。公式マーケットプレイスおよび組み込みバンドル以外のプラグインの場合、OTEL_LOG_TOOL_DETAILS=1 でない限り値は "third-party" になります
- plugin_id_hash（hook_source が "pluginHook" の場合）: プラグイン名とマーケットプレイスの決定論的なハッシュ。設定したエクスポーターにのみ送信されます。名前を記録せずに、フックを提供する個別のプラグインの数をカウントできます。Claude Code は**プラグイン読み込みイベント** (ページ 976) で説明している方法でこれを計算します

フック実行開始イベント

フックイベントに対して 1 つ以上のフックの実行が開始されたときにログに記録されます。

イベント名: claude_code.hook_execution_start

属性:

- すべての**標準属性** (ページ 956)
- event.name: "hook_execution_start"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- hook_event: フックイベントの種類。"PreToolUse" や "PostToolUse" など
- hook_name: matcher を含む完全なフック名。"PreToolUse:Write" など
- num_hooks: 一致したフックコマンドの数
- managed_only: 管理ポリシーのフックのみが許可されている場合は "true"
- hook_source: "policySettings" または "merged"
- safe_mode: セッションが *--safe-mode* (ページ 45) で開始された場合は "true"、それ以外は "false"。Claude Code v2.1.169 以降が必要です

- hook_definitions: JSON シリアライズされたフック設定。詳細なベータトレースと OTEL_LOG_TOOL_DETAILS=1 の両方が有効な場合にのみ含まれます

フック実行完了イベント

フックイベントに対するすべてのフックが完了したときにログに記録されます。

イベント名: claude_code.hook_execution_complete

属性:

- すべての**標準属性** (ページ 956)
- event.name: "hook_execution_complete"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- hook_event: フックイベントの種類
- hook_name: matcher を含む完全なフック名
- num_hooks: 一致したフックコマンドの数
- num_success: 正常に完了した数
- num_blocking: ブロッキングの決定を返した数
- num_non_blocking_error: ブロックせずに失敗した数
- num_cancelled: 完了前にキャンセルされた数
- total_duration_ms: 一致したすべてのフックの実経過時間
- stdout_chars: 成功した一致フック全体の stdout の合計文字数。Claude Code v2.1.280 以降が必要です
- additional_context_chars: 一致したフックが返した additionalContext の合計文字数。Claude Code v2.1.280 以降が必要です
- system_message_chars: 一致したフックが返した systemMessage の合計文字数。Claude Code v2.1.280 以降が必要です
- initial_user_message_chars: 一致したフックが返した initialUserMessage の合計文字数。Claude Code v2.1.280 以降が必要です
- num_outputs_persisted: *10,000* **文字の上限** (ページ 474) を超えたために Claude Code がファイルに保存したフック出力の数。Claude Code v2.1.280 以降が必要です
- managed_only: 管理ポリシーのフックのみが許可されている場合は "true"
- hook_source: "policySettings" または "merged"
- safe_mode: セッションが *--safe-mode* (ページ 45) で開始された場合は "true"、それ以外は "false"。Claude Code v2.1.169 以降が必要です

- hook_definitions: JSON シリアライズされたフック設定。詳細なベータトレースと OTEL_LOG_TOOL_DETAILS=1 の両方が有効な場合にのみ含まれます

フックプラグインメトリクスイベント

公式マーケットプレイスのプラグインのフックが呼び出しごとのメトリクスを出力したときにログに記録されます。これを出力できるのは、Anthropic の公式マーケットプレイスからインストールされたプラグインのみです。サードパーティのマーケットプレイスのプラグインやユーザーが設定したフックは、このイベントに出力しません。このイベントを使用すると、検出率、コスト、所要時間などのプラグインの挙動を独自のオブザーバビリティスタックから監視できます。

イベント名: claude_code.hook_plugin_metrics

属性:

- すべての**標準属性** (ページ 956)
- event.name: "hook_plugin_metrics"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- plugin_id: <name>@<marketplace> 形式のプラグイン識別子
- hook_event: メトリクスを出力したフックイベントの種類
- プラグインが出力する最大 20 個のメトリクスキー。名前は ^[a-z][a-z0-9_]{0,39}$ に一致します。値はブール値または数値です。

コンテキスト圧縮イベント

会話のコンテキスト圧縮が完了したときにログに記録されます。

イベント名: claude_code.compaction

属性:

- すべての**標準属性** (ページ 956)
- event.name: "compaction"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- trigger: "auto" または "manual"
- success: "true" または "false"
- duration_ms: 圧縮の所要時間
- pre_tokens: 圧縮前のおおよそのトークン数
- post_tokens: 圧縮後のおおよそのトークン数

- error: 圧縮が失敗した場合のエラーメッセージ
- precompute_reuse: trigger が "manual" の場合にのみ設定されます。自動圧縮では、コンテキストウィンドウがいっぱいになる前にバックグラウンドで要約を準備できます。この属性は、/compact がその準備済みの要約を再利用したかどうかを記録します。"hit" は再利用されたことを意味し、"miss_custom_instructions"、"miss_hook"、"miss_not_ready" は代わりに新しい要約が計算された理由を示します。Claude Code v2.1.153 以降が必要です

サブエージェント完了イベント

サブエージェント (ページ 317) が終了し、その結果を起動元の会話に返したときに記録されます。サブエージェントの種類ごとにツール使用と実行時間を集計する用途に使用します。トークンやコストを集計する場合は、query_source を "subagent" でフィルタリングした **トークンカウンター** (ページ 961) と **コストカウンター** (ページ 960) を使用してください。このイベントの total_tokens は最後のリクエストのみを対象とするためです。"subagent" カテゴリには、エージェントベースのフックからのリクエストも含まれますが、これらはサブエージェントイベントを出力しません。

イベント名: claude_code.subagent_completed

属性:

- すべての**標準属性** (ページ 956)
- event.name: "subagent_completed"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- agent_type: サブエージェントの種類。組み込みエージェント名と公式マーケットプレイスのプラグインからのエージェントはそのまま表示されます。その他のエージェント名は、OTEL_LOG_TOOL_DETAILS=1 が設定されていない限り "custom" に置き換えられます
- agent.source: エージェント定義の取得元。built-in、plugin、またはカスタムエージェントを定義した設定ソース（userSettings や projectSettings など）
- is_built_in: サブエージェントが組み込みのエージェントタイプかどうか
- is_async: サブエージェントが**バックグラウンド** (ページ 344) で実行されたかどうか
- total_tokens: サブエージェントの最後の API リクエストのトークン量。その 1 つのリクエストの入力、キャッシュ作成、キャッシュ読み取り、出力トークンであり、完了時点のサブエージェントのコンテキストサイズにおおよそ相当します。実行全体の合計ではありません
- total_tool_uses: サブエージェントが実行全体で行ったツール呼び出しの数
- duration_ms: 実行時間（ミリ秒）
- model: サブエージェントの実行に解決されたモデル
- final_model: サブエージェントの最終応答を生成したモデル。フォールバックなど実行途中で切り替えがあった場合は model と異なります。Claude Code v2.1.212 以降が必要です

- `model_swapped`: サブエージェントのリクエストを複数のモデルが処理したかどうか。Claude Code v2.1.212 以降が必要です
- `plugin_id_hash`、`plugin.name`: プラグインが提供するエージェントの場合に存在します。公式マーケットプレイスのプラグイン名はそのまま表示されます。その他のプラグイン名は、`OTEL_LOG_TOOL_DETAILS=1` が設定されていない限り `"third-party"` に置き換えられます

フィードバックアンケートイベント

セッション品質アンケートが表示されたとき、または回答されたときに記録されます。アンケートで収集される内容とその制御方法については、**セッション品質アンケート** (ページ 1106) を参照してください。

イベント名: `claude_code.feedback_survey`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"feedback_survey"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- `event_type`: アンケートのライフサイクルイベント。例: `"appeared"`、`"responded"`、`"transcript_prompt_appeared"`
- `appearance_id`: 1 つのアンケートインスタンスに対して出力されたイベントを結び付ける一意の ID
- `survey_type`: イベントを生成したアンケート。`"session"` は「Claude の調子はどうですか？」という評価プロンプトです
- `response`: `responded` イベントにおけるユーザーの選択
- `enabled_via_override`: `CLAUDE_CODE_ENABLE_FEEDBACK_SURVEY_FOR_OTEL` が設定されている場合は `true`。文字列ではなくブール値として出力されます。`session` アンケートイベントに存在します。この属性でフィルタリングすると、上書きがフリート全体に適用されていることを確認できます

保持期間スイープイベント

保持期間クリーンアップスイープの実行ごとに 1 回記録されます。このスイープは、`cleanupPeriodDays` 設定よりも古いセッショントランスクリプトやその他のアプリケーションデータを削除します。Claude Code はセッションごとに最大 1 回、バックグラウンドでスイープを実行し、何も削除しなかった実行でもイベントを出力します。同じマシン上のいずれかのセッションで過去 24 時間以内に Claude Code がスイープを実行していた場合、このセッションのスイープは少なくとも 10 分遅延されるため、それより早く終了したセッションは何も出力しません。`claude -p` を `--bare` とともに実行した場合、Claude Code はスイープを実行せず、何も出力しません。

このページのすべての OTel イベントと同様に、このイベントは設定したテレメトリバックエンドにのみ送信されます。Claude Code v2.1.227 以降が必要です。

Claude Code が保持期間を安全に判断できない場合、スイープを一時停止し、`result` を `"skipped"` に設定し、`skip_reason` を付けてイベントを出力します。管理設定で `cleanupPeriodDays` が設定されている場

合は、管理設定の値によって保持期間が固定され、より優先度の低いスコープの設定ファイルが壊れていたり無効であったりしてもスイープは実行されます。managed-settings.json 自体を読み取れない場合、管理層がサーバー管理設定や壊れたファイルの隣にある managed-settings.d/ ドロップインなど、他の場所から cleanupPeriodDays を提供しない限り、Claude Code はスイープを一時停止します。削除カウンター属性は、result が "complete" の場合にのみ存在します。

イベント名: claude_code.retention_sweep

属性:

- すべての**標準属性** (ページ 956)
- event.name: "retention_sweep"
- event.timestamp: ISO 8601 タイムスタンプ
- event.sequence: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています
- result: スイープが実行された場合は "complete"、Claude Code が一時停止した場合は "skipped"
- period_days: マージされた設定の cleanupPeriodDays の値（日数）。どのソースでも設定されていない場合は 30。skipped イベントでは、Claude Code が読み取れた設定ソースから算出した、スイープが使用するはずだった値
- used_default: 読み取り可能な設定ソースのいずれでも cleanupPeriodDays が設定されていない場合は "true"、それ以外は "false"。complete イベントでは、"true" は 30 日のデフォルトが適用されたことを意味します
- skip_reason: Claude Code がスイープを一時停止した理由。result が "skipped" の場合にのみ存在します:
  - "user_source_disabled": *--setting-sources* (ページ 51) フラグや SDK の setting Sources オプションなどによってユーザー設定が除外されており、有効なソースのいずれも cleanupPeriodDays を提供していない
  - "settings_unknowable": 設定ファイルを読み取りまたは解析できなかったため、cleanup PeriodDays または desktopSessionCleanupPeriodDays が Claude Code から見えない値に設定されている可能性がある
  - "settings_invalid_key_set": 設定に検証エラーがあり、かつ cleanupPeriodDays または desktopSessionCleanupPeriodDays が明示的に設定されているため、デフォルトにフォールバックするとその設定に反してファイルを削除または保持してしまう可能性がある
- transcripts_deleted: スイープが削除したセッショントランスクリプト（最上位の ~/.claude/projects/*/*.jsonl ファイル）の数
- transcripts_exempted_desktop: 保持期間を過ぎているものの、Claude Desktop と Cowork のルールによりスイープが保持したトランスクリプトの数。これらは files_past_cutoff にはカウントされません。Claude Code v2.1.248 以降が必要です
- session_files_deleted: セッションファイルスイープが削除したアーティファクトの数。トランスクリプトに加え、サイドカー、録画、ツール結果などのセッションごとの付随ファイルを含みます

- `artifacts_deleted`: スイープが対象とするデータディレクトリ全体で削除した項目の合計（セッションファイルを含む）。一部のスイープは削除したディレクトリツリー全体を 1 項目としてカウントし、いくつかのクリーンアップパスはカウンターに加算されないため、この値は正確なファイル数ではなく下限値として扱ってください
- `files_retained_fresh`: 検査されたものの、まだ保持期間内であるためそのまま残されたファイル。ファイル単位のスイープのみがこれらをカウントするため、この値は下限値です。ゼロ以外の値は通常の定常状態です
- `files_past_cutoff`: 保持期間より古いものの、権限エラーやファイルが開かれたままであることなどが原因でスイープが削除に失敗したファイル。ゼロより大きい値は、設定された保持期間を超えてファイルが残ったことを意味します。ただし、ディレクトリ全体の削除に失敗した場合は代わりに `error_count` にカウントされるため、ゼロであってもそのようなファイルがないことの証明にはなりません
- `error_count`: ファイルの一覧取得または削除中にスイープが遭遇したエラーの数

管理設定解決イベント

セッションが解決した管理設定とともに記録されます。セッション開始時に 1 回、セッション中に管理設定またはポリシーヘルパーの状態が変化したときに再度、そして `error.type` 属性に列挙されている理由のいずれかにより Claude Code が起動を拒否するかセッションを終了したときに記録されます。このイベントを使用して、予期しない管理ソースで実行されているマシン、ポリシーヘルパーが失敗しているマシン、およびマシンが起動を拒否した理由を特定できます。 Claude Code v2.1.274 以降が必要です。

デフォルトでは、このイベントには管理ソースとポリシーヘルパーの状態が含まれますが、設定自体は含まれません。編集済みの `managed_settings.settings` 属性と `managed_settings.resolved_sha256` ダイジェストを追加するには、`OTEL_LOG_MANAGED_SETTINGS=1` を設定します:

- 管理設定、ユーザー設定、または `--settings` の `env` ブロック、あるいは Claude Code を起動する環境で設定してください。プロジェクト設定やローカル設定の値では有効になりません。クローンしたリポジトリがこれらを書き込めるためです。
- サーバー管理設定では、セキュリティ承認ダイアログを表示せずにこれを設定できます。この変数は、組織がすでに受け取っているイベントに、組織自身の編集済みポリシーを追加するだけだからです。

信頼 (ページ 1066) していないフォルダーでの対話型セッションでは、Claude Code は拒否イベントをエクスポートしません。

イベント名: `claude_code.managed_settings_resolved`

属性:

- すべての**標準属性** (ページ 956)
- `event.name`: `"managed_settings_resolved"`
- `event.timestamp`: ISO 8601 タイムスタンプ
- `event.sequence`: イベントを順序付けるためのプロセスごとのカウンター。**イベント相関属性** (ページ 962) で説明しています

- managed_settings.trigger: セッション開始時のイベントの場合は "startup"、セッションの後半で管理設定またはポリシーヘルパーの状態が変化した場合は "change"、管理設定ポリシーによってセッションが停止された場合は "refused"。Claude Code は、最後に送信したイベントと属性が異なる場合にのみ change イベントを送信します。設定値の変更は、OTEL_LOG_MANAGED_SETTINGS がオフの場合でもカウントされます
- error.type: Claude Code がセッションを停止した理由。refused イベントにのみ存在します:
  - "helper_failed": ポリシーヘルパーの実行が失敗した
  - "policy_invalid": 管理設定に Claude Code の起動を妨げるエラーが含まれている、または管理ソースが読み取り拒否以外の理由で読み込みに失敗したため、Claude Code が組織ログインやプロバイダーの強制を確認できない
  - "provider_not_allowed": セッションが、管理設定の allowedProviders リストで許可されていない API プロバイダーを使用する、またはプロバイダーのトラフィックを許可されていないホストに送信しようとしている。Claude Code v2.1.285 以降が必要です
  - "consent_rejected": ユーザーがサーバー管理設定のセキュリティ承認ダイアログを拒否した
  - "force_refresh_failed": forceRemoteSettingsRefresh が必要とする設定の取得に失敗した
  - "gateway_rejected": Claude apps ゲートウェイが管理設定の読み込みに HTTP 403 で応答した
  - "version_below_minimum": この Claude Code のバージョンが requiredMinimumVersion を下回っているか、requiredMaximumVersion を上回っている
  - "_OTHER": Claude apps ゲートウェイの管理設定の読み込みがその他の理由で失敗した
- managed_settings.sources: 少なくとも 1 つのポリシーキーを提供するすべての管理ソース。優先度の高い順に並び、first-wins の下でキーが有効にならないソースも含まれます。値は、MDM または OS レベルのポリシーの場合は "remote"、"plist"、または "hklm"、管理設定ファイルとドロップインの場合は "file"、埋め込みホストが設定を提供する場合は "parent"、Claude Code が Windows HKCU レジストリ値を読み取る場合は "hkcu" です。制御キーのみを含むソースや、Claude Code が読み取れなかったソースは一覧に含まれません。文字列の配列として出力され、ポリシーキーを提供する管理ソースがない場合は空になります
- managed_settings.source_behavior: Claude Code が読み取った managedSourcesBehavior の値で、"first-wins" または "merge"。どのソースでもこのキーが設定されていない場合は "first-wins"
- managed_settings.helper.state: 選択された MDM またはファイルソースが設定するポリシーヘルパーの状態:
  - "ok": ヘルパーの出力が管理設定として使用されている
  - "bad_path"、"not_a_file"、"exit_nonzero"、"timed_out"、"oversize"、"parse_failed"、"envelope_invalid"、または "schema_rejected": ヘルパーの最後の実行が失敗した。各ケースについてはヘルパーの失敗で説明しています
  - "none": ヘルパーが設定されていない、またはヘルパーを設定するソースが MDM ポリシーや管理設定ファイルではない

- managed_settings.helper.applied: ヘルパー自身の出力が管理設定として使用されている間は "output"、そうでない場合は "none"
- managed_settings.helper.entry: Claude Code が policyHelper を選択した場合は "policyHelper"。ヘルパーを選択しなかった場合は存在しません
- managed_settings.helper.path: ヘルパーに設定された path。Claude Code がヘルパーを選択した場合は、OTEL_LOG_MANAGED_SETTINGS が設定されているかどうかにかかわらず常に存在します
- managed_settings.resolved_sha256 (OTEL_LOG_MANAGED_SETTINGS=1 の場合): 編集前の解決済み管理設定の SHA-256。キーを再帰的にソートし、空白なしの JSON としてシリアル化したものです。同じダイジェストを持つマシンは同じポリシーで実行されています。短いポリシーは推測値をハッシュ化することで復元できてしまうため、Claude Code はオプトインした場合にのみダイジェストを送信します。管理設定が解決されなかった場合、および refused イベントでは存在しません
- managed_settings.settings（OTEL_LOG_MANAGED_SETTINGS=1 の場合）: 解決済み管理設定の名前と構造を、値を編集した JSON 文字列として表したもの。refused イベントでは存在しません。Claude Code は設定スキーマに基づいてこれを構築します:
  - スキーマで宣言されている設定名はエクスポートされ、宣言されていないキーは除外されます
  - ブール値、数値、およびスキーマが固定の選択肢に制限している文字列値 (permissions.defaultMode など) はそのままエクスポートされます。sandbox.network.httpProxyPort と sandbox.network.socksProxyPort は "[REDACTED]" としてエクスポートされます
  - model、apiKeyHelper、すべての env の値、すべての URL、すべてのコマンドなど、その他のすべての文字列は "[REDACTED]" としてエクスポートされます
  - env の変数名やプラグイン ID など、マップのエントリ名はそのままエクスポートされます。vimInsertModeRemaps など、スキーマがエントリの型を定義していない設定は単一の "[REDACTED]" としてエクスポートされ、sandbox.ignoreViolations はコマンドパターンを除いたパスリストのリストとしてエクスポートされます
  - リストは長さを保持し、各エントリは同じルールで編集されます
  - permissions.allow、permissions.deny、または permissions.ask のルールは、ツールがこのバージョンの Claude Code に組み込まれている場合、または mcp__jira__create_issue のような mcp__ 参照である場合、Read([REDACTED]) のように内容を編集したツール名としてエクスポートされます。その他のルールは "[REDACTED]" としてエクスポートされます
  - フックも同じルールに従うため、type や timeout などの固定選択肢のフィールドや数値フィールドは表示されますが、各コマンド、URL、matcher、if 条件は "[REDACTED]" としてエクスポートされます

  たとえば、apiKeyHelper、2 つの env 変数、および拒否ルールを含む管理設定は、{"apiKeyHelper":"[REDACTED]","env":{"HTTPS_PROXY":"[REDACTED]","CLAUDE_CODE_ENABLE_TELEMETRY":"[REDACTED]"},"permissions":{"deny":["Read([REDACTED])"]}} としてエクスポートされます。

  Claude Code は値を UTF-8 で 8 KB に切り詰め、切り詰められた値は有効な JSON ではありません
- managed_settings.settings_truncated (managed_settings.settings が存在する場合): Claude Code が managed_settings.settings を 8 KB で切り詰めた場合は true、それ以外は false。文字列ではなくブール値として出力されます

42.6 メトリクスとイベントデータの解釈

エクスポートされたメトリクスとイベントは、さまざまな分析をサポートします:

42.6.1 使用状況監視

メトリクス	分析の機会
`claude_code.token.usage`	トークンの *type* (ページ 961)、ユーザー、チーム、モデル、`skill.name`、`plugin.name`、または `agent.name` 別に分類
`claude_code.session.count`	時間経過に伴う採用と関与を追跡
`claude_code.lines_of_code.count`	コード追加と削除を追跡して生産性を測定し、モデル別に分類
`claude_code.commit.count` & `claude_code.pull_request.count`	開発ワークフローへの影響を理解

42.6.2 コスト監視

`claude_code.cost.usage` メトリクスは以下に役立ちます:

- チームまたは個人全体の使用トレンドを追跡する
- 最適化のための高使用セッションを特定する
- `skill.name`、`plugin.name`、および `agent.name` 属性を介して、特定のスキル、プラグイン、またはサブエージェントタイプへの支出を属性付けする

注釈:

コストメトリクスは概算です。公式な請求データについては、API プロバイダー (Claude Console、Amazon Bedrock、または Google Cloud の Agent Platform) を参照してください。

Claude Code は、`ANTHROPIC_BASE_URL` の背後にあるゲートウェイまたはプロキシが複数のフレーム全体で使用状況をプログレッシブにストリーミングする場合を含め、各ストリーミングレスポンスをコストおよびトークンメトリクスに対して正確に 1 回カウントします。v2.1.214 より前では、複数のフレームで使用状況を含むストリームは、`claude_code.cost.usage` と `claude_code.token.usage` を追加フレームごとにおよそ 1 つの追加フルリクエスト分だけ増加させました。

42.6.3 アラートとセグメンテーション

検討すべき一般的なアラート:

- コストスパイク
- 異常なトークン消費
- 特定のユーザーからの高いセッションボリューム

すべてのメトリクスは、**標準属性** (ページ 956) でセグメント化できます。`model` 属性は `claude_code.token.usage`、`claude_code.cost.usage`、および v2.1.172 以降の `claude_code.lines_of_code.count` で利用可能です。

コミットのモデル別の内訳は、1 つのセッションが複数のモデルにまたがる可能性があるため、`session.id` でトークンまたはコストメトリクスに対して結合することによってのみ概算できます。トークンまたはコスト側をフィルタリングして、`query_source` が `"main"` である行のみにしてください。これにより、補助的なリクエストとサブエージェントリクエストが、セッションのコミットをそれらを作成しなかったモデルに属性付けしません。

42.6.4 再試行枯渇の検出

Claude Code は失敗した API リクエストを内部的に再試行し、あきらめた後にのみ単一の `claude_code.api_error` イベントを出力するため、イベント自体がそのリクエストの終端信号です。中間再試行試行は個別のイベントとしてログされません。

イベントの `attempt` 属性は、試行の総数を記録します。`CLAUDE_CODE_MAX_RETRIES` はデフォルトで 10 で、15 で上限です。v2.1.199 以降では、`CLAUDE_CODE_RETRY_WATCHDOG` を設定してデフォルトを引き上げ、上限を削除できます。

リクエストが一時的なエラーのすべての再試行を枯渇させた場合、`attempt` はその有効な制限より 1 つ多くなります: デフォルトでは 11、ウォッチドッグが設定されていない限り 16 を超えることはありません。より低い値は、`400` レスポンスなどの再試行不可能なエラー、または独自のより小さい再試行予算を持つ原因を示します。たとえば、Claude Code は AWS または Google Cloud 認証情報の読み込み失敗を最大 2 回再試行します。

セッションが回復したものと停止したものを区別するには、イベントを `session.id` でグループ化し、エラーの後に後続の `api_request` イベントが存在するかどうかを確認します。

42.6.5 イベント分析

イベントデータは Claude Code インタラクションに関する詳細な洞察を提供します:

ツール使用パターン: ツール結果イベントを分析して以下を特定します:

- 最も頻繁に使用されるツール
- ツール成功率
- 平均ツール実行時間
- ツールタイプ別のエラーパターン

パフォーマンス監視: API リクエスト期間とツール実行時間を追跡して、パフォーマンスボトルネックを特定します。

42.6.6 入力トークンを OpenTelemetry GenAI セマンティック規約にマッピングする

Claude Code は、入力トークン数を API レスポンスの usage ブロックに表示されるとおりにエクスポートするため、これらの値には プロンプトキャッシュ から読み取られたトークンやキャッシュに書き込まれたトークンは含まれません:

- *claude_code.llm_request* (ページ 945) スパンと *api_request* (ページ 966) イベントの input_tokens
- *claude_code.token.usage* (ページ 961) メトリクスの "input" タイプ

Claude Code は gen_ai.usage.* 属性を設定しません。OpenTelemetry GenAI セマンティック規約 では、gen_ai.usage.input_tokens にはキャッシュから読み取られたトークンとキャッシュに書き込まれたトークンを含めるべきとされています。その合計を計算するには:

- スパンまたはイベントから: input_tokens、cache_read_tokens、cache_creation_tokens を合計します
- claude_code.token.usage メトリクスから: その "input"、"cacheRead"、"cacheCreation" タイプを合計します

規約では、キャッシュ読み取りとキャッシュ書き込みに対して個別の属性も定義されています:

- cache_read_tokens は gen_ai.usage.cache_read.input_tokens に対応します
- cache_creation_tokens は gen_ai.usage.cache_write.input_tokens に対応します。古いバージョンの規約ではキャッシュ書き込み属性の名前が gen_ai.usage.cache_creation.input_tokens となっているため、バックエンドが想定する名前を使用してください。

42.7 監査セキュリティイベント

OpenTelemetry イベントは Claude Code アクティビティの監査データソースです。すべてのイベントは、ツール呼び出し、MCP アクティビティ、権限決定をそれらをトリガーしたユーザーに結び付ける ID 属性を持ち、OTLP ログエクスポーターは、これらのイベントを OTLP レシーバーを持つセキュリティ情報およびイベント管理（SIEM）プラットフォーム、または SIEM にフォワードする OpenTelemetry Collector に配信できます。

42.7.1 属性アクションをユーザーに関連付ける

各イベントの **標準属性** (ページ 956) には、認証されたユーザーの ID が含まれます：Claude アカウントでサインインしている場合は user.email、user.account_uuid、user.account_id、および organization.id、さらに **クラウドセッション** (ページ 223) では、セッション自体の認証情報がそれらを持つ場合、user.id とセッションごとの session.id。user.id はインストールスコープの識別子です。ただし、Claude apps gateway セッションでは /login を通じてサインインしている場合、ゲートウェイが発行したトークンからの IdP サブジェクトです。

開発者が開始したセッションでは、MCP ツール呼び出し、Bash コマンド、ファイル編集はその開発者に属性付けられます。Claude Code は個別のサービスアカウントの下では機能しません。各イベントに記録される ID は、開発者自身の Claude アカウント、または Claude apps gateway セッションでの開発者の IdP ID です。Claude Tag チャネルセッションでは、Claude はあなたの組織の 共有 ID として機能します。

Claude Code が直接 API キーで認証する場合、または Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry に対して認証する場合、セッションに Claude アカウントはなく、user.id と session.id のみが入力されます。これらのデプロイメントでは、OTEL_RESOURCE_ATTRIBUTES を使用してユーザー ID を自分で添付し、**管理設定** (ページ 936) ファイルまたはローンチラッパーを通じてユーザーごとに設定します。Claude apps gateway セッションはこれを必要としません：**標準属性** (ページ 956) を参照して、それらのエクスポートが持つ ID を確認してください。

```
export OTEL_RESOURCE_ATTRIBUTES="enduser.id=jdoe@example.com,enduser.directory_id=S-1-
↪5-21-..."
```

42.7.2 MCP アクティビティを監査する

完全なコール詳細で MCP サーバーアクティビティをキャプチャするには、ログエクスポーターを有効にし、OTEL_LOG_TOOL_DETAILS=1 を設定します。その後、各 MCP 操作は、標準 ID 属性と共にサーバー名、ツール名、呼び出し引数を含む構造化イベントを生成します：

イベント	MCP に対して記録するもの
mcp_server_connection	server_name、transport_type、server_scope、およびエラー詳細を含むサーバー接続、切断、接続失敗
tool_result	tool_name および mcp_server_scope を含む各 MCP ツール呼び出し、mcp_server_name および mcp_tool_name を含む tool_parameters ペイロード、および呼び出し引数を含む tool_input ペイロード
tool_decision	呼び出しが許可されたか拒否されたか、および決定が設定、フック、またはユーザーから来たかどうか、および mcp_server_name と mcp_tool_name を含む tool_parameters ペイロード

OTEL_LOG_TOOL_DETAILS がない場合、これらのイベントは識別詳細を削除します：

- tool_result：mcp_server_scope と、ユーザー設定サーバーの場合はリテラル "mcp_tool" に編集された tool_name を保持し、引数コンテンツを省略します。Claude Desktop の組み込みサーバーの場合、Claude Desktop が所有するセッションでは、tool_parameters 内の mcp_server_name/mcp_tool_name ペアも保持します。これは tool_decision と同じホスト作成例外です。Claude Code v2.1.214 以降が必要です
- tool_decision：tool_source と、ユーザー設定サーバーの場合はリテラル "mcp_tool" に編集された tool_name を保持し、引数コンテンツを省略します。Claude Desktop の組み込みサーバーの場合、Claude Desktop が所有するセッションでは、tool_parameters 内の mcp_server_name/mcp_tool_name ペアも保持します。tool_source と名前ペアの両方に Claude Code v2.1.214 以降が必要です
- mcp_server_connection：server_name とエラーメッセージを省略しますが、is_plugin、plugin_id_hash、および plugin.name を保持し、Anthropic 以外のプラグイン名はリテラル "third-party" に編集されるため、プラグイン提供サーバーは詳細ログなしで区別可能なままです

42.7.3 セキュリティの質問をイベントにマップする

検出ルールを構築する場合、監視したいシグナルを検索し、対応するイベントと属性についてバックエンドをクエリします：

シグナル	イベント	キー属性
ツール呼び出しが許可または拒否され、何によって	tool_decision	decision、source、tool_name、tool_parameters
権限モードのエスカレーション	permission_mode_changed	from_mode、to_mode、trigger
ポリシーフックがアクションをブロック	hook_execution_complete	hook_event、num_blocking
ログイン、ログアウト、認証失敗	auth	action、success、error_category
MCP サーバー接続または失敗	mcp_server_connection	status、server_name、is_plugin、error_code
プラグインがインストールされ、そのソース	plugin_installed	plugin.name、marketplace.name、marketplace.is_official
実行されたコマンドとタッチされたファイル	tool_result（実行）または tool_decision（拒否）（OTEL_LOG_TOOL_DETAILS=1 の場合）	tool_parameters；tool_input（tool_result のみ）
マシンが実行する管理設定ソース、そのポリシーヘルパーが正常かどうか、およびマシンが起動を拒否した理由	managed_settings_resolved	managed_settings.trigger、managed_settings.sources、managed_settings.source_behavior、managed_settings.helper.state、error.type；managed_settings.settings および managed_settings.resolved_sha256（OTEL_LOG_MANAGED_SETTINGS=1 の場合）

Claude Code は生のイベントストリームのみを出力します。異常検出、ベースライン化、セッション間の相関、アラートは SIEM または可観測性バックエンドの責任です。

42.7.4 SIEM にイベントを送信する

OTEL_EXPORTER_OTLP_LOGS_ENDPOINT を SIEM の OTLP レシーバーに、または SIEM のネイティブ取り込み API にフォワードする OpenTelemetry Collector に指定します。以下の管理設定の例は、MCP および Bash 監査のための完全なツール詳細を有効にして、イベントのみをエクスポートします：

```
{
  "env": {
    "CLAUDE_CODE_ENABLE_TELEMETRY": "1",
    "OTEL_LOGS_EXPORTER": "otlp",
    "OTEL_LOG_TOOL_DETAILS": "1",
    "OTEL_EXPORTER_OTLP_LOGS_PROTOCOL": "http/protobuf",
    "OTEL_EXPORTER_OTLP_LOGS_ENDPOINT": "https://siem.example.com:4318/v1/logs",
    "OTEL_EXPORTER_OTLP_HEADERS": "Authorization=Bearer your-siem-token"
  }
}
```

イベントが到着したことを確認するには、この設定で実行されているセッションでプロンプトを送信し、SIEM で `claude_code.user_prompt` イベントを確認します。何も到着しない場合は、`claude --debug-file <path>` で Claude Code を起動し、そのログで `[3P telemetry]` エクスポートエラーを確認します。

42.8 バックエンドに関する考慮事項

メトリクス、ログ、トレースバックエンドの選択により、実行できる分析のタイプが決まります:

42.8.1 メトリクスの場合

- **時系列データベース**: レート計算、集約メトリクス
- **カラムナーストア**: 複雑なクエリ、一意のユーザー分析
- **フル機能の可観測性プラットフォーム**: 高度なクエリ、可視化、アラート

42.8.2 イベント/ログの場合

- **ログ集約システム**: 全文検索、ログ分析
- **カラムナーストア**: 構造化イベント分析
- **フル機能の可観測性プラットフォーム**: メトリクスとイベント間の相関

42.8.3 トレースの場合

分散トレースストレージとスパン相関をサポートするバックエンドを選択します:

- **分散トレースシステム**: スパン可視化、リクエストウォーターフォール、レイテンシー分析
- **フル機能の可観測性プラットフォーム**: トレース検索とメトリクスおよびログとの相関

日次/週次/月次アクティブユーザー (DAU/WAU/MAU) メトリクスが必要な組織の場合は、効率的な一意値クエリをサポートするバックエンドを検討してください。

42.9 サービス情報

すべてのメトリクスとイベントは、以下のリソース属性でエクスポートされます:

- `service.name`: ターミナルセッションの場合は `claude-code`、*Claude Desktop* **アプリ** (ページ 137) のコードタブから開始されたセッションの場合は `claude-code-desktop`
- `service.version`: 現在の Claude Code バージョン、またはコードタブセッションの場合は Desktop アプリバージョン
- `os.type`: オペレーティングシステムタイプ (例: `linux`、`darwin`、`windows`)
- `os.version`: オペレーティングシステムバージョン文字列
- `host.arch`: ホストアーキテクチャ (例: `amd64`、`arm64`)
- `wsl.version`: WSL バージョン番号 (Windows Subsystem for Linux で実行している場合のみ存在)

- メーター名: `com.anthropic.claude_code`

`service.name = claude-code` でフィルタリングするコレクターパイプラインまたはダッシュボードがある場合は、コードタブセッションからのテレメトリもキャプチャするために、フィルターに `claude-code-desktop` を追加してください。

42.10 ROI 測定リソース

テレメトリセットアップ、コスト分析、生産性メトリクス、自動レポート生成を含む Claude Code の投資収益率（ROI）測定に関する包括的なガイドについては、Claude Code ROI 測定ガイドを参照してください。このリポジトリは、すぐに使用できる Docker Compose 設定、Prometheus と OpenTelemetry セットアップ、Linear などのツールと統合された生産性レポート生成テンプレートを提供します。

42.11 セキュリティとプライバシー

- OpenTelemetry エクスポートをバックエンドに送信することはオプトインであり、明示的な設定が必要です。Anthropic の個別の運用テレメトリと無効化方法については、**データ使用** (ページ 1109) を参照してください
- 生のファイルコンテンツとコードスニペットはメトリクスやイベントに含まれません。トレーススパンは別のデータパスです。以下の `OTEL_LOG_TOOL_CONTENT` の項目を参照してください
- OAuth 経由で認証されている場合、`user.email` はテレメトリ属性に含まれ、設定した OTel エンドポイントにのみ送信され、Anthropic には送信されません。これが組織にとって懸念事項である場合は、テレメトリバックエンドと協力してこのフィールドをフィルタリングまたは編集してください
- ユーザープロンプトコンテンツはデフォルトでは収集されません。プロンプト長のみが記録されます。プロンプトコンテンツを含めるには、`OTEL_LOG_USER_PROMPTS=1` を設定してください。詳細なベータトレースでは、この変数はプロンプトテキストよりも広い範囲に達します。これは *new_context* **スパン属性** (ページ 951) もゲートします。これは `claude_code.llm_request` スパンのツール結果を含みます
- アシスタント応答テキストはデフォルトでは収集されません。応答長のみが記録されます。応答テキストを含めるには、`OTEL_LOG_ASSISTANT_RESPONSES=1` を設定してください。Claude Code からのすべての OpenTelemetry データと同様に、応答テキストは設定した OTel エンドポイントにのみ送信され、Anthropic には送信されません。この変数が設定されていない場合、`OTEL_LOG_USER_PROMPTS` がフォールバックとして使用されるため、プロンプトコンテンツなしで応答コンテンツが必要な場合は `OTEL_LOG_ASSISTANT_RESPONSES=0` を設定してください
- ツール入力引数とパラメータはデフォルトではログに記録されません。これらを含めるには、`OTEL_LOG_TOOL_DETAILS=1` を設定してください。Claude Desktop の組み込みサーバーの場合、Claude Desktop が所有するセッションでは、`tool_decision` と `tool_result` は `mcp_server_name`/`mcp_tool_name` ペアを含みます。これはホストが作成した名前であり、フラグがオフの場合でも引数コンテンツではありません。この例外には Claude Code v2.1.214 以降が必要です。このデータは設定した OTEL エンドポイントにのみ送信され、Anthropic には送信されません。引数には機密値が含まれる可能性があるため、テレメトリバックエンドを設定してこれらの属性をフィルタリングまたは編集してください。有効にすると：

  - `tool_result` と `tool_decision` イベントには、Bash コマンド、MCP サーバーとツール名、およびスキル名を含む `tool_parameters` 属性が含まれます。`full_command` などのフィールドは切り詰められずに出力されます
  - `tool_result` イベントには、ファイルパス、URL、検索パターン、およびその他の引数を含む `tool_input` 属性も含まれます。512 文字を超える個別の値は切り詰められ、合計は約 4 K 文字に制限されます
  - `user_prompt` イベントには、カスタム、プラグイン、および MCP コマンドの逐語的な `command_name` が含まれます
  - **コストとトークンカウンター** (ページ 960) および `api_request`、`api_error`、および `api_refusal` イベントは、その属性の帰属に実際のエージェント、スキル、プラグイン、および MCP サーバーとツール名を含みます
  - トレーススパンには、同じ `tool_input` 属性と `file_path` などの入力派生属性が含まれ、`tool_input` と同じ切り詰めが行われます

- ツールコンテンツはデフォルトではトレーススパンにログに記録されません。これを含めるには、`OTEL_LOG_TOOL_CONTENT=1` を設定してください。その後、`claude_code.tool` スパンは、生のファイルコンテンツ、Bash コマンド出力、および MCP ツール、WebFetch、WebSearch が返すものを含む *tool.output* **スパンイベント** (ページ 949) を含みます。コンテンツは属性ごとにコンテンツ制限（デフォルトでは 60 KB）で切り詰められます。MCP ツール、WebFetch、WebSearch からの結果には Claude Code v2.1.283 以降が必要です。ツールコンテンツは *new_context* (ページ 951) を通じてスパンに到達します。このゲートはスパンごとに異なります。テレメトリバックエンドを設定してこれらの属性をフィルタリングまたは編集してください

- 生の Anthropic Messages API リクエストおよびレスポンスボディはデフォルトではログに記録されません。これらを含めるには、シェル、ユーザー設定、または管理設定で `OTEL_LOG_RAW_API_BODIES` を設定してください。プロジェクトおよびローカル設定では無視されます。ボディには、システムプロンプト、すべての以前のユーザーとアシスタントのターン、およびツール結果を含む完全な会話履歴が含まれるため、これを有効にすることは、他の `OTEL_LOG_*` コンテンツフラグが明かすすべてのことへの同意を意味します。Claude Code は、他の設定に関係なく、これらのボディから Claude の拡張思考コンテンツを常に編集します。設定する値は、Claude Code がボディを配信する方法を決定します：
  - `=1` の場合、Claude Code は各 API 呼び出しに対して `api_request_body` と `api_response_body` ログイベントを出力します。イベントの `body` 属性は JSON シリアル化されたペイロードを含み、コンテンツ制限（デフォルトでは 60 KB）で切り詰められます
  - `=file:<dir>` の場合、Claude Code は切り詰められていないボディをそのディレクトリの `.request.json` と `.response.json` ファイルに書き込み、イベントはインラインボディの代わりに `body_ref` パスを含みます。テレメトリストリームではなく、ログコレクターまたはサイドカーでディレクトリを送信してください。

    各成功したレスポンスについて、Claude Code はそのディレクトリの `index.jsonl` に 1 行追加し、レスポンスファイルをそれを生成したリクエストファイルおよびそれが成為したトランスクリプトメッセージにリンクします。各行はメッセージコンテンツを含まず、*API* **レスポンスボディイベント** (ページ 970) セクションがそのフィールドをリストします。インデックスファイルには Claude Code v2.1.274 以降が必要です

42.12 Amazon Bedrock での Claude Code の監視

Amazon Bedrock での Claude Code 使用状況監視ガイダンスの詳細については、Claude Code 監視実装（Amazon Bedrock）を参照してください。

第 43 章　Claude があなたのプロジェクトを記憶する方法

CLAUDE.md ファイルで Claude に永続的な指示を与え、自動メモリで Claude が自動的に学習を蓄積できるようにします。

Claude Code の各セッションは、新しいコンテキストウィンドウで始まります。2 つのメカニズムがセッション間で知識を保持します。

- **CLAUDE.md ファイル**: Claude に永続的なコンテキストを与えるために書く指示。Claude はリポジトリの *AGENTS.md* **ファイル** (ページ 1008) も読むことができます。CLAUDE.md と一緒に使用することも、単独で使用することもできます
- **自動メモリ**: あなたの修正と好みに基づいて Claude が自分自身で書くメモ

このページでは、以下の方法について説明します。

- *CLAUDE.md* **ファイルを書いて整理する** (ページ 998)
- **既存の** *AGENTS.md* **をプロジェクト指示として使用する** (ページ 1008)。CLAUDE.md と一緒に使用することも、単独で使用することもできます
- *.claude/rules/* **で特定のファイルタイプにルールをスコープする** (ページ 1003)
- **自動メモリを設定する** (ページ 1013) ので Claude が自動的にメモを取ります
- **指示が従われていない場合のトラブルシューティング** (ページ 1016)

43.1 CLAUDE.md と自動メモリ

Claude Code には 2 つの相互補完的なメモリシステムがあります。どちらも各会話の開始時に読み込まれます。Claude はこれらをコンテキストとして扱い、強制的な設定ではありません。アクションをブロックするには、Claude の判断に関わらず *PreToolUse hook* (ページ 411) を使用してください。より具体的で簡潔な指示ほど、Claude はそれに従う可能性が高くなります。

	CLAUDE.md ファイル	自動メモリ
誰が書くか	あなた	Claude
何が含まれるか	指示とルール	学習とパターン
スコープ	プロジェクト、ユーザー、または組織	リポジトリごと、ワーキングツリー全体で共有
読み込まれる場所	すべてのセッション	すべてのセッション（最初の 200 行または 25KB）

次のページに続く

表 1 – 前のページからの続き

	CLAUDE.md ファイル	自動メモリ
用途	コーディング標準、ワークフロー、プロジェクトアーキテクチャ	あなたの好み、Claude に与える修正、Claude がコードから導き出せないプロジェクトコンテキスト

Claude の動作をガイドしたい場合は CLAUDE.md ファイルを使用します。自動メモリにより、Claude は手動の作業なしにあなたの修正から学習できます。

Subagent も独自の自動メモリを保持できます。詳細については、*subagent* **設定** (ページ 337) を参照してください。

43.2 CLAUDE.md ファイル

CLAUDE.md ファイルは、プロジェクト、個人のワークフロー、または組織全体に対して Claude に永続的な指示を与えるマークダウンファイルです。これらのファイルはプレーンテキストで記述し、Claude はすべてのセッションの開始時にそれらを読みます。リポジトリが代わりに `AGENTS.md` を使用している場合は、*AGENTS.md* (ページ 1008) を参照してください。

43.2.1 CLAUDE.md に追加するタイミング

CLAUDE.md を、そうでなければ何度も説明し直すことになる内容を記述する場所として扱ってください。以下の場合に追加してください。

- Claude が同じ間違いを 2 回目に犯した場合
- コードレビューで Claude がこのコードベースについて知っておくべきだったことが指摘された場合
- 前回のセッションで入力した同じ修正または説明をチャットに再度入力する場合
- 新しいチームメンバーが生産性を高めるために同じコンテキストが必要な場合

すべてのセッションで Claude が保持すべき事実に限定してください。ビルドコマンド、規約、プロジェクトレイアウト、「常に X を実行する」ルールなどです。エントリが複数ステップの手順である場合、またはコードベースの 1 つの部分にのみ関連する場合は、代わりに **スキル** (ページ 275) または **パススコープ付きルール** (ページ 1003) に移動してください。拡張機能の概要 では、各メカニズムをいつ使用するかについて説明しています。

43.2.2 CLAUDE.md ファイルの配置場所を選択する

CLAUDE.md ファイルはいくつかの場所に配置でき、それぞれ異なるスコープを持ちます。以下の表は、ロード順に、最も広いスコープから最も具体的なスコープまでリストしているため、プロジェクト指示はユーザー指示の後のコンテキストに表示されます。

スコープ	場所	目的	ユースケース例	共有対象
管理ポリシー	• macOS: `/Library/Application Support/ClaudeCode/CLAUDE.md` • Linux と WSL: `/etc/claude-code/CLAUDE.md` • Windows: `C:\Program Files\ClaudeCode\CLAUDE.md`	IT/DevOps が管理する組織全体の指示	企業のコーディング標準、セキュリティポリシー、コンプライアンス要件	組織内のすべてのユーザー
ユーザー指示	`~/.claude/CLAUDE.md`	すべてのプロジェクトの個人的な設定	コードスタイルの設定、個人的なツーリングショートカット	自分のみ（すべてのプロジェクト）
プロジェクト指示	`./CLAUDE.md` または `./.claude/CLAUDE.md`。`./AGENTS.md` がそれらの代わりに、または一緒にロードされるタイミングについては *AGENTS.md* (ページ 1008) を参照してください	プロジェクトのチーム共有指示	プロジェクトアーキテクチャ、コーディング標準、一般的なワークフロー	ソース管理を通じたチームメンバー
ローカル指示	`./CLAUDE.local.md`	個人的なプロジェクト固有の設定。`.gitignore` に追加してください	サンドボックス URL、推奨テストデータ	自分のみ（現在のプロジェクト）

作業ディレクトリより上のディレクトリ階層内の CLAUDE.md および CLAUDE.local.md ファイルは起動時にロードされます。サブディレクトリ内のファイルは、Claude がそれらのディレクトリ内のファイルを読むときにオンデマンドでロードされます。完全な解決順序については、*CLAUDE.md* **ファイルのロード方法** (ページ 1002) を参照してください。

大規模なプロジェクトの場合、**プロジェクトルール** (ページ 1003) を使用してトピック固有のファイルに指示を分割できます。ルールを使用すると、特定のファイルタイプまたはサブディレクトリに指示をスコープできます。

43.2.3 プロジェクト CLAUDE.md をセットアップする

プロジェクト CLAUDE.md は `./CLAUDE.md` または `./.claude/CLAUDE.md` に保存できます。このファイルを作成し、プロジェクトで作業する誰もが適用される指示を追加してください。ビルドおよびテストコマンド、コーディング標準、アーキテクチャの決定、命名規約、一般的なワークフローなどです。これらの指示はバージョン管理を通じてチームと共有されるため、個人的な設定ではなくプロジェクトレベルの

標準に焦点を当ててください。ファイルがロードされたことを確認するには、セッションで `/context` を実行し、**Memory files** の下のリストを確認してください。

Tip:

`/init` を実行して、開始用の CLAUDE.md を自動的に生成します。Claude はコードベースを分析し、発見したビルドコマンド、テスト指示、プロジェクト規約を含むファイルを作成します。CLAUDE.md が既に存在する場合、`/init` は上書きするのではなく改善を提案します。そこから Claude が自分で発見しない指示で洗練させてください。

代わりにインタラクティブなマルチフェーズフローの場合は、`/init` を実行する前に `CLAUDE_CODE_NEW_INIT` 環境変数を `1` に設定してください。シェルまたは 環境変数を設定する に示されているように設定ファイルの `env` ブロックで設定します。設定すると、`/init` はセットアップするアーティファクト（CLAUDE.md ファイル、スキル、フック）を尋ねます。その後、サブエージェントでコードベースを探索し、フォローアップの質問を通じてギャップを埋め、ファイルを書き込む前に確認可能な提案を提示します。変数は `/init` の実行方法のみを変更するため、設定したままにしておくことができます。

43.2.4 効果的な指示を書く

Claude は CLAUDE.md ファイルをコンテキストとして扱い、強制的な設定ではないため、指示の書き方は Claude がそれらに従う信頼性に影響します。検証可能なほど具体的な指示を書いてください。

- 「コードを適切にフォーマットする」ではなく「2 スペースのインデントを使用する」
- 「変更をテストする」ではなく「コミット前に `npm test` を実行する」
- 「ファイルを整理しておく」ではなく「API ハンドラーは `src/api/handlers/` に存在する」

ファイルを短く、整理され、一貫性のあるものに保ってください。

- **サイズ**: CLAUDE.md ファイルあたり 200 行以下を目標にしてください。より長いファイルはより多くのコンテキストを消費し、遵守を減らします。コードベースの一部にのみ関連する指示を **パススコープ付きルール** (ページ 1004) に移動してください。これらは、一致するファイルで Claude が作業するときにのみロードされます。**インポート** (ページ 1001) は長いファイルを整理するのに役立ちますが、インポートされたファイルも起動時にロードされるため、コンテキストコストは削減されません。
- **構造**: 関連する指示をマークダウンヘッダーと箇条書きの下にグループ化してください。整理されたセクションは、密集した段落よりも Claude が従いやすくなります。
- **一貫性**: 2 つの指示が矛盾している場合、Claude は 1 つを任意に選択する可能性があります。CLAUDE.md ファイル、サブディレクトリ内のネストされた CLAUDE.md ファイル、および *`.claude/rules/`* (ページ 1003) を定期的に確認して、古い指示または矛盾する指示を削除してください。Claude に見つけてもらうには、**プロンプト監査を実行** (ページ 1001) してください。

指示ファイルを監査する

Claude に指示ファイルで古い内容または矛盾する内容をチェックさせるには、セッションで `/doctor prompt-audit` を実行してください。Claude は、古いモデル用に書かれた指示、存在しないファイルまたはコマンドへの参照、互いに矛盾するファイルなどの問題を探します。提案された編集を含む調査結果のレポートが表示され、Claude に適用するよう依頼するまでファイル内の何も変更されません。

デフォルトでは、監査は CLAUDE.md、CLAUDE.local.md、AGENTS.md ファイル、および `.claude/` と `~/.claude/` の下のルール、スキル、コマンド、サブエージェント、出力スタイルをカバーします。代わりに 1 つのファイルまたはディレクトリを監査するには、そのパスを渡してください。例えば `/doctor prompt-audit .claude/skills/deploy` です。

監査はバンドルされた `/claude-api` スキルを通じて実行されます。そのスキルが *skillOverrides* (ページ 306) で無効になっているか、`disableBundledSkills` で無効になっている場合は利用できません。`/doctor prompt-audit` には Claude Code v2.1.283 以降が必要です。

43.2.5 追加ファイルをインポートする

CLAUDE.md ファイルは `@path/to/import` 構文を使用して追加ファイルをインポートできます。インポートされたファイルは展開され、それらを参照する CLAUDE.md と一緒に起動時にコンテキストにロードされます。

相対パスと絶対パスの両方が許可されます。相対パスは、作業ディレクトリではなく、インポートを含むファイルを基準に解決されます。インポートされたファイルは他のファイルを再帰的にインポートでき、最大深度は 4 ホップです。

パスにスペースが含まれるファイルをインポートするには、各スペースの前にバックスラッシュを付けます。バックスラッシュがない場合、インポートが独立した行にあっても、パスは最初のスペースで終了します。引用符で囲まれたパスは、バックスラッシュの有無にかかわらず、まったくインポートされません。このインポートは `Design Docs` という名前のフォルダからファイルをロードします。

```
- API conventions @Design\ Docs/api-conventions.md
```

インポート解析はマークダウンコードスパンとフェンスコードブロックをスキップします。CLAUDE.md でパスをインポートせずに言及するには、バックティックで囲んでください。`` `@README` `` と書くとテキストはリテラルのままですが、バックティックの外の `@README` はファイルをインポートします。

README、package.json、ワークフローガイドを取得するには、CLAUDE.md の任意の場所で `@` 構文を使用してそれらを参照してください。

```
See @README for project overview and @package.json for available npm commands for␣
↪this project.

# Additional Instructions
- git workflow @docs/git-instructions.md
```

バージョン管理にチェックインすべきではない個人的なプロジェクト固有の設定については、プロジェクトルートに `CLAUDE.local.md` を作成してください。これは `CLAUDE.md` と一緒にロードされ、同じ方法で扱われます。`CLAUDE.local.md` を `.gitignore` に追加して、コミットされないようにしてください。`CLAUDE_`

CODE_NEW_INIT=1 が設定されている場合、/init を実行して個人オプションを選択するとこれが自動的に行われます。

同じリポジトリの複数の git worktree で作業する場合、gitignore された CLAUDE.local.md は作成した worktree にのみ存在します。worktree 間で個人的な指示を共有するには、代わりにホームディレクトリからファイルをインポートしてください。

```
# Individual Preferences
- @~/.claude/my-project-instructions.md
```

警告:

プロジェクトレベルのメモリファイル内のインポートは、上記のホームディレクトリのインポートのように、パスが作業ディレクトリの外に解決される場合、外部インポートとなります。Claude Code が初めてプロジェクトで外部インポートを検出すると、ファイルをリストする承認ダイアログが表示されます。拒否すると、インポートは無効のままになり、ダイアログは再度表示されません。
Claude Code はダイアログを表示して、共有プロジェクトに他の人がコミットしたファイルから保護します。~/.claude/CLAUDE.md や ~/.claude/rules/ などのユーザースコープのメモリファイルは、自分で書いたファイルです。デスクトップでの Cowork セッションを除き、Claude Code はダイアログなしでそれらのインポートをロードし、他の個人設定と同様に信頼します。
デスクトップの Cowork セッションでは、Claude Code はユーザースコープファイル内のセッションの作業ディレクトリの外に解決されるパスを持つインポートをスキップし、ファイルの残りをロードします。これらのセッションでは、それ自体がシンボリックリンクまたはハードリンクである ~/.claude/CLAUDE.md と、作業ディレクトリの外を指すシンボリックリンクされた ~/.claude/rules/ ディレクトリまたはルールファイルもスキップします。

43.2.6 CLAUDE.md ファイルのロード方法

Claude Code は、現在の作業ディレクトリとそれより上のすべてのディレクトリから CLAUDE.md と CLAUDE.local.md をロードします。foo/bar/ で Claude Code を実行すると、foo/bar/CLAUDE.md、foo/CLAUDE.md、およびそれらと一緒にある CLAUDE.local.md ファイルから指示をロードします。

発見されたすべてのファイルは、互いに上書きするのではなく、コンテキストに連結されます。ディレクトリツリー全体で、コンテンツはファイルシステムルートから作業ディレクトリまで順序付けられます。foo/bar/ の例では、foo/CLAUDE.md は foo/bar/CLAUDE.md の前にコンテキストに表示されるため、Claude を起動した場所に近い指示が最後に読まれます。各ディレクトリ内で、CLAUDE.local.md は CLAUDE.md の後に追加されるため、個人的なメモはそのレベルで Claude が読む最後のものです。

Claude はまた、現在の作業ディレクトリの下のサブディレクトリで CLAUDE.md と CLAUDE.local.md ファイルを発見します。起動時にロードするのではなく、Claude がそれらのサブディレクトリ内のファイルを読むときに含まれます。.claude/worktrees/ の下にある worktree 内のファイルについては、worktree でサブエージェントを分離する を参照してください。

大規模なモノレポで他のチームの CLAUDE.md ファイルが取得される場合は、*claudeMdExcludes* (ページ 1007) を使用してそれらをスキップしてください。ルートおよびディレクトリごとの CLAUDE.md ファイルとルールの完全なレイアウトについては、モノレポと大規模リポジトリ を参照してください。

CLAUDE.md ファイル内のブロックレベル HTML コメント（<!-- maintainer notes -->）は、コンテンツが Claude のコンテキストに注入される前に削除されます。コンテキストトークンを消費せずに、人間のメンテナーのためのメモを残すために使用してください。コードブロック内のコメントは保持されます。Read ツールで CLAUDE.md ファイルを直接開くと、コメントは表示されたままになります。

追加ディレクトリからロードする

--add-dir フラグは、メインの作業ディレクトリの外の追加ディレクトリに Claude がアクセスできるようにします。デフォルトでは、これらのディレクトリからの CLAUDE.md ファイルはロードされません。

追加ディレクトリからメモリファイルもロードするには、CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD 環境変数を設定してください。

```
CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1 claude --add-dir ../shared-config
```

インライン形式は、Bash または Zsh でその 1 回の起動に対して変数を設定します。すべてのセッションで有効にするには、環境変数を設定する に示されているように ~/.claude/settings.json の env ブロックに追加してください。

これは追加ディレクトリから CLAUDE.md、.claude/CLAUDE.md、.claude/rules/*.md、および CLAUDE.local.md をロードします。CLAUDE.local.md は *--setting-sources* (ページ 45) から local を除外する場合はスキップされます。

43.2.7 .claude/rules/ でルールを整理する

大規模なプロジェクトの場合、.claude/rules/ ディレクトリを使用して指示を複数のファイルに整理できます。これにより、指示がモジュール化され、チームが保守しやすくなります。ルールは **特定のファイルパスにスコープ** (ページ 1004) することもできるため、Claude が一致するファイルで作業するときにのみコンテキストにロードされ、ノイズを減らしてコンテキストスペースを節約します。

注釈:

ルールはすべてのセッションまたは一致するファイルが開かれるときにコンテキストにロードされます。タスク固有の指示で常にコンテキストに含まれる必要がない場合は、代わりに **スキル** (ページ 275) を使用してください。これは、呼び出すときまたは Claude がプロンプトに関連していると判断したときにのみロードされます。

ルールをセットアップする

プロジェクトの .claude/rules/ ディレクトリにマークダウンファイルを配置してください。各ファイルは 1 つのトピックをカバーし、testing.md や api-design.md のような説明的なファイル名を持つ必要があります。すべての .md ファイルは再帰的に発見されるため、frontend/ や backend/ のようなサブディレクトリにルールを整理できます。

```
your-project/
├─── .claude/
│    ├─── CLAUDE.md           # Main project instructions
│    └─── rules/
```

(次のページに続く)

(前のページからの続き)

```
│       ├── code-style.md   # Code style guidelines
│       ├── testing.md      # Testing conventions
│       └── security.md     # Security requirements
```

paths **フロントマター** (ページ 1004) のないルールは、`.claude/CLAUDE.md` と同じ優先順位で起動時にロードされます。

--setting-sources (ページ 45) から `project` を除外する場合、プロジェクトルールはスキップされます。v2.1.211 より前では、パススコープ付きルールやネストされた `.claude/rules/` ディレクトリ内のルールを含む、オンデマンドでロードするルールは、`project` が除外されている場合でもロードされました。

パススコープ付きルール

ルールは YAML フロントマターと `paths` フィールドを使用して特定のファイルにスコープできます。これらの条件付きルールは、指定されたパターンに一致するファイルで Claude が作業するときにのみ適用されます。

```
---
paths:
  - "src/api/**/*.ts"
---

# API Development Rules

- All API endpoints must include input validation
- Use the standard error response format
- Include OpenAPI documentation comments
```

`paths` フィールドのないルールは無条件にロードされ、すべてのファイルに適用されます。パススコープ付きルールは、すべてのツール使用時ではなく、Claude がパターンに一致するファイルに対して Read、Write、または Edit ツールを使用するときにトリガーされます。マッチングは、Claude がプロジェクトディレクトリへのシンボリックリンクされたパスを通じてファイルに到達する場合（例えば、シンボリックリンクされたチェックアウト）にも機能します。

`paths` フィールドでグロブパターンを使用して、拡張子、ディレクトリ、またはそれらの組み合わせでファイルをマッチさせてください。

パターン	マッチ
`**/*.ts`	任意のディレクトリ内のすべての TypeScript ファイル
`src/**/*`	`src/` ディレクトリの下のすべてのファイル
`*.md`	プロジェクトルート内のマークダウンファイル
`src/components/*.tsx`	特定のディレクトリ内の React コンポーネント

複数のパターンを指定し、ブレース展開を使用して 1 つのパターンで複数の拡張子をマッチさせることができます。

```
---
paths:
  - "src/**/*.{ts,tsx}"
  - "lib/**/*.ts"
  - "tests/**/*.test.ts"
---
```

各ブレースグループは展開されたパターンの数を乗算します。`src/*.{ts,tsx}` は 2 つのパターンに展開され、`{a,b}/{c,d}/*.{ts,tsx}` は 8 つに展開されます。展開を制限するために、ルール全体の `paths` リストは 1,000 展開パターンと 4 MiB の 1 つの予算を共有し、ブレースのないパターンはそれに対してカウントされません。

Claude Code は予算を超えるパターンを展開されていない状態で使用し、その文字通りのブレースはファイルをマッチさせません。v2.1.217 より前では、多くのブレースグループを持つ `paths` 値は起動時に CLI をスタールまたはクラッシュさせました。

グロブ構文は `[` をブラケット式（`[abc]` など）の開始として扱います。`photos [2024/**` のようにブラケット式として読むことができない `[` を持つパターンは無効です。何もマッチせず、ルールの他のパターンは機能し続けます。ファイル名内のリテラル `[` をマッチさせるには、`photos \[2024/**` のようにエスケープしてください。v2.1.207 より前では、1 つの無効なパターンは、何もマッチしないのではなく、ルールが評価されたすべてのファイルに対して Read ツールを失敗させました。

ルールのフロントマターリファレンス

ファイル先頭の `---` マーカーの間に YAML **フロントマター** を記述してルールを設定します。`paths` は Claude Code がルールから読む唯一のフィールドです。他のフィールドはエラーなしで無視されます。Claude Code はルールをコンテキストにロードする前にフロントマターを削除します。

フィールド	必須	説明
`paths`	いいえ	**ルールを一致するファイルにスコープする** (ページ 1004) グロブパターン。YAML リストまたはカンマ区切り文字列を受け入れます

マーカー間の YAML が解析できない場合、Claude Code はフロントマターを無視し、`paths` がないかのようにルールをロードします。`claude --debug` を実行して解析エラーを確認してください。

シンボリックリンクを使用してプロジェクト間でルールを共有する

`.claude/rules/` ディレクトリはシンボリックリンクをサポートしているため、共有ルールセットを保持し、複数のプロジェクトにリンクできます。循環シンボリックリンクは検出され、適切に処理されます。

Claude Code は、ターゲットが作業ディレクトリの外にあるシンボリックリンクを **外部インポート** (ページ 1001) のように扱います。リンクされたルールは、プロジェクトの外部インポートを承認するまでロードされず、その後は *paths* **フィールド** (ページ 1004) のないものだけがロードされます。Claude Code がこの承認を求めるのは、プロジェクトのメモリファイルが `@path` で作業ディレクトリ外のファイルをインポートする場合のみで、シンボリックリンクだけでは求めません。承認なしで共有ルールをロードするに

は、*~/.claude/rules/* (ページ 1006) に保持してください。そこではマシン上のすべてのプロジェクトに適用されます。

この例は、共有ディレクトリと個別ファイルの両方をリンクします。

```
ln -s ~/shared-claude-rules .claude/rules/shared
ln -s ~/company-standards/security.md .claude/rules/security.md
```

.claude/rules/ または CLAUDE.md のシンボリックリンクを、UNC 共有 \\server\share や /net または /Network の下のパスなどのネットワークパスに向けた場合、リンクされた指示はロードされません。そのようなパスを検索すると、その名前が示すホストに接続する可能性があるため、Claude Code はリンクをたどりません。\\wsl$ パスはネットワークパスとしてカウントされません。

ユーザーレベルルール

~/.claude/rules/ の個人的なルールはマシン上のすべてのプロジェクトに適用されます。プロジェクト固有ではない設定に使用してください。

```
~/.claude/rules/
├─── preferences.md    # Your personal coding preferences
└─── workflows.md      # Your preferred workflows
```

Claude Code はプロジェクトルールの前にユーザーレベルルールをロードするため、プロジェクトルールはユーザールールより後に Claude のコンテキストに表示されます。どちらのセットも他方を上書きしません。ユーザールールとプロジェクトルールが矛盾する場合、Claude はどちらかに従う可能性があるため、2 つを一貫性のあるものに保ってください。

43.2.8 大規模なチーム向けに CLAUDE.md を管理する

Claude Code を複数のチームにデプロイする組織の場合、指示を一元化し、どの CLAUDE.md ファイルがロードされるかを制御できます。

組織全体の CLAUDE.md をデプロイする

組織は、マシン上のすべてのユーザーに適用される一元管理の CLAUDE.md をデプロイできます。このファイルは個別の設定で除外することはできません。

1. **管理ポリシーの場所にファイルを作成する**
   - macOS: /Library/Application Support/ClaudeCode/CLAUDE.md
   - Linux と WSL: /etc/claude-code/CLAUDE.md
   - Windows: C:\Program Files\ClaudeCode\CLAUDE.md
2. **設定管理システムでデプロイする**

   MDM、グループポリシー、Ansible、または同様のツールを使用して、開発者マシン全体にファイルを配布してください。その他の組織全体の設定オプションについては、管理設定 を参照してください。

claudeMd キーを使用すると、別のファイルをデプロイする代わりに、管理された CLAUDE.md コンテンツを managed-settings.json に直接配置できます。

スコープ: マシン上のすべての Claude Code セッション、すべてのリポジトリで。リポジトリ固有のガイダンスについては、プロジェクト CLAUDE.md をコミットしてください。

優先順位: 管理された CLAUDE.md ファイルと同じ。ユーザーおよびプロジェクト CLAUDE.md の前にロードされます。

どこで尊重されるか: 管理設定およびポリシー設定のみ。ユーザー、プロジェクト、またはローカル設定で claudeMd を設定しても効果がありません。

以下の例は、管理設定ファイルに直接動作指示を追加します。

```
{
  "claudeMd": "Always run `make lint` before committing.\nNever push directly to main.
↪"
}
```

管理された CLAUDE.md と 管理設定 は異なる目的を果たします。技術的な強制には設定を、動作ガイダンスには CLAUDE.md を使用してください。

懸念事項	設定対象
特定のツール、コマンド、またはファイルパスをブロックする	管理設定: permissions.deny
サンドボックス分離を強制する	管理設定: sandbox.enabled
環境変数と API プロバイダーのルーティング	管理設定: env
ログイン方法と組織の制限	管理設定: forceLoginMethod、forceLoginOrgUUID
コードスタイルと品質ガイドライン	管理 CLAUDE.md
データ処理とコンプライアンスのリマインダー	管理 CLAUDE.md
Claude の動作指示	管理 CLAUDE.md

設定ルールは、Claude が何をするかに関わらず、クライアントによって強制されます。CLAUDE.md 指示は Claude の動作を形作りますが、ハード強制レイヤーではありません。

特定の CLAUDE.md ファイルを除外する

大規模なモノレポでは、祖先の CLAUDE.md ファイルに、作業に関連しない指示が含まれている可能性があります。claudeMdExcludes 設定を使用すると、パスまたはグロブパターンで特定のファイルをスキップできます。

この例は、トップレベルの CLAUDE.md と親フォルダのルールディレクトリを除外します。.claude/settings.local.json に追加して、除外をマシンにローカルに保ってください。

```
{
  "claudeMdExcludes": [
    "**/monorepo/CLAUDE.md",
```

(次のページに続く)

(前のページからの続き)

```
    "/home/user/monorepo/other-team/.claude/rules/**"
  ]
}
```

パターンは、グロブ構文を使用して絶対ファイルパスに対してマッチされます。`claudeMdExcludes` は任意の **設定レイヤー** (ページ 711)（ユーザー、プロジェクト、ローカル、または管理ポリシー）で設定できます。配列はレイヤー全体でマージされます。

シンボリックリンク (ページ 1005) を通じて到達するルールファイルを除外するには、ファイルとそのディレクトリのどちらがリンクであっても、`.claude/rules/` の下のファイルのパスまたはそのリンクターゲットのいずれかのパスに対してパターンを記述してください。どちらかのパスに一致するパターンはファイルを除外します。v2.1.239 より前では、リンクターゲットに一致するパターンのみがファイルを除外しました。

管理ポリシー CLAUDE.md ファイルは除外できません。これにより、個別の設定に関わらず、組織全体の指示が常に適用されることが保証されます。

43.3 AGENTS.md

Claude Code は *AGENTS.md* をプロジェクト指示として読み込むことができるため、他のコーディングエージェント向けにセットアップされたリポジトリは、`CLAUDE.md`、インポート、またはセッティングを追加することなく動作します。このテーブルは、リポジトリ内の指示ファイルの組み合わせごとに Claude が既定で読み込むものを示しています。

リポジトリに含まれるもの	Claude が読み込むもの
`AGENTS.md` があり、ワーキングディレクトリまたはその上位ディレクトリに `CLAUDE.md` または `CLAUDE.local.md` がない	`AGENTS.md`
`AGENTS.md` と、ワーキングディレクトリまたはその上位ディレクトリに `CLAUDE.md` または `CLAUDE.local.md` がある	`CLAUDE.md` ファイルのみ
`AGENTS.md` を既に **インポートしている** (ページ 1011) `CLAUDE.md` がある	`CLAUDE.md`（インポートを通じて `AGENTS.md` を含む）

既定の動作を変更する場合（例えば、Claude に常に両方のファイルを読み込ませたい、`CLAUDE.md` のみを読み込ませたい、または組織が管理する指示のみを読み込ませたい場合）は、*Project instructions* **セッティングを変更** (ページ 1009) してください。

注釈:

`AGENTS.md` を直接読み込むには Claude Code v2.1.277 以降が必要です。一部のセッションでは Claude が *AGENTS.md* **を読み込むことができない** (ページ 1010) ため、代わりに *CLAUDE.md* **からインポート** (ページ 1011) してください。

43.3.1 Claude Code が AGENTS.md を読み込むとき

既定では、Claude はワーキングディレクトリまたはその上位ディレクトリに CLAUDE.md がない場合にのみ AGENTS.md を読み込みます。このチェックに該当するファイルは以下の通りです。

- **該当するため、Claude は AGENTS.md の代わりにこれらを読み込む**: ワーキングディレクトリまたはその上位ディレクトリにある CLAUDE.md、.claude/CLAUDE.md、または CLAUDE.local.md
- **該当しないため、AGENTS.md と一緒に読み込み続ける**: ~/.claude/CLAUDE.md、組織が管理する CLAUDE.md、および .claude/rules/ ファイル

該当するものがない場合、Claude が読み込むものと確認方法は以下の通りです。

- **セッション開始時**: ワーキングディレクトリとその上位ディレクトリ内のすべての AGENTS.md と .claude/AGENTS.md。インタラクティブセッションでは、会話に no CLAUDE.md found; AGENTS.md loaded: /home/you/repo/AGENTS.md のような行が表示されます
- **Claude がサブディレクトリで作業するとき**: Claude が Read ツールでそこにファイルを開き、そのサブディレクトリに 3 つの CLAUDE.md ファイルのいずれも含まれていない場合、サブディレクトリの AGENTS.md
- **各 AGENTS.md 内**: *@path* **インポート** (ページ 1001) が展開され、*claudeMdExcludes* (ページ 1007) パターンが適用され、**プロジェクト指示をスキップ** (ページ 350) するサブエージェントがこれらのファイルもスキップします
- **読み込まれない**: AGENTS.local.md、AGENTS.override.md、または .agents/ ディレクトリ下のすべてのもの

注釈:

CLAUDE.local.md が該当するため、AGENTS.md に依存するプロジェクトに CLAUDE.local.md を追加して独自のコミットされていない指示を保持すると、Claude は AGENTS.md を読み込まなくなります。CLAUDE.local.md を保持しながら Claude に AGENTS.md を読み込ませるには、**Project instructions** を *claude-md-and-agents-md* (ページ 1009) に設定してください。

43.3.2 Claude が読み込む指示ファイルを選択する

Claude が読み込むファイルを変更するには、Claude Code セッションで /config と入力してセッティングパネルを開き、**Project instructions** を以下のいずれかの値に設定します。

値	Claude が読み込むもの
claude-md-or-agents-md	CLAUDE.md ファイル、またはワーキングディレクトリまたはその上位ディレクトリに CLAUDE.md または CLAUDE.local.md がない場合は AGENTS.md ファイル。これが既定値です
claude-md-and-agents-md	CLAUDE.md と AGENTS.md ファイルを一緒に、各ディレクトリの CLAUDE.md ファイルを最初に、その後に AGENTS.md を読み込みます。Claude Code は既に読み込んだ AGENTS.md をスキップするため、CLAUDE.md がインポートまたはシンボリックリンクしているものは 2 回読み込まれません

次のページに続く

表 7 – 前のページからの続き

値	Claude が読み込むもの
claude-md	CLAUDE.md ファイルのみ
managed-only	起動時に組織が管理する CLAUDE.md と **自動メモリ** (ページ 1013) のみ。プロジェクト、ローカル、ユーザーの CLAUDE.md ファイル、claude/rules/ ファイル、およびすべての AGENTS.md は除外されます。Claude がそこでファイルを読み込むとき、サブディレクトリの CLAUDE.md と .claude/rules/ ファイル、および **パススコープ規則** (ページ 1004) は引き続き読み込まれます

/config の代わりにセッティングファイルで値を設定することもできます。~/.claude/settings.json、--settings ファイル、または 管理セッティング の pluginConfigs 下の組み込み agents-md プラグインの ID に追加してください。Claude Code はプロジェクトおよびローカルセッティングファイルではこれを無視します。この例は Claude に両方のファイルを読み込ませます。

リスト 1: settings.json

```
{
  "pluginConfigs": {
    "agents-md@builtin": {
      "options": { "instructionFiles": "claude-md-and-agents-md" }
    }
  }
}
```

変更は次のメッセージから、およびすべての新しいセッションで適用されます。

43.3.3 AGENTS.md サポートが利用できないとき

これらのセッションでは Claude は CLAUDE.md ファイルのみを読み込み、**Project instructions** は /config セッティングパネルに表示されません。

- Claude Code v2.1.277 より前のバージョンを使用している
- 組み込み agents-md プラグインを /plugin で無効にしました
- 一部の場合、v2.1.276 以前から アップグレード した後の最初のセッションです。次のセッションから Claude は AGENTS.md を読み込みます

v2.1.281 より前では、Amazon Bedrock 上のセッションやテレメトリが無効になっているセッションなど、一部のセッションは CLAUDE.md ファイルのみを読み込みます。これらのバージョンでは Claude Code を更新してください。これらのセッションのいずれかで Claude に AGENTS.md を提供するには、*CLAUDE.md* **からインポート** (ページ 1011) してください。

43.3.4 AGENTS.md が CLAUDE.md と異なる点

Claude が **Project instructions** セッティングを通じて読み込む AGENTS.md は、これらの場所で CLAUDE.md と異なります。

	CLAUDE.md	Project instructions セッティングを通じて読み込まれた AGENTS.md
InstructionsLoaded **フック** (ページ 486)	発火する	発火しない。CLAUDE.md がインポートまたはシンボリックリンクしている AGENTS.md に対しては通常通り発火します
CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD (ページ 1003) が設定されている間に --add-dir で追加するディレクトリ	それらの CLAUDE.md が読み込まれる	それらの AGENTS.md は読み込まれません
ワーキングディレクトリの外のファイルの @path インポート	Claude Code はあなたに **外部インポート** (ページ 1001) の承認を求めます	このプロジェクトに対して既に外部インポートを承認している場合のみ読み込まれ、プロンプトはありません

43.3.5 以前の AGENTS.md 回避策を削除する

Claude Code が独自に AGENTS.md を読み込む前にそれを読み込むようにセットアップした場合、各一般的なセットアップで以下を実行してください。

- **@AGENTS.md を含む** CLAUDE.md: そのままにしておくことができます。インポートを保持することで、使用する **Project instructions** 値に関係なく Claude が AGENTS.md を 2 回読み込むことはありません。CLAUDE.md が他に何も含まない場合は削除するか、一部のセッションが *AGENTS.md* **を直接読み込むことができない** (ページ 1010) 場合は保持してください。
- **Claude に言葉で** AGENTS.md **を読み込むよう指示する** CLAUDE.md: Claude がファイルを開くことを決定した場合のみ AGENTS.md が表示されます。CLAUDE.md を削除して Claude が AGENTS.md を直接読み込むようにするか、文を @AGENTS.md インポートに置き換えてください。
- AGENTS.md **にシンボリックリンクされた** CLAUDE.md: 何もしないか、シンボリックリンクを削除してください。どちらの場合でも Claude はコンテンツを 1 回読み込みます。
- AGENTS.md **を出力する** SessionStart **フック**: 削除してください。Claude が AGENTS.md を直接読み込むと、フックはコンテキストに 2 番目のコピーを追加します。

43.3.6 他のコーディングツールと 1 つのファイルを共有する

Claude が AGENTS.md を直接読み込んでいない場合でも、AGENTS.md の隣に CLAUDE.md に @AGENTS.md インポートを配置することで、すべてのツールが共有する 1 つのファイルとして保持できます。プロジェクトに CLAUDE.md もある場合、**Project instructions** を claude-md に設定した場合、または *AGENTS.md* **を読み込むことができない** (ページ 1010) セッションで実行してください。インポートの下に Claude 固有の指示を追加すると、Claude はインポートされたファイルを最初に読み込み、その後残りを読み込みます。

リスト 2: CLAUDE.md

```
@AGENTS.md

## Claude Code

`src/billing/` 下の変更にはプランモードを使用してください。
```

Claude 固有のコンテンツが不要な場合、シンボリックリンクも機能します。

```
ln -s AGENTS.md CLAUDE.md
```

コマンドは成功時に出力を出力しません。シンボリックリンクをインポートより選択する前に、これらの制約を確認してください。

- **編集**: Claude はリンクを通じて `CLAUDE.md` を読み込みますが、Edit および Write ツールは シンボリックリンクを通じた書き込みを拒否 し、拒否はリンクのターゲットである `AGENTS.md` を代わりに編集するよう Claude に指示します
- **Windows**: リポジトリをクローンする人が Windows で作業する場合は、`@AGENTS.md` インポートを代わりに使用してください。そこでシンボリックリンクを作成するには管理者権限または開発者モードが必要であり、`core.symlinks` が有効になっていない限り Git はコミットされたシンボリックリンクをプレーンテキストファイルとしてチェックアウトします。これにより、そのクローンに指示の代わりに 1 行の `CLAUDE.md` が残ります

どちらのアプローチでも、次のセッションで `/context` を実行し、**Memory files** の下に `CLAUDE.md` が表示されることを確認してください。

43.3.7 他のツールから指示を移行する

`/init` を実行すると、他のツールの指示ファイルが読み込まれ、関連する部分が生成された `CLAUDE.md` に組み込まれます。

- `.cursor/rules/` または `.cursorrules` の Cursor ルール
- `.github/copilot-instructions.md` の Copilot ルール
- `CLAUDE_CODE_NEW_INIT=1` が設定されている場合: `AGENTS.md`、`.devin/rules/`、`.windsurf/rules/` または `.windsurfrules`、および `.clinerules`

`/import` を実行して、サポートされているコーディングエージェントの設定を Claude Code に取り込むこともできます。これにより、`AGENTS.md` などの指示ファイルの 1 回限りのコピーが一致する `CLAUDE.md` に追加され、MCP サーバー、コマンド、サブエージェント、およびスキルが引き継がれます。Claude Code v2.1.213 以降が必要です。

43.4 自動メモリ

自動メモリを使用すると、Claude は何も書かずにセッション間で知識を蓄積できます。Claude は作業中に自分自身のためにメモを 4 種類保存します。Claude はメモリファイルのフロントマターに `type` フィールドとして種類を記録します。

- `user`：あなたの役割、専門知識、および作業の好み
- `feedback`：Claude に与える修正と確認するアプローチ
- `project`：進行中の作業、期限、および Claude がコードまたは git 履歴から導き出せない決定
- `reference`：issue トラッカーやダッシュボードなど、プロジェクト外の情報を見つける場所

Claude はアーキテクチャ、ファイルパス、デバッグ修正など、コードベースから導き出せるものはすべてスキップします。また、CLAUDE.md ファイルが既に述べていることもスキップします。

Claude はすべてのセッションで何かを保存するわけではありません。情報が将来の会話で役立つかどうかに基づいて、何を記憶する価値があるかを決定します。

43.4.1 自動メモリを有効または無効にする

自動メモリはローカルセッションではデフォルトで有効です。Claude Tag セッション以外では、セルフホスト環境のセッションはデフォルトで自動メモリがオフの状態で実行されます。

切り替えるには、セッションで `/memory` を開き、自動メモリトグルを使用します。これにより `autoMemoryEnabled` が `~/.claude/settings.json` のユーザー設定に保存されます。

次のセッションでは、トグルで自動メモリをオフにすることはできますが、再びオンにすることはできません。

- バックグラウンドセッション
- 別の Claude Code セッションが開始したセッション（Claude が Bash ツールを通じて `claude` を実行した場合など）

これらのセッションで自動メモリがオフの間、トグルには `off · can't be turned on here; use a session started outside Claude Code` と表示されます。自動メモリを再びオンにするには、ターミナルで直接 `claude` を実行し、そのセッションで `/memory` トグルを使用します。

単一のプロジェクトに対してオフにするには、そのプロジェクトの設定で `autoMemoryEnabled` を設定します。

```
{
  "autoMemoryEnabled": false
}
```

環境変数を使用して自動メモリを無効にするには、`CLAUDE_CODE_DISABLE_AUTO_MEMORY=1` を設定します。

43.4.2 ストレージの場所

各プロジェクトは ~/.claude/projects/<project>/memory/ に独自のメモリディレクトリを取得します。<project> パスは git リポジトリから派生しているため、同じリポジトリ内のすべてのワーキングツリーとサブディレクトリは 1 つの自動メモリディレクトリを共有します。git リポジトリの外では、プロジェクトルートが代わりに使用されます。

CLAUDE_CODE_PROJECT_DIR_NAME を CLAUDE_CONFIG_DIR の横に設定した場合、Claude Code はその名前を、どのリポジトリを起動しても <config dir>/projects/ の下の <project> ディレクトリとして使用するため、その設定ディレクトリで起動されたプロジェクトは 1 つの自動メモリディレクトリを共有します。Claude Code v2.1.234 以降が必要です。詳細は CLAUDE_CODE_PROJECT_DIR_NAME を参照してください。

自動メモリを別の場所に保存するには、settings.json で autoMemoryDirectory を設定します。これは任意の**設定スコープ** (ページ 719) から読み込まれます。ユーザー、プロジェクト、ローカル、ポリシー、または --settings です。

```
{
  "autoMemoryDirectory": "~/my-custom-memory-dir"
}
```

値は絶対パスであるか、~/ で始まる必要があります。

プロジェクトの .claude/settings.json または .claude/settings.local.json で設定する場合、Claude Code はそれを**設定ファイル内の *hooks* と同じワークスペーストラストルール** (ページ 1066) の下で尊重します。permissions.blockReadsOutsideWorkingDirectories がオンの間、Claude Code は**リポジトリ提供の設定ファイル** (ページ 1065) が選択するディレクトリからは自動メモリを読み込まず、そこに保存もしません。そのディレクトリがどこにあるかに関わらず。

ディレクトリには MEMORY.md インデックスとメモリごとに 1 つのトピックファイルが含まれます。

```
~/.claude/projects/<project>/memory/
├── MEMORY.md          # インデックス、メモリごとに 1 行、すべてのセッションに読み込まれます
├── user_role.md       # 1 つのメモリ
├── feedback_testing.md # 1 つのメモリ
└── ...                # Claude が作成するその他のトピックファイル
```

MEMORY.md はメモリディレクトリのインデックスとして機能します。Claude はセッション全体を通じてこのディレクトリ内のファイルを読み書きし、MEMORY.md を使用して保存されている内容を追跡します。

自動メモリはマシンローカルです。同じ git リポジトリ内のすべてのワーキングツリーとサブディレクトリは 1 つの自動メモリディレクトリを共有します。ファイルはマシン間またはクラウド環境全体で共有されません。

Claude Code は cleanupPeriodDays 保持期間後に古いセッショントランスクリプトを削除しますが、メモリディレクトリ内のメモリファイルをその保持スイープから除外します。MEMORY.md とトピックファイルは、あなたまたは Claude が編集または削除するまで保持されます。

43.4.3 仕組み

MEMORY.md の最初の 200 行、または最初の 25KB のいずれか先に来る方が、すべての会話の開始時に読み込まれます。そのしきい値を超えるコンテンツはセッション開始時に読み込まれません。Claude は詳細なメモを別のトピックファイルに移動することで、MEMORY.md を簡潔に保ちます。

Claude が MEMORY.md に書き込んだ後、Claude Code はファイルを 200 行および 25KB の読み込み制限に対して測定します。ファイルが制限に近い場合、Claude Code は Claude に短縮するよう促します。エントリごとに 1 行を保持し、詳細をトピックファイルに移動し、古いエントリをマージまたは削除します。ファイルが制限を超えている場合、書き込みは成功しますが、Claude Code は Claude にインデックスを書き直すよう指示するエラーを返します。次の読み込み時に制限を超えるすべてが削除されるためです。

この制限は MEMORY.md にのみ適用されます。Claude Code は最大 4 MiB の CLAUDE.md ファイルを完全に読み込み、より大きいファイルはスキップします。より短いファイルはより良い遵守を生成します。

user_role.md または feedback_testing.md などのトピックファイルは起動時に読み込まれません。Claude は標準ファイルツールを使用してセッション中にオンデマンドで読み込み、情報が必要な場合に読みます。

メイン会話の自動メモリは**サブエージェント** (ページ 350) に読み込まれません。例外は**フォーク** (ページ 354) で、これは親会話とシステムプロンプトを継承します。サブエージェント memory フィールドで有効にされたサブエージェント自身の自動メモリは、別のディレクトリです。

Claude はセッション中にメモリファイルを読み書きします。Claude Code インターフェイスで「Saved 2 memories」または「Recalled 2 memories」などのメッセージが表示されたら、Claude は ~/.claude/projects/<project>/memory/ から積極的に更新または読み込みを行っています。

Claude が YAML フロントマターで始まるメモリファイルを書き込む場合、Claude Code は書き込み時刻を ISO 8601 タイムスタンプとして modified フロントマターフィールドに記録します。タイムスタンプは、事実がどの程度最新であるかを、あなたと Claude がメモリを読み戻すときの両方に示します。フロントマターを持つすべてのファイルは、以前のバージョンで作成されたファイルを含め、Claude が次に書き込むときにフィールドを取得します。Claude Code はフロントマターを持たないファイルにフロントマターを追加することはありません。modified フィールドには Claude Code v2.1.214 以降が必要です。

43.4.4 メモリを監査および編集する

自動メモリファイルはプレーンマークダウンで、いつでも編集または削除できます。*/memory* (ページ 1015) を実行して、セッション内からメモリファイルを参照して開きます。

43.5 /memory で表示および編集する

/memory コマンドは、ユーザーおよびプロジェクトスコープ全体にわたる CLAUDE.md、CLAUDE.local.md、およびその他のメモリファイルの場所をリストします。これには、まだ存在しないファイルのユーザーおよびプロジェクト CLAUDE.md エントリも含まれます。また、自動メモリのオン/オフを切り替えたり、自動メモリフォルダを開くオプションを提供したりできます。任意のファイルを選択してエディタで開きます。まだ存在しないファイルを選択すると、最初にそれが作成されます。現在のセッションに実際に読み込まれたファイルを確認するには、/context を実行します。

VS Code などの GUI エディタはファイルを別のウィンドウで開き、ファイルが開いている間もセッションを使用し続けることができます。v2.1.216 より前は、`/memory` はファイルを閉じるまで応答を待っていました。Vim などのターミナルエディタはターミナルを占有し、終了するまで制御します。

Claude に何かを記憶するよう求めるとき、「常に npm ではなく pnpm を使用する」または「API テストがローカル Redis インスタンスを必要とすることを覚えておく」のように、Claude はそれを自動メモリに保存します。代わりに CLAUDE.md に指示を追加するには、Claude に直接「これを CLAUDE.md に追加する」と尋ねるか、`/memory` を通じてファイルを自分で編集します。

43.6 メモリの問題をトラブルシューティングする

CLAUDE.md と自動メモリに関する最も一般的な問題と、それらをデバッグするための手順を以下に示します。

43.6.1 Claude が CLAUDE.md に従っていない

CLAUDE.md のコンテンツは、システムプロンプト自体の一部ではなく、システムプロンプトの後のユーザーメッセージとして配信されます。Claude はそれを読んで従おうとしますが、特に曖昧または矛盾した指示の場合、厳密な準拠を保証することはできません。

デバッグするには：

- `/context` を実行し、**Memory files** の下のリストを確認して、CLAUDE.md と CLAUDE.local.md ファイルが読み込まれたことを確認します。`CLAUDE.md` ファイルがそこにない場合、Claude はそれを見ることができません。`/memory` を使用してファイルを開いて編集します。
- 関連する CLAUDE.md がセッションに読み込まれる場所にあることを確認します（*CLAUDE.md* **ファイルをどこに配置するかを選択する** (ページ 998) を参照）。
- 指示をより具体的にします。「Use 2-space indentation」は「format code nicely」よりも効果的です。
- CLAUDE.md ファイル全体で矛盾する指示を探します。2 つのファイルが同じ動作に対して異なるガイダンスを提供する場合、Claude は任意に 1 つを選択する可能性があります。
- Claude Code が独自に追加するガイダンスと指示が競合しているかどうかを確認します。CLAUDE.md がコミットまたはプルリクエストルールを設定する場合、`includeGitInstructions` で組み込みのものをオフにし、`attribution` で属性テキストを設定します。

指示が各コミットの前やファイル編集後など、特定の時点で実行する必要があるものである場合は、代わりに *hook* (ページ 411) として記述します。Hook はシェルコマンドとして固定されたライフサイクルイベントで実行され、Claude が何をすることに決めたかに関係なく適用されます。

システムプロンプトレベルで指示が必要な場合は、*`--append-system-prompt`* (ページ 72) を使用します。起動時に渡すため、対話的な使用よりもスクリプトと自動化に適しています。会話を再開するときの動作については、**再開された会話でのシステムプロンプトフラグ** (ページ 73) を参照してください。

Tip:

InstructionsLoaded hook (ページ 486) を使用して、どの `CLAUDE.md` とルールファイルが読み込まれたか、いつ読み込まれたか、なぜ読み込まれたかをログに記録します。これは、パス固有のルールまたはサブディレクトリ内の遅延読み込みファイルをデバッグするのに役立ちます。

43.6.2 AGENTS.md が読み込まれていない

リポジトリに AGENTS.md があり、Claude がそれが何を言っているかを知らないようである場合、通常の原因はプロジェクトパスのどこかに CLAUDE.md があることです。デフォルトでは、Claude は、作業ディレクトリまたはそれより上のディレクトリに CLAUDE.md または CLAUDE.local.md がない場合にのみ AGENTS.md を読み込みます（~/.claude/CLAUDE.md を除く）。以下の順序で確認します：

1. 作業ディレクトリまたはそれより上のディレクトリ（~/.claude/CLAUDE.md を除く）で CLAUDE.md、.claude/CLAUDE.md、または CLAUDE.local.md を探します。見つかった場合、**Project instructions** を claude-md-and-agents-md に設定しない限り、Claude はそれを AGENTS.md の代わりに読み込みます。

2. claude --version を実行し、v2.1.277 以降であることを確認します。v2.1.281 より前では、Amazon Bedrock 上のセッションやテレメトリが無効なセッションなど、一部のセッションは *AGENTS.md* **を読み込むことができない** (ページ 1010) ため、これらのバージョンでは v2.1.281 以降に更新してください。

3. セッションで /config を入力して設定パネルを開き、**Project instructions** が claude-md または managed-only に設定されていないことを確認します。そこに設定が表示されない場合、セッションは *AGENTS.md* **を読み込むことができない** (ページ 1010) セッションです。

Claude が AGENTS.md を読み込んだかどうかを確認するには、/memory を実行し、リストでそのパスを探します。

v2.1.280 より前では、/memory と /context は Claude が直接読み込んだ AGENTS.md をリストしていませんでした。これらのバージョンでは、代わりに Claude にプロジェクト指示が何であるかを尋ねます。

見つけた CLAUDE.md を保持したい場合、またはセッションが AGENTS.md を読み込むことができない場合は、*AGENTS.md* **の横に** *AGENTS.md* **をインポートする** *CLAUDE.md* **を追加します** (ページ 1011)。

43.6.3 自動メモリが何を保存したかわからない

/memory を実行し、自動メモリフォルダを選択して、Claude が保存したものを参照します。すべてはプレーンマークダウンであり、読み取り、編集、または削除できます。

43.6.4 CLAUDE.md が大きすぎる

200 行を超えるファイルはより多くのコンテキストを消費し、準拠を減らす可能性があります。Claude Code は 4 MiB を超えるファイルをスキップします。**パススコープルール** (ページ 1004) を使用して、Claude が一致するファイルで作業する場合にのみ指示を読み込むか、すべてのセッションで必要でないコンテンツをトリミングします。*@path* **インポート** (ページ 1001) に分割すると、組織化に役立ちますが、インポートされたファイルは起動時に読み込まれるため、コンテキストは削減されません。

/doctor チェックアップは、チェックインされた CLAUDE.md のトリミングを提案します。コードベースから派生できるコンテンツ（ディレクトリレイアウト、依存関係リスト、アーキテクチャの概要など）を削除し、落とし穴、根拠、およびツールのデフォルトと異なる規約を保持します。トリムチェックには Claude Code v2.1.206 以降が必要です。

43.6.5 `/compact` 後に指示が失われたようです

プロジェクトルート CLAUDE.md は圧縮後も保持されます。`/compact` の後、Claude はディスクから再度読み込み、セッションに再度注入します。サブディレクトリ内のネストされた CLAUDE.md ファイルと *paths: frontmatter* (ページ 1004) を持つルールは、Claude がそれらが適用されるファイルを読み込むときに再度読み込まれます。

圧縮後に指示が消えた場合、それは会話でのみ与えられたか、まだ再度読み込まれていないネストされた CLAUDE.md に存在するか、またはまだ再度マッチしたファイルがないパススコープルールです。会話のみの指示を CLAUDE.md に追加して、それらを永続化させます。完全な内訳については、圧縮後に保持されるもの を参照してください。

効果的な指示を書くためのガイダンスについては、**効果的な指示を書く** (ページ 1000) を参照してください。

43.7 関連リソース

- 設定をデバッグする: CLAUDE.md または設定が有効にならない理由を診断する
- *Skills* (ページ 275): オンデマンドで読み込まれる反復可能なワークフローをパッケージ化する
- *Settings* (ページ 711): 設定ファイルで Claude Code の動作を設定する
- *Subagent* **メモリ** (ページ 337): subagent が独自の自動メモリを保持できるようにする

第44章　チェックポイント

Claude のエディット内容と会話を追跡、巻き戻し、要約してセッション状態を管理します。

Claude Code は、作業中に Claude が行ったファイルエディットを自動的に追跡し、変更をすばやく取り消したり、問題が発生した場合に以前の状態に巻き戻したりできます。

44.1 チェックポイントの仕組み

Claude で作業する際、チェックポイント機能は送信する各プロンプトがターンを開始する前のコード状態を自動的にキャプチャします。

44.1.1 自動追跡

Claude Code は、ファイル編集ツールで行われたすべての変更を追跡します。

- 送信する各プロンプトがターンを開始するたびに新しいチェックポイントが作成されます
- Claude Code はセッション内の最新 100 個のチェックポイントのファイルスナップショットを保持します。古いチェックポイントを破棄すると、残りのチェックポイントが参照しないスナップショットファイルが削除されます。ただし、各ファイルの最初のスナップショットは除外されます。これは VS Code 拡張機能がセッション diff のベースラインとして使用します。
- チェックポイントはセッションと共に保存されるため、再開したセッションでも `/rewind` で戻ることができます
- Claude Code は保持期間スイープでセッションのファイルスナップショットを削除します。デフォルトではセッションが最後に保存されてから約 30 日後です。スナップショットが削除されたチェックポイントに巻き戻すと、`No files were restored` エラーで失敗する可能性があります。スナップショットをより長く保持するには、`cleanupPeriodDays` を設定してください。

44.1.2 巻き戻しと要約

`/rewind` を実行するか、プロンプト入力が空の状態で `Esc` キーを 2 回押して、巻き戻しメニューを開きます。

注釈:

プロンプト入力にテキストが含まれている場合、ダブル `Esc` はメニューを開く代わりにテキストをクリアします。クリアされたテキストは入力履歴に保存されるため、巻き戻しメニューを終了した後に `Up` キーを押して呼び出すことができます。

巻き戻しメニューには、セッション中に送信した各プロンプトが表示されます。ただし、**ターン中に送信されたメッセージ** (ページ 1022) は除外されます。操作したいポイントを選択してから、アクションを選択します。

- **コードと会話を復元**: コードと会話の両方をそのポイントに戻します
- **会話を復元**: 現在のコードを保持しながら、そのメッセージに巻き戻します
- **コードを復元**: 会話を保持しながら、ファイルの変更を戻します
- **ここから要約**: このポイント以降の会話を圧縮して要約し、コンテキストウィンドウスペースを解放します
- **ここまで要約**: このポイント前の会話を要約に圧縮し、後のメッセージはそのまま保持します
- **キャンセル**: 変更を加えずにメッセージリストに戻ります

2 つのコード復元オプションは、選択したチェックポイントが戻すべき追跡されたファイル変更を持つ場合にのみ表示されます。そのポイント以降にファイル編集がキャプチャされていない場合、メニューは **会話を復元**、要約オプション、および **キャンセル** のみを提供します。

会話を復元または「ここから要約」を選択した後、選択したメッセージからの元のプロンプトが入力フィールドに復元されるため、再送信または編集できます。

「ここまで要約」を選択すると、会話の最後に留まり、入力フィールドは空になります。どちらの要約オプションでも、圧縮されたメッセージがあった会話内に **要約された会話** マーカーが表示されます。

クリアされた会話を超えて巻き戻す

同じ Claude Code プロセスの前の段階で `/clear` を実行した場合、巻き戻しメニューはリストの最上部に `/resume <session-id>`（前のセッション） というラベルの追加エントリを表示します。これを選択して、`/clear` が実行される前にアクティブだった会話を再開します。このエントリは Claude Code を終了するか別のセッションを再開するまで利用可能です。

要約をガイドする

要約はディスク上のファイルを変更せず、元のメッセージはセッショントランスクリプトに保持されるため、Claude は詳細を参照できます。要約が焦点を当てるべき内容をガイドするには、矢印キーで **要約** オプションをハイライトし、行が **コンテキストを追加（オプション）** と表示されている場所に指示を入力してから、`Enter` キーを押します。数字キーでオプションを選択すると、指示なしで即座に要約されます。

注釈:

要約はセッションを同じ状態に保ち、コンテキストを圧縮します。これは対象を絞った `/compact` のようなものです。元のセッションを保持したまま異なるアプローチを試したい場合は、代わりに `/branch` または `claude --continue --fork-session` を使用してください。

44.2 一般的なユースケース

チェックポイントは以下の場合に特に便利です。

- **代替案の検討**: 開始点を失わずに異なる実装アプローチを試します
- **ミスからの回復**: バグを導入したり機能を破損させた変更をすばやく取り消します
- **機能の反復**: 動作状態に戻すことができるという確信を持って変更を試験します
- **コンテキストスペースの解放**: 冗長なデバッグセッションを中間地点から要約し、初期指示を保持します

44.3 制限事項

44.3.1 Bash コマンドの変更は追跡されません

チェックポイント機能は、Bash コマンドで変更されたファイルを追跡しません。たとえば、Claude Code が以下を実行する場合：

```
rm file.txt
mv old.txt new.txt
cp source.txt dest.txt
```

これらのファイル変更は巻き戻しで取り消すことはできません。Claude のファイル編集ツールで行われた直接的なファイルエディットのみが追跡されます。

44.3.2 サブエージェントの編集は復元されません

サブエージェント (ページ 317) は Claude のファイル編集ツールで編集を行いますが、Claude Code は通常、セッションのチェックポイントでこれらの編集をキャプチャしません。巻き戻しがそれらを復元するかどうかは、サブエージェントの実行方法によって異なります：

- **フォアグラウンドフォークスキル**：*context: fork* **を持つスキル** (ページ 303) がフォアグラウンドで実行され、自分のターン中にワーキングツリーを編集するため、巻き戻しは通常どおりその編集を復元します。`background: false` を設定してフォークをフォアグラウンドで実行します。いくつかの状況（**スキルページに記載** (ページ 303)）では、この設定に関係なくそこで実行されます。
- **その他のサブエージェント**：巻き戻しは編集を復元しません。git を使用してそれらを元に戻してください。これには、バックグラウンドで実行されるフォークスキル（デフォルト）と、バックグラウンド `/code-review --fix` 実行が含まれます。

44.3.3 外部の変更は追跡されません

チェックポイント機能は、現在のセッション内で編集されたファイルのみを追跡します。Claude Code の外部で手動で行ったファイルの変更や、他の同時セッションからのエディットは、通常キャプチャされません。ただし、現在のセッションと同じファイルを変更する場合は除きます。

44.3.4 ターン中に送信されたメッセージはチェックポイントされません

Claude **が作業中にキューに入れたメッセージ** (ページ 90) が実行中のターン内に Claude に到達すると、新しいターンを開始する代わりにそのターンに参加します。メッセージは会話に表示されますが、Claude Code はそれのチェックポイントを作成せず、巻き戻しメニューにはリストされません。Claude Code が独自のターンとして送信するキューに入れたメッセージは、通常どおりチェックポイントを取得します。複数のキューに入れたメッセージが**そのターンを共有** (ページ 90) する場合も含まれます。

そのようなメッセージを削除するか、メッセージの後に Claude が行った編集を取り消すには、ターンを開始したプロンプトに巻き戻します。これにより、メッセージが到達する前に Claude が行った作業を含む、ターン全体が巻き戻されます。

44.3.5 シンボリックリンクおよびハードリンクパスは復元されません

チェックポイント機能は、シンボリックリンクまたはハードリンクされたファイルを巻き戻しません。`/rewind` メニューから **Restore code**（コードを復元）または **Restore code and conversation**（コードと会話を復元）を選択すると、Claude Code は追跡されたパスのうち、シンボリックリンクまたはハードリンクであるパスをスキップし、`Restored the code, but skipped N files`（コードを復元しましたが、N 個のファイルをスキップしました）という警告を表示します。スキップされたファイルは現在の内容を保持します。セッションの変更の 1 つを取り消すには、Claude に編集を逆にするよう依頼するか、ファイルを自分で編集してください。ドットファイルマネージャーがプロジェクトにシンボリックリンクする設定ファイルと、pnpm がその場所にハードリンクするファイルの両方がこのカテゴリに該当します。

復元がスキップするパスを確認するには、復元する前に `/debug` でデバッグログを有効にします：`~/.claude/debug/<session-id>.txt` のデバッグログには、スキップされた各パスが記載されます。スキップの理由と復旧手順については、エラーリファレンスのスキップされたファイルエントリを参照してください。

44.3.6 バージョン管理の代替ではありません

チェックポイントは、クイックなセッションレベルの復旧用に設計されています。永続的なバージョン履歴とコラボレーションの場合は、Git などのバージョン管理を引き続き使用して、コミット、ブランチ、長期履歴を管理してください。

44.4 関連項目

- *Interactive mode* (ページ 75) - キーボードショートカットとセッションコントロール
- Commands - `/rewind` を使用したチェックポイントへのアクセス
- *CLI reference* (ページ 45) - コマンドラインオプション

第45章　開発コンテナ

チーム全体で一貫した分離環境を実現するため、開発コンテナ内で Claude Code を実行します。

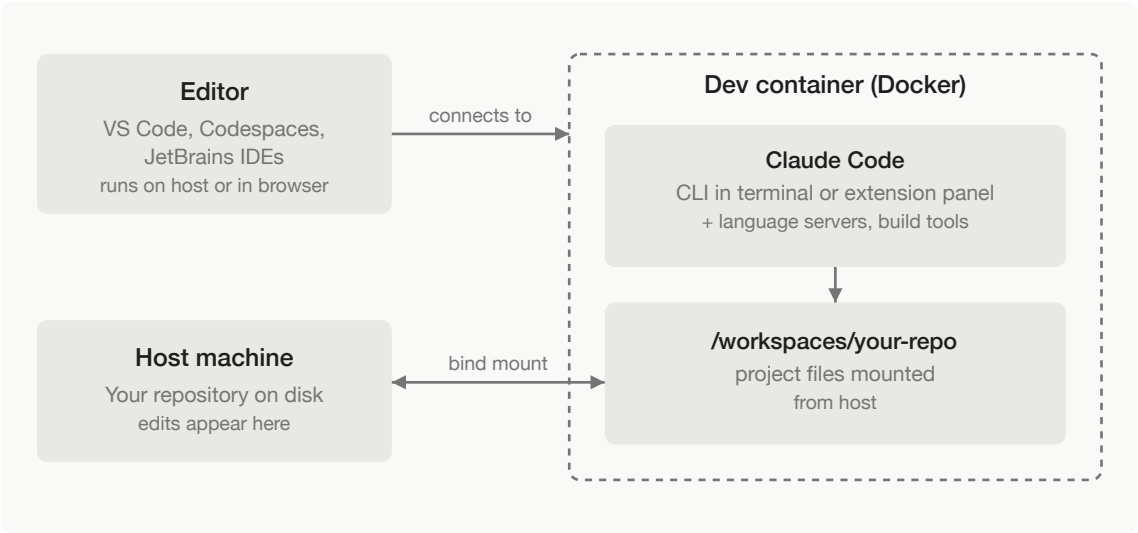
開発コンテナ（dev container）を使用すると、チームのすべてのエンジニアが実行できる同一の分離環境を定義できます。Claude Code がそのコンテナにインストールされている場合、Claude が実行するコマンドはホストマシンではなくコンテナ内で実行され、プロジェクトファイルへの編集はローカルリポジトリに表示されます。

このページでは、**開発コンテナに *Claude Code* をインストール** (ページ 1024) する方法と、その後の自己完結型の設定トピックについて説明します。認証をリビルド全体で保持する、組織ポリシーを適用する、ネットワークエグレスを制限する、権限プロンプトなしで実行するなどです。セットアップに合致するものをお読みください。

警告:

開発コンテナは実質的な保護を提供していますが、すべての攻撃に完全に耐性のあるシステムはありません。 `--dangerously-skip-permissions` で実行する場合、開発コンテナは、`~/.claude` に保存されている Claude Code の認証情報を含む、コンテナ内でアクセス可能なものを悪意のあるプロジェクトが流出させることを防ぎません。信頼できるリポジトリで開発する場合にのみ開発コンテナを使用し、Claude のアクティビティを監視してください。 `~/.ssh` やクラウド認証情報ファイルなどのホストシークレットをコンテナにマウントすることは避け、リポジトリスコープまたは短期間有効なトークンを使用してください。

開発コンテナがエディタとどのように連携するか



開発コンテナは Docker コンテナとして実行され、マシン上または GitHub Codespaces などのクラウドホスト上で実行されます。Dev Containers 仕様をサポートするエディタ（VS Code、GitHub Codespaces、JetBrains IDE、Cursor など）がそのコンテナに接続します。通常どおりエディタでファイルを参照お

よび編集しますが、統合ターミナル、言語サーバー、ビルドツールはすべてホストではなくコンテナ内で実行されます。プレーン Vim などの開発コンテナをサポートしていないエディタはこのワークフローの対象外です。

Claude Code はコンテナ内で実行されるため、プロジェクトのツールチェーンの残りの部分と同じファイル、依存関係、ツールが表示されます。VS Code では、*Claude Code* **拡張機能パネル** (ページ 173) を使用するか、統合ターミナルで `claude` を実行できます。どちらもコンテナ内で実行され、同じ `~/.claude` 設定を共有します。

45.1 開発コンテナに Claude Code を追加する

Claude Code は、Claude Code Dev Container Feature を通じて任意の開発コンテナにインストールされます。

設定は、VS Code、GitHub Codespaces、JetBrains IDE など、Dev Containers 仕様をサポートするあらゆるツールで機能します。以下の手順では、例として VS Code を使用しています。

VS Code または Codespaces でコンテナを開くと、機能は Claude Code VS Code 拡張機能も追加します。他のエディタはその部分を無視します。

Tip:

開発コンテナが初めてですか？ VS Code Dev Containers チュートリアルでは、Docker、拡張機能、最初のコンテナを開く方法について説明しています。ファイアウォールと永続ボリュームを備えた、より完全な強化例については、**リファレンスコンテナを試す** (ページ 1028) を参照してください。

1. **devcontainer.json を作成または更新する**

   以下をリポジトリの `.devcontainer/devcontainer.json` として保存するか、既存ファイルに `features` ブロックを追加します。

   末尾のバージョンタグ（`:1.0` など）は、Claude Code リリースではなく、機能のインストールスクリプトをピン留めします。機能は最新の Claude Code をインストールし、Claude Code はデフォルトでコンテナ内で自動更新されます。

   CLI バージョンをピン留めするか、自動更新を無効にするには、**組織ポリシーを適用する** (ページ 1026) を参照してください。

リスト 1: .devcontainer/devcontainer.json

```
{
  "image": "mcr.microsoft.com/devcontainers/base:ubuntu",
  "features": {
    "ghcr.io/anthropics/devcontainer-features/claude-code:1.0": {}
  }
}
```

   `image` 行をプロジェクトのベースイメージに置き換えるか、既存ファイルが Dockerfile を使用している場合は削除します。

   Claude Code 機能は、ベースイメージが Node.js を提供しない場合、Node.js 自体をインストールします。そのインストールが失敗し、ビルドが `Failed to install Node.js and npm` で停止する場合

は、`"ghcr.io/devcontainers/features/node:1": {}` を Claude Code 機能の上の `features` ブロックに追加して、再構築してください。

2. **コンテナを再構築する**

   Mac では `Cmd+Shift+P`、Windows と Linux では `Ctrl+Shift+P` で VS Code コマンドパレットを開き、**Dev Containers: Rebuild Container** を実行します。

   他のツールについては、そのツールの再構築アクション（GitHub Codespaces での再構築、Dev Containers CLI、または IDE の開発コンテナドキュメント）に従ってください。

3. **Claude Code にサインインする**

   再構築されたコンテナでターミナルを開き、`claude` を実行して、認証プロンプトに従います。

認証プロンプトで表示される内容は、プロバイダーによって異なります：

- **Anthropic**：Claude または Anthropic Console アカウントでブラウザ経由でサインイン
- *Amazon Bedrock、Google Cloud の Agent Platform、または Microsoft Foundry* (ページ 847)：Claude Code はクラウドプロバイダーの認証情報を使用し、ブラウザプロンプトはありません

クラウドプロバイダーの場合、ホストから認証情報ファイルをマウントするのではなく、`containerEnv`、Codespaces シークレット、またはクラウドのワークロード ID を通じて認証情報をコンテナに渡します。Claude Code が読み取る認証情報チェーンについては、*Amazon Bedrock* (ページ 863)、*Google Cloud の Agent Platform* (ページ 881)、または *Microsoft Foundry* (ページ 891) を参照してください。

どのパスが組織に適しているかを決定するには、API プロバイダーを選択するを参照してください。

注釈:

ブラウザサインインが完了しても、コールバックがコンテナに到達しない場合は、ブラウザに表示されているコードをコピーして、ターミナルの `Paste code here if prompted` プロンプトに貼り付けます。これは、エディタのポート転送が localhost コールバックをルーティングしない場合に発生する可能性があります。

45.2 再構築時に認証と設定を保持する

デフォルトでは、コンテナのホームディレクトリは再構築時に破棄されるため、エンジニアは毎回サインインし直す必要があります。Claude Code は認証トークン、ユーザー設定、セッション履歴を `~/.claude` ディレクトリに保存します。OAuth アカウント、個人用 MCP サーバー、プロジェクトごとの信頼設定は `~/.claude.json` に保存されます。これはそのディレクトリの外にある別ファイルであるため、`~/.claude` だけにボリュームをマウントしても、サインイン状態は保持されません。`~/.claude` に名前付きボリュームをマウントし、`CLAUDE_CONFIG_DIR` を同じパスに設定して、Claude Code が `.claude.json` をボリューム内に書き込むようにします。

以下の例は、`remoteUser` が `node` であるコンテナのボリュームをマウントし、`CLAUDE_CONFIG_DIR` を設定します：

リスト 2: devcontainer.json

```
"mounts": [
  "source=claude-code-config,target=/home/node/.claude,type=volume"
],
"containerEnv": {
  "CLAUDE_CONFIG_DIR": "/home/node/.claude"
}
```

/home/node をコンテナの remoteUser のホームディレクトリに置き換えます。既に containerEnv を設定している場合（例えば **組織ポリシーの適用** (ページ 1026) で設定している場合）、新しいオブジェクトを追加するのではなく、そのオブジェクトに CLAUDE_CONFIG_DIR を追加します。

プロジェクトごとに状態を分離して、すべてのリポジトリ間で 1 つのボリュームを共有しないようにするには、ソース名に ${devcontainerId} 変数を含めます。リファレンス設定はこの目的で source=claude-code-config-${devcontainerId} を使用しています。

GitHub Codespaces では、~/.claude は codespace の停止と開始の間で保持されますが、コンテナを再構築するときはクリアされるため、上記の設定がそこにも適用されます。

codespace 間で認証を実行するには、Codespaces シークレットとして ANTHROPIC_API_KEY または *claude setup-token* (ページ 860) からの CLAUDE_CODE_OAUTH_TOKEN を保存します。Codespaces はシークレットを自動的にコンテナ内の環境変数として利用可能にします。

45.3 組織ポリシーを適用する

開発コンテナは、同じイメージと設定がすべてのエンジニアのマシンで実行されるため、組織ポリシーを適用するのに便利な場所です。

Claude Code は Linux で /etc/claude-code/managed-settings.json を読み取り、**設定階層** (ページ 719) で最高の優先度で適用するため、そこの値はエンジニアが ~/.claude またはプロジェクトの .claude/ ディレクトリで設定したものをオーバーライドします。Dockerfile からファイルをコピーして配置します：

リスト 3: Dockerfile

```
RUN mkdir -p /etc/claude-code
COPY managed-settings.json /etc/claude-code/managed-settings.json
```

Dockerfile はリポジトリに存在するため、書き込みアクセス権を持つ誰でもこのステップを変更または削除できます。エンジニアがリポジトリファイルを編集してバイパスできないポリシーについては、サーバー管理設定または MDM を通じて管理設定を配信します。利用可能なキーと他の配信パスについては、管理設定ファイルを参照してください。

コンテナ内のすべての Claude Code セッションに適用される環境変数を設定するには、devcontainer.json の containerEnv に追加します。以下の例は、テレメトリとエラーレポートをオプトアウトし、Claude Code がインストール後に自動更新されるのを防ぎます：

リスト 4: devcontainer.json

```
"containerEnv": {
  "CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC": "1",
  "DISABLE_AUTOUPDATER": "1"
}
```

CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC は、Remote Control と他のフィーチャーフラグ取得が必要な機能が依存するフィーチャーフラグ評価も無効にするため、コンテナ内のセッションはそれらを使用できません。

Dev Container Feature は常に最新の Claude Code リリースをインストールします。再現可能なビルドのために特定の Claude Code バージョンをピン留めするには、機能を使用する代わりに Dockerfile から `npm install -g @anthropic-ai/claude-code@X.Y.Z` でインストールし、`containerEnv` で `DISABLE_AUTOUPDATER` を `1` に設定します。

権限ルール、ツール制限、MCP サーバーアローリストを含むポリシーコントロールの完全なリストについては、組織向けに Claude Code をセットアップするを参照してください。

MCP **サーバー** (ページ 363) をコンテナ内で利用可能にするには、リポジトリルートの `.mcp.json` ファイルで**プロジェクトスコープ** (ページ 380) で定義して、開発コンテナ設定と一緒にチェックインします。ローカル stdio サーバーが依存するバイナリを Dockerfile にインストールし、リモートサーバードメインをネットワークアローリストに追加します。

45.4 ネットワークエグレスを制限する

コンテナのアウトバウンドトラフィックを Claude Code が必要とするドメインのみに制限できます。推論と認証ドメインについては**ネットワークアクセス要件** (ページ 904) を参照し、オプションのテレメトリとエラーレポート接続およびそれらを無効にする方法については**テレメトリサービス** (ページ 1109) を参照してください。

リファレンスコンテナには、スクリプトが許可する宛先へのアウトバウンドトラフィックを制限する `init-firewall.sh` スクリプトが含まれています。コンテナ内でファイアウォールを実行するには追加の権限が必要なため、リファレンスは `runArgs` を通じて `NET_ADMIN` と `NET_RAW` 機能を追加します。ファイアウォールスクリプトとこれらの機能は Claude Code 自体には必須ではありません。これらを除外して、代わりに独自のネットワークコントロールに依存することができます。

45.5 権限プロンプトなしで実行する

コンテナは Claude Code を非ルートユーザーとして実行し、コマンド実行をコンテナに限定するため、無人操作のために `--dangerously-skip-permissions` を渡すことができます。CLI はルートとして起動された場合、このフラグを拒否するため、`remoteUser` が非ルートアカウントに設定されていることを確認します。

権限プロンプトをスキップすると、実行前にツール呼び出しを確認する機会が失われます。Claude はバインドマウントされたワークスペース内のあらゆるファイルを変更でき、これはホストに直接表示され、コンテナのネットワークポリシーが許可するものに到達できます。このフラグを上記の**ネットワークエグレス制限** (ページ 1027) と組み合わせて、バイパスされたセッションが到達できるものを制限します。

安全チェックを無効にせずにプロンプトを減らしたい場合は、代わりに自動モードを検討してください。これは、実行前にアクションを確認するための分類器を備えています。エンジニアが `--dangerously-skip-permissions` をまったく使用できないようにするには、管理設定で `permissions.disableBypassPermissionsMode` を `"disable"` に設定します。

45.6 リファレンスコンテナを試す

`anthropics/claude-code` リポジトリには、CLI、エグレスファイアウォール、永続ボリューム、Zsh ベースのシェルを組み合わせた開発コンテナの例が含まれています。これは、保守されたベースイメージではなく、動作する例として提供されています。独自の設定に適用する前に、ピースがどのように組み合わさるかを確認するために使用してください。

1. **前提条件をインストールする**

   VS Code と Dev Containers 拡張機能をインストールします。

2. **リファレンスをクローンする**

   Claude Code リポジトリをクローンして、VS Code で開きます。

3. **コンテナで再度開く**

   プロンプトが表示されたら、**Reopen in Container** をクリックするか、コマンドパレットから **Dev Containers: Reopen in Container** を実行します。

4. **Claude Code を開始する**

   コンテナのビルドが完了したら、`Ctrl+`` でターミナルを開き、`claude` を実行してサインインし、最初のセッションを開始します。

この設定を独自のプロジェクトで使用するには、`.devcontainer/` ディレクトリをリポジトリにコピーして、ツールチェーン用に Dockerfile を調整するか、**開発コンテナに *Claude Code* を追加する** (ページ 1024)に戻って、既に持っている設定に機能のみを追加します。

リファレンス設定は 3 つのファイルで構成されています。機能を通じて独自の開発コンテナに Claude Code を追加する場合、これらは必須ではありませんが、ピースを組み合わせる 1 つの方法を示しています。

ファイル	目的
`devcontainer.json`	ボリュームマウント、`runArgs` 機能、VS Code 拡張機能、`containerEnv`
`Dockerfile`	ベースイメージ、開発ツール、Claude Code インストール
`init-firewall.sh`	許可されたドメイン以外のすべてのアウトバウンドネットワークトラフィックをブロック

45.7 次のステップ

Claude Code が開発コンテナで実行されたら、以下のページは組織ロールアウトの残りの部分をカバーしています。認証パスの選択、リポジトリ外での管理ポリシーの配信、使用状況の監視、Claude Code が保存および送信するものの理解です。

- 組織向けに Claude Code をセットアップする：認証プロバイダーを選択し、ポリシーがデバイスに到達する方法を決定し、ロールアウトを計画します
- サーバー管理設定：claude.ai 管理コンソールから管理ポリシーを配信して、エンジニアがリポジトリファイルを編集してバイパスできないようにします
- **使用状況の監視と監査アクティビティ** (ページ 935)：OpenTelemetry メトリクスをエクスポートして、チームが実行しているものを確認します
- **ネットワークアクセス要件** (ページ 904)：プロキシとファイアウォール用の完全なドメインアローリスト
- **テレメトリサービスとオプトアウト** (ページ 1109)：Claude Code がデフォルトで送信するもの、およびそれを無効にする環境変数
- .claude ディレクトリを探索する：ボリュームマウントが保持するもの（認証情報、設定、セッション履歴を含む）
- サンドボックス環境：開発コンテナと組み込み Bash サンドボックス、カスタムコンテナ、VM を比較します
- **セキュリティモデル** (ページ 1033)：Claude Code の権限システム、サンドボックス、プロンプトインジェクション保護がどのように組み合わさるか
- 権限モード：プランモードから自動モードからバイパスまでの完全な範囲、および各モードを使用する場合

第VIII部

セキュリティ・プライバシー

第46章　セキュリティ

Claude Code のセキュリティ対策とセキュアな使用方法のベストプラクティスについて学びます。

46.1 セキュリティへのアプローチ方法

46.1.1 セキュリティの基盤

コードのセキュリティは最優先事項です。Claude Code はセキュリティを中核に据えて構築されており、Anthropic の包括的なセキュリティプログラムに従って開発されています。詳細情報とリソース（SOC 2 Type 2 レポート、ISO 27001 証明書など）については、Anthropic Trust Center をご覧ください。

46.1.2 パーミッションベースのアーキテクチャ

セッションの権限モードは、Claude が事前に確認せずに実行できるアクションを決定します。auto モードは、対話型のターミナルセッションと VS Code セッションにおける組み込みの開始権限モードです。セッションがどのモードで開始されるか では、以前のバージョン、その他のサーフェス、および開始権限モードを変更する設定について説明しています。

- **auto モード**: 別の分類器モデルがユーザーの代わりにアクションをレビューし、安全でないと判断したものをブロックします。分類器がアクションを評価する方法 では、Claude Code が直接承認するアクション、分類器に送信するアクション、および Claude Code が引き続きユーザーに確認するアクションを一覧表示しています。明示的な ask ルールと deny ルールは引き続き適用され、組織は auto モードをオフにする ことができます
- **Manual モード**: Claude Code は読み取り専用の権限で開始されます。ファイルを編集したり、テストを実行したり、コマンドを実行したりする必要がある場合は、まずユーザーに確認し、ユーザーはアクションを 1 回だけ承認するか、それ以降は常に許可するかを選択できます。`ls`、`cat`、`git status` などの **読み取り専用コマンド** (ページ 1049) の組み込みセットは、確認なしで実行されます

ユーザーと組織は、これらの権限を直接設定します。詳細な権限設定については、*Permissions* (ページ 1039) を参照してください。

46.1.3 組み込み保護機能

agentic システムのリスクを軽減するために：

- **サンドボックス化された bash ツール**: *Sandbox* (ページ 1069) bash コマンドをファイルシステムとネットワークの分離で実行し、権限プロンプトを減らしながらセキュリティを維持します。`/sandbox` で設定して、Claude Code が自律的に動作できる境界を定義します

- **作業ディレクトリの境界**: Manual モードでは、Claude Code のファイルツールが、起動されたフォルダとそのサブフォルダの外部で読み取りまたは書き込みを行う前に、ユーザーに確認します。この境界は権限プロンプトであるため、ユーザーが承認した Bash コマンドは、ユーザーアカウントが書き込み可能な任意の場所に書き込むことができます
  - 確認なしでフォルダを読み取るには、そのフォルダを **追加ディレクトリ** (ページ 1060) として追加します
  - オペレーティングシステムレベルで Bash コマンドを制限するには、**サンドボックス化** (ページ 1086) をオンにします
- **プロンプト疲労の軽減**: ユーザーごと、コードベースごと、または組織ごとに頻繁に使用される安全なコマンドのホワイトリスト化をサポート
- **Accept Edits モード**: ファイル編集と `mkdir`、`touch`、`rm`、`mv`、`cp`、`sed` などの固定セットのファイルシステム Bash コマンドを作業ディレクトリ内のパスに対して自動承認します。その他の Bash コマンドとスコープ外のパスはプロンプトが表示されます

46.1.4 ユーザーの責任

承認前に、提案されたコードとコマンドの安全性を確認する責任があります。

46.2 プロンプトインジェクションから保護する

プロンプトインジェクションは、攻撃者が悪意のあるテキストを挿入することで AI アシスタントの指示をオーバーライドまたは操作しようとする手法です。Claude Code にはこれらの攻撃に対する複数のセーフガードが含まれています：

46.2.1 コア保護機能

- **権限モード**: Manual モードでは、機密操作には明示的な承認が必要です
- **ネットワークコマンド承認**: `curl` や `wget` などのウェブからコンテンツを取得するコマンドはデフォルトでは自動承認されません。Manual モードでは他の読み取り専用以外の Bash コマンドと同様にプロンプトが表示されるため、一度承認するか、`Bash(curl *)` のような明示的な許可ルールを追加できます。Claude がこれらを実行しないようにするには、*`permissions.deny`* (ページ 1047) に追加してください。deny ルールは**書かれたとおりの** (ページ 1048) コマンドにマッチします。コマンドテキストに依存しないネットワーク強制については、*sandbox* **ネットワーク分離** (ページ 1087) を参照してください

46.2.2 プライバシーセーフガード

データを保護するために、複数のセーフガードを実装しています：

- 機密情報の保持期間の制限（詳細については Privacy Center を参照してください）
- ユーザーセッションデータへのアクセス制限
- データトレーニング設定に対するユーザーコントロール。コンシューマーユーザーは プライバシー設定 をいつでも変更できます。

詳細については、Commercial Terms of Service（Team、Enterprise、API ユーザー向け）または Consumer Terms（Free、Pro、Max ユーザー向け）および Privacy Policy をご確認ください。

46.2.3 追加のセーフガード

- **ネットワークリクエスト承認**: Manual モードでは、ネットワークリクエストを行うほとんどのツールはデフォルトでユーザー承認が必要です
- **Web ページの要約**: ほとんどのフェッチでは、WebFetch がページに対して別のモデル呼び出しを実行し、Claude は生のページではなくその呼び出しの回答を受け取ります。WebFetch ツールの動作 を参照してください
- **信頼検証**: 対話型セッションでは、まだ信頼していないフォルダで Claude Code を起動すると、ワークスペースの信頼ダイアログが表示されます。プロジェクトの `.mcp.json` 内のサーバーには独自の承認プロンプトがあり、**プロジェクトスコープ** (ページ 382) にはそのプロンプトをスキップするセッションが記載されています
  - 注：`-p` セッションではどちらのプロンプトも表示されません。そこでリポジトリのファイルが実行できるものについては、**フォルダを信頼する前に実行されるもの** (ページ 1066) に記載されています
  - 注：Claude Code をホームディレクトリで直接起動する場合、信頼受け入れは現在のセッションのみ保持され、ディスクに書き込まれないため、起動するたびにプロンプトが再度表示されます。これを永続化するための設定はありません。代わりに、プロジェクトサブディレクトリから Claude Code を起動してください。そこでは信頼受け入れはディレクトリごとに保存されます
- **コマンドインジェクション検出**: Manual モードでは、Claude Code は完全に分析できない Bash コマンドを実行する前に確認を求めます。`Bash(git *)` のようなコマンドの一部に対する許可ルールがあっても、このプロンプトはスキップされません。**サンドボックス化されたコマンド** (ページ 1063) はこのプロンプトなしで実行できます
- **フェイルクローズドマッチング**: Manual モードでは、マッチしないコマンドはデフォルトで承認が必要です
- **セキュアな認証情報ストレージ**: API キーとトークンは、利用可能な場合は macOS Keychain に保存されます。Linux ではモード `0600` のファイルに保存され、Windows ではユーザープロファイルディレクトリのアクセス制御を継承するファイルに保存されます。*Credential Management* (ページ 857) を参照してください

警告:

Windows WebDAV セキュリティリスク: Windows で Claude Code を実行する場合、WebDAV を有効にしたり、Claude Code に `\\*` などの WebDAV サブディレクトリを含む可能性のあるパスへのアクセスを許可することはお勧めしません。WebDAV は Microsoft によって非推奨になっています セキュリティリスクのため。WebDAV を有効にすると、Claude Code がリモートホストへのネットワークリクエストをトリガーし、パーミッションシステムをバイパスする可能性があります。

信頼できないコンテンツを使用する場合のベストプラクティス：

1. 承認前に提案されたコマンドを確認します
2. 信頼できないコンテンツを Claude に直接パイプすることを避けます

3. 重要なファイルへの提案された変更を確認します
4. 仮想マシン（VM）を使用してスクリプトを実行し、ツール呼び出しを行います。特に外部 Web サービスと対話する場合
5. `/feedback` で疑わしい動作を報告します

警告:

これらの保護機能はリスクを大幅に軽減しますが、どのシステムもすべての攻撃に完全に免疫があるわけではありません。AI ツールを使用する場合は常に良好なセキュリティプラクティスを維持してください。

46.3 MCP セキュリティ

Claude Code を Model Context Protocol（MCP）サーバーに接続できます。プロジェクトスコープのサーバーは `.mcp.json` で定義され、このファイルはソース管理にチェックインできます。**その他のスコープ** (ページ 380) のサーバーや *claude.ai* **コネクタ** (ページ 398) はリポジトリの外部で設定され、プラグインもサーバーを追加できるため、`.mcp.json` を確認しても、セッションが読み込む可能性のあるすべてのサーバーがわかるわけではありません。組織内で実行されるサーバーを制限するには、マネージド MCP 設定を参照してください。

独自の MCP サーバーを作成するか、信頼できるプロバイダーからの MCP サーバーを使用することをお勧めします。Claude Code パーミッションを MCP サーバー用に設定できます。Anthropic は MCP サーバーをリスティング基準 に照らして確認してから Anthropic Directory に追加しますが、MCP サーバーのセキュリティ監査または管理は行いません。

46.4 IDE セキュリティ

IDE で Claude Code を実行する場合の詳細については、*VS Code security and privacy* (ページ 200) を参照してください。

46.5 クラウド実行セキュリティ

クラウドセッション (ページ 223) を使用する場合、追加のセキュリティ制御が実施されます。組織が self-hosted environment にルーティングするセッションは独自のインフラストラクチャで実行され、分離、ネットワーク出力、および Git 認証情報は展開の責任です。Anthropic ホスト環境では：

- **分離された仮想マシン**: 各クラウドセッションは分離された Anthropic 管理 VM で実行されます
- **ネットワークアクセス制御**: ネットワークアクセスはデフォルトで制限され、無効にするか特定のドメインのみを許可するように設定できます
- **認証情報保護**: GitHub 認証情報は Anthropic のサーバーに暗号化されて保存され、セッション VM に入ることはありません。VM はそのセッションにスコープされた短命の認証情報を保持し、GitHub トラフィックは GitHub 認証情報をサーバー側で付与する Anthropic プロキシ を通じて流れます。アクセスを付与する方法については、*GitHub* **認証オプション** (ページ 224) を参照してください

- **プッシュ制限**: GitHub プロキシ は、ブランチの削除と、タグなどブランチ以外のもののプッシュを拒否します。セッションがどのブランチを更新できるかは、接続した GitHub アクセスに対してリポジトリのブランチ保護ルールとルールセットを適用することで GitHub が決定します。そのアクセスがバイパスできるルールでは、セッションのプッシュはブロックされません
- **監査ログ**: クラウドセッション内のすべての操作はコンプライアンスと監査目的でログされます
- **自動クリーンアップ**: セッション VM は非アクティブ期間後に回収されます
- **削除**: **セッションを削除** (ページ 233) することはいつでも可能です。クラウドセッションについて Anthropic が保存する内容については、**クラウド実行データフロー** (ページ 1109) を参照してください

クラウド実行の詳細については、*Claude Code* **をクラウドで使用する** (ページ 223) を参照してください。クラウドセッションのネットワークアクセスを設定するには、クラウド環境を設定する を参照してください。

Remote Control セッションは異なる方法で動作します：Web インターフェースはローカルマシンで実行されている Claude Code プロセスに接続します。すべてのコード実行とファイルアクセスはローカルに留まり、セッショントラフィックは TLS 経由で Anthropic API を通じて流れます。接続中、セッショントランスクリプトはデバイス間で会話を同期するために Anthropic サーバーに保存されます。これは Connection and security で説明されています。クラウド VM またはサンドボックスは関与しません。接続は複数の短命で狭くスコープされた認証情報を使用し、各認証情報は特定の目的に限定され、独立して有効期限が切れ、単一の侵害された認証情報のブラストラディウスを制限します。

46.6 セキュリティベストプラクティス

46.6.1 機密コードの使用

- 承認前にすべての提案された変更を確認してください
- 機密リポジトリにはプロジェクト固有のパーミッション設定を使用してください
- 追加の分離のために *dev containers* (ページ 1023) の使用を検討してください
- `/permissions` で定期的にパーミッション設定を監査してください

46.6.2 チームセキュリティ

- *managed settings* (ページ 711) を使用して組織標準を実施してください
- 承認されたパーミッション設定をバージョン管理を通じて共有してください
- チームメンバーにセキュリティベストプラクティスについてトレーニングを行ってください
- *OpenTelemetry metrics* (ページ 935) を通じて Claude Code の使用を監視してください
- *ConfigChange hooks* (ページ 531) でセッション中の設定変更を監査またはブロックしてください

46.6.3 セキュリティ問題の報告

Claude Code でセキュリティ脆弱性を発見した場合：

1. 公開で開示しないでください

2. HackerOne program を通じて報告してください
3. 詳細な再現手順を含めてください
4. 公開開示前に問題に対処する時間を与えてください

46.7 関連リソース

- Security guidance plugin：Claude がセッション中に独自のコード変更の脆弱性をレビューして修正します
- `/security-review`：現在のブランチの変更に対してオンデマンドのセキュリティパスを実行します
- Sandbox environments：分離アプローチを比較し、脅威モデルに合わせて選択します
- *Sandboxing* (ページ 1069)：Bash コマンドのファイルシステムとネットワーク分離
- *Permissions* (ページ 1039)：パーミッションとアクセス制御を設定します
- *Monitoring usage* (ページ 935)：Claude Code アクティビティを追跡および監査します
- *Development containers* (ページ 1023)：セキュアで分離された環境
- Anthropic Trust Center：セキュリティ認証とコンプライアンス
- CISO's guide to agentic AI：agentic AI デプロイメントを評価するためのセキュリティリーダーのフレームワーク

第47章　権限を設定する

きめ細かい権限ルール、モード、管理ポリシーを使用して、Claude Code がアクセスして実行できる内容を制御します。

Claude Code は、エージェントが実行できることと実行できないことを正確に指定できるようにするため、きめ細かい権限をサポートしています。権限設定はバージョン管理にチェックインでき、組織内のすべての開発者に配布できるほか、個々の開発者がカスタマイズできます。

47.1 権限システム

Claude Code は、パワーと安全性のバランスを取るために、段階的な権限システムを使用しています。表は、各ツールタイプについて、Manual モードがアクションの実行前に確認するかどうかを示しています。その他の**権限モード** (ページ 1042) は、これらのどれがあなたに確認するかを変更します。auto モードでは、分類器があなたの代わりにアクションをレビューし、分類器がアクションをどのように評価するかは、それが見るアクションをリストアップしています。

ツールタイプ	例	承認が必要	「はい、今後は聞かない」の動作
読み取り専用	ファイル読み取り、Grep	いいえ、**作業ディレクトリと追加ディレクトリ** (ページ 1060) 内	N/A
Bash コマンド	シェル実行	はい、**読み取り専用コマンド** (ページ 1049) の組み込みセットを除く	リポジトリとコマンドごとに永続的
ファイル変更	Edit/Write ファイル	はい	セッション終了まで
Web フェッチ	WebFetch	はい、事前承認されたドキュメンテーションドメインの組み込みセットを除く	リポジトリとドメインごとに永続的
Web 検索	WebSearch	はい	リポジトリごとに永続的

権限プロンプトには、Claude が実行しようとしている内容が表示され、その後に選択肢が続きます。次の例は、Manual モードのセッションにおける Bash コマンドのプロンプトです。

```
Bash command
Tip: auto mode handles these prompts for you — choose "switch to auto mode"
below
Run the test suite
--------------------------------------------------------------------------------
npm test
--------------------------------------------------------------------------------
This command requires approval

Do you want to proceed?
❯ 1. Yes
  2. Yes, and don't ask again for: npm test *
  3. Yes, and switch to auto mode · auto mode handles these prompts for you
  4. No

Esc to cancel · Tab to amend
```

3 番目のオプションである **Yes, and switch to auto mode** は、すべてのプロンプトに表示されるわけではありません。

「はい、今後は聞かない」を選択し、承認が永続的に保存される場合（Bash コマンドや WebFetch ドメインなど）、Claude Code はルールを git リポジトリのルートにある `.claude/settings.local.json` に保存します。これは worktrees を通じてメインチェックアウトに解決されます。ルールは、そのリポジトリ内のサブディレクトリで開始されたセッションや worktrees 内のセッションを含む、そのリポジトリ内の将来のセッションに適用されます。ファイル変更の承認はファイルに保存されません。表が示すように、セッション終了まで続きます。git リポジトリの外部や Windows 上など、場合によっては Claude Code はリポジトリルートを使用しません。*Claude Code* **が各ファイルを探す場所** (ページ 714) は、これらのケースと代わりにルールを保存する場所をリストアップしています。

v2.1.211 より前では、Claude Code は常にルールを開始ディレクトリに保存していたため、worktree またはサブディレクトリで付与された承認はリポジトリの残りの部分に適用されませんでした。以前のバージョンがサブディレクトリまたは worktree に保存したルールは、そこで開始されたセッションに引き続き適用されます。

場合によっては、権限プロンプトは 1 回限りの承認のみを提供し、「今後は聞かない」オプションもセッションの残りの部分のアクションを許可するオプションもありません。Claude Code はプロンプトがそれらが許可するすべてのものをあなたに表示できる場合にのみ、これらのオプションを提供するため、プロンプトから保存するルールは、その名前のオプションが許可するものだけをカバーします。プロンプトが 1 回限りの承認のみを提供する場合、アクションを 1 回承認するか、*/permissions* (ページ 1041) でルール自体を追加してください。

47.1.1 権限プロンプトに回答するときにコメントを追加する

権限プロンプトに回答するときに、単一のアクションを承認または拒否するときに、Claude にメモを添付できます。Bash、PowerShell、ファイル、MCP ツールプロンプトを含むほとんどの権限プロンプトで、**はい**または**いいえ**に移動し、Tab を押してそのオプションのコメントフィールドを開きます。WebFetch とブラウザプロンプトはフィールドを提供しません。セッションの残りの部分のアクションを許可するか、ルールを保存するオプションも 1 つを取りません。

フィールドが開いている状態で、コメントを入力してから、これらのキーのいずれかを押します。

- `Enter`：コメントが添付された状態で回答を送信します。フィールドを空のままにすると、Claude Code はコメントなしで回答を送信します。
- `Tab`：フィールドを閉じて回答しません。Claude Code は入力したテキストを保持し、そのオプションで回答する場合でも送信します。
- `Shift+Tab`：ファイルプロンプト（Edit または Write プロンプトなど）では、フィールドを `Tab` と同じように閉じます。v2.1.235 より前では、フィールド内で `Shift+Tab` を押すと、セッションの残りの部分のアクションを許可するオプションが選択されたため、Claude Code はセッションの残りの部分のアクションを承認し、コメントを破棄しました。

Claude Code は、回答方法に応じてコメントを異なる方法で配信します。

- **はい**：Claude Code はアクションを実行し、結果の後に Claude にコメントを送信します。
- **いいえ**：Claude Code はコメントを Claude に拒否の理由として送信し、Claude は作業を続けます。メインの会話からのプロンプトでコメントなしで**いいえ**を選択すると、Claude Code はターンを停止します。

47.2 権限を管理する

`/permissions` を使用して、Claude Code のツール権限を表示および管理できます。このダイアログは、すべての権限ルールと、各ルールが取得される `settings.json` ファイルをリストします。Claude が作業中にダイアログを開くことができます。ルールを追加または削除すると、Claude Code は同じターン内の Claude の次のツール呼び出しから変更を適用します。v2.1.234 より前では、Claude Code はターンが終了するまでコマンドをキューに入れていました。

- **Allow** ルールは、Claude Code が手動承認なしで指定されたツールを使用できるようにします。
- **Ask** ルールは、Claude Code が指定されたツールを使用しようとするたびに確認を促します。
- **Deny** ルールは、Claude Code が指定されたツールを使用することを防止します。

ルールは順序で評価されます。deny、ask、allow の順です。その順序での最初のマッチがアウトカムを決定し、ルールの特異性は順序を変更しません。

`Bash(aws *)` のような広い deny ルールは、`Bash(aws s3 ls)` のようなより狭い allow ルールにもマッチする呼び出しを含む、マッチするすべての呼び出しをブロックします。allow ルールは deny ルールから例外を作成することはできません。ask と allow の間にも同じ優先順位が適用されます。マッチする ask ルールは、同じ呼び出しにマッチするより具体的な allow ルールがある場合でも、プロンプトを表示します。

Deny ルールは、ツール名を指定するか、ツール内のパターンをスコープするかによって異なる動作をします。`Bash` のようなベアツール名は、ツールを Claude のコンテキストから完全に削除するため、Claude はそれを見ることはありません。セッション中にそのようなルールを追加する場合、Claude は次のツール呼び出しからそのツールを呼び出すことができません。ツール全体を拒否するでは、Claude が既に見た定義に何が起こるかについて説明しています。`Bash(rm *)` のようなスコープ付きルールは、ツールを利用可能なままにし、Claude が試みたときにマッチする呼び出しをブロックします。

ベア名削除はすべてのツールに適用されます（`EndConversation` を除く）。deny ルールは他のツールが残っている間はそれを削除できず、ask ルールはそれに対してプロンプトを表示しません。

注釈:

権限ルールは Claude Code によって実装されており、モデルによってではありません。プロンプトまたは `CLAUDE.md` の指示は、Claude が何をしようとするかを形作りますが、Claude Code が許可する内容は変わりません。アクセスを付与または取り消すには、`/permissions`、ここで説明されているルール、permission mode、または *PreToolUse hook* (ページ 1059) を使用してください。

auto mode がセッションで利用可能な場合、ダイアログには auto mode classifier rules も含まれます。**Auto mode** タブを選択して、それらを表示します。

47.3 権限モード

Claude Code は、ツール呼び出しの承認方法を制御するいくつかの権限モードをサポートしています。権限モードを参照して、各モードをいつ使用するかを確認してください。セッションが開始される際のモードを変更するには、**設定ファイル** (ページ 711) で `defaultMode` を設定してください。セッションが開始されるモードでは、各プランの組み込みデフォルトと VS Code 拡張機能が読み込む内容について説明しています。

モード	説明
`default`	各ツールの最初の使用時に権限を促します。CLI、VS Code と JetBrains 拡張機能、およびデスクトップアプリでは Manual とラベル付けされており、Claude Code は `manual` をエイリアスとして受け入れます。ラベルとエイリアスには Claude Code v2.1.200 以降が必要です。デスクトップアプリのラベルは CLI バージョンに依存しません
`acceptEdits`	ファイル編集と一般的なファイルシステムコマンド（`mkdir`、`touch`、`mv`、`cp` など）を、作業ディレクトリまたは `additionalDirectories` 内のパスに対して自動的に受け入れます
`plan`	Claude はファイルを読み取り、読み取り専用シェルコマンドを実行して探索しますが、ソースファイルを編集しません。auto モードが利用可能で、分類器が承認したコマンドも実行されます。CLI および VS Code 拡張機能では Plan とラベル付けされています
`auto`	ルーチンプロンプトなしで実行されます。シェルコマンドやネットワークリクエストなどのアクションが実行される前に、バックグラウンド分類器がそれらがリクエストと一致することを確認します
`dontAsk`	その他の場合はプロンプトを表示するすべての呼び出しを自動的に拒否します。作業ディレクトリ内のファイル読み取りおよび承認が不要なその他のアクションは実行されます。`/permissions` または `permissions.allow` ルール経由で事前に承認されたツールも実行されます。`AskUserQuestion`、MCP ツール（*`requiresUserInteraction`* (ページ 404) とマークされたもの）、およびコネクタツール（**組織が *ask* に設定したもの** (ページ 399)）は、その設定が Claude Code に到達するセッションでは、許可していてもすべて拒否されます
`bypassPermissions`	権限プロンプトをスキップします。ただし、どのモードも自動承認しないアクションは除きます

警告:

bypassPermissions モードでは、Claude Code は権限プロンプトをスキップします。これには 保護されたパス（.git や .claude など）への書き込みも含まれます。クロスセッションメッセージングセーフガードは引き続き適用されます。このモードは、Claude Code が損害を引き起こせないコンテナや VM などの隔離された環境でのみ使用してください。

bypassPermissions または auto モードが使用されるのを防ぐには、任意の**設定ファイル** (ページ 711) で permissions.disableBypassPermissionsMode または permissions.disableAutoMode を "disable" に設定してください。これらは、オーバーライドできない**管理設定** (ページ 1063) で最も有用です。

47.4 権限ルール構文

権限ルールは、Tool または Tool(specifier) の形式に従います。スペシファイア内の括弧はリテラルであるため、括弧を含むコマンドまたはパスはエスケープが不要です。

47.4.1 ツールのすべての使用をマッチさせる

ツールのすべての使用をマッチさせるには、括弧なしでツール名を使用します。

ルール	効果
Bash	すべての Bash コマンドをマッチさせます
WebFetch	すべてのウェブフェッチリクエストをマッチさせます
Read	すべてのファイル読み取りをマッチさせます

Bash(*) は Bash と同等で、すべての Bash コマンドをマッチさせます。拒否ルールとして、両方の形式は Claude のコンテキストからツールを削除します。

47.4.2 細かい制御のためにスペシファイアを使用する

括弧内にスペシファイアを追加して、特定のツール使用をマッチさせます。

ルール	効果
Bash(npm run build)	正確なコマンド npm run build をマッチさせます
Read(./.env)	現在のディレクトリの .env ファイルを読み取ることをマッチさせます
WebFetch(domain:example.com)	example.com へのフェッチリクエストをマッチさせます

47.4.3 入力パラメータでマッチさせる

拒否ルールと確認ルールは、Tool(param:value) を使用して任意のビルトインツール上のトップレベル入力パラメータをマッチさせることができます。

MCP ツール上のパラメータをマッチさせるには、*--disallowedTools* (ページ 51) を使用して拒否ルール

を渡します。Claude Code が設定ファイルを読み込むとき、括弧を持つ mcp__ ルールをスキップします。Claude Code は、インタラクティブセッションが開始されるときに無効な設定ダイアログでスキップされたルールをリストし、claude doctor 出力に表示します。

パラメータルールは、Claude がそのパラメータをその正確な値に設定してツールを呼び出すときにマッチします。1 つのパラメータ値に対する許可ルールは、その呼び出しが全体的に安全であることを確立しないため、許可ルールは各ツール独自のスペシファイア構文を使用し続けます。これはツールが受け入れるスカラーパラメータで機能します。

ルール	マッチ
Agent(model:opus)	Opus モデルティアをリクエストする Agent 呼び出し
Agent(isolation:worktree)	git worktree をリクエストする Agent 呼び出し
Bash(run_in_background:true)	バックグラウンドで実行される Bash 呼び出し

パラメータマッチングは以下のルールに従います。

- パラメータ名は Agent ツール上の model など、ツールの入力の直接フィールドである必要があります。オブジェクトまたは配列内にネストされたフィールドはマッチ可能ではありません
- 各ルールは 1 つのパラメータに名前を付けます。model と isolation の両方でゲートするには、1 つのルールで組み合わせるのではなく、Agent(model:opus) と Agent(isolation:worktree) の 2 つのルールを記述します
- 値は * をワイルドカードとしてサポートし、任意の文字シーケンスにマッチするため、Agent(isolation:*) は任意の明示的な isolation 値にマッチします。* がない場合、マッチは正確です
- モデルが省略するパラメータは決してマッチしないため、Agent(model:*) は model が設定されていない呼び出しにはマッチしません
- 値は Claude が送信するリテラル入力と比較され、正規化の前です。Agent(model:opus) は別名 opus にマッチしますが、完全なモデル ID にはマッチしません
- Skill(skill:<name>) 拒否ルールは代わりに **スキルをそのいずれかの名前でマッチさせます** (ページ 305)。別名や表示名など
- *--verbose* (ページ 45) で実行して、各ツール呼び出しの正確なパラメータ名と値を確認してください
- コロンの周りのホワイトスペースは無視されます

ツールのプライマリコンテンツフィールドはこの方法ではマッチ可能ではありません。Bash と PowerShell の command、Read、Edit、Write の file_path、Grep と Glob の path、NotebookEdit の notebook_path、WebFetch の url です。Bash(command:rm *) のようなルールはコンパウンドコマンドでバイパス可能であるため、Claude Code はそれを無視し、スタートアップ警告を発行します。代わりに Bash(rm *)、Read(./path)、または WebFetch(domain:host) を使用してください。

47.4.4 ワイルドカードパターン

Bash ルールの * は、スペースを含む任意のテキストにマッチするため、1 つのルールはコマンドのファミリーをカバーします。* のないルールは 1 つの正確なコマンドにマッチします。

警告:

* をサブコマンドの後に配置します。`git log --oneline main` では、`git` はプログラムで `log` はサブコマンドです。サブコマンドはプログラムが何をするかを決定する単語です。Claude Code は最初の * の前のすべてをそのまま書かれたようにマッチさせるため、これらの単語がルールを制限するものです。`Bash(git log *)` は `git log` コマンドのみを許可し、`Bash(git *)` はすべての git コマンドを許可します。Claude Code は、`Bash(git * main)` のようなサブコマンドの前に * を持つ許可ルールについて スタートアップで警告 します。

Claude が質問なしで実行するコマンドを記述し、変わる部分を * に置き換えます。この設定により、Claude Code は npm スクリプトと git コミットを質問なしで実行し、`git push` で始まるコマンドを拒否します。`git -C . push` のように別の書き方をした push はマッチしません。*Bash* **ルールがマッチしないもの** (ページ 1048) を参照してください。

```
{
  "permissions": {
    "allow": [
      "Bash(npm run *)",
      "Bash(git commit *)"
    ],
    "deny": [
      "Bash(git push *)"
    ]
  }
}
```

* はルール内の任意の場所に配置できます。開始時、中間、または終了時です。各行はルール、マッチするコマンド、マッチしない近くのコマンドを示します。

記述内容	マッチ	マッチしない
`Bash(npm run build)`	`npm run build`	`npm run build --watch`
`Bash(npm run *)`	`npm run build`、`npm run test --watch`、`npm run`	`npm install`
`Bash(git log * main)`	`git log --oneline main`、`git log -5 main`、`git log --output=<file> main`	`git log main`、`git push origin main`
`Bash(git * main)`	`git merge main`、`git push origin main`、`git -c core.fsmonitor=<script> diff main`	`git log`
`Bash(* --version)`	`node --version`、`bash -c 'echo hi' --version`	`node -v`

次のページに続く

表 6 – 前のページからの続き

記述内容	マッチ	マッチしない
Bash(ls *)	ls -la、ls	lsof
Bash(ls*)	ls -la、lsof	
Bash(* --help *)	npm --help x	npm --help

3 つのマッチングルールがこれらの行を生成します。

- *** はその場所にあるテキストの代わりになります。** Bash(git * main) では、サブコマンドの代わりになるため、Claude Code はすべての git サブコマンドとその前のすべてのオプションにマッチします。これには -c が含まれます。これは git に名前を付けたプログラムを実行させます。Bash(* --version) では、* はプログラムの代わりになるため、任意のプログラムがマッチします。
- **末尾の * とその前のスペースは、ベアコマンドにもマッチします。** Bash(ls *) は ls にマッチし、Bash(git log *) は git log にマッチします。これは末尾の * がルールの唯一のワイルドカードである場合にのみ成立します。Bash(* --help *) は npm --help x にマッチしますが npm --help にはマッチしません。
- **末尾の * の前のスペースはルールの一部です。** Bash(ls *) は ls の後にスペースが必要であるため、lsof はマッチしません。Bash(ls*) にはスペースがないため、lsof にもマッチします。

:* サフィックスは末尾のワイルドカードを記述する同等の方法であるため、Bash(ls:*) は Bash(ls *) と同じコマンドをマッチさせます。

権限ダイアログは、コマンドプレフィックスに対して「はい、今後は聞かない」を選択すると、スペース区切り形式を書き込みます。:* 形式はパターンの末尾でのみ認識されます。Bash(git:* push) のようなパターンでは、コロンはリテラル文字として扱われ、git コマンドにはマッチしません。

47.4.5 ツール名ワイルドカード

拒否ルールと確認ルールは、ツール名の位置でもグロブパターンを受け入れます。パターンはツール名全体にマッチする必要があります。"*" はすべてのツールにマッチし、"mcp__*" はすべてのサーバー全体のすべての MCP ツールにマッチします。ベアネーム glob 拒否ルールでマッチしたツールは Claude のコンテキストから削除されます。これはベアツール名と同じです。EndConversation 例外を含みます。glob 拒否は他のツールが残っている間はそれを削除できず、glob 確認はそれに対してプロンプトを表示しません。この設定はすべての MCP ツールを拒否します。

```
{
  "permissions": {
    "deny": [
      "mcp__*"
    ]
  }
}
```

許可ルールは、リテラル mcp__<server>__ プレフィックスの後でのみツール名 glob を受け入れます。サーバーセグメントは glob フリーである必要があるため、ルールは設定した特定のサーバーに名前を付けます。mcp__puppeteer__* は puppeteer サーバーからのすべてのツールにマッチし、mcp__github__get_* は

その `get_` ツールにマッチします。`"*"`、`"B*"`、または `"mcp__*"` などのアンカーなし許可 glob は警告とともにスキップされ、自動承認されません。

ツール名がマッチしない既知のツールを持つ拒否ルールまたは確認ルールは、タイプミスをキャッチするためにスタートアップ警告を生成します。`_` または `*` を含むツール名はチェックから除外されます。また、Claude Code が削除したツール（`TaskOutput` など）の名前も除外されます。

トランスクリプトと権限ダイアログに表示されるツールのラベルは、その正規名と異なる場合があります。たとえば、トランスクリプトで `Stop Task` というラベルが付いているツールの正規名は `TaskStop` です。権限ルールと *hook* **マッチャー** (ページ 443) はラベルにはマッチしないため、`Stop Task` として記述されたルールはマッチしません。拒否ルールと確認ルールの場合、上記のスタートアップ警告がミスマッチをキャッチします。ツール参照 に記載されている正規名を使用してください。

47.5 ツール固有の権限ルール

47.5.1 Bash

Bash 権限ルールはコマンド全体をマッチさせ、`*` は任意のテキストを表します。**ワイルドカードパターン** (ページ 1045) は各ルール形状がマッチするコマンドと `*` の配置場所を示しています。このセクションの残りは、Claude Code が複合コマンドとラッパーをどのようにマッチさせるか、ルールがマッチしないもの、読み取り専用コマンド、およびリダイレクションについて説明しています。

複合コマンド

Tip:

Claude Code はシェルオペレータを認識しているため、`Bash(safe-cmd *)` のようなルールは、`safe-cmd && other-cmd` コマンドを実行する権限を与えません。認識されるコマンド区切り文字は `&&`、`||`、`;`、`|`、`|&`、`&`、および改行です。ルールは各サブコマンドを独立して個別にマッチさせる必要があります。

Deny ルールと ask ルールは、サブシェル内のネストされたコマンド、コマンド置換、または `for` ループなどの制御フロー本体内のコマンドを含む、マッチするサブコマンドが存在する場合に適用されます。`Bash(git clean *)` のような ask ルールは、auto モードでも、`cd /tmp && git clean -f` または `echo "$(git clean -f)"` に対してプロンプトを表示します。

`&&` または `||` の後に何もない場合（`npm test &&` など）、Claude Code はコマンドを解析不可能として扱い、allow ルールマッチングのためにサブコマンドに分割しないため、`Bash(npm *)` のようなルールはそれを承認しません。

「はい、今後は聞かない」で複合コマンドを承認すると、Claude Code は複合文字列全体の単一ルールではなく、承認が必要な各サブコマンドの個別ルールを保存します。たとえば、`git status && npm test` を承認すると、`npm test` のルールが保存されるため、将来の `npm test` 呼び出しは `&&` の前に何があるかに関係なく認識されます。`cd` をサブディレクトリに移動するようなサブコマンドは、そのパスの独自の Read ルールを生成します。単一の複合コマンドに対して最大 5 つのルールが保存される場合があります。

ラッパー

Bash ルールをマッチさせる前に、Claude Code は固定されたラッパーセットをストリップするため、`Bash(npm test *)` のようなルールは `timeout 30 npm test` もマッチさせます。ストリップされるラッパーは `timeout`、`time`、`nice`、`nohup`、`stdbuf`、およびシェルビルトイン `command` と `builtin`、および zsh の `noglob` です。各ラッパーは引数を実際のコマンドとして実行します。ストリップされない関連する 2 つの形式があります。コマンドを検索する `command -v` クエリ形式と、zsh の `nocorrect` です。

Claude Code はまた、既知の安全な環境変数の先頭の割り当てをストリップするため、`Bash(npm test *)` は `NODE_ENV=test npm test` にマッチします。Allow ルールは他の変数の割り当てを超えてマッチしません。Deny ルールまたは ask ルールは任意の先頭の割り当てを超えてマッチするため、`Bash(rm *)` を deny で使用すると、`FOO=bar rm -rf tmp/` にマッチします。

ベア `xargs` もストリップされるため、`Bash(grep *)` は `xargs grep pattern` にマッチします。ストリップは `xargs` にフラグがない場合にのみ適用されます。`xargs -n1 grep pattern` のような呼び出しは `xargs` コマンドとしてマッチされるため、内部コマンド用に記述されたルールはそれをカバーしません。

このラッパーリストは組み込まれており、設定不可能です。`direnv exec`、`devbox run`、`mise exec`、`npx`、`docker exec` などの開発環境ランナーはリストに含まれていません。これらのツールは引数をコマンドとして実行するため、`Bash(devbox run *)` のようなルールは `run` の後に続くものをマッチさせます。これには `devbox run rm -rf .` が含まれます。環境ランナー内での作業を承認するには、ランナーと内部コマンドの両方を含む特定のルールを記述します。例えば `Bash(devbox run npm test)`。許可する内部コマンドごとに 1 つのルールを追加します。

`watch`、`setsid`、`ionice`、`flock` などの Exec ラッパーは、`Bash(watch *)` のようなプレフィックスルールで自動承認することはできず、Manual モードでは常にプロンプトを表示します。同じことが `-exec` または `-delete` を使用する `find` にも適用されます。`Bash(find *)` ルールはこれらの形式をカバーしません。特定の呼び出しを承認するには、完全なコマンド文字列の正確一致ルールを記述します。

Bash ルールがマッチしないもの

Bash ルールは Claude が記述したコマンドテキストにマッチします。Claude Code が**複合コマンド** (ページ 1047) を分割し、**ラッパー** (ページ 1048) をストリップした後です。同じプログラムを別の形式で呼び出した場合、マッチしません。そのため、deny または ask ルールは Claude が通常生成する呼び出しをカバーし、プログラムの周りのセキュリティ境界ではありません。deny または ask のこれらのルールは最初の形式を停止し、他の形式は停止しません。

ルール	停止	停止しない
`Bash(curl *)`	`curl https://example.com`	`/usr/bin/curl https://example.com`、`sh -c 'curl https://example.com'`
`Bash(rm *)`	`rm -rf build/`	`/bin/rm -rf build/`、`bash -c 'rm -rf build/'`
`Bash(git push *)`	`git push origin main`	`git -C . push origin main`、`git -c push.default=current push origin main`、`git 'push' origin main`

最後の列のコマンドは、他のルールと権限モードによって決定されます。

コマンドテキストに依存しないファイルシステムとネットワークの強制には、**サンドボックス** (ページ 1069) を使用してください。実行前に独自のロジックで完全なコマンドテキストを検査するには、*PreToolUse* **フック** (ページ 1059) を使用してください。

読み取り専用コマンド

Claude Code は、Bash コマンドの組み込みセットを読み取り専用として認識し、`permissions.blockReadsOutsideWorkingDirectories` がフェンスするパスを除き、すべてのモードで権限プロンプトなしで実行します。セットには `ls`、`cat`、`echo`、`pwd`、`head`、`tail`、`grep`、`find`、`wc`、`which`、`diff`、`stat`、`du`、`cd`、および `git` の読み取り専用形式が含まれます。セットは設定不可能です。これらのコマンドの 1 つにプロンプトを要求するには、それに対して `ask` または `deny` ルールを追加します。auto モードでは、これらのコマンドは分類器のレビューを待つこともできます。分類器がアクションを評価する方法を参照してください。

`ls > out.txt` などのリダイレクトはターゲットのチェックを追加します。**リダイレクション** (ページ 1050) を参照してください。

すべてのフラグが読み取り専用であるコマンドに対しては、引用符なしのグロブパターンが許可されるため、`ls *.ts` および `wc -l src/*.py` はプロンプトなしで実行されます。

Manual モードでは、このセットのコマンドは以下の場合にもプロンプトを表示します。

- **書き込み可能なフラグを持つコマンドの引用符なしグロブ**：`find`、`sort`、`sed`、`git` などの書き込み可能または実行可能なフラグを持つコマンドは、グロブが `-delete` のようなフラグに展開される可能性があるため、引用符なしのグロブが存在する場合にプロンプトを表示します。
- **別のデーモンを指す** `docker`：読み取り専用形式の `docker` は、`-H`、`--context`、または Podman の `--url` と `--connection` などの異なるデーモンを選択するフラグを持つコマンドでプロンプトを表示します。
- **パス開放フラグを持つ** `file`：`file` は `-m`/`--magic-file` または `-f`/`--files-from` を渡す場合にプロンプトを表示します。これらのフラグにより、`file` はフラグの値で指定されたパスを開くためです。
- **Windows 上のネットワークパス**：`\\server\share\file` などのネットワーク（UNC）パスを含む引数を持つコマンドは、ネットワークパスへのアクセスが Windows 認証情報をそれが指すホストに送信する可能性があるため、プロンプトを表示します。同じチェックが PowerShell ツールコマンドに適用されます。
- **特殊シェル変数への書き込み**：`PATH` または `IFS` などの特定の特殊シェル変数を設定、設定解除、またはループするコマンドは、コマンドの残りが読み取り専用であっても、プロンプトを表示します。
- **解析できないコマンド**：Claude Code がコマンドを完全に解析できない場合、読み取り専用として扱う代わりに承認を求めます。10,000 文字を超えるコマンドは、解析が超過するため常にプロンプトを表示します。

作業ディレクトリまたは**追加ディレクトリ** (ページ 1060) 内のパスへの `cd` も読み取り専用です。`cd packages/api && ls` のような複合コマンドは、各部分が独立して適格である場合、プロンプトなしで実行されます。これらの組み合わせは、各部分が読み取り専用であっても、プロンプトを表示します。

- `cd` **と** `git`：`cd` が異なるディレクトリに変更される場合にプロンプトを表示します。新しいディレクトリで `git` を実行するとそのディレクトリのフックを実行できるためです。ターゲットが現在の作業ディレクトリに解決される `cd` は、操作なしであり、プロンプトをトリガーしません。

- **cd とリダイレクト**：Claude Code が cd 実行後にリダイレクトターゲットが解決するディレクトリを判定できない場合にプロンプトを表示します。唯一のリダイレクトターゲットが /dev/null であるコマンド（cd app; grep -r pattern . 2>/dev/null など）は、/dev/null は作業ディレクトリに依存しないため、プロンプトを表示しません。

警告:

コマンド引数を制約しようとする Bash 権限パターンは脆弱です。たとえば、Bash(curl http://github.com/ *) は curl を GitHub URL に制限することを意図していますが、次のようなバリエーションにはマッチしません。

- URL の前のオプション：curl -X GET http://github.com/...
- 異なるプロトコル：curl https://github.com/...
- リダイレクト：curl -L http://short.example.com/xyz（GitHub にリダイレクト）
- 変数：URL=http://github.com && curl $URL

より信頼性の高い URL フィルタリングについては、以下を検討してください。

- **Bash ネットワークツールを制限する**：deny ルールを使用して curl、wget などのコマンドをブロックし、許可されたドメインに対して WebFetch(domain:github.com) 権限で WebFetch ツールを使用します。deny ルールは、同じプログラムをパスで呼び出した場合や sh -c 内で呼び出した場合にはマッチしないため、制限を確実に保持する必要がある場合は**サンドボックスネットワーク許可リスト** (ページ 1087) と組み合わせてください。*Bash* **ルールがマッチしないもの** (ページ 1048) を参照してください。
- **PreToolUse フックを使用する**：Bash コマンドの URL を検証し、許可されていないドメインをブロックするフックを実装します。
- **CLAUDE.md ガイダンスを追加する**：CLAUDE.md で許可された curl パターンについて説明します。これは Claude が試みることを形作りますが、境界を強制しないため、上記のオプションの 1 つと組み合わせてください。

WebFetch のみを使用しても、ネットワークアクセスは防止されません。Bash が許可されている場合、Claude は curl、wget または他のツールを使用して任意の URL に到達できます。

リダイレクション

コマンドが出力または入力をリダイレクトする場合、Claude Code はリダイレクトターゲットをファイルルールに対してチェックします。Claude が直接そのファイルを書き込みまたは読み取ったかのようにです。

- **出力リダイレクト**：> file、>> file、または 2> file の場合、チェックは Edit allow ルールと deny ルール、保護されたパス、および**作業ディレクトリ** (ページ 1060) をカバーします。Bash(git commit *) のようなルールはコマンドを許可し、ターゲットは許可しません。~ で始まるターゲットまたはグロブ文字を含むターゲットは承認が必要です。
- **入力リダイレクト**：< file の場合、チェックは Read allow ルールと deny ルール、および作業ディレクトリをカバーします。作業ディレクトリの外のターゲットは、allow ルールがカバーしない限り承認が必要です。グロブパターンを含むターゲット、または同じコマンド内の cd に続く相対パスは、allow ルールがカバーしている場合でも承認が必要です。Claude Code は v2.1.257 以降で入力ターゲットをチェックします。

ターゲットの背後にファイルがない場合はチェックされません。/dev/null、2>&1 や <&3 などのファイルディスクリプタ形式、および here-docs と here-strings です。

Claude Code はまた、tee コマンドが書き込むファイルをチェックします。これには make | tee build.log などのパイプラインが含まれます。チェックは Edit allow ルールと deny ルール、保護されたパス、および**作業ディレクトリ** (ページ 1060) をカバーします。Bash(tee *) のような allow ルールは、作業ディレクトリの外の宛先をカバーしません。Claude Code は v2.1.269 以降で tee ターゲットをチェックします。

47.5.2 PowerShell

PowerShell 権限ルールは Bash ルールと同じ形状を使用しています。* を使用したワイルドカードは任意の位置でマッチし、:* サフィックスは末尾の * と同等であり、ベア PowerShell または PowerShell(*) はすべてのコマンドをマッチさせます。この設定により、Get-ChildItem および git commit コマンドが許可され、Remove-Item がブロックされます。

```
{
  "permissions": {
    "allow": [
      "PowerShell(Get-ChildItem *)",
      "PowerShell(git commit *)"
    ],
    "deny": [
      "PowerShell(Remove-Item *)"
    ]
  }
}
```

一般的なエイリアスはマッチング前に正規化されます。コマンドレット名用に記述されたルールはそのエイリアスもマッチさせるため、PowerShell(Get-ChildItem *) は gci、ls、dir もマッチさせます。マッチングは大文字と小文字を区別しません。

Claude Code は PowerShell AST を解析し、複合コマンド内の各コマンドを独立してチェックします。パイプオペレータ |、ステートメント区切り文字 ;、および PowerShell 7 以降のチェーンオペレータ && と || は複合コマンドをサブコマンドに分割します。複合コマンドが許可されるには、ルールがすべてのサブコマンドをマッチさせる必要があります。

47.5.3 Read と Edit

Claude のファイルツールがファイルまたはディレクトリを読み取るのをブロックするには、Read(./.env) または Read(./secrets/**) などのパスに対して Read deny ルールを追加します。機密ファイルを除外にはペースト可能な例があります。プロジェクトに .claudeignore ファイルがある場合、そのファイルは効果がないため、そのエントリを Read deny ルールに移してください。

Edit ルールはファイルを編集するすべての組み込みツールに適用されます。Claude は、Grep や Glob などのファイルを読み取るすべての組み込みツール、プロンプト内の @file メンション、および接続された *IDE* (ページ 200) が Claude と共有する選択およびオープンファイルコンテキストに Read ルールを適用するためにベストエフォートを試みます。

Read deny ルールは、同じパスの Edit と Write ツールもブロックします。これには新しいファイルの作成も含まれます。NotebookEdit はカバーされないため、ツールが変更できないパスに対して Edit deny ルールを

追加してください。チェックには編集時に Claude Code v2.1.208 以降が必要です。書き込み時には v2.1.228 以降が必要です。

Claude Code は `Edit(path)` と `Read(path)` ルールに対してのみファイル権限をチェックします。代わりに `Write`、`NotebookEdit`、`Glob`、またはレガシー `MultiEdit` ツール用にパスルールを記述する場合、Claude Code はルールを受け入れますが、それを参照することはなく、起動時に警告を表示します。ただし、`--allowedTools` で渡された `Glob` ルールは除きます。`Write(docs/**)`、`NotebookEdit(docs/**)`、または `MultiEdit(docs/**)` の代わりに `Edit(docs/**)` を使用し、`Glob(docs/**)` の代わりに `Read(docs/**)` を使用してください。Claude Code は `Write` の deny ルールなど、パスのないツール名ルールについては警告しません。それはどこでもツールレベルでそのルールをマッチさせます。v2.1.210 以降が必要です。

警告:

Read と Edit deny ルールは Claude の組み込みファイルツール、`cat`、`head`、`tail`、`sed` などの Claude Code が認識する Bash ファイルコマンド、および `> file` と `< file` などの Bash **リダイレクション** (ページ 1050) のターゲットに適用されます。これらは、そのファイルがあるディレクトリから実行する `grep -r pattern .` のような、ファイルを名前で指定せずに読み取るコマンドや、Python または Node スクリプトがファイルを自分で開くような、ファイルを間接的に読み書きする任意のサブプロセスには適用されません。パスへのすべてのプロセスのアクセスをブロックする OS レベルの強制については、**サンドボックスを有効にしてください** (ページ 1069)。

Read と Edit ルールの両方は、gitignore パターン構文を使用し、4 つの異なるパターンタイプがあります。単一セグメントディレクトリパターンの場合、マッチング深度はルールタイプにも依存し、このセクションの後半で説明されています。

パターン	意味	例	マッチ
`//path`	ファイルシステムルートからの絶対パス	`Read(//Users/alice/secrets/**)`	`/Users/alice/secrets/**`
`~/path`	ホームディレクトリからのパス	`Read(~/Documents/*.pdf)`	`/Users/alice/Documents/*.pdf`
`/path`	設定ソースからの相対パス	`Edit(/src/**/*.ts)`	プロジェクト設定の `<primary working directory>/src/**/*.ts`
`path` または `./path`	現在のディレクトリからの相対パス	`Read(*.env)`	`<cwd>/*.env`

警告:

`/Users/alice/file` のようなパターンは絶対パスではありません。単一の先頭スラッシュは設定ソースにアンカーされており、ファイルシステムルートではありません。絶対パスには `//Users/alice/file` を使用してください。

`/path` パターンは、それを定義する設定ソースに関連付けられたディレクトリにアンカーされるため、同じルールは配置場所に応じて異なる場所にマッチします。

ルール定義場所	/path の解決先
.claude/settings.json のプロジェクト設定	<primary working directory>/path
.claude/settings.local.json のローカル設定	<primary working directory>/path
~/.claude/settings.json のユーザー設定	~/.claude/path
--settings <file> で渡されたファイル	<directory of file>/path
CLI フラグまたはセッションルール	<primary working directory>/path

/permissions を通じて追加するルールは、保存先の設定ファイルの行に従います。

ローカル設定ルールは、v2.1.211 以降で Claude Code が **ファイルを保存** (ページ 1039) するリポジトリルートではなく、セッションの *primary working directory* (ページ 1060) にアンカーされます。リポジトリルートで開始されたセッションでは、2 つのディレクトリは同じです。worktree セッションでは、Edit(/src/**) などの共有ルールはそのワークツリー独自の src/ ディレクトリをマッチさせます。

Read(/secrets/**) のような deny ルールをユーザー設定で記述すると、プロジェクト内の secrets ディレクトリではなく、~/.claude/secrets/** をブロックします。すべてのプロジェクト内で適用されるルールをユーザー設定で記述するには、代わりに // 絶対パスまたは ~/ ホーム相対パスを使用してください。

Windows では、パスはマッチング前に POSIX 形式に正規化されます。C:\Users\alice は /c/Users/alice になるため、そのドライブ上の任意の場所の .env ファイルをマッチさせるには //c/**/.env を使用します。すべてのドライブにわたってマッチさせるには、//**/.env を使用します。

例：

- Edit(/docs/**)：<primary working directory>/docs/ での編集。/docs/ や <primary working directory>/.claude/docs/ ではありません
- Read(~/.zshrc)：ホームディレクトリの .zshrc の読み取り
- Edit(//tmp/scratch.txt)：絶対パス /tmp/scratch.txt の編集
- Read(src/**)：allow ルールとしては <current-directory>/src/ からの読み取りのみ。deny ルールまたは ask ルールとしては、現在のディレクトリの下の任意の深さにある src ディレクトリにマッチします

ルールはそのアンカーの下のファイルのみをマッチさせます。その範囲内で、マッチング深度はパターン形状に依存し、単一セグメントディレクトリパターンの場合、ルールタイプにも依存します。以下で説明されています。ベアファイル名は gitignore セマンティクスに従い、任意の深さでマッチするため、Read(.env) と Read(**/.env) は同等です。

Deny ルール	ブロック	ブロックしない
Read(.env) または Read(**/.env)	現在のディレクトリ以下の任意の .env	親ディレクトリまたは別のプロジェクト内の .env
Read(//**/.env)	ファイルシステム上の任意の場所の .env	なし。ルールはファイルシステムルートにアンカーされています

単一ディレクトリセグメント（src/** など）を持つ相対パターンは、ルールタイプに応じて異なる深さでマッチします。

- **Allow ルール**：`Edit(src/**)` は `<cwd>/src` とその下のファイルのみをマッチさせます。任意の深さでディレクトリ名を許可するには、`Edit(**/src/**)` を記述してください。
- **Deny と ask ルール**：`Read(secrets/**)` は現在のディレクトリの下の任意の深さで `secrets` という名前のディレクトリをマッチさせるため、ルールはネストされたコピーにも適用されます。

他のすべてのパターン形状は、すべてのルールタイプで同じ深さでマッチします。`Edit(/src/**)` と `Edit(src/components/**)` はそのアンカーされた場所でのみマッチし、`Edit(**/src/**)` は任意の深さでマッチします。

次の例は、トップレベルの `src/` ディレクトリと `vendor/` の下のネストされたコピーを持つプロジェクトに対する各パターン形状を示しています。

```
<current-directory>/
├── src/
│   └── app.ts
└── vendor/
    └── pkg/
        └── src/
            └── lib.js
```

ルール	src/app.ts をマッチ	vendor/pkg/src/lib.js をマッチ
Allow ルールとしての `Edit(src/**)`	はい	いいえ
Deny または ask ルールとしての `Edit(src/**)`	はい	はい
任意のルールタイプの `Edit(/src/**)`	はい	いいえ
任意のルールタイプの `Edit(**/src/**)`	はい	はい

注釈:

gitignore パターンでは、* は単一のパスセグメント内でマッチし、パターン内の任意の位置に表示できます。一方、** はディレクトリ全体でマッチします。

「はい、今後は聞かない」でファイルパスを承認すると、Claude Code は [、]、* などの gitignore パターン文字をそのパスでエスケープするため、生成されたルールは承認したリテラルパスのみをマッチさせます。自分で記述したルールはエスケープされません。v2.1.202 より前では、Claude Code はパスをエスケープなしで保存していたため、`[2024-06] Reports` という名前のディレクトリの生成されたルールは、独自のパスをマッチさせできなかったり、意図しない兄弟ディレクトリをマッチさせたりする可能性がありました。

パスに括弧をエスケープする必要はないため、`Edit(./Finance (2024)/**)` は `Finance (2024)` フォルダをそのまま記述してマッチさせます。

パスが gitignore パターンとして使用可能でない deny または ask ルールは、その正確なパスを保護します。使用可能でないパターンを持つ allow ルールは何も承認しません。

deny または ask パターンが！で始まる場合、gitignore 否定です。それは、その前にリストされた `path` ま

たは `./path` ルールから、それがマッチするパスを削除します。1 つの設定ファイルの `deny` リストで、`Read(*.env)` の後に `Read(!sample.env)` が続く場合、名前が `.env` で終わるすべてのファイルを任意の深さでブロックしますが、`sample.env` という名前のファイルは除きます。最初にリストされた `!` ルールは何も削除しません。

削除は同じソースからのルールにのみ到達します。プロジェクト設定または `--disallowedTools` の `Read(!.env)` は、管理設定または他の設定ファイルからの `Read(./.env)` deny をキャンセルしません。

2 つの制限は、`!` パターンが削除できるものを狭めます。

- Claude Code は、`!` の後に `/`、`~/`、または `//` が続く場合でも、`!` パターンを現在のディレクトリから相対的に読み取るため、パターンはこれらのプレフィックスの 1 つでアンカーされたルールに到達できません。`Read(!~/notes/public/**)` は `Read(~/notes/**)` から何も削除しません。
- 削除は、ルールが全体としてブロックするディレクトリ内のファイルを再度開くことはできません。`Read(secrets/**)` と `Read(!secrets/public/**)` では、Claude Code は `secrets/public` をそれ以外の `secrets` と一緒にブロックします。

シンボリックリンク

Claude がシンボリックリンクを通じてファイルパスにアクセスするとき、権限チェックは 2 つのパスをカバーします。Claude が要求したパスと、それが解決するファイルです。これは macOS、Linux、Windows 上のシンボリックリンク、および Windows 上のディレクトリジャンクションに適用されます。

ルールがシンボリックリンク付きパスをマッチさせる方法

Allow ルールと deny ルールは、要求されたパスと解決先のファイルを異なる方法で扱います。

- **Allow ルール**：要求されたパスと解決先のファイルの両方がマッチする場合にのみ適用されます。許可されたディレクトリ内のシンボリックリンクがそれの外を指している場合、ルールはマッチしません。
- **Deny ルール**：要求されたパスまたは解決先のファイルのいずれかがマッチする場合に適用されます。拒否されたファイルを指すシンボリックリンク自体が拒否されます。たとえば、`Read(./project/**)` が許可され、`Read(~/.ssh/**)` が拒否されている場合、`./project/key` にあるシンボリックリンクが `~/.ssh/id_rsa` を指している場合、ターゲットが allow ルールに失敗し、deny ルールにマッチするため、ブロックされます。

macOS と Linux では、シンボリックリンク付きディレクトリを通じて記述された deny または ask ルール（`//`、`~/`、または `/` パターン）は、そのディレクトリの実際の場所にも適用されます。たとえば macOS では、`/etc` が `/private/etc` に解決される場合、`Read(//etc/**)` は `/private/etc/hosts` もブロックします。v2.1.268 より前では、シンボリックリンク付きディレクトリを通じて記述された deny または ask ルールは、その実際の場所で指定されたパスに適用されませんでした。

Grep と Glob は `path` 引数が解決するディレクトリを検索します。Claude Code はそのディレクトリに `Read` deny ルールを適用します。

シンボリックリンクを通じた書き込み

Claude が編集または書き込みを要求するパスがそれ自体がシンボリックリンクである場合、Edit と Write ツールは 書き込みを拒否し、Claude をリンクのターゲットに向ける。

ディレクトリへのパスがシンボリックリンクである場合、またはファイルへのパスがシンボリックリンクを通じている場合、または Bash または PowerShell コマンドが書き込みを行う場合、書き込みはシンボリックリンクを通じて渡される可能性があります。これらの書き込みについて、何が起こるかは、書き込みが解決するファイルが *working directories* (ページ 1060) と protected paths に対してどこに位置するかに依存します。

- **作業ディレクトリの外に解決**：要求されたパスが作業ディレクトリ内にあり、解決先のファイルがそうでない場合、書き込みは acceptEdits モードで自動承認されません。auto モードでは、allow ルールが書き込みを承認しない限り、分類器が決定する代わりに、書き込みについてプロンプトが表示されます。プロンプトは、書き込みが解決するパスを指定します。
- **要求されたパスが指定しない保護されたパスに解決**：protected paths テーブルは各権限モードの結果を示しますが、テーブルが書き込みを分類器にルーティングする場合を除き、この書き込みはプロンプトを表示します。

解決できないパスまたは変更されるパス

Claude Code がディスク上のパスがどこに導くかを判定できない場合（たとえば、シンボリックリンクがループを形成する場合）、Read、Edit、Write ツールは 操作を拒否。

ツールが承認されたファイルを開くとき、パスが権限チェックが承認した場所にまだ解決されることを確認。

47.5.4 WebFetch

WebFetch ルールは `domain:` プレフィックスを使用し、リクエストされた URL のホスト名に対してマッチします。マッチングは大文字と小文字を区別せず、`*` ワイルドカードをサポートし、ルールとホスト名の両方から末尾の `.` をストリップするため、`example.com.` と `example.com` は同じものとして扱われます。

- `WebFetch(domain:example.com)` は `example.com` へのリクエストをマッチさせます
- `WebFetch(domain:*.example.com)` は `api.example.com` や `a.b.example.com` などの任意の深さのサブドメインをマッチさせますが、`example.com` 自体はマッチさせません
- `WebFetch(domain:*)` はすべてのドメインをマッチさせます。ベア `WebFetch` ルールと同じではありません。**すべてのフェッチを許可または拒否** (ページ 1057) を参照してください

先頭の `*.` またはベア `*` 以外の任意の位置では、ワイルドカードは 2 つのドット間のテキストのみをマッチさせます。`WebFetch(domain:example.*)` は `example.org` にマッチします。ここで `*` は `org` になりますが、`example.evil.com` にはマッチしません。ここで `*` は `evil.com` になり、ドットを越えます。これにより、末尾のワイルドカードが攻撃者が登録できるドメインをマッチさせるのを防ぎます。

WebFetch ルールのワイルドカードは、フェッチをマッチさせるために Claude Code v2.1.172 以降が必要です。

すべてのフェッチを許可または拒否

ベア WebFetch ルールは、"deny": ["WebFetch"] などの domain: 部分のないツール名です。それと WebFetch(domain:*) の両方はすべての URL をカバーしますが、Claude Code はそれらを異なる方法で適用し、domain: 形式のみがそのドメインをサンドボックスの **許可または拒否ドメインリスト** (ページ 1087) に追加します。そのセクションはサンドボックスが尊重するワイルドカード形式とそれを追加したバージョンをリストしています。

各行は、allow リストと deny リストでルールが何をするかを示しています。

ルール	allow で	deny で
WebFetch	Claude はプロンプトなしでフェッチします。サンドボックス化されたコマンドが到達できるホストを変更しません。	Claude Code は WebFetch ツールを削除するため、Claude はまったくフェッチできません。サンドボックス化されたコマンドが到達できるホストを変更しません。
WebFetch(domain:*)	Claude はプロンプトなしでフェッチし、サンドボックス化されたコマンドは任意のホストに到達できます。	Claude Code はツールを保持し、各フェッチを拒否し、サンドボックス化されたコマンドはホストに到達できません。

2 つの形式は artifacts（Artifact ツールが claude.ai に公開するページ）の読み取りについても異なります。ベア WebFetch deny または ask ルールはこれらの読み取りに適用されません。claude.ai または *.claudeusercontent.com コンテンツホストをカバーする domain: ルール（WebFetch(domain:claude.ai) または WebFetch(domain:*) など）は、各読み取りを拒否するか、その前にプロンプトを表示します。Artifact ルールも同じことを行います。

ルールが読み取りをブロックするとき、拒否はルールを指定します。v2.1.268 より前では、ベア WebFetch deny ルールはすべての artifact 読み取りをブロックし、ベア ask ルールはそれぞれの前にプロンプトを表示していました。

Claude がフェッチを自由に行えるようにしながら、サンドボックス許可リストをそのままにするには、ベア形式を使用してください。この settings.json はそれを行います。

```
{
  "permissions": {
    "allow": ["WebFetch"]
  }
}
```

Claude にページをフェッチするよう求めると、プロンプトなしでフェッチします。**サンドボックス化** (ページ 1069) された curl をサンドボックス許可リストの外のホストに対して実行するよう求めると、Claude Code はそのホストに対してプロンプトを表示します。ベア形式はホストを許可リストに追加しなかったためです。

auto モードでは、Claude は代わりにホストを分類器がレビューするコマンドの *per-command allowed domains* (ページ 1090) に指定します。

47.5.5 MCP

MCP ルールは Claude Code で設定されたサーバー名を使用し、オプションでそのサーバーからのツールの名前が続きます。

- `mcp__puppeteer` は `puppeteer` サーバーによって提供されるツールをマッチさせます
- `mcp__puppeteer__*` ワイルドカード構文を使用し、`puppeteer` サーバーからのすべてのツールもマッチさせます
- `mcp__puppeteer__puppeteer_navigate` は `puppeteer` サーバーによって提供される `puppeteer_navigate` ツールをマッチさせます

組織が *claude.ai* **コネクタ** (ページ 399) ツールを `ask` に設定している場合、そのツールの allow ルールは有効になりません。Claude Code は `auto` および `bypassPermissions` モードでも、すべての呼び出しでプロンプトを表示します。プロンプトを表示しない `dontAsk` モードでは、Claude Code は代わりに呼び出しを拒否します。Claude Code が自分で取得するコネクタからのツールは `mcp__claude_ai_<server>__<tool>` として表示されます。

Claude Desktop アプリの Cowork セッションでは、Claude は組み込み `Bash` ツールではなく Cowork の `mcp__workspace__bash` ツールを通じてシェルコマンドを実行し、Cowork は同様に `mcp__workspace__web_fetch` をウェブフェッチに提供します。Claude Code はまた、全体の `Bash` または `WebFetch` ツールを指す deny ルールをこれらの Cowork ツールに適用するため、管理された `Bash` deny ルールは Claude が Cowork でシェルコマンドを実行するのを停止します。Claude Code がそのような呼び出しをブロックするとき、メッセージは Cowork ツールを指定します。`Permission to use mcp__workspace__bash has been denied.` Allow ルールは引き継がれません。Claude Code は `Bash` allow ルールを `mcp__workspace__bash` に適用することはありません。

47.5.6 Agent（subagents）

`Agent(AgentName)` ルールを使用して、Claude が使用できる *subagents* (ページ 317) を制御します。

- `Agent(Explore)` は Explore subagent をマッチさせます
- `Agent(Plan)` は Plan subagent をマッチさせます
- `Agent(my-custom-agent)` は `my-custom-agent` という名前のカスタム subagent をマッチさせます

これらのルールを設定の `deny` 配列に追加するか、`--disallowedTools` CLI フラグを使用して特定のエージェントを無効にします。Explore エージェントを無効にするには：

```
{
  "permissions": {
    "deny": ["Agent(Explore)"]
  }
}
```

47.5.7 Cd

Cd ルールは、/cd コマンドがセッションを移動できるディレクトリを制御します。Cd はモデル呼び出し可能なツールではありません。Claude はそれを呼び出すことはできず、ルールは自分で /cd を実行する場合にのみ適用されます。

ベア Cd deny ルールは /cd を完全に無効にします。Cd(<path-pattern>) deny ルールはマッチするターゲットをブロックします。Deny ルールはターゲットのすべてのスペルをチェックします。これには、それが解決する各シンボリックリンクホップが含まれるため、1 つのパス用に記述されたルールは、それに解決するターゲットもブロックします。

任意の Cd allow ルールを追加すると、/cd をホワイトリストモードに切り替えます。解決されたターゲットディレクトリは、allow ルールの 1 つにマッチする必要があります。そうでない場合、/cd は拒否します。Cd ルールが設定されていない場合、/cd はデフォルト動作を保持し、見慣れないディレクトリを信頼するようにプロンプトを表示します。

パスパターンは *Read と Edit ルール* (ページ 1051) から //、~/、/ アンカーを共有しますが、マッチングはディレクトリパス全体にアンカーされます。gitignore スタイルではなく、* は正確に 1 つのパスセグメントをマッチさせ、** はセグメント全体でマッチさせます。末尾の /** はその名前付きルートもマッチさせます。

ルール	マッチ	マッチしない
Cd(~/code/*)	~/code/app	~/code/app/src、~/code
Cd(~/code/**)	~/code およびその下のディレクトリ	~/code の外のディレクトリ
Cd(**/node_modules)	現在のディレクトリ配下の任意の深さにある任意の node_modules ディレクトリ	node_modules/pkg

47.6 フックで権限を拡張する

Claude Code フック (ページ 411) は、実行時に権限評価を実行するカスタムシェルコマンドを登録する方法を提供します。Claude Code がツール呼び出しを行うと、PreToolUse フックは権限プロンプトの前に実行されます。ただし、EndConversation を除くすべてのツールに対して実行されます。フック出力はツール呼び出しを拒否し、プロンプトを強制し、またはプロンプトをスキップしてコールを続行させることができます。

PreToolUse フック決定は権限ルールをバイパスしません。Claude Code は deny ルールと ask ルールを、フックが何を返すかに関係なく評価します。マッチする deny ルールはコールをブロックし、マッチする ask ルールはフックが "allow" または "ask" を返した場合でもプロンプトを表示します。これは、**権限を管理する** (ページ 1041) で説明されている deny 優先の優先順位を保持し、管理設定で設定された deny ルールを含みます。

その優先順位は、設定ファイル内のフックとプラグインの hooks/hooks.json 内のフックをカバーしています。インストールした mod が tool.check を処理する場合、ルールと PreToolUse フックが決定した後に応答し、その応答はそれらの決定を置き換えることができます。

- **Ask ルール**: mod は ask ルールがプロンプトを表示するコールを承認できます

- `PreToolUse` **フックからのブロック**: mod はコールを承認できます。ただし、フックが管理設定にある場合を除きます
- **自動モード分類器**: 自動モードでは、mod が承認するコールは分類器チェックなしで実行されます
- **Deny ルール**: 管理設定を持つマシン上、または Team または Enterprise プランでサインインしている場合、deny ルールはデフォルトで mod より優先され、組織はそれを変更できます。その他の場所では、mod は deny ルールが拒否するコールを承認できます

mod を信頼するかどうかを決定するを参照するか、管理設定をデプロイする場合は組織の mod を管理するを参照してください。

`requiresUserInteraction` (ページ 404) でマークされた MCP ツールも、フックが `"allow"` を返した場合でもプロンプトを表示します。コネクタツール（**組織が *ask* に設定** (ページ 399)）も同様に、その設定が Claude Code に到達するセッションではプロンプトを表示します。

ブロッキングフックは allow ルールよりも優先されます。終了コード 2 で終了するフックは、権限ルールが評価される前にツール呼び出しを停止するため、allow ルールがコールを許可する場合でもブロックが適用されます。プロンプトなしですべての Bash コマンドを実行し、ブロックしたい少数のコマンドを除外するには、allow リストに `"Bash"` を追加し、それらの特定のコマンドを拒否する PreToolUse フックを登録します。適応できるフックスクリプトについては、**保護されたファイルへの編集をブロックする** (ページ 417) を参照してください。

47.7 作業ディレクトリ

デフォルトでは、Claude は起動されたディレクトリ内のファイルにアクセスできます。そのディレクトリはセッションの主要な作業ディレクトリであり、***/cd* でセッションを移動** (ページ 1061) するまで変わりません。このアクセスを拡張できます。

- **起動時**：`--add-dir <path>` CLI 引数を使用します
- **セッション中**：`/add-dir` コマンドを使用します
- **永続的な設定**：**設定ファイル** (ページ 711) の `additionalDirectories` に追加します

追加ディレクトリ内のファイルは、元の作業ディレクトリと同じ権限ルールに従います。プロンプトなしで読み取り可能になり、ファイル編集権限は現在の権限モードに従います。

ほとんどのネットワークパス（`\\server\share` などの UNC 共有など）は、作業ディレクトリとして追加できません。これは、ルックアップがそれが指す名前のホストに接続する可能性があるためです。Windows では、代わりに共有をドライブ文字にマップし、起動時に `--add-dir` でドライブを渡します。

`permissions.blockReadsOutsideWorkingDirectories` を設定して、ファイルツールがすべての権限モードで囲まれたパスを拒否するようにします。自動モードでは、Claude Code は Claude が作業ディレクトリの外側を読み取る最初の時点でこれをオンにするよう提案します。

macOS のバックグラウンドセッションでは、セッションホストは `~/Desktop`、`~/Documents`、`~/Downloads` などの保護されたフォルダへのアクセスをターミナルとは別に要求します。Claude がそこでファイルを読み取りまたは書き込む必要がある場合、`Operation not permitted` で読み取りが失敗する場合は、バックグラウンドセッションにフォルダアクセスを許可する方法を参照してください。

47.7.1 セッションを別のディレクトリに移動する

セッションを別の主要な作業ディレクトリに移動するには、現在のディレクトリの横に**ディレクトリを追加** (ページ 1060) するのではなく、`/cd <path>` を実行します。Claude Code は会話を保持し、新しいディレクトリの `CLAUDE.md` を読み込み、以前にそこで作業していない場合は**ワークスペースを信頼** (ページ 1064) するよう促します。その後、新しいディレクトリから `--resume` を実行すると、Claude Code は移動されたセッションを検出します。

移動するとすぐに、Claude Code は新しいディレクトリのプロジェクト設定を適用します。

- プロジェクト設定（権限ルールと *hooks* (ページ 443) を含む）
- *.mcp.json* **サーバー** (ページ 382)（起動時と同じ**サーバー承認** (ページ 370) の対象であり、それに登録した**ローカルスコープ** (ページ 381) MCP サーバー）
- 設定が有効にするプラグイン、その**スキル** (ページ 281)、およびその**サブエージェント** (ページ 317)
- `env` 値（前のディレクトリの設定から適用された環境変数の上に適用され、有効なままです）

Claude Code はまた、前のディレクトリのプロジェクトと**ローカルスコープ** (ページ 381) MCP サーバーを切断し、移動後に有効でなくなった**プラグイン** (ページ 378) のサーバーを切断します。前のディレクトリの設定ではなく、新しいディレクトリの設定から**追加ディレクトリ** (ページ 1060) を取得し、`--add-dir` または `/add-dir` で追加したディレクトリを保持します。移動が有効にする Hooks は、セッションが開始されたプロジェクトルートに設定された *`${CLAUDE_PROJECT_DIR}`* (ページ 463) を受け取ります。

新しいディレクトリがまだ信頼されていない場合、Claude Code は信頼プロンプトでディレクトリの設定が有効にするアロールール、追加ディレクトリ、hooks、およびヘルパーコマンドをリストアップするため、受け入れる前に確認できます。拒否した場合、セッションはそのままです。v2.1.246 より前では、`/cd` は再開するまで新しいディレクトリの設定、hooks、MCP サーバー、またはスキルを適用しませんでした。また、信頼プロンプトはディレクトリの設定が何を有効にするかをリストアップしませんでした。

Cd **権限ルール** (ページ 1059) で `/cd` ターゲットを制限または無効にします。

47.7.2 追加ディレクトリはファイルアクセスを許可し、設定ではありません

ディレクトリを追加すると、Claude がファイルを読み取りおよび編集できる場所が拡張されます。そのディレクトリを完全な設定ルートにはしません。ほとんどの `.claude/` 設定は追加ディレクトリから検出されませんが、いくつかのタイプは例外として読み込まれます。

これらの例外は、`--add-dir` フラグまたは `/add-dir` コマンドで追加されたディレクトリにのみ適用されます（Agent SDK がフラグを通じて追加するディレクトリを含む）。設定ファイルの `permissions.additionalDirectories` にリストされているディレクトリは、ファイルアクセスのみを許可し、以下の設定は読み込みません。

Agent SDK の TypeScript の `additionalDirectories` オプションと Python の `add_dirs` オプションは、TypeScript オプションが設定キーと同じ名前を共有していても、例外を受け取ります。SDK は各エントリを Claude Code に `--add-dir` として渡すため、これらのディレクトリはフラグで追加されたディレクトリのように動作します。任意のフラグで追加されたディレクトリからのスキル、コマンド、およびサブエージェントは、プロジェクト設定ソースを通じて読み込まれるため、CLI で *`--setting-sources`* (ページ 45) を使用するか SDK で `settingSources` を使用してそのソースを除外した場合は読み込まれず、**ベアモード** (ページ 696) はそれらの中のコマンドとサブエージェントをスキップします。

次の設定タイプは --add-dir ディレクトリから読み込まれます。

設定	--add-dir から読み込まれます
.claude/skills/ の**スキル** (ページ 275)	はい、ライブリロード付き
.claude/commands/ の**コマンドファイル** (ページ 279)	はい、ライブリロードなし。追加されたディレクトリとプロジェクトの両方が同じ名前のコマンドを定義する場合、Claude Code はプロジェクトのコマンドを実行します
.claude/agents/ の**サブエージェント** (ページ 317)	はい、ライブリロードなし
.claude/settings.json および .claude/settings.local.json の**設定** (ページ 711)	enabledPlugins および extraKnownMarketplaces キーのみ
CLAUDE.md (ページ 997) ファイル、.claude/rules/、および CLAUDE.local.md	CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1 が設定されている場合のみ。CLAUDE.local.md はさらに local 設定ソースが必要です。これはデフォルトで有効になっています

主要な作業ディレクトリ (ページ 1060) のサブディレクトリからスキル、コマンド、およびサブエージェントをセッション中に読み込むには、そのサブディレクトリのパスで /add-dir を実行します。Claude Code はサブディレクトリが既に読み取り可能であるため、プロンプトを表示したり作業ディレクトリを追加したりせずに、セッションの残りの間それらを読み込みます。これには Claude Code v2.1.257 以降が必要です。

Claude Code は現在の作業ディレクトリとその親、~/.claude/ のユーザーディレクトリ、および管理設定から出力スタイルを検出します。Hooks およびその他の .claude/settings.json キーは、現在の作業ディレクトリの .claude/ フォルダから親ディレクトリへのフォールバックなしで読み込まれ、ユーザーの ~/.claude/settings.json および管理設定と共に読み込まれます。.claude/settings.local.json は git リポジトリルートから読み込まれます。Claude Code がサブディレクトリで起動された場合でも、Windows など Claude Code が**リポジトリルートを使用しない** (ページ 714) 場合を除きます。v2.1.211 より前では、これも現在の作業ディレクトリからのみ読み込まれました。Agent SDK セッションはすべてのバージョンで作業ディレクトリから読み込みます。

その設定をプロジェクト全体で共有するには、次のいずれかのアプローチを使用します。

- **ユーザーレベルの設定**：~/.claude/agents/、~/.claude/output-styles/、または ~/.claude/settings.json にファイルを配置して、すべてのプロジェクトで利用可能にします
- **プラグイン**：設定をプラグインとしてパッケージ化および配布し、チームがインストールできるようにします
- **設定ディレクトリから起動する**：使用する .claude/ 設定を含むディレクトリから Claude Code を実行します

47.8 権限とサンドボックス化がどのように相互作用するか

権限と**サンドボックス化** (ページ 1069) は補完的なセキュリティレイヤーです。

- **権限**は Claude Code が使用できるツール、およびアクセスできるファイルやドメインを制御します。これらは Bash、Read、Edit、WebFetch、MCP、およびその他すべてのツールに適用されます。ただし、他のツールが残っている場合、deny ルールまたは ask ルールは `EndConversation` をブロックできません。
- **サンドボックス化**は OS レベルの強制を提供し、シェルコマンドのファイルシステムとネットワークアクセスを制限します。これは Bash、PowerShell、および Monitor コマンドとその子プロセスにのみ適用されます。

防御の多層化のために両方を使用してください。プロンプトインジェクションが Claude の意思決定をバイパスしても、サンドボックス制限は引き続き適用されます。サンドボックス設定と権限ルールからのパスとドメインは**最終的なサンドボックス構成にマージされます** (ページ 1091)。

サンドボックス化を有効にして `autoAllowBashIfSandboxed` をデフォルトの `true` のままにすると、サンドボックス化された Bash コマンドは、権限に bare `Bash` ask ルール、または**同等の *Bash(*)* フォーム** (ページ 1043) が含まれている場合でも、プロンプトなしで実行されます。サンドボックス境界がそのツール全体のプロンプトの代わりになります。

プランモードでは、Claude Code はこの置き換えをスキップします。ask ルールがない場合、**組み込みの読み取り専用コマンド** (ページ 1049) は引き続きプロンプトなしで実行され、その他のシェルコマンドはプランニング中に通常の権限フローを通過します。プランモードの詳細については、プランモードを参照して、Claude Code がそこでコマンドをどのようにゲートするかを確認してください。bare `Bash` ask ルールがある場合、サンドボックス化された読み取り専用コマンドを含むすべての Bash コマンドがプロンプトされます。これはサンドボックス化の外側と同じです。v2.1.212 より前では、置き換えはプランモードでも適用されていました。

これらのチェックは引き続き適用されます。

- `Bash(git push *)` のようなコンテンツスコープの ask ルールは引き続きプロンプトを強制します
- 明示的な deny ルールは引き続き適用されます
- 重要なパスをターゲットとする `rm` または `rmdir` コマンドは引き続き通常の権限フローを通過します

除外されたコマンドなど、サンドボックス化されて実行されないコマンドは、通常どおり bare `Bash` ask ルールを尊重します。この動作を変更するには、**サンドボックスモード** (ページ 1074) を参照してください。

47.9 管理設定

一元的な制御が必要な組織の場合、管理者はユーザーおよびプロジェクト設定でオーバーライドできない管理設定をデプロイします。ただし、いくつかの**セキュリティに関連するキー** (ページ 722) は例外です。管理設定をデプロイするでは、配信メカニズム、管理層内での優先順位、および管理設定のみが設定できるキーについて説明しています。

これらのキーの 1 つである `allowManagedPermissionRulesOnly` は、管理設定を権限ルールの唯一の設定ソースにします。そのエントリには、Claude Code がその後無視するすべてのソースが記載されています。

`disableBypassPermissionsMode` は通常、組織ポリシーを強制するために管理設定に配置されますが、任意のスコープから機能します。ユーザーは独自の設定で設定して、自分自身をバイパスモードからロックアウトできます。

47.10 設定の優先順位

権限ルールは、他のすべての Claude Code 設定と同じ**設定優先順位** (ページ 719) に従います。管理設定が最も高い優先度を持ちます。コマンドライン引数を含む他のレベルは、管理権限ルールをオーバーライドできません。

ツールがいずれかのレベルで拒否されている場合、他のレベルはそれを許可できません。たとえば、管理設定の deny は `--allowedTools` でオーバーライドできず、`--disallowedTools` は管理設定が定義する内容を超えて制限を追加できます。

設定スコープ全体でも同じことが当てはまります。ユーザー設定で権限が許可されており、プロジェクト設定で拒否されている場合、拒否ルールがそれをブロックします。逆も同様です。ユーザーレベルの deny がプロジェクトレベルの allow をブロックします。これは、任意のスコープからの deny ルールが allow ルールの前に評価されるためです。

この優先順位は、設定ファイルとコマンドライン引数の間のものです。deny ルールがインストールした mod に対して有効かどうかについては、**フックで権限を拡張する** (ページ 1059) を参照してください。

埋め込みホストは、SDK の `managedSettings` オプションを介して追加の管理ポリシーを提供できます。これには、管理者が `allowManaged*Only` ロックを設定していない限り、権限許可ルールが含まれます。Claude Desktop セッションにポリシーを配信するでは、埋め込み元ポリシーがいつ適用されるかについて説明しています。

47.11 プロジェクトの許可ルールとワークスペーストラスト

プロジェクトの `.claude/settings.json` 内の `permissions.allow` ルールと `permissions.additionalDirectories` エントリは機能を付与するため、Claude Code はそのフォルダの**ワークスペーストラストダイアログ** (ページ 1035) を受け入れた後にのみ適用します。ダイアログにはフォルダが付与するルールとディレクトリが表示されるため、受け入れる前に確認できます。`deny` ルールと `ask` ルールは制限のみを行うため、影響を受けません。

Claude Code はワークスペーストラストをそれを起動した場所に応じてキーとして保存します。

- リポジトリ内では、Claude Code は git リポジトリルートをキーとしてトラストを保存するため、トラストはリポジトリ全体をカバーします。ただし、サブモジュールなどのネストされた git リポジトリは除きます。worktree では、**保存されたルール** (ページ 1039) の場合と同様にメインチェックアウトのルートを使用します。
- リポジトリ外では、Claude Code はそれを起動したディレクトリをキーとしてトラストを保存し、そのディレクトリのサブディレクトリをカバーします。ただし、クローンなどのネストされた git リポジトリは除きます。カバーされた各サブディレクトリは、その親を信頼したフォルダとしてカウントされます。
- ホームディレクトリから起動した場合、Claude Code はトラストを現在のセッションのみ保持し、ディスクに書き込みません。**追加のセーフガード** (ページ 1035) に関する注記を参照してください。

Claude Code はインタラクティブセッションでのみトラストダイアログを表示します。`claude -p` 実行または SDK セッションはダイアログを表示しません。親フォルダを信頼してもこれらのルールにはカウントされないため、**フォルダを信頼する前に実行されるもの** (ページ 1066) は、これら 2 つの状況で Claude Code がどのリポジトリコンテンツを使用するかを説明しています。

バックグラウンドセッションを開始または再開する前に、Claude Code はセッションが実行されるディレクトリのワークスペーストラストもチェックします。信頼していないディレクトリのターミナルから `claude --bg` を実行した場合、トラストダイアログが最初に表示され、それを受け入れるとセッションが開始されます。スクリプト内など、ダイアログが表示できない場所では、コマンドは代わりに `Workspace not trusted` エラーで終了します。

47.11.1 ローカル設定ファイルがトラストを必要とする場合

`.claude/settings.local.json` は通常あなた自身のファイルであるため、Claude Code はトラストステップなしにその許可ルールと追加ディレクトリを適用します。ファイルが git で追跡されている場合、または `.claude` がシンボリックリンクである場合、Claude Code はそれをリポジトリ提供として扱い、フォルダを信頼するまでそのルールを保持します。

Claude Code は git を実行して 2 つを区別し、フォルダを信頼した後にのみ git を実行します。つまり、そのトラストダイアログを受け入れたか、トラストがそれに拡張される親ディレクトリを受け入れたか、`-p` または SDK セッションにいます。これはトラストが受け入れられたとしてカウントされます。それまでは、Claude Code を起動した場所がファイルのルールに何が起こるかを決定します。

- **設定ホーム内：** Claude Code は git を実行せずにそのフォルダの `.claude/settings.local.json` をすぐに適用します。設定ホームはホームディレクトリ、または `.claude` サブディレクトリを `CLAUDE_CONFIG_DIR` として設定したディレクトリです。その `CLAUDE_CONFIG_DIR` ディレクトリが git リポジトリ内にあり、Claude Code が**リポジトリルートでローカル設定を保持** (ページ 714) する場合、他の場所と同様にルールを保持します。
- **その他の場所：** Claude Code はプロジェクト設定と同様にファイルのルールを保持します。チェックが実行されると、Claude Code は追跡されていないファイルのルール、またはどの git リポジトリの外にあるディレクトリ内のファイルのルールを適用します。その正確なフォルダを信頼していなくても同様です。

注釈:

設定ホーム例外はトラストステップのみをスキップします。`~/.claude/settings.local.json` は依然として**ローカルスコープ** (ページ 712) であるため、Claude Code はホームディレクトリ自体で起動したセッションでのみそれを読み込み、すべてのプロジェクトでは読み込みません。すべてのプロジェクトに許可ルールを適用するには、代わりにユーザー設定に追加してください。`~/.claude/settings.json`、または `CLAUDE_CONFIG_DIR` が設定されている場合は `$CLAUDE_CONFIG_DIR/settings.json`。

バージョン 2.1.196 から 2.1.199 では、Claude Code は設定ホームと git リポジトリ外のファイルのルールを保持し、そこに `this workspace has not been trusted` 警告を出力していました。v2.1.207 より前では、Claude Code はダイアログを受け入れる前に追跡されていないファイルのルールを適用していました。

47.11.2 フォルダを信頼する前に実行されるもの

各行はリポジトリが提供できるコンテンツの 1 種類です。列は、フォルダ自体を信頼していない 2 つの状況です。親フォルダのみを信頼したか、そこで `claude -p` または SDK を実行しました。これはトラストダイアログを表示しません。親フォルダ列は**ネストされたリポジトリ** (ページ 1064) 内には適用されません。インタラクティブセッションでは Claude Code はそれのトラストダイアログを表示し、`claude -p` または SDK 実行はそこで `claude -p` 列に従います。

リポジトリが提供するもの	親フォルダのみを信頼した	`claude -p` または SDK、フォルダは信頼されていない
設定ファイル内の *Hooks* (ページ 443)、`env` ブロック、`apiKeyHelper` などのヘルパーコマンド、およびプロジェクトスキルの *hooks* (ページ 465) と *`allowed-tools`* (ページ 298)	使用	使用。ワークスペーストラストはどのセッションでもスキルの `allowed-tools` をゲートしません
`.claude/settings.json` 内の `permissions.allow` ルールと `additionalDirectories`	トラストダイアログを受け入れるまで使用されません。ダイアログは再度表示され、それらをリストします	使用されません。Claude Code は stderr に `this workspace has not been trusted` 警告を出力します
プロジェクト *subagent* (ページ 341) のフロントマターフック、プロジェクト `@skills-dir` プラグイン、およびリポジトリまたは `--add-dir` ディレクトリからの `extraKnownMarketplaces` エントリ	使用されず、ダイアログは提供されません	使用されません
リポジトリまたは `--add-dir` ディレクトリからの subagent のフロントマター内のインライン *mcpServers* (ページ 334)	使用されず、ダイアログは提供されません	使用されません
`.mcp.json` 内のサーバー。リポジトリが**独自の設定で承認** (ページ 370) するものを含む	Claude Code は接続する前にあなたに尋ねます。リポジトリ独自の承認はカウントされません	承認されているかどうかに関わらず接続されます。SDK はセッティングソースがプロジェクト設定を含む場合にのみそれらを読み込みます。同じフォルダの `claude mcp list` はそのようなサーバーを保留中として報告します

次のページに続く

表 15 – 前のページからの続き

リポジトリが提供するもの	親フォルダのみを信頼した	claude -p または SDK、フォルダは信頼されていない
.mcp.json 内のサーバー上の *headersHelper* (ページ 394)	トラストダイアログを受け入れるまで実行されません。ダイアログは再度表示され、ヘルパーが宣言されている場所を名前で指定します。Claude Code はそれまでサーバーを静的 headers のみで接続します	実行されません。Claude Code はサーバーを静的 headers のみで接続し、サーバーごとに stderr に headersHelper not run 行を出力します

このフォルダを信頼する必要がある行については、手動で信頼してください。~/.claude.json で projects["<path>"].hasTrustDialogAccepted を true に設定します。<path> はリポジトリルート、またはリポジトリ外のフォルダ自体です。Claude Code はスキップされた subagent フックまたはインライン MCP サーバーのデバッグログ行、スキップされた許可ルールの stderr 警告、およびスキップされたヘルパーの headersHelper not run 行に正確なキーを出力します。

あなたが書いていないリポジトリで claude -p を実行する前に、マシンで実行される可能性があるものを決定してください。

- --setting-sources user を渡すか、SDK の settingSources をプロジェクト設定なしで設定して、Claude Code がプロジェクトの設定ファイルも .mcp.json も読み込まないようにします
- *--bare* (ページ 696) で起動して、Claude Code がプロジェクトから hooks、skills、カスタムコマンド、subagents、プラグイン、または .mcp.json サーバーを読み込まないようにします。プロジェクトの env ブロックと awsAuthRefresh などのヘルパーはその設定ファイルで依然として適用され、Claude Code は apiKeyHelper を --settings からのみ読み込みます
- --settings '{"disableAllHooks": true}' を渡して、その実行の ***hooks* をオフにします** (ページ 466)。ユーザー設定のみで設定するのは十分ではありません。リポジトリのプロジェクト設定があなたのものより優先され、それを false に戻すことができるためです
- disabledMcpjsonServers エントリをすべてのセッションタイプで名前で .mcp.json サーバーを拒否するために追加します

47.12 設定例

このリポジトリには、一般的なデプロイメントシナリオのスターター設定が含まれています。これらを出発点として使用し、ニーズに合わせて調整してください。

47.13 関連項目

- すべての設定：権限キーを含むすべての設定キー
- auto モードを設定する：auto モード分類器に組織が信頼するインフラストラクチャを伝えます
- **サンドボックス** (ページ 1069)：Bash コマンドの OS レベルのファイルシステムとネットワーク分離
- **認証** (ページ 851)：Claude Code へのユーザーアクセスを設定します

- **セキュリティ** (ページ 1033)：セキュリティ保護とベストプラクティス
- **フック** (ページ 411)：ワークフローを自動化し、権限評価を拡張します

第48章　サンドボックス化されたBashツールを設定する

組み込みのサンドボックスを使用して、Claude Code のシェルコマンドがアクセスできるファイルとネットワークホストを制限します。サンドボックスをオンにし、境界を設定し、それによって生じる問題を解決します。

Bash サンドボックスは、Claude がユーザーのマシン上で実行するシェルコマンドの周囲に、オペレーティングシステムが適用する境界です。これらのコマンドがアクセスできるファイルとネットワークドメインを設定すると、その制限は Bash、PowerShell、Monitor の各コマンドと、それらが起動するプロセスに適用されます。コマンドの実行中にオペレーティングシステムが制限を適用するため、Claude Code はサンドボックス化されたコマンドを 1 つずつ承認を求めることなく**実行できます** (ページ 1074)。

サンドボックスの対象はシェルコマンドのみです。Claude のファイルツール、MCP サーバー、フックは**サンドボックスの外で実行されます** (ページ 1070)。

サンドボックスは macOS、Linux、WSL2 で動作します。ネイティブ Windows では、Claude Code はコマンドをサンドボックス化せずに実行します。Windows マシンでサンドボックスを使用するには、WSL2 ディストリビューション内で Claude Code を実行してください。

注釈:

このページでは、ユーザー自身のマシン上のシェルコマンドを囲むサンドボックスについて説明します。関連する内容は他のページで説明しています。

- クラウドセッションがどのように分離されるかについては、**セキュリティと分離** (ページ 235) を参照してください
- dev container、カスタムコンテナ、仮想マシンなど、他の分離方法を比較するには、サンドボックス環境を参照してください
- Bash 以外のツールの権限プロンプトを減らすには、権限モードを参照してください

48.1 サンドボックスが制限する内容

サンドボックスがオンの間、Claude が実行するシェルコマンドはその境界内で起動し、それらのコマンドが起動するプロセスも同様に境界内で起動します。サンドボックスはデフォルトでオフです。オンにするには、**はじめに** (ページ 1071) で示すようにセッションで `/sandbox` を実行するか、`~/.claude/settings.json` などの**設定ファイル** (ページ 711) で `sandbox.enabled` を `true` に設定します。

次の表は、サンドボックス化されたコマンドがデフォルトでアクセスできる範囲と、各デフォルトを変更する設定を示しています。

アクセス	デフォルト	変更方法
書き込み	作業ディレクトリ、ユーザーごとの一時ディレクトリ、および**追加したディレクトリ** (ページ 1061)。**保護されたパス** (ページ 1087) は書き込みが拒否されたままです	`filesystem.allowWrite`、`filesystem.denyWrite`
読み取り	`~/.ssh` や `~/.aws/credentials` などの認証情報ファイルを含む、マシンの大部分	`filesystem.denyRead`、*`credentials`* (ページ 1082)
ネットワーク	外部への直接のルートはありません。接続はマシン上のプロキシを経由し、プロキシが各ホストを許可ドメインと照合します。許可ドメインは最初は空です。**その他のホストの扱い** (ページ 1089) は権限モードによって決まります	`network.allowedDomains`、`network.deniedDomains`
環境変数	Claude Code から継承され、その環境内のシークレットも含まれます	*`credentials`* (ページ 1082)、`CLAUDE_CODE_SUBPROCESS_ENV_SCRUB`

Claude Code は、オープンソースの `@anthropic-ai/sandbox-runtime` パッケージを基にサンドボックスをビルドしています。

48.1.1 サンドボックスの外で実行されるもの

サンドボックスはシェルコマンドをラップします。次のツールとプロセスはサンドボックスの外で実行されます。

- **組み込みのファイルツールと Web ツール**：Read、Edit、Write、WebFetch、WebSearch などのツールは、代わりに**権限ルール** (ページ 1039) に従います。`denyRead` エントリは Read ツールを止めず、`allowedDomains` は WebFetch を制限しません
- **Claude Code が起動するその他のプロセス**：コマンド**フック** (ページ 443)、ローカルの *MCP* **サーバー** (ページ 363)、プラグインモニター、LSP サーバー、および**ステータスライン** (ページ 785) コマンドや `apiKeyHelper` などのヘルパーコマンドは、ユーザーの完全なアクセス権で実行されます

設定によっては、一部のシェルコマンドもサンドボックスの外で実行されます。

- **自分で入力したコマンド**： シェルモードのプロンプト <interactive-mode--shell-mode-with-prefix>で入力したコマンドは、ほとんどのセッションでサンドボックス化されずに実行されます。入力したコマンドがサンドボックス内で実行されるセッションについては、**厳格サンドボックスモード** (ページ 1076) を参照してください
- **除外されたコマンド**：*`excludedCommands`* (ページ 1078) に一致するコマンドは、サンドボックス化されずに実行されます
- **サンドボックス外での再試行**：Claude は、通常はサンドボックス内でコマンドが失敗した後に、**そのコマンドをサンドボックス外で実行するよう求める** (ページ 1075) ことがあります

このセクションで挙げたツール、プロセス、コマンドを 1 つの境界の内側に置くには、Claude Code プロセス自体をコンテナ、仮想マシン、またはサンドボックスランタイム内で実行します。

48.2 はじめに

サンドボックスは Claude Code に組み込まれています。インストールが必要なものはプラットフォームによって異なります。

- **macOS**：サンドボックス化には組み込みの Seatbelt フレームワークが使われるため、そのまま手順に進めます
- **Linux と WSL2**：サンドボックスは `bubblewrap` と `socat` に依存しています。これらについては *Linux と WSL2* **のセットアップ** (ページ 1072) で説明しています。まだインストールしていなくても、`/sandbox` から始められます。そのパネルに不足しているものが表示されるためです

1. **/sandbox を実行する**

   Claude Code のセッションを開始し、`/sandbox` コマンドを実行します。

   ```
   /sandbox
   ```

   これによりサンドボックスパネルが開きます。パネルには 3 つのタブがあり、Linux でオプションの seccomp フィルターが不足している場合はさらに Dependencies タブが表示されます。

   - **Mode**：サンドボックス化されたコマンドの承認方法を選択します。次のステップで説明します
   - **Overrides**：サンドボックス内で失敗したコマンドを、サンドボックス外での実行にフォールバックできるかどうかを選択します。これは `allowUnsandboxedCommands` 設定です
   - **Config**：解決済みのサンドボックス設定を表示します

   パネルに Dependencies タブしか表示されない場合は、必須パッケージが不足しています。*Linux と WSL2* **のセットアップ** (ページ 1072) の説明に従ってインストールし、Claude Code を再起動してから、もう一度 `/sandbox` を実行してください。

2. **モードを選択する**

   Mode タブで、auto-allow または regular permissions を選択します。auto-allow ではサンドボックス化されたコマンドがプロンプトなしで実行され、regular permissions ではコマンドがサンドボックス化されていても通常の権限プロンプトが維持されます。auto-allow モードでもプロンプトが表示されるコマンドについては、**サンドボックスモード** (ページ 1074) を参照してください。

3. **Bash コマンドを実行する**

   ビルドやテストスイートなどのコマンドを実行するよう Claude に依頼します。デフォルトでは、サンドボックス内のコマンドは、作業ディレクトリ、ユーザーごとの一時ディレクトリ、および `--add-dir`、`/add-dir`、`permissions.additionalDirectories` で**追加したディレクトリ** (ページ 1061) に書き込めます。

   コマンドが新しいネットワークドメインを初めて必要とするとき、Claude Code は承認を求めるプロンプトを表示します。auto モードでは代わりに、コマンドが必要とするホストを Claude が**コマンド自体に** (ページ 1090) 記載し、分類器がそれをコマンドとともに審査します。

   サンドボックスが許可する範囲を広げたり狭めたりするには、**サンドボックス化の設定** (ページ 1077) を参照してください。

   コンテナ内でサンドボックス化されたコマンドが `Operation not permitted` で失敗する場合は、**コンテナ内で** *Bubblewrap* **の起動に失敗する** (ページ 1099) を参照してください。

パネルでモードを選択すると、Claude Code はそれをプロジェクトのローカル設定 `.claude/settings.local.json` に保存します。この設定は現在のプロジェクトに適用されます。Claude Code はそこに設定を保存するときに、そのファイルをグローバル gitignore に追加します。すべてのプロジェクトでサンドボックスを有効にするには、ユーザー設定 `~/.claude/settings.json` で `sandbox.enabled` を `true` に設定します。組織内のすべての開発者にサンドボックス化を強制するには、**管理設定** (ページ 1093) を使用します。

設定ファイルに書き込まずに 1 つのセッションだけサンドボックスを変更するには、*`--settings`* (ページ 717) を付けて Claude Code を起動します。たとえば次のコマンドは、ブロックされたコマンドを Claude がサンドボックス外で再試行できない、サンドボックス化されたセッションを開始します。

```
claude --settings '{"sandbox": {"enabled": true, "allowUnsandboxedCommands": false}}'
```

警告:

デフォルトでは、依存関係が不足している、またはプラットフォームがサポートされていないためにサンドボックスを起動できない場合、Claude Code はサンドボックス化せずにコマンドを実行します。代わりに起動時に Claude Code を終了させるには、`sandbox.failIfUnavailable` を `true` に設定します。サンドボックス化をセキュリティゲートとして必須とする管理されたデプロイでは、この設定を使用できます。

48.2.1 コマンドがサンドボックス内で実行されることを確認する

サンドボックスが機能していることを確認するには、表の各行を実行するよう Claude に依頼します。 プロンプト <sandboxing--what-runs-outside-the-sandbox>で入力したものは通常サンドボックス外で実行されるため、自分で行を入力してもテストにはなりません。

コマンド	サンドボックス内での結果
`touch ~/sandbox-probe`	macOS では `Operation not permitted`、Linux と WSL2 では `Read-only file system` で失敗します
`curl --noproxy '*' https://example.com`	コマンドにはサンドボックスプロキシを迂回する経路がないため、`Could not resolve host` で失敗します

失敗したコマンドをサンドボックス外で再試行するよう Claude が求めてきた場合は、再試行を拒否してください。`touch` が成功し、ホームディレクトリがサンドボックスでコマンドの書き込みが許可されているディレクトリに含まれていない場合は、`~/sandbox-probe` を削除してください。その後、`/sandbox` を実行して、サンドボックスがオンになっていることと、その依存関係がインストールされていることを確認します。

48.2.2 Linux と WSL2 のセットアップ

Linux と WSL2 では、サンドボックスは次のパッケージに依存しています。

- `bubblewrap`：ファイルシステムの分離を強制する、特権不要のサンドボックス化ツール
- `socat`：ネットワークトラフィックをサンドボックスプロキシ経由でルーティングするために使われるリレー

ディストリビューションのパッケージマネージャーでインストールします。

Ubuntu/Debian

```
sudo apt-get install bubblewrap socat
```

Fedora

```
sudo dnf install bubblewrap socat
```

依存関係が不足している場合、/sandbox の Dependencies タブには、ripgrep、bubblewrap、socat、seccomp フィルターのうちプラットフォームに不足しているものが一覧表示されます。インストールして Claude Code を再起動した後にこのタブが表示されなければ、すべての依存関係がそろっています。

Ripgrep はネイティブの Claude Code バイナリに同梱されています。seccomp フィルターはオプションで、Unix ドメインソケットのブロックを追加します。不足している場合は npm install -g @anthropic-ai/sandbox-runtime でインストールしてください。

必須の依存関係が不足している場合、インストールするまで Dependencies タブだけが表示されます。オプションの seccomp フィルターだけが不足している場合は、Dependencies タブが他のタブと並んで表示されます。依存関係のチェックは起動時に実行されるため、/sandbox にパッケージを検出させるには、インストール後に Claude Code を再起動してください。

Ubuntu 24.04 以降：bubblewrap によるユーザー名前空間の作成を許可する

Ubuntu 24.04 以降では、デフォルトの AppArmor ポリシーにより、bubblewrap が分離に必要なユーザー名前空間を作成できません。

WSL2 内も含め、環境がこの制限を強制しているかどうかを確認するには、sysctl kernel.apparmor_restrict_unprivileged_userns を実行します。コマンドが 0 を返す場合は、このステップをスキップしてください。No such file or directory エラーが表示される場合は、キーが存在しないため、このステップをスキップできます。1 を返す場合は、bwrap にこの機能を付与する AppArmor プロファイルを追加します。

```
sudo tee /etc/apparmor.d/bwrap > /dev/null <<'EOF'
abi <abi/4.0>,
include <tunables/global>

profile bwrap /usr/bin/bwrap flags=(unconfined) {
  userns,
  include if exists <local/bwrap>
}
EOF
```

このプロファイルは bwrap 自体にのみ適用され、サンドボックス内で bwrap が実行するコマンドには適用されません。適用するには AppArmor を再読み込みします。

```
sudo systemctl reload apparmor
```

WSL2 に関する注意事項

PowerShell から wsl -l -v を実行して WSL のバージョンを確認します。Sandboxing requires

`WSL2` と表示される場合、ディストリビューションは WSL1 で実行されています。WSL2 にアップグレードするか、サンドボックス化せずに Claude Code を実行してください。

WSL2 では、`cmd.exe`、`powershell.exe`、または `/mnt/c/` 配下のものなど Windows バイナリの起動を、WSL が Unix ソケット経由で Windows ホストに引き渡します。そのため、サンドボックス化されたコマンドがそれらを起動できるかどうかは、サンドボックスの Unix ソケット設定に従います。そもそもソケットをブロックするには、オプションの seccomp フィルターがインストールされている必要があります。これらの起動を許可するには、`allowAllUnixSockets` を設定します。これにより、サンドボックス化されたコマンドにすべての Unix ソケットが開放されます。

48.2.3 サンドボックスモード

Claude Code には 2 つのサンドボックスモードがあります。どちらのモードでも、サンドボックスは同じファイルシステムとネットワークの制限を強制します。違いは、サンドボックス化されたコマンドが自動承認されるか、明示的な権限が必要かという点だけです。

auto-allow モード

コマンドがサンドボックス内で実行される場合、Claude Code はプロンプトなしでそのコマンドを自動的に承認します。コマンドが *`excludedCommands`* (ページ 1078) に一致するため、または Claude が**サンドボックス外で再試行する** (ページ 1075) ためにサンドボックス外で実行される場合、そのコマンドは通常の**権限フロー** (ページ 1039) を経由します。

許可していないホストに接続するサンドボックス化されたコマンドは、サンドボックス内にとどまります。接続を通すかどうかを誰が決めるかについては、**許可されたドメイン外のホスト** (ページ 1089) で説明しています。

auto-allow モードでも、次の点は引き続き適用されます。

- 明示的な**拒否ルール** (ページ 1039) は常に尊重されます
- 重要なパスを対象とする `rm` または `rmdir` コマンドは、引き続き通常の権限フローを経由します
- `Bash(git push *)` のような内容を限定した**確認ルール** (ページ 1039) は、サンドボックス化されたコマンドであっても引き続きプロンプトを強制します
- 単独の `Bash` 確認ルール、またはそれと同等の `Bash(*)` 形式は、サンドボックス化されて実行されるコマンドではスキップされます。通常の権限フローにフォールバックするコマンドには引き続き適用されます。plan モードでは、このルールはスキップされず、読み取り専用のものも含め、サンドボックス化されたコマンドに対してもプロンプトを表示します

注釈:

auto-allow モードは権限モードの設定とは独立して動作しますが、例外が 3 つあります。plan モード、**コマンドごとの許可ドメイン** (ページ 1090) を持つ auto モードのコマンド、そして auto モードでのサンドボックス化されたコマンドに対するサーバー側の分類器による審査です。「accept edits」モードでなくても、auto-allow が有効な場合、サンドボックス化された Bash コマンドは自動的に実行されます。つまり、ファイル編集ツールであればプロンプトが表示される Manual モードでも、サンドボックスの境界内でファイルを変更する Bash コマンドはプロンプトなしで実行されます。
plan モードでは、auto-allow によって承認の範囲は広がりません。計画中に Claude Code がコマンドをどのように制御するかについては、plan モードを参照してください。

regular permissions モード

すべての Bash コマンドは、サンドボックス化されている場合でも通常の権限フローを経由します。より細かく制御できますが、より多くの承認が必要になります。

サンドボックス外での再試行という抜け道

サンドボックス外での再試行は、サンドボックスと互換性のないツールなど、サンドボックス内で失敗するコマンドのための抜け道です。サンドボックスがネットワーク接続をブロックすると、Claude Code はコマンドの結果の中で拒否されたホストを示すため、Claude は何がブロックされたかを把握できます。Claude は失敗を分析し、`dangerouslyDisableSandbox` パラメーターを付けてコマンドを再試行することがあります。

再試行されたコマンドはサンドボックス外で実行されます。インタラクティブなターミナルセッションでは、誰が承認するかは権限モードによって異なります。

- `bypassPermissions` **モード**：再試行はプロンプトなしで実行されます
- **Manual モードと** `acceptEdits` **モード**：「Bash command (unsandboxed)」というタイトルのプロンプトが表示されます
- auto モード：別の分類器モデルが基になるコマンドを評価します
- `dontAsk` **モード**：Claude Code は再試行を拒否します
- **plan モード**：計画中に Claude Code がコマンドをどのように制御するかを参照してください

次のルールと設定によって、再試行を誰が承認するかが変わります。

- **一致する許可ルール**：`Bash(curl *)` などの許可ルールがコマンドに一致する場合、そのルールは再試行も承認するため、コマンドはプロンプトなしでサンドボックス外で実行されます
- **パラメーターに対する確認ルール**：Bash の再試行時にプロンプトを表示させるには、`Bash(dangerouslyDisableSandbox:true)` に対する**確認ルール** (ページ 1043) を追加します。auto モードと `bypassPermissions` モードでもプロンプトが表示され、このルールは一致する許可ルールより優先されます
- `permissions.blockReadsOutsideWorkingDirectories`：これがオンの間にプロンプトが表示される再試行については、どのモードでも自動承認されないアクションで説明しています

strict sandbox モードで再試行をオフにする

サンドボックス設定で "allowUnsandboxedCommands": false を設定すると、サンドボックス外での再試行を無効にできます。再試行が無効になると、Claude Code は dangerouslyDisableSandbox パラメーターを無視します。これにより、サンドボックスが動作している間、Claude が実行するコマンドは excludedCommands のエントリに一致しない限りサンドボックス化されます。サンドボックスを起動できないときに Claude Code がコマンドをサンドボックス外で実行しないようにするには、failIfUnavailable も設定します。/sandbox の **Overrides** タブでは、この設定は **Strict sandbox mode** として表示されます。

ユーザー設定、--settings、または管理設定での false は、プロジェクトの設定で true が設定されていても維持されます。ユーザー設定での false によってサンドボックスが管理者必須になることはないため、プロジェクトのその他のサンドボックス設定は引き続き適用されます。v2.1.285 より前は、プロジェクトの true がユーザー設定の false を上書きしていました。

ユーザーまたは管理者が管理設定または --settings フラグで再試行を無効にすると、サンドボックスは管理者必須になります。その場合、Claude Code は、excludedCommands のエントリも含め、リポジトリのファイル内にあるサンドボックスを緩める設定を無視します。それらの一覧は、**管理者必須のサンドボックスにおけるリポジトリ設定** (ページ 1094) に記載されています。

strict sandbox モードは、Claude が実行するコマンドに適用されます。 シェルモードのプロンプト <interactive-mode--shell-mode-with-prefix>で自分で入力したコマンドは、セッションが次のいずれかでない限り、サンドボックス外で実行されます。

- バックグラウンドセッション：strict sandbox モードはシェルモードのコマンドにも適用されます
- CLAUDE_CODE_SUBPROCESS_ENV_SCRUB **が設定された Linux セッション**：シェルモードのコマンドも含め、すべてのコマンドがサンドボックス化されて実行されます

v2.1.260 より前は、strict sandbox モードはすべてのセッションでシェルモードのコマンドをサンドボックス化していました。

一時ディレクトリ

デフォルトでは、作業ディレクトリに加えて、ユーザーごとの一時ディレクトリにもサンドボックス内から書き込めます。**ファイルシステムの分離を無効にする** (ページ 1080) 場合を除き、Claude Code はサンドボックス化されたコマンドに対して $TMPDIR をこのディレクトリに設定するため、一時ファイルを書き込むツールは追加の設定なしで動作します。

サンドボックス化されていないコマンドは、シェルの $TMPDIR が設定されていればそれを継承します。そのため、ファイルシステムの分離がオンの間は、サンドボックス化されたコマンドとされていないコマンドで $TMPDIR が異なるディレクトリに解決されます。シェルで $TMPDIR が未設定または空の場合、$TMPDIR を参照するサンドボックス外のコマンドには、CLAUDE_CODE_TMPDIR による上書き値が渡されます。上書き値を設定していない場合や上書き値が長いパスである場合はオペレーティングシステムの一時ディレクトリが渡されるため、変数が空文字列に展開されることはありません。両者の間で一時ファイルを受け渡すには、代わりに作業ディレクトリ配下に書き込んでください。

48.3 サンドボックスを設定する

サンドボックスの動作は `settings.json` ファイルでカスタマイズできます。設定の完全なリファレンスについては、設定を参照してください。

デフォルトでは、サンドボックス化されたコマンドが書き込めるのは、現在の作業ディレクトリ、ユーザーごとの一時ディレクトリ、そして `--add-dir`、`/add-dir`、または `permissions.additionalDirectories` で**追加したディレクトリ** (ページ 1061) です。`kubectl`、`terraform`、`npm` などのサブプロセスコマンドがこれらのディレクトリの外に書き込む必要がある場合は、`sandbox.filesystem.allowWrite` を使用して特定のパスへのアクセスを付与します。

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "allowWrite": ["~/.kube", "/tmp/build"]
    }
  }
}
```

これらのパスは OS レベルで適用されるため、サンドボックス内で実行されるすべてのコマンドは、その子プロセスも含めてこれらに従います。ツールが特定の場所への書き込みアクセスを必要とする場合は、`excludedCommands` でツールをサンドボックスから完全に除外するのではなく、この方法を使用することを推奨します。

同じファイルシステム配列を複数の**設定スコープ** (ページ 719) で定義した場合、Claude Code はそれらをマージし、あるスコープの配列を別のスコープの配列で置き換えるのではなく、すべてのスコープのパスを結合します。**開発者がポリシーを広げないようにする** (ページ 1094) で説明しているロックがエントリを対象としている場合、Claude Code はそのエントリをマージから除外します。

CLI の *`--setting-sources`* (ページ 45) や Agent SDK の `settingSources` で設定ソースを除外した場合、Claude Code はサンドボックス設定を構築する際に、そのソースの `sandbox.filesystem` エントリ、`Edit` 権限ルール、`Read` 拒否ルールを無視します。Claude Code v2.1.246 以降が必要です。

セッション中にこれらのファイルシステムリストを編集すると、Claude Code は**実行中のセッションに変更を適用する** (ページ 717) ため、次にサンドボックス化されたコマンドは新しいパスのもとで実行されます。

サンドボックスのファイルシステムパスは標準的な規則に従います。`/tmp/build` は絶対パスで、`~/.kube` はホームディレクトリからの相対パスです。これは、絶対パスに `//path`、プロジェクト相対パスに `/path` を使用する *Read と Edit の権限ルール* (ページ 1051) とは異なります。相対パス、末尾のスラッシュ、ワイルドカードについては、サンドボックスのパスプレフィックスを参照してください。

`sandbox.filesystem.denyWrite` と `sandbox.filesystem.denyRead` を使用して書き込みや読み取りのアクセスを拒否することもでき、`sandbox.filesystem.allowRead` を使用して拒否された領域内の特定のパスを再び許可することもできます。読み取りルールが重なる場合は、より狭いパスのルールが適用されます。

ルールの例	結果
"denyRead": ["~/"] と "allowRead": ["~/projects"]	~/projects は読み取り可能で、ホームディレクトリの残りはブロックされたままになります。より狭い許可が、拒否された領域のその部分を再び開きます
"allowRead": ["~/"] と "denyRead": ["~/.env"]	~/.env はブロックされたままで、ホームディレクトリの残りは読み取り可能です。拒否はより広い許可の内側でも維持されるため、広範な許可によってシークレットが気付かないうちに再び公開されることはありません
"allowRead": ["~/"] と "denyRead": ["~/**/.env"]	ホームディレクトリ配下のすべての .env はブロックされたままで、残りは読み取り可能です。ワイルドカードによる拒否も、正確なパスと同じようにより広い許可の内側で維持されます

以下の例では、現在のプロジェクトからの読み取りを許可しつつ、ホームディレクトリ全体からの読み取りをブロックします。相対パス . がプロジェクトルートに解決されるのは設定がプロジェクト設定にある場合のみなので、この設定はプロジェクトの .claude/settings.json に配置してください。

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "denyRead": ["~/"],
      "allowRead": ["."]
    }
  }
}
```

同じ設定を ~/.claude/settings.json に配置した場合、. は代わりに ~/.claude に解決され、プロジェクトファイルは denyRead ルールによってブロックされたままになります。

作業ディレクトリを読み取り可能に保ちつつ、サンドボックス化されたコマンドによるホームディレクトリやマウントされたボリュームへの読み取りアクセスを拒否するには、パスルールを書く代わりに permissions.blockReadsOutsideWorkingDirectories を設定します。

48.3.1 excludedCommands でコマンドをサンドボックスの外で実行する

sandbox.excludedCommands にコマンドパターンを記載すると、一致するコマンドがサンドボックスの外で実行されます。つまり、ファイルシステムの制限もネットワークプロキシもありません。サンドボックス内では動作せず、完全なアクセス権を委ねても信頼できるツールに使用してください。ディレクトリやホストが 1 つ追加で必要なだけのツールであれば、コマンドをサンドボックス化したままにできる allowWrite や allowedDomains で動作する場合があります。

この例では、docker compose コマンドをサンドボックスから外します。すべてのプロジェクトに適用するには、~/.claude/settings.json に保存してください。

```
{
  "sandbox": {
    "enabled": true,
    "excludedCommands": ["docker compose *"]
  }
}
```

Claude Code は、Bash と Monitor の各呼び出しに対してエントリを照合します。呼び出しとは Claude が送信するコマンドライン全体であり、複数のコマンドが連結されている場合があります。呼び出しがサンドボックスの外に出るかどうかは、以下のルールによって決まります。

- **パターンの末尾は * にする**: エントリは `Bash(...)` の**権限ルール** (ページ 1043) と同じ構文を使用し、ワイルドカードのないパターンは完全一致になります。`docker` は引数のない `docker` のみに一致します。`docker *` は引数の有無にかかわらず `docker` に一致します
- **呼び出し内のすべてのコマンドが一致する必要がある**: `npm ci && docker compose build` は、別のエントリが `npm ci` をカバーしていない限り、サンドボックス化されたままです
- **Claude Code は呼び出しのテキストを照合する**: 内部で `docker` を呼び出すスクリプトや `make` ターゲットは一致せず、`/usr/local/bin/docker` も一致しません
- **サンドボックス化されたままになる呼び出しがある**: ファイルへのリダイレクト、`cd`、または `$(...)` のようなコマンド置換があると、呼び出し全体がサンドボックス化されたままになります。サンドボックス化されたままになるその他の呼び出しについては、リファレンスのエントリに記載されています
- **エントリの保存場所が影響する場合がある**: サンドボックスが**管理者によって必須化** (ページ 1094) されている間、Claude Code は `.claude/settings.json` と `.claude/settings.local.json` 内のエントリを無視します

除外されたコマンドは、通常の権限フローを経由します。

- **読み取り専用コマンド** (ページ 1049) と、許可ルールでカバーされているコマンドは、プロンプトなしで実行されます
- auto モードでは、その他の除外されたコマンドを分類器がレビューします
- `bypassPermissions` モードでは、除外されたコマンドは確認ルールに一致しない限りプロンプトなしで実行されます

エントリが一致することを確認するには、Manual モードに切り替えて、`docker compose up -d` のような何かを変更する一致コマンドを実行するよう Claude に依頼します。権限プロンプトのタイトルは「Bash command (unsandboxed)」になります。

警告:

除外されたコマンドは、ユーザーの完全なアクセス権で実行されます。`docker *` のような広範なエントリは、そのツールができることすべてをカバーします。インタープリター、作業ディレクトリ内のスクリプト、またはそこにあるファイルに作用するツール（`docker compose` が compose ファイルに対して行うように）をカバーするパターンを書くと、Claude はそのファイルを書き込み、その後サンドボックスの外で実行できてしまいます。パターンを狭くするほど、Claude がサンドボックスの外で実行できるものは少なくなります。

48.3.2 ファイルシステム分離を無効にする

`sandbox.filesystem.disabled` を `true` に設定すると、ネットワーク分離を維持したままファイルシステム分離をスキップできます。以下の例では、ネットワークドメインの許可リストを維持したまま、ファイルシステム分離をオフにします。

```
{
  "sandbox": {
    "enabled": true,
    "filesystem": {
      "disabled": true
    },
    "network": {
      "allowedDomains": ["github.com", "*.npmjs.org"]
    }
  }
}
```

サンドボックスには独立した 2 つのレイヤーがあります。**ファイルシステム分離** (ページ 1086) はサンドボックス化されたコマンドが読み書きできるパスを制御し、**ネットワーク分離** (ページ 1087) はそれらが到達できるドメインを制御します。ファイルシステムレイヤーをオフにすると、サンドボックス化されたコマンドはホストのファイルシステムに対して無制限の読み書きアクセスを得ますが、ネットワークの送信先は許可したドメインに限定されたままです。コマンドが何を書き込むかではなく、どこに接続するかを制御する目的でサンドボックスを使う場合に、このレイヤーをオフにしてください。

`sandbox.filesystem.disabled` のデフォルトは `false` です。Claude Code v2.1.216 以降が必要です。

警告:

ファイルシステム分離がオフでコマンドが自動許可されている場合、サンドボックス化されたコマンドは、シェルの起動ファイル、`$PATH` 上の実行ファイル、`~/.claude/settings.json` など、後続のコマンドが実行または読み取るファイルを書き込み、それを利用して次回の実行時に自身のアクセス権を広げることができます。`filesystem.disabled` を `true` に設定するのは、自身のアクセス権を昇格させないと信頼できるワークロードに限定してください。*`allowManagedDomainsOnly`* (ページ 1094) でネットワークドメインをロックするとリスクは狭まりますが、このロックはサンドボックス内で実行されるコマンドにのみ適用されるため、リスクがなくなるわけではありません。

無効にできる設定

ファイルシステム分離をオフにすると、サンドボックス化されたコマンドができることが広がるため、Claude Code は以下の設定ソースからの `filesystem.disabled` のみを尊重します。

- ユーザー設定、管理設定、および `--settings` CLI フラグで設定できます。`.claude/settings.json` と `.claude/settings.local.json` のプロジェクト設定では設定できないため、チェックアウトしたプロジェクトがファイルシステム分離をオフにすることはできません。
- 管理設定が `sandbox.filesystem` を何らかの形で設定している場合、または `"mode": "deny"` の `sandbox.credentials.files` エントリを記載している場合は、管理設定のみがこのキーを設定できます。これにより、管理者がデプロイしたファイルシステムの制限が有効なまま維持されます。そのようなデプロイを緩和するには、管理設定で `"disabled": true` を設定します。
- `CLAUDE_CODE_SUBPROCESS_ENV_SCRUB` が設定されている場合、Claude Code は管理設定を含むすべてのソースからの `filesystem.disabled` を無視し、ファイルシステム分離をオンのまま維持します。

有効な `mask` エントリは、起動時に Claude Code がそのエントリについて *deny* **にフォールバック** (ページ 1085) した場合でも、このキーを固定しません。認証情報ディレクトリのようにマスクできないパスは、管理設定で明示的な `deny` エントリとして記載してください。これによりキーが固定されます。

ファイルシステム分離がオフのときに変わること

`filesystem.disabled` を設定すると、ファイルシステムレイヤー自体が適用している保護が解除されます。他のレイヤーが適用している保護は引き続き適用されます。

保護	ファイルシステム分離がオフの場合
`filesystem.denyRead` と *credentials.files* (ページ 1082) の `deny` による読み取りブロック	適用されません。どちらもファイルシステムレイヤーが適用しています
`credentials.envVars` の `deny` と `mask` エントリ	適用されます。環境変数の除去はファイルシステムレイヤーから独立しています
マスクとして適用された *credentials.files* **の** *mask* **エントリ** (ページ 1085)	適用されます。マスキングはファイルシステムレイヤーから独立しています。*deny* **にフォールバックした** (ページ 1085) エントリは、他の `deny` エントリと同様に適用されません

その他に 2 つの点が変わります。

- サンドボックス化されたコマンドは、ユーザーごとの一時ディレクトリではなく、シェルの `$TMPDIR` を継承します。すべての一時ディレクトリが書き込み可能になり、Claude Code がコマンドをユーザーごとの一時ディレクトリにリダイレクトしなくなるためです。

  Linux では、親シェルでこの変数が設定されていないことがよくあります。Bash ツールのガイダンスは、`$TMPDIR` に頼るのではなく `mktemp -d` で作業用ディレクトリを作成するよう Claude に指示します。

- `autoAllowBashIfSandboxed` のデフォルトは引き続き `true` であるため、サンドボックス化されたコマンドはプロンプトなしで実行され続けます。サンドボックス化されたコマンドでプロンプトを表示するには、`false` に設定します。

48.3.3 認証情報を保護する

`sandbox.credentials` 設定では、サンドボックス化されたコマンドから保護する認証情報ファイルと環境変数を宣言します。各エントリには、ファイルパスまたは環境変数と、`mode` を指定します。専用の `credentials` ブロックを使うことで、認証情報のルールをまとめて、一般的なファイルシステムのルールとは分けて管理できます。

`"mode": "deny"` のエントリでは、ファイルパスはサンドボックス内での読み取りが拒否され（`filesystem.denyRead` が適用するのと同じ制限）、環境変数はサンドボックス化された各コマンドの実行前に未設定にされます。ファイルの保護はファイルシステムレイヤーの一部であるため、**ファイルシステム分離を無効にした** (ページ 1080) 場合は適用されませんが、環境変数の保護は引き続き適用されます。

以下の例では、AWS の認証情報ファイルと SSH ディレクトリの読み取りをブロックし、サンドボックス化されたコマンドの環境から `GITHUB_TOKEN` と `NPM_TOKEN` を削除します。

```
{
  "sandbox": {
    "enabled": true,
    "credentials": {
      "files": [
        { "path": "~/.aws/credentials", "mode": "deny" },
        { "path": "~/.ssh", "mode": "deny" }
      ],
      "envVars": [
        { "name": "GITHUB_TOKEN", "mode": "deny" },
        { "name": "NPM_TOKEN", "mode": "deny" }
      ]
    }
  }
}
```

環境変数のエントリとファイルのエントリは `"mode": "mask"` も受け付けます。これについては**認証情報をマスクする** (ページ 1083) で説明します。

ファイルパスは、`sandbox.filesystem.*` 設定と同じプレフィックスのルールに従います。

Claude Code は、セッションが読み込むすべての**設定スコープ** (ページ 719) の deny エントリをマージします。deny エントリはアクセスを狭めることしかしないため、どのスコープでも追加できますが、別のスコープが追加したエントリをどのスコープも削除することはできません。

設定ソースを除外した (ページ 1077) 場合:

- **プロジェクト設定またはローカル設定**: Claude Code はそれらの `credentials` エントリをいずれも適用しません。Claude Code v2.1.246 以降が必要です。
- **ユーザー設定**: Claude Code は `~/.claude/settings.json` 内の deny エントリを引き続き適用し、**ファイルの *mask* エントリ** (ページ 1085) も制限として維持します（ただし、それらはプロキシが実際の値に置換することを認可しなくなります）が、**環境変数の *mask* エントリ** (ページ 1083) は破棄します。

組み込みの認証情報拒否リストはないため、制限されるのは記載したファイルと変数のみです。

sandbox.credentials は、サンドボックス化された Bash コマンドにのみ影響します。サンドボックス化にかかわらずすべてのサブプロセスから認証情報を除去するには、CLAUDE_CODE_SUBPROCESS_ENV_SCRUB を設定します。

48.3.4 認証情報をマスクする

認証情報をマスクすると、Claude Code はサンドボックス化されたコマンドにセンチネルと呼ばれるセッションごとのプレースホルダーを見せ、**サンドボックスプロキシ** (ページ 1087) が許可したホストへの送信リクエストで実際の値に置き換えます。**認証情報を保護する** (ページ 1082) で説明した deny エントリは、代わりに認証情報をブロックします。macOS 上のファイルについては、Claude Code はマスクする代わりに**ファイルをブロックします** (ページ 1085)。すべてのフィールドは sandbox.credentials のリファレンスに記載されています。

マスキングには以下が必要です。

- **TLS 終端**: プロキシはリクエストの内容の中で実際の値を置換するため、その内容を参照できる必要があります。プロキシ自体が TLS を終端するように、network.tlsTerminate を設定してください。これを設定しない場合、マスキングは何も漏らさずに失敗します。コマンドにはセンチネルしか見えませんが、センチネルはそのままサーバーに届き、認証が失敗します。Claude Code は起動時にこの設定ミスを報告します。
- **許可された送信先**: 各 mask エントリには injectHosts（実際の値の送信先として許可されるホスト）を記載できます。プロキシは**ドメイン許可リスト** (ページ 1087) が許可する接続でのみ注入を行うため、各 injectHosts のホストは network.allowedDomains を通じても到達可能である必要があります。injectHosts のない mask エントリの場合、プロキシは network.allowedDomains 内のすべてのホストへのリクエストで実際の値に置換します。
- **信頼できる設定スコープ**: マスキングはプロキシが実際の認証情報をどこかに送信することを認可するため、Claude Code は mask エントリ、network.tlsTerminate、credentials.allowPlaintextInject、awsPairs、sigv4 を、ユーザー設定、管理設定、および --settings フラグからのみ尊重します。リポジトリの .claude/settings.json や .claude/settings.local.json 内のこれらは無視されます。管理者がサーバー管理設定を通じて mask エントリ、network.tlsTerminate、または credentials.allowPlaintextInject を配布する場合、それらは承認が必要な設定として扱われます。

環境変数をマスクする

環境変数をマスクするには、その credentials.envVars エントリに "mode": "mask" を設定します。コマンドやそれがログに記録するものが実際の認証情報を保持することはありませんが、リクエストは引き続き認証されます。同じ変数がいずれかのスコープで deny として記載されている場合は、deny が優先されます。

以下の例では、2 つのトークンをマスクします。GH_TOKEN は api.github.com へのリクエストでのみ置換され、NPM_TOKEN は injectHosts がないため、network.allowedDomains 内のすべてのホストへのリクエストで置換されます。

```
{
  "sandbox": {
    "enabled": true,
```

(次のページに続く)

(前のページからの続き)

```
    "network": {
      "tlsTerminate": {},
      "allowedDomains": ["*.github.com", "registry.npmjs.org"]
    },
    "credentials": {
      "envVars": [
        { "name": "GH_TOKEN", "mode": "mask", "injectHosts": ["api.github.com"] },
        { "name": "NPM_TOKEN", "mode": "mask" }
      ]
    }
  }
}
```

マスキングはデフォルトで値全体を置き換えます。DATABASE_URL 接続文字列や JWT のように構造を持つ値には、extract、decode、maskClaims、onExtractNoMatch フィールドを使用して、値を解析するツールが動作し続けるようにしてください。

IPv6 の送信先は、2 つのリストで異なる書き方をしてください。

- network.allowedDomains: "[::1]" のような角括弧付きの形式
- injectHosts: "::1" のような、正規の圧縮形式による角括弧なしのアドレス

プロキシはポートを無視して各 injectHosts エントリを接続の角括弧なしの送信先アドレスと照合するため、角括弧付き、ゾーン ID 付き、または異なる圧縮方法で書かれたものは決して一致しません。claude doctor は、決して一致しないエントリを Sandbox credential injectHosts entries can never match their destination という警告で指摘します。このチェックには Claude Code v2.1.229 以降が必要です。

AWS リクエストを再署名する

AWS リクエストはリクエストの内容に対する SigV4 署名を持つため、AWS_ACCESS_KEY_ID と AWS_SECRET_ACCESS_KEY は一緒にマスクしてください。プロキシはアクセスキーの**センチネル** (ページ 1083) によって SigV4 リクエストを検出し、実際の値でリクエストを再署名します。これには Claude Code v2.1.221 以降が必要です。シークレットのみをマスクすると、リクエストはプロキシが検出できないプレースホルダーで署名されるため、AWS で失敗します。

Claude Code は、慣例的な変数である AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY、AWS_SESSION_TOKEN の値全体をマスクすると、それらを自動的に 1 つの認証情報として関連付けます。AWS の認証情報が別の名前の変数にある場合は、credentials.awsPairs でグループ化してください。これには Claude Code v2.1.224 以降が必要です。

ストリーミングアップロード、署名付き URL、SigV4A リクエストは、プロキシが再計算できない署名を持ちます。これらのリクエストがマスクされたペアのプレースホルダーで署名されている場合、プロキシは壊れた署名を転送するのではなく、リクエストを失敗させます。マスクされていない認証情報で署名されたリクエストが影響を受けることはありません。これらのリクエスト形式のいずれかを代わりに転送するには、credentials.sigv4 を使用します。これには Claude Code v2.1.224 以降が必要です。AWS は依然としてリクエストを拒否するため、呼び出し元のツールはプロキシエラーではなく AWS 自身の拒否レスポンスを受け取ります。

認証情報ファイルをマスクする

認証情報ファイルをマスクするには、その `credentials.files` エントリに `"mode": "mask"` を設定します。ファイルのマスキングには Claude Code v2.1.221 以降が必要です。サンドボックス化されたコマンドに何が見えるかは、プラットフォームによって異なります。

- **Linux と WSL2**: サンドボックス化されたコマンドはファイルの**センチネル** (ページ 1083) コピーを読み取り、プロキシが送信リクエストで実際の値に置換します。
- **macOS**: サンドボックス化されたコマンドはファイルをまったく読み取れません。Claude Code はセンチネルコピーを作成しないため、そのファイルを使って認証するツールはサンドボックス内で動作しません。これは deny と同じ効果です。**ファイルシステム分離を無効にした** (ページ 1080) 場合でも、読み取りブロックは維持されます。

以下の例では、`~/.config/gh/hosts.yml` に保存された GitHub トークンをマスクします。`extract` パターンはファイルのどの部分がシークレットであるかを示すため、Linux と WSL2 では `gh` が設定の残りの部分を引き続き解析できます。

```
{
  "sandbox": {
    "enabled": true,
    "network": {
      "tlsTerminate": {},
      "allowedDomains": ["*.github.com"]
    },
    "credentials": {
      "files": [
        {
          "path": "~/.config/gh/hosts.yml",
          "mode": "mask",
          "extract": "oauth_token:\\s*(\\S+)",
          "injectHosts": ["api.github.com"]
        }
      ]
    }
  }
}
```

マスクが有効であることを確認するには、サンドボックス化されたコマンドで `cat ~/.config/gh/hosts.yml` を実行するよう Claude に依頼します。Linux と WSL2 では出力にトークンの代わりにセンチネルが表示され、macOS では読み取りが失敗します。

`extract` または `decode` がない場合、Claude Code はファイル全体を 1 つのセンチネルに置き換えます。これは、単独のシークレットだけを保持するファイルに適しています。部分的なマスキングと、パターンが何にも一致しない場合の動作を制御するには、extract、decode、maskClaims、onExtractNoMatch、maskDuplicates フィールドを使用してください。

警告:

照合でマスクするものが見つからなかった場合、`onExtractNoMatch` のデフォルト値である `warn` はエントリをスキップするため、サンドボックス化されたコマンドはマスクされていない実際のファイルを読み取れます。macOS では、ファイルシステム分離がオンのときは常に Claude Code がパターンの実行前に `mask` エントリを `deny` として適用するため、一致しない場合の結果が有効になるのは**ファイルシステム分離がオフ** (ページ 1080) の場合のみです。このデフォルトは正当に存在しない場合がある認証情報に適しています。シークレットが存在する可能性があるもののパターンがそれを見逃す可能性がある場合は、`deny` を使用してください。

`mask` は単一のファイルに適用されるため、各認証情報ファイルを個別に記載してください。Claude Code は、安全にマスクできない `mask` エントリ（ディレクトリパス、glob パターン、8 MiB を超えるファイル、UTF-8 テキストではないファイル）については `deny` にフォールバックします。

48.4 サンドボックス化の仕組み

48.4.1 ファイルシステムの分離

サンドボックス化された Bash ツールは、ファイルシステムへのアクセスを特定のディレクトリに制限します。

- **デフォルトの書き込み動作**: 現在の作業ディレクトリとそのサブディレクトリ、`--add-dir`、`/add-dir`、または `permissions.additionalDirectories` で追加したディレクトリ、さらに `$TMPDIR` が指すユーザーごとの一時ディレクトリへの読み取りおよび書き込みアクセス
- **デフォルトの読み取り動作**: 特定の拒否されたディレクトリを除き、コンピューター全体への読み取りアクセス。このデフォルトでは認証情報ファイルの読み取りも許可されるため、コマンドに読み取らせたくない**認証情報を保護** (ページ 1082) してください。
- **読み取りブロック**: `permissions.blockReadsOutsideWorkingDirectories` をオンにすると、サンドボックス化されたコマンドは、ブロック下のサンドボックス化されたコマンドに記載されたパスを除き、ホームディレクトリおよびユーザーファイルを保持するその他のディレクトリへの読み取りアクセスも失います。このセクションでは、ブロックのこの部分が適用されない場合についても説明しています。
- **Git worktree**: 作業ディレクトリがリンクされた git worktree である場合、サンドボックスはメインリポジトリの共有 `.git` ディレクトリへの書き込みも許可するため、`git commit` などのコマンドで ref やインデックスを更新できます。そのディレクトリ内の `hooks/` と `config` への書き込みは引き続き拒否されます。

ネットワークの分離を維持したままファイルシステムの分離を完全にスキップするには、*`sandbox.filesystem.disabled`* (ページ 1080) を設定します。

48.4.2 保護されたパス

サンドボックス化されたコマンドが書き込み可能なディレクトリ内であっても、サンドボックスは Claude Code が設定やコードを読み込むファイルへの書き込みを引き続き拒否します。これらのファイルを編集できるコマンドは、自身に権限を付与したり、Claude Code がサンドボックスの外で実行するフックや MCP サーバーを追加したりできてしまうためです。権限システムには独自の保護されたパスがあり、ツールの実行前に Claude Code が何を承認するかを制御します。サンドボックスのリストは、すでに実行中のコマンドに適用されます。対象となるパスは次の 4 つのグループです。

- **作業ディレクトリとその上位のディレクトリ**: `.claude` 設定ファイル、`.claude/skills`、`.claude/agents`、`.claude/commands`、`.claude/hooks` ディレクトリ、`.mcp.json`、および `.claude/workflows` や `.claude/scheduled_tasks.json` など Claude Code が自ら実行するファイル
- **作業ディレクトリのみ**: `.bashrc` や `.zshrc` などのシェル起動ファイル、`.gitconfig`、`.vscode` および `.idea` ディレクトリ、`.git` 内の `hooks` と `config`
- **作業ディレクトリをベア git リポジトリに変えてしまうファイル**: 最上位の `HEAD`、`objects`、`refs`、および `HEAD` が隣にある場合の既存の `config` と `hooks` エントリ。`config` という名前のファイルは、`HEAD` がなくても拒否されます。Linux と WSL2 では、サンドボックス化されたコマンドの実行中に最上位の `HEAD` ファイルや `objects` または `refs` ディレクトリが出現すると、サンドボックスがそれを削除します
- **`~/.claude`、または `CLAUDE_CONFIG_DIR` が指すディレクトリ内**: その内容の大部分、および `~/.claude.json` と `.credentials.json` 認証情報ストア

セッション中に保護された設定ファイルのパスにシンボリックリンクが出現した場合、サンドボックスは次のコマンドから、そのリンク先のファイルへの書き込みも拒否します。

これらのパスのいずれかを除外する方法はありません。パスを対象とする `allowWrite` エントリや `Edit` 許可ルールでは保護は解除されません。保護をオフにする唯一の方法は *`filesystem.disabled`* (ページ 1080) で、これはすべてのパスでファイルシステムの分離をオフにします。ご使用のマシンで解決されたこれらのパスの大部分を確認するには、`/sandbox` を実行して **Config** タブを開きます。ここでは、これらのパスがユーザー自身の `denyWrite` エントリと混在して **Denied within allowed** の下に一覧表示されます。

これらのパスのいずれかで `git merge` または `git checkout` が `unable to unlink old` で失敗する場合は、*git* **コマンドが** *unable to unlink old* **で失敗する** (ページ 1098) を参照してください。

48.4.3 ネットワークの分離

サンドボックス化されたコマンドには、ネットワークへの直接の経路がありません。

- **Linux と WSL2**: コマンドは、ユーザーのネットワークに接続されていない別のネットワーク名前空間で実行されます
- **macOS**: Seatbelt サンドボックスフレームワークが、デフォルトでサンドボックスプロキシへの接続以外の接続をブロックします

Claude Code はサンドボックスの外、ユーザーのマシン上でサンドボックスプロキシを実行し、`HTTP_PROXY`、`HTTPS_PROXY`、`ALL_PROXY` および関連する環境変数を使ってコマンドをプロキシに向けます。プロキシは、各接続のホスト名を許可ドメインおよび拒否ドメインと照合します。

ツールが到達できる範囲は、そのツールがプロキシを使用するかどうかによって異なります。

- **プロキシ変数を読み取るツール**: curl、npm、HTTPS 経由の git などのツールは、ホストが許可されると接続できます。ポートを指定しない allowedDomains エントリは、そのホストのすべてのポートを許可します
- **プロキシ変数を無視するツール**: 素の ssh、ほとんどのデータベースドライバーなどのツールは、許可されたホストであっても接続できません。**データベースクライアントやその他の非 *HTTP* ツールが許可されたホストに到達できない** (ページ 1100) を参照してください
- **TCP 以外のもの**: UDP、QUIC 上の HTTP/3、および ping などの ICMP ツールはサンドボックスの外に出られません

以下の設定と動作により、プロキシが許可するホストが制御されます。

- **ドメイン制限**: 許可ドメインは最初は空です。コマンドが新しいドメインを初めて必要とした場合の動作については、**許可ドメイン外のホスト** (ページ 1089) で説明しています。
- **承認の選択**: プロンプトで Yes を選択すると、Claude Code は現在のセッションの残りの間、そのホストを許可します。「Yes, and don't ask again」を選択すると、Claude Code は WebFetch(domain:...) 許可ルールを**ローカル設定** (ページ 1039) に保存するため、今後のセッションでもそのホストは許可されたままになります。サンドボックスが**管理者必須** (ページ 1094) の場合、Claude Code はルールをユーザー設定に保存し、すべてのプロジェクトに適用されます。
- **事前許可ドメイン**: allowedDomains でドメインを事前に許可すると、プロンプトを完全に回避できます。**権限ルール** (ページ 1091) で説明しているように、Claude Code は WebFetch(domain:...) 許可ルールのドメインも事前に許可します。
- **厳格な許可リスト**: ユーザー設定、管理設定、または CLI の --settings 設定で strictAllowlist を true に設定すると、Claude Code はプロンプトを表示する代わりに、サンドボックス化されたコマンドから許可リスト外のホストへのアクセスを拒否します。許可リストは allowedDomains と WebFetch(domain:...) 許可ルールのドメインの合計で、allowManagedDomainsOnly が設定されている場合は管理設定のエントリのみとなります。リポジトリのエントリについては、**管理者必須のサンドボックスなしで適用されるロック** (ページ 1095) で説明しています。Claude Code はこれをサンドボックス化されたコマンドにのみ適用します。WebFetch などのプロセス内ツールは引き続き**権限ルール** (ページ 1091) に従います。リポジトリの .claude/settings.json または .claude/settings.local.json で設定しても効果はありません。Claude Code v2.1.219 以降が必要です。
- **管理設定によるロックダウン**: 管理設定で allowManagedDomainsOnly が設定されている場合、許可されていないドメインはプロンプトを表示せずに自動的にブロックされ、管理設定の allowedDomains と WebFetch(domain:...) 許可ルールのみが適用されます。
- **企業プロキシ**: ネットワークで送信トラフィックを企業プロキシ経由にする必要がある場合は、**プロキシ設定** (ページ 897) の説明に従って HTTPS_PROXY、HTTP_PROXY、NO_PROXY を設定します。**バックグラウンドエージェント** (ページ 901) にも適用されるよう設定の env ブロックで設定するか、Claude Code を起動する環境で設定してください。Claude Code はドメイン許可リストを適用したうえで、許可された接続をその上流プロキシ経由でトンネリングします。http:// と https:// のプロキシ URL が使用でき、必要に応じて URL に Basic 認証を含めることもできます。

WebFetch(domain:...) ルールでは、サンドボックスは 2 つのワイルドカード形式を認識します。*.example.com のような先頭の *. と、単独の * です。単独の * 形式には Claude Code v2.1.186 以降が必要です。WebFetch(domain:example.*) のようにそれ以外の位置にあるワイルドカードは、フェッチには引き続き一致しますが、サンドボックス化されたコマンドには効果がありません。

注釈:

組み込みプロキシは、要求されたホスト名に基づいて許可リストを適用し、デフォルトでは TLS トラフィックの終端や検査を行いません。実験的な `network.tlsTerminate` 設定を使用すると、組み込みプロキシ自体が TLS を終端します。これは *mask* **認証情報エントリ** (ページ 1083) に必要です。デフォルトの影響については**セキュリティ上の制限** (ページ 1103) を、脅威モデルで TLS の検査が必要な場合は**カスタムプロキシ設定** (ページ 1096) を参照してください。

許可ドメイン外のホスト

サンドボックス化されたコマンドが許可ドメインにないホストに接続すると、コマンドはサンドボックス内にとどまり、判断を待ちます。インタラクティブなターミナルセッションでは、判断は権限モードによって異なります。

権限モード	接続の扱い
`bypassPermissions` モード、および権限のバイパスが利用可能な plan モード	プロンプトなしで許可
手動モード、`acceptEdits` モード、およびそれ以外の plan モード	プロンプトが表示される
auto モード	コマンドが**ホストを列挙** (ページ 1090) し、分類器がそのリストを承認した場合を除き拒否
`dontAsk` モード	拒否

`strictAllowlist` または `allowManagedDomainsOnly` がオンの場合、組み込みのサンドボックスプロキシはすべての権限モードで接続を拒否します。`bypassPermissions` モードでは、これらのいずれかがオンでない限り、許可ドメイン外のホストは許可されます。そのモードでコマンドがサンドボックスの外に出られる場合については、**サンドボックスなしでの再試行というエスケープハッチ** (ページ 1075) で説明しています。`deniedDomains` にあるホストへの接続も、すべての権限モードで拒否されます。

ローカルアドレスに解決されるホスト名

ホスト名が許可リストを通過した後、サンドボックスプロキシはその名前を解決し、ローカルアドレスのみに解決される場合は接続を拒否します。ローカルアドレスには、`127.0.0.1` などのループバックアドレス、`169.254.169.254` クラウドメタデータエンドポイントなどのリンクローカルアドレス、およびユーザー自身のマシンに割り当てられたアドレスが含まれます。`localhost` および `*.localhost` という名前は、ループバックへの解決が許可されます。

`10.0.0.0/8` などのプライベート範囲に解決される許可済みのイントラネットホスト名は接続できます。名前が拒否されるアドレスに解決されることを許可するには、`"127.0.0.1:8080"` のように、その IP アドレスを `allowedDomains` に追加します。

このチェックはホスト名に適用されます。IP アドレスへの接続は、許可ドメインと権限モードによって判断されます。また、プロキシは上流の企業プロキシ経由で送信する接続についてはこのチェックをスキップします。名前の解決はその企業プロキシが行うためです。

auto モードでのコマンドごとの許可ドメイン

サンドボックス化がオンの auto モードでは、Claude は接続ごとにネットワーク承認をトリガーする代わりに、コマンドが必要とするホストをコマンド自体に指定します。サンドボックス内で実行される各 Bash、PowerShell、または Monitor コマンドは、サンドボックスの許可リストを超えるホストのリストを持つことができます。`registry.npmjs.org` のようなドメイン、`*.pythonhosted.org` のようなワイルドカード、または IP アドレスで、それぞれにオプションで `:port` を付けられます。分類器はホストをコマンドと一緒に審査します。Claude Code v2.1.271 以降が必要です。

承認されたリストは、そのコマンドの実行中に限り、そのコマンドに対してのみそれらのホストを開放します。セッションの許可ホストや設定には何も追加されず、次のコマンドは独自のホストを指定します。

ホストを持つコマンドは、権限ルールやサンドボックスの**自動許可モード** (ページ 1074) によって承認されるのではなく、分類器に送られます。**確認ルール** (ページ 1041) によってコマンドにプロンプトが強制される場合、ターミナルの権限ダイアログではコマンドの横にホストが一覧表示され、そこで承認すると両方が対象になります。

コマンドごとのリストは、サンドボックスがデフォルトで拒否する範囲のみを広げます。`deniedDomains` のエントリは引き続きブロックされます。`strictAllowlist` または `allowManagedDomainsOnly` が許可リストをロックしている場合、Claude Code はコマンドごとのリストを拒否します。

コマンドごとのリストが適用されている間、Claude Code は、承認されたどのコマンドにも列挙されていないホストへの接続を、プロンプトや分類器のチェックなしに拒否します。拒否の際はコマンドの結果にホスト名が示され、Claude はそのホストを追加してコマンドを再実行します。

ドメインリスト内の IPv6 アドレス

`allowedDomains`、`deniedDomains`、または `WebFetch(domain:...)` ルールで IPv6 アドレスに一致させるには、アドレスを角括弧で囲んで記述します。`"[::1]"` はすべてのポートでそのアドレスに一致し、`"[::1]:443"` はポート 443 でのみ一致します。角括弧形式には Claude Code v2.1.229 以降が必要です。

`::1:443` のような角括弧のないエントリは、アドレスとポート付きのアドレスのどちらとも解釈できるため曖昧です。

- **拒否リスト**: Claude Code はエントリが解釈され得るすべての読み方を拒否するため、意図した読み方がどちらであってもブロックされます。解釈可能な読み方がないエントリについては、Claude Code は何もブロックしません
- **許可リスト**: Claude Code は記述された以上のものを許可しません。曖昧なエントリは、ホストとポートとしての読み方が正しく解析できる場合はその読み方に書き換え、許可リストを広げるよりはエントリを完全に破棄することがあります

曖昧なエントリを見つけるには、ターミナルで `claude doctor` を実行し、`Sandbox network domain entries have unreliable spellings` という警告を確認します。曖昧なエントリはそれぞれ角括弧形式で書き直してください。

48.4.4 OS レベルの適用

サンドボックス化された Bash ツールは、オペレーティングシステムのセキュリティプリミティブを使用します。

- **macOS**: サンドボックスの適用に Seatbelt を使用します
- **Linux**: 分離に bubblewrap を使用します
- **WSL2**: Linux と同じく bubblewrap を使用します

`@anthropic-ai/sandbox-runtime` パッケージを単独で実行して、Claude Code プロセスをラップすることもできます。サンドボックスランタイムを参照してください。

48.5 サンドボックスが権限と権限モードにどのように関連するか

サンドボックス、**権限ルール** (ページ 1039)、および権限モードは補完的なレイヤーです。以下のセクションでは、サンドボックスが各レイヤーとどのように相互作用するかについて説明します。

48.5.1 権限ルール

権限ルールとサンドボックスは異なるものを制御します。

- **権限ルール**は Claude Code が使用できるツールを制御し、ツールが実行される前に評価されます。Bash、Read、Edit、WebFetch、MCP、およびその他のツールを含むすべてのツールに適用されます。ただし、deny ルールまたは ask ルールは、他のツールが残っている間は `EndConversation` をブロックできません。
- **サンドボックス**は OS レベルの強制を提供し、シェルコマンドがファイルシステムおよびネットワークレベルでアクセスできるものを制限します。Bash、PowerShell、および Monitor コマンドとその子プロセスにのみ適用されます。

この 2 つのレイヤーは、強制方法も異なります。Claude Code は、コマンド文字列に基づいて、またはオートモードでは別の分類器がコマンドが安全かどうかについての判断に基づいて、コマンドが実行される前に権限の決定を評価します。オペレーティングシステムは、実行中のプロセスにサンドボックス境界を強制するため、モデルが実行することを選択したものに関係なく、また許可されたコマンドがその名前が示唆するもの以上のことを行う場合でも、それが保持されます。

ファイルシステムおよびネットワーク制限は、サンドボックス設定と権限ルールの両方を通じて構成されます。

設定またはルール	機能
`sandbox.filesystem.allowWrite`	作業ディレクトリ外のパスへのサブプロセス書き込みアクセスを許可します
`sandbox.filesystem.denyWrite` および `sandbox.filesystem.denyRead`	特定のパスへのサブプロセスアクセスをブロックします
`sandbox.filesystem.allowRead`	`denyRead` 領域内の特定のパスの読み取りを再度許可します

次のページに続く

表 6 – 前のページからの続き

設定またはルール	機能
sandbox.filesystem.disabled (ページ 1080)	ネットワーク分離を維持しながら、ファイルシステムレイヤーを完全にオフにします
Edit allow ルール	sandbox.filesystem.allowWrite と同じ方法で、特定のパスへの書き込みアクセスを許可します
Read および Edit deny ルール	特定のファイルまたはディレクトリへのアクセスをブロックします
WebFetch(domain:...) allow および deny ルール	ドメインアクセスを制御します
サンドボックス allowedDomains	Bash コマンドが到達できるドメインを制御します
サンドボックス deniedDomains	より広い allowedDomains ワイルドカードが許可する場合でも、特定のドメインをブロックします

サンドボックス設定と権限ルールの両方からのパスとドメインは、最終的なサンドボックス構成にマージされます。

claude-code リポジトリの examples ディレクトリには、サンドボックス固有の例を含む、一般的なデプロイメントシナリオ用のスターター設定構成が含まれています。これらを出発点として使用し、ニーズに合わせて調整してください。

48.5.2 権限モード

/sandbox は権限モードではありません。権限モードは、ツール呼び出しが実行されるかどうか、および最初にプロンプトが表示されるかどうかを決定しますが、サンドボックスは Bash コマンドが実行されたら何にアクセスできるかを制限します。制御対象と、アクション単位のプロンプトに代わるものが異なります。

	制御対象	プロンプトに代わるもの
/sandbox	Bash コマンドが実行されたら何にアクセスできるか	**オートアロー モード** (ページ 1074) のサンドボックス境界自体
オートモード	各ツール呼び出しが実行されるかどうか	アクションをレビューする分類器
--dangerously-skip-permissions	各ツール呼び出しが実行されるかどうか	なし。保護されたパスチェックもスキップされます。モードが自動承認しないアクションは引き続き適用されます

サンドボックスの**オートアロー モード** (ページ 1074) はオートモードとは別です。オートアロー はサンドボックス境界がそれらを含むため Bash コマンドを承認し、オートモードはアクションをレビューするために分類器を使用します。この 2 つは独立して機能し、**サンドボックス モード** (ページ 1074) の下にリストされている例外を除いて組み合わせることができます。無人実行の分離境界を選択するには、サンドボックス環境を参照してください。各フラグを開始する一般的な権限モードとサンドボックスペアリングのテーブルについては、一般的なセットアップを参照してください。

48.6 組織のサンドボックスを設定する

管理者はすべてのユーザーにサンドボックス化を要求し、開発者がポリシーを広げるのを防ぎ、サンドボックストラフィックを企業プロキシを通じてルーティングできます。

48.6.1 管理設定でサンドボックス化を実施する

すべての開発者にサンドボックスを要求するには、管理設定を通じて sandbox キーを配信します。MDM で管理されるファイルまたは claude.ai のサーバー管理設定を通じて配信します。

以下の管理設定はサンドボックスを有効化し、プラットフォームがサポートされていない場合や依存関係が不足している場合は Claude Code の起動を拒否し、モデルがサンドボックス外でコマンドを再試行するのを防止します。

```
{
  "sandbox": {
    "enabled": true,
    "failIfUnavailable": true,
    "allowUnsandboxedCommands": false
  }
}
```

enabled 以外の 2 つのキーは、サンドボックスがコマンドを実行できない場合に何が起こるかを制御します。

- failIfUnavailable：Linux の bubblewrap などの依存関係が不足している場合、サンドボックス化されていない実行にフォールバックするのではなく、Claude Code の起動をブロックします
- allowUnsandboxedCommands: false：Claude Code は dangerouslyDisableSandbox エスケープハッチを無視するため、サンドボックス内でコマンドが失敗しても、Claude はそれをサンドボックス外で再試行できません

これらと併せて、次の追加を検討してください。

- 分離なしで実行する必要がある組織承認済みのツールについて excludedCommands を追加します。この設定により、**リポジトリの設定でコマンドをサンドボックスの外に出すことができなくなる** (ページ 1094) ためです
- ~/.aws や ~/.ssh などの認証情報ディレクトリと、秘密の環境変数について *sandbox.credentials* (ページ 1082) エントリを追加します。デフォルトの読み取りポリシーではこれらが引き続き許可されるためです

この設定は Claude が実行するコマンドをサンドボックス化します。開発者は依然として シェルモードプロンプト <interactive-mode--shell-mode-with-prefix>でコマンドを入力し、Claude Code の外の任意のターミナルで既に持っているのと同じアクセス権でサンドボックス外で実行できます。入力されたコマンドがサンドボックス内で実行されるセッションについては、*strict* **サンドボックスモード** (ページ 1076) を参照してください。

サンドボックスはネイティブ Windows では実行されないため、failIfUnavailable が設定されていると、それらのマシンでは Claude Code が起動時に終了します。フリートに Windows ホストが含まれている場合は、次の方法を取れます。

- **オペレーティングシステムごとに設定を配信する**：MDM を通じて、または管理設定ファイルとして、macOS と Linux のマシンにのみデプロイします。サーバー管理設定は組織内のすべてのユーザーに適用されます
- **Windows ユーザーをサポートされている環境に移行する**：WSL2 またはコンテナ内で Claude Code を実行してもらいます

48.6.2 開発者がポリシーを広げるのを防ぐ

管理設定で `enabled` や `failIfUnavailable` などのブール値キーを設定した場合、Claude Code は管理値を使用し、開発者がローカルで設定したものを無視します。`allowRead` などの配列キーの場合、Claude Code はセッションが読み込むスコープからエントリをマージするため、そのキーがロックの対象になっていない限り、開発者はポリシーを広げるエントリを追加できます。

管理設定で設定されていない限り、開発者のユーザー設定または `--settings` で次のキーをオンにできます。サンドボックスが**管理者必須** (ページ 1094) でない限り、リポジトリの `.claude/settings.json` でもオンにできます。いずれもサンドボックスを弱めるため、使用させたくない場合は管理設定で `false` に設定してください。

- `enableWeakerNestedSandbox`
- `enableWeakerNetworkIsolation`
- `network.allowAllUnixSockets`
- `network.allowLocalBinding`
- `allowAppleEvents`（リポジトリではオンにできません）

管理設定で `allowManagedReadPathsOnly` を `true` に設定して、管理設定からの `allowRead` エントリのみが尊重されるようにします。これにより、開発者が組織承認済みのパスを超えて読み取りアクセスを広げるのを防止します。

ネットワークドメインを同じ方法で管理値にロックするには、`allowManagedDomainsOnly` を設定します。このロックがオンの場合、**プロキシポート** (ページ 1096) を設定できるのは管理設定のみです。

管理設定が `sandbox.filesystem` を設定するか、`"mode": "deny"` を含む `sandbox.credentials.files` エントリをリストする場合、管理設定のみが *`filesystem.disabled`* (ページ 1080) を設定できるため、開発者は管理者がデプロイしたファイルシステム制限をオフにすることはできません。有効な `mask` エントリはキーをロックしません。**どの設定がそれを無効にできるか** (ページ 1081) を参照してください。

管理者必須のサンドボックスにおけるリポジトリ設定

次のいずれかの設定が有効な間、サンドボックスは管理者必須になります。

- `allowUnsandboxedCommands` が管理設定で `false` に設定されている場合、または管理設定で `true` に設定されていない限り `--settings` フラグで `false` に設定されている場合
- `allowManagedDomainsOnly` が管理設定で `true` に設定されている場合

これらの設定はサンドボックスをオンにしないため、`enabled` も設定してください。

サンドボックスが管理者必須である間、Claude Code はサンドボックスを緩める設定を、管理設定、--settings フラグ、および各開発者の ~/.claude/settings.json からのみ取得します。リポジトリの .claude/settings.json および .claude/settings.local.json にある次の設定は無視されます。

リポジトリの設定	Claude Code が無視するもの
excludedCommands、ignoreViolations、network.allowedDomains、network.allowUnixSockets、network.allowMachLookup、network.httpProxyPort、network.socksProxyPort	すべてのエントリ
filesystem.allowWrite、Edit(...) 許可ルール、permissions.additionalDirectories	各エントリがサンドボックス化されたコマンドに与える書き込みアクセス。Claude のファイルツールは引き続き Edit(...) ルールと追加ディレクトリに従います
WebFetch(domain:...) 許可ルール	各ルールがサンドボックスの許可リストに追加するホスト。WebFetch ツールは引き続きそのルールに従います
enableWeakerNestedSandbox、enableWeakerNetworkIsolation、network.allowAllUnixSockets、network.allowLocalBinding	true。false は引き続き適用されます
enabled、failIfUnavailable	開発者の ~/.claude/settings.json が true を設定している場合の false
filesystem.allowRead	管理設定、--settings、またはユーザー設定で読み取りが拒否されているパスまたはその配下のエントリ、あるいはそれに一致する可能性のある glob

サンドボックスが管理者必須である間も、次の設定は引き続き適用されます。

- **リポジトリのファイル内**：deny エントリと autoAllowBashIfSandboxed の値。リポジトリによる変更を防ぐには、管理設定でこのキーを設定してください
- **開発者自身の設定内**：表にある設定は、allowManagedDomainsOnly などの管理専用ロックの対象でない限り、~/.claude/settings.json または --settings から引き続き適用されます。excludedCommands や filesystem.allowWrite など、そのほとんどには管理専用ロックがありません

管理設定でサンドボックス化を実施する (ページ 1093) の設定により、サンドボックスは管理者必須になります。リポジトリからは指定できないため、承認済みのツールに必要な excludedCommands、allowWrite、およびソケットのエントリは管理設定に追加してください。

Claude Code v2.1.285 以降が必要です。v2.1.282 から v2.1.284 では、同じ設定によって Claude Code はリポジトリの excludedCommands エントリを無視していました。

管理者必須のサンドボックスがなくても適用されるロック

一部の設定は、サンドボックスが管理者必須でない場合でも、1 つの制限を直接上書きするリポジトリのキーを Claude Code に無視させます。各設定がこの効果を持つのは、その行に記載されたファイルで設定した場合のみであり、リポジトリのその他のサンドボックス設定は引き続き適用されます。Claude Code v2.1.285 以降が必要です。

設定	設定する場所	Claude Code がリポジトリの設定で無視するもの
network.deniedDomains または WebFetch(domain:...) 拒否ルール	管理設定、--settings	httpProxyPort と socksProxyPort
network.strictAllowlist	管理設定、--settings、ユーザー設定	プロキシポート、allowedDomains、および WebFetch(domain:...) 許可ルール
filesystem.denyRead、Read(...) 拒否ルール、または credentials.files エントリ	管理設定、--settings	管理設定、--settings、またはユーザー設定で読み取りが拒否されているパスまたはその配下にある allowRead、allowWrite、Edit(...) 許可、または additionalDirectories のエントリ、あるいはそれに一致する可能性のある glob

これらのロックは、サンドボックス化されたコマンドがアクセスできる範囲を変更します。WebFetch ツールと Claude のファイルツールは、引き続きリポジトリのルールと追加ディレクトリに従います。

48.6.3 カスタムプロキシ設定

独自のツールでサンドボックストラフィックを検査、フィルタリング、またはログに記録するには、組み込みのサンドボックスプロキシを、同じマシン上で実行する独自のプロキシに置き換えます。

ネットワーク上の別の場所にある企業プロキシを通じてサンドボックストラフィックをルーティングするには、代わりに **ネットワーク分離** (ページ 1087) の **企業プロキシ** の項目で説明されているように HTTPS_PROXY を設定します。そうすることで、Claude Code の許可リストが引き続き適用されます。

サンドボックス化されたコマンドをプロキシに向けるには、サンドボックス設定でプロキシがリッスンする localhost のポートを設定します。

```
{
  "sandbox": {
    "network": {
      "httpProxyPort": 8080,
      "socksProxyPort": 8081
    }
  }
}
```

ポートを設定し、さらに HTTPS_PROXY または HTTP_PROXY も設定した場合、Claude Code はサンドボックス化されたコマンドが独自のプロキシに送信したものを、これらの変数で指定されたプロキシに転送しません。企業プロキシにアクセスするには、独自のプロキシがそこへ転送するように設定してください。

どのファイルでポートを設定できるかは、その他のサンドボックス設定によって異なります。最初に一致するケースが適用されます。

- `allowManagedDomainsOnly` **がオンの場合**：管理設定のみ
- **サンドボックスが管理者必須** (ページ 1094) **である場合、または**より限定的なネットワークロック (ページ 1095) **が適用される場合**：管理設定、`--settings`、およびユーザー設定
- **それ以外の場合**：任意の設定ファイル

Claude Code はそれ以外の場所で設定されたポートを無視します。v2.1.285 より前は、任意の設定ファイルでポートを設定できました。

警告:

いずれかのポートが適用されると、そのプロキシに送信されるすべてのものをフィルタリングする責任は独自のプロキシが負います。`allowedDomains`、`deniedDomains`、`strictAllowlist`、承認プロンプト、**ローカルアドレスのチェック** (ページ 1089) など、Claude Code 自体のネットワーク制御はそのトラフィックには適用されなくなります。サンドボックス化されたコマンドはどちらのプロキシにも接続できるため、ポートを 1 つだけ設定した場合、もう一方のプロキシにおける Claude Code のドメインリストでは、コマンドが独自のプロキシを通じてアクセスする先を制限できません。

48.7 トラブルシューティング

一部のコマンドは、サンドボックス外では機能するにもかかわらず、サンドボックス内では失敗します。症状やエラーメッセージに一致する見出しを探してください。

組織のサンドボックスが**管理者必須** (ページ 1094) の場合、Claude Code はプロジェクトの設定ファイル内にある、これらの修正で挙げる設定を無視します。そのため、すべてのプロジェクトで適用される `~/.claude/settings.json` に保存してください。それでも修正が効果を持たない場合は、組織の管理設定がそのキーを設定している可能性があります。

`excludedCommands` パターンを追加する修正では、そのパターンに一致するコマンドからサンドボックスが外れます。**除外されたコマンドでできること** (ページ 1078) を参照してください。

48.7.1 コマンドがホスト許可なしエラーで失敗する

多くの CLI ツールは特定のホストに到達する必要があります。プロンプトが表示されたらホストを承認するか、`allowedDomains` に追加してください。組織が `allowManagedDomainsOnly` で許可リストをロックしている場合はプロンプトが表示されないため、管理者にホストの追加を依頼してください。

48.7.2 `jest` がハングまたは失敗する

`watchman` はサンドボックスと互換性がありません。代わりに `jest --no-watchman` を実行してください。

48.7.3 Go ベースの CLI が macOS で TLS 検証に失敗する

`gh`、`gcloud`、`terraform` などのツールは *Seatbelt* (ページ 1091) の下で TLS 検証に失敗する可能性があります。これらのツールをサンドボックス外で実行するには、各ツールのパターン（`gh *` など）を *`excludedCommands`* (ページ 1078) に追加してください。そのツールはユーザーの完全なアクセス権と保存された認

証情報で実行されます。httpProxyPort を MITM プロキシとカスタム CA で使用している場合は、代わりに enableWeakerNetworkIsolation を true に設定してください。

48.7.4 open、osascript、またはブラウザベースの認証フローが macOS でエラー -600 で失敗する

サンドボックスはデフォルトで Apple Events をブロックします。ユーザー、管理、または CLI 設定で allowAppleEvents を true に設定して、それらを許可してください。Claude Code はプロジェクト設定ではこのキーを無視します。

allowAppleEvents を有効にするとコード実行の分離が削除されます。サンドボックス化されたコマンドはユーザープロンプトなしで他のアプリケーションをサンドボックス化されていない状態で起動でき、macOS オートメーション同意プロンプト（TCC）の対象となる実行中のアプリケーションに AppleScript コマンドを送信できるためです。または、open * などのパターンを *excludedCommands* (ページ 1078) に追加してください。その場合、各 open 呼び出しは権限フローを経由します。また、open は Claude が書いたものを含め、任意のファイルやアプリを起動できます。

48.7.5 docker コマンドが失敗する

docker はサンドボックスと互換性がありません。必要な docker コマンドを、docker compose * などの excludedCommands パターンでサンドボックス外に出してください。除外された docker コマンドが到達できる範囲については、*excludedCommands* **でサンドボックス外でコマンドを実行する** (ページ 1078) で説明しています。パターンを狭くするほど、サンドボックス外に出るコマンドは少なくなります。

48.7.6 pbcopy、xclip、または wl-copy がクリップボードを更新しない

pbcopy、xclip、wl-copy のクリップボードユーティリティはサンドボックス内からシステムクリップボードに到達できない場合があり、その場合はパイプされたテキストが到達しません。

Claude の出力をクリップボードに配置するには、Claude に応答で出力するよう依頼してから、/copy を実行してください。/copy はサンドボックス化されたコマンドではなく Claude Code プロセスからクリップボードに書き込みます。

Claude がテキストをこれらのツールの 1 つにパイプする場合、ツールを excludedCommands に追加しても、それだけではその呼び出しはサンドボックス外に出ません。

48.7.7 git コマンドが unable to unlink old で失敗する

git merge、git checkout などのコマンドは、サンドボックスが書き込みを拒否するファイルを置き換える必要がある場合に unable to unlink old で失敗します。Linux と WSL2 ではエラーは Read-only file system で終わります。そのファイルは次のいずれかの場所にある可能性があります：

- .claude/skills などの**保護されたパス** (ページ 1087) の下
- denyWrite エントリの 1 つの下
- サンドボックスがコマンドに書き込みを許可するディレクトリの外

失敗後、Claude は**コマンドをサンドボックス外で再実行することを提案** (ページ 1075) する場合があります。その再試行を承認するか、別のターミナルで git コマンドを自分で実行してください。`allowUnsandboxedCommands` を `false` に設定している場合、Claude は再試行を提案できないため、コマンドを自分で実行してください。

48.7.8 Bubblewrap がコンテナ内で起動に失敗する

非特権コンテナでは、*bubblewrap* (ページ 1091) は新しい `/proc` ファイルシステムをマウントできないため、サンドボックス化されたコマンドは `bwrap` エラー（`Can't mount proc on /newroot/proc: Operation not permitted` など）で失敗します。`enableWeakerNestedSandbox` を `true` に設定して、サンドボックスが代わりにコンテナの既存の `/proc` をバインドマウントするようにしてください。この設定は、外部コンテナが既に必要な分離境界を提供する場合にのみ使用してください。新しい `/proc` マウントであれば隠されるプロセス情報を、この設定はサンドボックス化されたコマンドに公開するためです。

48.7.9 0 バイトの読み取り専用ファイルが .claude 設定パスに表示され、「はい、今後は聞かない」が保存されない

Linux と WSL2 では、サンドボックスはサンドボックス化されたコマンドが実行されている間に、まだ存在しないファイルに対する書き込み拒否を保持するために、そこに 0 バイトの読み取り専用プレースホルダーを作成します。サンドボックスはその後プレースホルダーを削除します。SIGKILL などによってセッションがそのクリーンアップが実行される前に強制終了された場合、プレースホルダーは残ります。後のセッションは毎回起動時にそれらを読み取り専用で再度バインドするため、プレースホルダーが残っている箇所では、権限の選択を保存するなどの設定書き込みが失敗します。

ターミナルで `claude doctor` を実行して、残されたプレースホルダーファイルをリストアップしてください。`Stale sandbox mask files left by a killed session` 警告はその一部の名前を表示し、残りをカウントします。そのプロジェクトで他の Claude Code セッションが実行されていない間に、`rm` で各ファイルを削除してください。v2.1.257 より前では、Claude Code は同じプレースホルダーを残していましたが、警告していませんでした。

48.7.10 サンドボックスがオンの状態で SSH 経由の git が失敗する

macOS では、SSH リモートに対する `git fetch`、`git pull`、`git push` は、ホストが許可されていてもサンドボックス内で失敗します。Linux と WSL2 では、ホストが許可されれば動作します。Claude Code は git の SSH 接続を**サンドボックスプロキシ** (ページ 1087) 経由でトンネリングしますが、macOS のトンネルはそのプロキシに対して認証できません。

Linux と WSL2 でそれでも接続が失敗する場合は、以下を確認してください：

- **ホストがポート 22 で許可されている**：`"git.example.com"` のようにポートを指定しない `allowedDomains` エントリで対象になります
- **企業プロキシがポート 22 を許可している**：ネットワークでアップストリームプロキシが必要な場合、トンネルもそれを経由します
- **鍵がファイルとして読み取り可能である**：サンドボックスは `ssh-agent` ソケットをブロックする場合があり、`~/.ssh` に対する `denyRead` または `credentials` エントリは鍵ファイルを隠します

macOS では、リモートを HTTPS に切り替えてください。これには個人用アクセストークンなどの HTTPS 認証情報が必要です：

```
git remote set-url origin https://git.example.com/example-org/example-repo.git
```

SSH リモートを維持する必要がある場合は、*excludedCommands* (ページ 1078) で git のネットワークコマンドをサンドボックス外に出してください：

```
{
  "sandbox": {
    "excludedCommands": ["git fetch *", "git pull *", "git push *"]
  }
}
```

これらのエントリは `git push origin main` に一致します。`cd` を追加する呼び出し、`git -C` を使用する呼び出し、またはコマンド置換を含む呼び出しは、サンドボックス内のままです。除外された git コマンドは、`allowedDomains` にあるホストだけでなく、任意のホストに到達できます。

SSH 経由の単純な `ssh`、`scp`、`rsync` は、**データベースクライアントの項目** (ページ 1100) で説明している理由で失敗します。

48.7.11 データベースクライアントやその他の非 HTTP ツールが許可されたホストに到達できない

プロキシの環境変数を無視するツールは、`allowedDomains` にあるホストであっても、サンドボックス内から接続できません。サンドボックス化されたコマンドには**ネットワークへの直接の経路がない** (ページ 1087) ため、独自に接続を開くツールは失敗します。ほとんどのデータベースドライバー、単純な `ssh`、UDP を使用するツールはこのように動作します。

失敗はネットワークエラーまたは名前解決エラーのように見えます：

- **macOS**：`Operation not permitted`、または `Could not resolve host` などの名前解決エラー
- **Linux と WSL2**：`Network is unreachable`、または `Temporary failure in name resolution` などの名前解決エラー

プロキシを使用するツールは、ホストが許可されていない場合に異なる形で失敗します。ネットワークのプロンプトが表示されるか、ツールがプロキシから `403` レスポンスを受け取ります。

ツールが接続できるようにするには、それを必要とするコマンドを *excludedCommands* (ページ 1078) でサンドボックス外で実行してください。この例では 1 つのスクリプトを除外し、*ask* **ルール** (ページ 1039) を追加して、実行ごとに承認するようにしています：

```
{
  "sandbox": {
    "excludedCommands": ["python scripts/load_orders.py *"]
  },
  "permissions": {
    "ask": ["Bash(python scripts/load_orders.py *)"]
```

(次のページに続く)

(前のページからの続き)

```
  }
}
```

スクリプトはユーザーの完全なアクセス権で実行され、Claude は作業ディレクトリ内にあるスクリプトを編集できるため、プロンプトが表示されたらスクリプトを確認してください。

48.7.12 コマンドが localhost 上のサーバーに到達できない

デフォルトでは、サンドボックス化されたコマンドは、開発サーバーやコンテナ内のデータベースなど、マシン上でサンドボックス外で実行されているサーバーに直接接続できません。変更できる内容はプラットフォームによって異なります：

- **macOS**：`network.allowLocalBinding` を `true` に設定します。これにより、サンドボックス化されたコマンドはネットワークポートでリッスンし、localhost の任意のポートに接続できるようになります。これには、そこでリッスンしている他のすべてのサービスが含まれます。その結果、認証を必要としない localhost サービス（デバッガーなど）がサンドボックス外でコマンドの代わりに動作できるようになり、非ループバックアドレスでリッスンするコマンドは他のマシンからの接続を受け入れます
- **Linux と WSL2**：サンドボックス化されたコマンドの `localhost` はそのコマンド専用です。コマンドはポートでリッスンでき、自身が起動したサーバーに到達できます。`localhost` または `127.0.0.1` への直接接続はホスト上のサーバーには到達せず、`allowLocalBinding` は効果がありません。ホストのサーバーを必要とするコマンドは、*excludedCommands* (ページ 1078) でサンドボックス外で実行してください。そこではファイルシステムやネットワークの制限はありません。サンドボックスプロキシを経由する接続については、**ローカルアドレスに解決されるホスト名** (ページ 1089) を参照してください

この例では macOS でこの設定をオンにします：

```
{
  "sandbox": {
    "network": {
      "allowLocalBinding": true
    }
  }
}
```

`localhost` に対する `allowedDomains` エントリはプロキシを経由する接続に適用されるため、直接接続には影響しません。Claude Code はサンドボックス化されたコマンドに `NO_PROXY` を設定し、プロキシ経由ではなく `localhost` に直接接続するようにしています。また、このエントリは、プロキシを使用するコマンドに対して、マシンの localhost のすべてのポートを公開します。`127.0.0.1` を指す開発用ホスト名については、**許可されたホスト名が** *resolved to a loopback address* **で拒否される** (ページ 1102) を参照してください。

48.7.13 許可されたホスト名が resolved to a loopback address で拒否される

サンドボックスプロキシは、**ローカルアドレスに解決される** (ページ 1089) 許可されたホスト名を拒否します。これは 127.0.0.1 を指す myapp.test などの開発用の名前に影響します。コマンドは 403 レスポンスを受け取り、その本文には Connection to myapp.test blocked: resolved to a loopback address のようにアドレスの種類が示されます。

名前の解決先の IP アドレスをホスト名と並べて allowedDomains に追加し、それぞれにサーバーがリッスンするポートを指定してください：

```
{
  "sandbox": {
    "network": {
      "allowedDomains": ["myapp.test:3000", "127.0.0.1:3000"]
    }
  }
}
```

ポートを指定しない IP アドレスのエントリでは、サンドボックス化されたコマンドがそのアドレスでリッスンしているすべてのサービスに到達できるようになります。

v2.1.284 より前では、プロキシは許可されたホスト名がどのアドレスに解決されても接続していました。

48.7.14 /sandbox が Sandbox settings are overridden by a higher-priority configuration で失敗する

上位の**設定レベル** (ページ 719) が sandbox.enabled、sandbox.autoAllowBashIfSandboxed、または sandbox.allowUnsandboxedCommands を設定している場合、/sandbox はパネルを開く代わりに Error: Sandbox settings are overridden by a higher-priority configuration and cannot be changed locally. を出力します。パネルは選択内容を .claude/settings.local.json に保存しますが、そこに保存された値はそれらのレベルを上書きできません。

管理設定と --settings はローカル設定より優先されます。このセッションでどれが読み込まれたかを確認するには、/status を実行して Setting sources 行を確認してください：

- Command line arguments：*--settings* (ページ 717) を指定して Claude Code を起動した場合は、渡したファイルまたは JSON がこれらのキーのいずれかを設定しているか確認してください。設定している場合は、そこで値を変更するか、これらのキーを含めずに Claude Code を再起動してください。
- Enterprise managed settings：組織の管理設定が読み込まれています。それらがこれらのキーのいずれかを設定している場合、そのキーは /sandbox からも、ユーザーが管理するどの設定ファイルからも変更できないため、管理者に問い合わせてください。

48.8 制限事項

サンドボックス化はリスクを軽減しますが、完全な分離境界ではありません。ハードセキュリティ制御として依存する前に、以下の制限事項を確認してください。

48.8.1 セキュリティ上の制限

- **ネットワークフィルタリング**：サンドボックスは、プロセスが接続できるドメインを制限します。デフォルトでは、組み込みプロキシは発信トラフィックの TLS を終端または検査しないため、暗号化された接続の内容は検査されません。実験的な `network.tlsTerminate` 設定は、*mask* **認証情報置換** (ページ 1083) のためにプロキシで TLS を終了しますが、コンテンツフィルタリングは追加しません。ポリシーで許可されるのは信頼できるドメインのみであることを確認する責任があります。

警告:

`github.com` などの広いドメインを許可すると、データ流出のパスが作成される可能性があります。プロキシは TLS を検査せずにクライアント提供のホスト名から許可決定を行うため、サンドボックス内で実行されるコードはドメインフロンティングまたは同様の技術を使用して許可リスト外のホストに到達する可能性があります。脅威モデルがより強力な保証を必要とする場合は、TLS を終了してトラフィックを検査し、CA 証明書をサンドボックス内にインストールする **カスタムプロキシ** (ページ 1096) を設定してください。より強力な TLS 対応ネットワーク分離は開発の活発な領域です。

- **Unix ソケットを通じた権限昇格**：`allowUnixSockets` 設定は、サンドボックスバイパスにつながる可能性のあるシステムサービスへのアクセスを不注意に付与する可能性があります。たとえば、`/var/run/docker.sock` へのアクセスを許可すると、Docker ソケットを通じてホストシステムへのアクセスが効果的に付与されます。サンドボックスを通じて許可する Unix ソケットを慎重に検討してください。

- **ファイルシステム権限昇格**：過度に広いファイルシステム書き込み権限は権限昇格攻撃を有効にする可能性があります。`$PATH` の実行可能ファイルを含むディレクトリ、システム設定ディレクトリ、またはユーザーシェル設定ファイル（`.bashrc` または `.zshrc`）への書き込みを許可すると、他のユーザーまたはシステムプロセスがこれらのファイルにアクセスするときに異なるセキュリティコンテキストでコード実行につながる可能性があります。

- **Linux サンドボックス強度**：Linux 実装は強力なファイルシステムとネットワーク分離を提供しますが、特権付き名前空間のない Docker 環境内で動作できるようにする `enableWeakerNestedSandbox` モードが含まれています。このオプションはセキュリティを大幅に弱め、追加の分離が別の方法で実施される場合にのみ使用する必要があります。

- **macOS での Apple Events**：macOS サンドボックスはデフォルトで Apple Events をブロックします。`allowAppleEvents` 設定はこの制限を解除して、`open` や `osascript` などのツールが動作するようにしますが、コード実行分離を削除します。サンドボックス化されたコマンドは、ユーザープロンプトなしで他のアプリケーションをサンドボックス化されていない状態で起動でき、実行中のアプリケーションに AppleScript コマンドを送信できます。これはアプリごとの macOS オートメーション同意プロンプト（TCC）の対象です。これはユーザー、管理、または CLI 設定からのみ有効です。プロジェクト設定では有効にできません。

48.8.2 スコープ

サンドボックスはシェルコマンドとその子プロセスを分離します。サンドボックスの対象外となるツールとヘルパープロセスは、**サンドボックス外で実行されるもの** (ページ 1070) に記載されています。コンピュータ使用とサブエージェントとサンドボックスの関係は次のとおりです。

- **コンピュータ使用**：Claude がアプリを開いてスクリーンを制御する場合、分離された環境ではなく実際のデスクトップで実行されます。アプリごとの権限プロンプトが各アプリケーションをゲートします。CLI でのコンピュータ使用または *Desktop* **でのコンピュータ使用** (ページ 146) を参照してください。
- **サブエージェント**：*subagents* (ページ 317) は親セッションと同じプロセスで実行され、同じサンドボックス設定を使用します。親セッションでサンドボックス化が有効な場合、サブエージェント内の Bash コマンドはサンドボックス化されます。
- **Mod**：mod は Claude Code 内で独自のコードを実行するプラグインであり、mod が起動するプロセスはサンドボックス外で実行されます。mod がアクセスできる範囲を参照してください。

警告:

効果的なサンドボックス化にはファイルシステムとネットワークの両方の分離が必要です。ネットワーク分離がない場合、侵害されたエージェントは SSH キーなどの機密ファイルを流出させる可能性があります。ファイルシステム分離がない場合、それが制限の緩いポリシーによるものであれ、**ファイルシステムレイヤーを無効化** (ページ 1080) したことによるものであれ、侵害されたエージェントはシステムリソースにバックドアを仕掛けてネットワークアクセスを取得する可能性があります。デフォルトを広げるときは、`allowWrite` パス、広い `allowedDomains` エントリ、または `excludedCommands` 例外が反対側の制限を元に戻さないことを確認してください。

48.9 関連項目

- Sandbox environments：組み込みサンドボックスと dev コンテナ、コンテナ、VM を比較する
- *Security* (ページ 1033)：包括的なセキュリティ機能とベストプラクティス
- *Permissions* (ページ 1039)：許可設定とアクセス制御
- All settings：すべての設定キー
- *CLI reference* (ページ 45)：コマンドラインオプション

第 49 章　データ使用

Anthropic の Claude のデータ使用ポリシーについて学習します

49.1 データポリシー

49.1.1 データトレーニングポリシー

コンシューマーユーザー（Free、Pro、Max プラン）：将来の Claude モデルの改善のためにデータを使用することを許可するかどうかを選択できます。この設定がオンの場合、Free、Pro、Max アカウントからのデータを使用して新しいモデルをトレーニングします（これらのアカウントから Claude Code を使用する場合も含みます）。

商用ユーザー：（Team および Enterprise プラン、API、サードパーティプラットフォーム、Claude Gov）既存のポリシーを維持します。Anthropic は、商用条件の下で Claude Code に送信されたコードまたはプロンプトを使用して生成モデルをトレーニングしません。ただし、カスタマーがモデル改善のためにデータを提供することを選択した場合は除きます（例えば、Developer Partner Program）。

49.1.2 Development Partner Program

Development Partner Program などを通じて、トレーニング用の資料を提供する方法に明示的にオプトインした場合、提供された資料を使用してモデルをトレーニングする可能性があります。組織管理者は、組織の Development Partner Program に明示的にオプトインできます。このプログラムは Anthropic ファーストパーティ API でのみ利用可能であり、Amazon Bedrock または Google Cloud の Agent Platform ユーザーは利用できないことに注意してください。

49.1.3 `/feedback` コマンドを使用したフィードバック

`/feedback` コマンドを使用して Claude Code に関するフィードバックを送信することを選択した場合、製品とサービスを改善するためにフィードバックを使用する可能性があります。`/feedback`、`/bug`、`/share` を通じて共有されたトランスクリプトは、同じパスを通じてレポートされ、5 年間保持されます。

Claude が作成したフィードバック では、Claude はフィードバックレポートを作成し、レビュー用にマシンにキューイングすることもできます。Claude Code は、ドラフトを送信することを選択するまで何も送信しません。送信されたドラフトは、他の `/feedback` レポートと同じ送信パスと保持期間を通じて処理されます。

49.1.4 セッション品質調査

Claude Code で「How is Claude doing this session?」プロンプトが表示されたときに、この調査に応答する場合（「Dismiss」を選択する場合を含む）、数値評価のみが記録されます。この調査の一部として、会話トランスクリプト、入力、出力、またはその他のセッションデータは収集または保存されません。サムズアップ/ダウンフィードバックまたは `/feedback` レポートとは異なり、このセッション品質調査は単純な製品満足度メトリックです。

数値評価プロンプトの後、「Can Anthropic look at your session transcript to help us improve Claude Code?」と尋ねる別の追加フォローアップが表示される場合があります。これは数値評価とは異なるオプションの 2 番目のステップです。

- **Yes**：会話トランスクリプト、サブエージェントトランスクリプト、ディスクからの生のセッションログファイルを Anthropic にアップロードします。既知の API キーとトークンパターンはアップロード前に削除されます。ソースコード、ファイルコンテンツ、およびその他の会話コンテンツはそのままアップロードされます。共有されたトランスクリプトは最大 6 ヶ月間保持されます。Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry、およびサインイン済みの Claude apps gateway セッションでは、Yes は同じペイロードを `~/.claude/feedback-bundles/` の下のローカルアーカイブに書き込みます。アップロードの代わりに、ファイルを転送するまで何もマシンから出ません。
- **No**：何も送信せずに拒否します
- **Don't ask again**：拒否し、今後のセッションでこのフォローアップが表示されなくなります

Yes を明示的に選択しない限り、何もアップロードされません。ゼロデータ保持 を設定している組織、または組織ポリシーで製品フィードバックが無効になっている組織、または `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` が設定されている組織は、このフォローアップを表示しません。数値評価プロンプトの後に送信されたセッショントランスクリプトを含む、この調査への応答は、データトレーニング設定に影響を与えず、AI モデルをトレーニングするために使用することはできません。

これらの調査を無効にするには、`CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1` を設定します。調査は、`DISABLE_TELEMETRY`、`DO_NOT_TRACK`、または `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` が設定されている場合にも無効になります。無効にする代わりに頻度を制御するには、設定ファイルで `feedbackSurveyRate` を `0` から 1 の間の確率に設定します。非必須トラフィックをブロックしているが、独自の *OpenTelemetry collector* (ページ 935) を通じて調査応答をキャプチャしている組織は、`CLAUDE_CODE_ENABLE_FEEDBACK_SURVEY_FOR_OTEL=1` を設定することで調査をオプトバックインできます。調査は、設定されたコレクターのみに数値評価をログします。トランスクリプト共有フォローアップおよび他のすべての Anthropic バウンドフィードバックトラフィックは無効のままです。

49.1.5 データ保持

Anthropic は、アカウントタイプと設定に基づいて Claude Code データを保持します。

コンシューマーユーザー（Free、Pro、Max プラン）：

- モデル改善のためのデータ使用を許可するユーザー：モデル開発とセキュリティ改善をサポートするための 5 年間の保持期間
- モデル改善のためのデータ使用を許可しないユーザー：30 日間の保持期間
- プライバシー設定は、claude.ai/settings/data-privacy-controls でいつでも変更できます。

商用ユーザー（Team、Enterprise、API）：

- 標準：30 日間の保持期間
- ゼロデータ保持：Claude for Enterprise の Claude Code で利用可能。ZDR は組織ごとに有効になります。新しい各組織は、アカウントチームによって個別に ZDR を有効にする必要があります
- ローカルキャッシング：Claude Code クライアントは、セッション再開を有効にするために、`~/.claude/projects/` の下にセッショントランスクリプトをプレーンテキストでローカルに 30 日間保存します。`cleanupPeriodDays` で期間を調整できます。application data を参照して、何が保存されているか、およびそれをクリアする方法を確認してください。

  Claude Desktop または Cowork で開始または最後に続行されたセッションのトランスクリプトは、デフォルトではその制限から除外されます。

Web 上の個別の Claude Code セッションはいつでも削除できます。セッションを削除すると、セッションのイベントデータが永久に削除されます。セッションの削除方法については、*Delete sessions* (ページ 233)を参照してください。

データ保持慣行の詳細については、Privacy Center を参照してください。

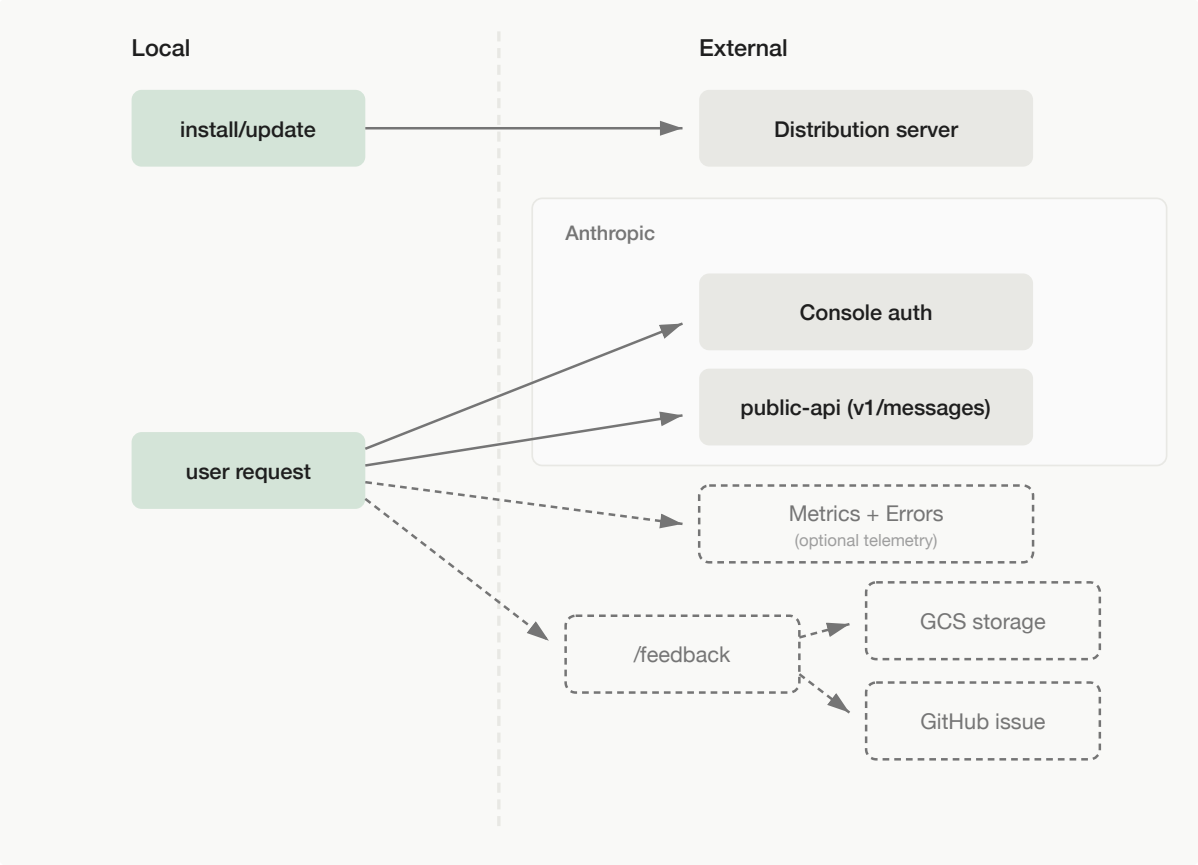
詳細については、Commercial Terms of Service（Team、Enterprise、API ユーザー向け）または Consumer Terms（Free、Pro、Max ユーザー向け）および Privacy Policy を確認してください。

49.2 データアクセス

すべてのファーストパーティユーザーの場合、**ローカル** *Claude Code* (ページ 1107) および **リモート** *Claude Code* (ページ 1109) に対してログされるデータについて詳しく知ることができます。Remote Control セッションは、すべての実行がマシン上で行われるため、ローカルデータフローに従います。接続中、セッショントランスクリプトは Connection and security に記載されているように、デバイス間で会話を同期するために Anthropic サーバーにも保存されます。リモート Claude Code の場合、Claude は Claude Code セッションを開始するリポジトリにアクセスします。Claude は接続したが、セッションを開始していないリポジトリにはアクセスしません。

49.3 ローカル Claude Code：データフローと依存関係

下の図は、Claude Code がインストール中および通常の操作中に外部サービスにどのように接続するかを示しています。実線は必須の接続を示し、破線はオプションまたはユーザーが開始するデータフローを表します。



Claude Code はローカルで実行されます。LLM と相互作用するために、Claude Code はネットワーク経由でデータを送信します。このデータには、すべてのユーザープロンプトとモデル出力が含まれ、TLS 1.2 以上で転送中に暗号化されます。Claude Code はほとんどの一般的な VPN と LLM プロキシと互換性があります。

保存時の暗号化はモデルプロバイダーによって異なります：

プロバイダー	保存時の暗号化
Anthropic API	インフラストラクチャレベルのディスク暗号化（AES-256）。サーバー側の永続化がないようにするには、Zero Data Retention を有効にしてください。
Amazon Bedrock	AWS 管理キーを使用した AES-256。AWS KMS 経由でカスタマー管理キーを利用可能です。
Google Cloud の Agent Platform	Google 管理の暗号化キー。CMEK を利用可能です。
Microsoft Foundry	デプロイメントのホスティングオプションによって異なります。Azure でホストされているデプロイメントの場合、プロンプトと完了は Azure 内に留まり、使用状況メタデータと Anthropic のセーフティシステムでフラグが付けられたコンテンツのみが Anthropic に送信されます。Anthropic でホストされているデプロイメントの場合、リクエストは AES-256 ディスク暗号化を使用して Anthropic インフラストラクチャにルーティングされます。

Claude Code は Anthropic の API 上に構築されています。API ロギング手順を含む API セキュリティコントロールの詳細については、Anthropic Trust Center のコンプライアンスアーティファクトを参照してください。

49.3.1 クラウド実行：データフローと依存関係

クラウドセッション (ページ 223) は、デフォルトではローカルではなく Anthropic 管理の仮想マシンで実行されます。組織が自己ホスト環境にルーティングするセッションは、制御するインフラストラクチャ上で実行されます。マシン上に留まるもの、および Anthropic に送信されるものについては、インフラストラクチャに留まるものを参照してください。Anthropic ホスト型クラウドセッションでは：

- **コードとデータストレージ：** リポジトリは分離された VM にクローンされます。Anthropic はセッショントランスクリプトを保存して、後でセッションに戻ることができるようにします。コードとセッションデータは、アカウントタイプの**保持および使用ポリシー** (ページ 1106) の対象となります
- **認証情報：** GitHub 認証情報は Anthropic のサーバーに暗号化されて保存され、VM に入ることはありません。VM からの GitHub トラフィックは、Anthropic プロキシを通じて送信され、サーバー側でそれらを接続します
- **ネットワークトラフィック：** すべてのアウトバウンドトラフィックは、監査ログと不正使用防止のためのセキュリティプロキシを通じて送信されます
- **セッションデータ：** プロンプト、コード変更、および出力は、ローカル Claude Code 使用と同じデータポリシーに従います

クラウド実行のセキュリティの詳細については、**セキュリティ** (ページ 1036) を参照してください。

49.4 テレメトリサービス

Claude Code は 2 種類の運用テレメトリを送信します。使用メトリクスとエラーレポートです。以下の環境変数を使用して各々をオフにすることができます。または `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` を設定することで、すべての非必須トラフィックを一度に無効にできます。`DISABLE_TELEMETRY` または `CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` を設定すると、Remote Control が依存するフィーチャーフラグ評価も無効になります。`DISABLE_ERROR_REPORTING` はそうではありません。

メトリクス：レイテンシ、信頼性、使用パターンは、Anthropic およびサードパーティのログインフラストラクチャに TLS 経由で送信されます。メトリクスにはコード、プロンプト、またはファイルパスが含まれることはありません。`DISABLE_TELEMETRY=1` を設定してオプトアウトします。

エラーレポート：Claude Code の内部からのエラーメッセージとスタックトレースは、TLS 経由でサードパーティのエラー追跡サービスに送信されます。Claude Code は、マシンから何かが出ていく前に、既知のシークレット、ファイルパス、メールアドレス、およびその他の個人情報のパターンを削除します。`DISABLE_ERROR_REPORTING=1` を設定してオプトアウトします。

エラーレポートは、以下のすべてが当てはまる場合にのみオンになります。

- Claude Pro または Max サブスクリプションでサインインしている
- Claude Code v2.1.198 以降を実行している
- Claude API に直接接続している

- 組織がゼロデータ保持または HIPAA 契約を持っていない

`/feedback` コマンドを実行すると、コードを含む会話履歴のコピーが Anthropic に送信されます。`/bug` および `/share` コマンドは同じパスを通じて送信されます。フィードバックダイアログからレポートを送信する前に、含める履歴の量を選択します。デフォルトは現在のセッションのみですが、同じプロジェクトの過去 24 時間または 7 日間の他のセッションも含めることができます。Claude Code は Claude が作成したフィードバック を同じパスを通じて送信します。ドラフトのレビュー画面で含めることを選択した場合、トランスクリプトを含めます。データは TLS 経由で転送中に暗号化され、Google Cloud Storage に保存されます。Google Cloud Storage はデフォルトで保存時のデータを暗号化します。オプションで、公開リポジトリに GitHub イシューが作成されます。オプトアウトするには、`DISABLE_FEEDBACK_COMMAND` 環境変数を 1 に設定します。

Amazon Bedrock や Google Cloud の Agent Platform などのサードパーティプロバイダーを使用している場合、または Anthropic 認証情報が設定されていない場合、`/feedback` はレポートを Anthropic に送信する代わりに、`~/.claude/feedback-bundles/` の下のローカルアーカイブに書き込みます。既知の API キーおよびトークンパターンは、アーカイブが書き込まれる前に削除されます。Anthropic アカウント担当者にファイルを送信するか、サポートリクエストに添付するまで、何もマシンから出ません。

49.5 API プロバイダーのデフォルト動作

デフォルトでは、Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry、または Claude Platform on AWS を使用する場合、エラーレポート、テレメトリ、およびバグレポートは無効になります。セッション品質調査と WebFetch ドメインセーフティチェックは例外であり、プロバイダーに関係なく実行されます。署名済みの Claude apps gateway セッションでは、Anthropic への使用分析、エラーレポート、および調査評価はゲートウェイ認証情報自体によって無効になり、それらを再度有効にする設定はありません。`CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC` を設定することで、調査を含むすべての非必須トラフィックをオプトアウトできます。この変数は WebFetch チェックや公式プラグインマーケットプレイスの自動インストールに影響を与えません。それぞれに独自のオプトアウトがあります。WebFetch の場合は *settings* (ページ 711) の `skipWebFetchPreflight`、マーケットプレイスの場合は `CLAUDE_CODE_DISABLE_OFFICIAL_MARKETPLACE_AUTOINSTALL` です。以下は完全なデフォルト動作です：

サービス	Claude API	Google Cloud の Agent Platform API	Amazon Bedrock API	Microsoft Foundry API	Claude Platform on AWS
メトリクス	デフォルトオン。`DISABLE_TELEMETRY=1` で無効にします。	デフォルトオフ。`CLAUDE_CODE_USE_VERTEX` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_BEDROCK` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_FOUNDRY` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_ANTHROPIC_AWS` は 1 である必要があります。
エラーレポート	v2.1.198 以降の Pro および Max サインインではオン、それ以外はオフ。`DISABLE_ERROR_REPORTING=1` で無効にします。	デフォルトオフ。`CLAUDE_CODE_USE_VERTEX` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_BEDROCK` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_FOUNDRY` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_ANTHROPIC_AWS` は 1 である必要があります。
Claude API（`/feedback` レポート）	デフォルトオン。`DISABLE_FEEDBACK_COMMAND=1` で無効にします。	デフォルトオフ。`CLAUDE_CODE_USE_VERTEX` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_BEDROCK` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_FOUNDRY` は 1 である必要があります。	デフォルトオフ。`CLAUDE_CODE_USE_ANTHROPIC_AWS` は 1 である必要があります。

次のページに続く

表 2 – 前のページからの続き

サービス	Claude API	Google Cloud の Agent Platform API	Amazon Bedrock API	Microsoft Foundry API	Claude Platform on AWS
セッション品質調査	デフォルトオン。CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 で無効にします。	デフォルトオン。CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 で無効にします。	デフォルトオン。CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 で無効にします。	デフォルトオン。CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 で無効にします。	デフォルトオン。CLAUDE_CODE_DISABLE_FEEDBACK_SURVEY=1 で無効にします。
WebFetch ドメインセーフティチェック	デフォルトオン。*settings* (ページ 711) で skipWebFetchPreflight: true で無効にします。	デフォルトオン。*settings* (ページ 711) で skipWebFetchPreflight: true で無効にします。	デフォルトオン。*settings* (ページ 711) で skipWebFetchPreflight: true で無効にします。	デフォルトオン。*settings* (ページ 711) で skipWebFetchPreflight: true で無効にします。	デフォルトオン。*settings* (ページ 711) で skipWebFetchPreflight: true で無効にします。

すべての環境変数は settings.json にチェックインできます（settings reference を参照）。

ホストプラットフォームが CLAUDE_CODE_PROVIDER_MANAGED_BY_HOST を設定する場合、メトリクスは Google Cloud の Agent Platform、Amazon Bedrock、Microsoft Foundry、および Claude Platform on AWS でデフォルトでオンになり、標準の DISABLE_TELEMETRY オプトアウトに従います。エラーレポートと /feedback レポートは、これらのプロバイダーではデフォルトでオフのままです。

49.5.1 WebFetch ドメインセーフティチェック

URL をフェッチする前に、WebFetch ツールは要求されたホスト名を api.anthropic.com に送信して、Anthropic が管理するセーフティブロックリストに対してチェックします。ホスト名のみが送信され、完全な URL、パス、またはページコンテンツは送信されません。Claude Code はチェックに合格したホスト名を 5 分間キャッシュし、ブロックされたまたは失敗したホスト名は次のリクエストで再チェックします。

このチェックは、使用するモデルプロバイダーに関係なく実行され、CLAUDE_CODE_DISABLE_NONESSENTIAL_TRAFFIC の影響を受けません。ネットワークが api.anthropic.com をブロックしている場合、WebFetch リクエストはドメインをホワイトリストに登録するか、*settings* (ページ 711) で skipWebFetchPreflight: true を設定するまで失敗します。チェックを無効にすると、WebFetch はブロックリストに相談せずに任意の URL を取得しようとするため、Claude が到達できるドメインを制限する必要がある場合は *WebFetch permission rules* (ページ 1056) と組み合わせてください。

第 50 章　法的および規制対応

Claude Code の法的契約、規制認証、およびセキュリティ情報。

50.1 法的契約

50.1.1 ライセンス

Claude Code の使用は、以下の対象となります。

- 商用条件 - Team、Enterprise、および Claude API ユーザー向け
- コンシューマー利用規約 - Free、Pro、および Max ユーザー向け

50.1.2 商用契約

Claude API を直接（1P）使用している場合でも、Amazon Bedrock または Google Cloud の Agent Platform を通じてアクセスしている場合（3P）でも、既存の商用契約が Claude Code の使用に適用されます。ただし、相互に別途合意した場合を除きます。

50.1.3 カスタマーは自社製品で Claude Code を提供できますか？

相互に別途合意した場合を除き、自社製品またはサービス（ホストされたサンドボックスやその他のエージェントインフラストラクチャなど）に Claude Code をプリインストールまたは実行することには、当社の 商用条件 に同意し、以下の条件に準拠する必要があります。

- **Claude Code バイナリは変更してはいけません。** Claude Code は Anthropic によって公開されたとおりにインストールおよび実行する必要があり、カスタマーは、Claude アカウントでのサインインまたはユーザー自身の API キーを許可するメソッドを含む、組み込まれた認証メソッドを削除、無効化、または制限することはできません。
- **カスタマーは、エンドユーザーに代わって Claude の使用に対して支払い、再販売、または仲介することはできません。** 各エンドユーザーは、自身の Anthropic API キー、Claude サブスクリプションプランの認証情報、または 3P 推論プロバイダーの認証情報（Amazon Bedrock、Google Cloud の Agent Platform、Microsoft Foundry）で認証する必要があります。その使用は、Anthropic との自身の契約、または第三者推論プロバイダーの場合は該当するプロバイダーとの契約に基づいて、エンドユーザーに直接請求されます。

Claude Code の名前とロゴの使用。 自社製品に Claude Code がプリインストールされている、または Claude Code を実行していることを、プレーンテキストで正確に述べることができます。ただし、Claude Code または Anthropic の名前またはロゴを、自社製品、機能、または会社名の一部として、自社ロゴで、または Anthropic が自社製品を構築、推奨、またはパートナーシップしていることを示唆する方法で使用すること

はできません。Anthropic の名前またはロゴのその他の使用は、当社の 商標ガイドライン に準拠し、当社の書面による許可が必要です。

Claude Code は、アクセスするプラットフォームに関係なく、Anthropic の標準条件（上記のライセンスおよび商用契約セクションを参照）に準拠し続けます。

50.2 規制対応

50.2.1 ヘルスケア規制対応（BAA）

カスタマーが Anthropic と業務提携契約（BAA）を締結しており、関連する組織に対して ゼロデータ保持（ZDR） が有効化されている場合、その BAA は Claude Code を通じて流れるカスタマーの API トラフィックに拡張されます。

50.3 使用ポリシー

50.3.1 許可される使用

Claude Code の使用は、Anthropic 使用ポリシーの対象となります。Pro および Max プランの広告表示される使用制限は、Claude Code および Agent SDK の通常の個人使用を想定しています。

50.3.2 認証と認証情報の使用

Claude Code は、OAuth トークンまたは API キーを使用して Anthropic のサーバーで認証します。これらの認証方法は異なる目的に対応しています。

- **OAuth 認証**は、Claude Free、Pro、Max、Team、および Enterprise サブスクリプションプランの購入者専用に設計されており、Claude Code および他のネイティブ Anthropic アプリケーションの通常の使用をサポートするために設計されています。ログインの手順については、Claude アカウントへのログインを参照してください。Claude Code が OAuth 認証を実行する方法については、**認証** (ページ 851)を参照してください。
- **開発者**が Claude の機能と相互作用する製品またはサービスを構築している場合（Agent SDK を使用している場合を含む）、Claude Console またはサポートされているクラウドプロバイダーを通じて API キー認証を使用する必要があります。Anthropic は、サードパーティの開発者が Claude.ai ログインを自社アプリケーションに提供したり、ユーザーに代わって Free、Pro、または Max プランの認証情報を通じてリクエストをルーティングしたりすることを許可していません。さらに、開発者は Claude.ai の認証情報またはセッショントークンを収集、保存、または仲介することはできません。Claude アカウントへのサインインは、Anthropic 独自のフローを通じて完了する必要があります。

これは、顧客が自身の API キーまたはサードパーティ推論プロバイダーの認証情報をどのように提供および管理するかを制限するものではありません。たとえば、開発環境、シークレットマネージャー、または機械イメージで API キーを設定して、顧客の認可されたユーザーが使用できるようにすることは許可されています。ただし、その結果の使用が、Anthropic（または該当するプロバイダー）との契約に基づいてキー所有者に請求され、上記で説明されているように再販売または仲介されていない場合に限ります。また、プラットフォームが上記の「顧客は自社製品で Claude Code を提供できますか？」で説明されているように Claude Code をホストしている場合を含め、エンドユーザーが自身の Claude サブスクリプションで未修正の Claude Code バイナリにサインインすることも妨げません。

Anthropic は、これらの制限を実施するための措置を講じる権利を留保し、事前通知なしにそうする場合があります。

ユースケースに対して許可される認証方法に関する質問については、営業に連絡してください。

50.4 セキュリティと信頼

50.4.1 信頼とセーフティ

詳細については、Anthropic Trust Center および Transparency Hub を参照してください。

50.4.2 セキュリティ脆弱性報告

Anthropic は HackerOne を通じてセキュリティプログラムを管理しています。このフォームを使用して脆弱性を報告してください。

© Anthropic PBC. All rights reserved. Use is subject to applicable Anthropic Terms of Service.

用語集

Claude Code の用語の定義。agentic loop、compaction、CLAUDE.md、hooks、subagents、MCP などのコア概念の意味を学びます。

この用語集は Claude Code の用語を定義しています。各エントリは、その概念について詳しく説明されているページにリンクしています。トークン、temperature、RAG などのモデルレベルの概念については、プラットフォーム用語集を参照してください。Claude Desktop のデスクトップ拡張機能、MCPB、DXT などの用語については、Claude ヘルプセンターを参照してください。

A

AGENTS.md

AI コーディングエージェント向けに作成するプロジェクト指示のマークダウンファイル。リポジトリに AGENTS.md があり、*CLAUDE.md* がない場合、Claude はこれをプロジェクト指示として読み込みます。別のファイルを追加する必要はありません。`/config` の **Project instructions** 設定を変更して、Claude が両方のファイルを読み込むか、CLAUDE.md のみを読み込むかを指定できます。AGENTS.md を直接読み込むには、Claude Code v2.1.277 以降が必要です。一部のセッションでは Claude が *AGENTS.md* **を読み込めない** (ページ 1010) ため、代わりに *CLAUDE.md* **からインポート** (ページ 1011) してください。

詳細情報: *AGENTS.md* (ページ 1008)

Agent teams

複数の独立した Claude Code セッションがチームリーダーによって調整され、共有タスクリストとピアツーピアメッセージングを備えています。単一のセッション内で実行され、親にのみレポートする *subagents* とは異なり、チームメイトはそれぞれ独自のコンテキストウィンドウを持ち、任意のメンバーと直接対話できます。Agent teams は実験的機能であり、デフォルトでは無効になっています。Enable agent teams を参照してください。

詳細情報: Run agent teams

Agentic coding

AI がファイルを読み取り、コマンドを実行し、自律的に変更を加えることができるワークフロー。あなたが見守ったり、リダイレクトしたり、立ち去ったりできます。これは、テキストのみで応答するチャットベースのアシスタントとは異なり、自分で適用する必要があります。Claude Code は agentic です。なぜなら、アドバイスするだけでなく、行動できる *tools* を持っているからです。

詳細情報: How Claude Code works

Agentic harness

言語モデルを有能なコーディングエージェントに変える、ツール、コンテキスト管理、実行環境。

Claude Code はハーネスです。Claude はその中のモデルです。ハーネスはファイルアクセス、シェル実行、権限ゲーティング、メモリロード、およびアクションをチェーンするループを提供します。

詳細情報: How Claude Code works

Agentic loop

Claude がすべてのタスクで実行するサイクル: コンテキストを収集し、アクションを実行し、結果を検証し、完了するまで繰り返します。各ツール使用は次のステップに情報を提供します。ループはいつでも中断してリダイレクトできます。*hooks*、*skills*、*MCP* を含むほとんどの拡張ポイントは、このループの特定のフェーズにプラグインします。

詳細情報: How Claude Code works

Artifact

Claude Code がセッションから claude.ai 上のプライベート URL に公開するライブでインタラクティブな Web ページ。出力を視覚的に確認したり、ターミナルテキストを読む代わりに共有したりできます。セッションが再公開されると、ページはその場で更新されます。Claude Code から作成した Artifact は、claude.ai の会話から作成した Artifact と同じギャラリーに表示されます。共有オプションはプランに依存します: Share an artifact を参照してください。

詳細情報: Share session output as artifacts

Auto memory

Claude が自分自身のために書いたメモ。あなたの修正と設定に基づいて、git リポジトリごとに `~/.claude/projects/` に保存されます。同じリポジトリのすべてのワークツリーは 1 つの auto memory ディレクトリを共有します。`MEMORY.md` インデックスの最初の 200 行または 25 KB がすべてのセッションの開始時にロードされます。Auto memory は、あなたが書く *CLAUDE.md* に対する Claude が書いた対応物です。

詳細情報: *Auto memory* (ページ 1013)

Auto mode

permission mode の一種。承認プロンプトを表示する代わりに、別の分類器モデルが各アクションをレビューするため、Claude Code はほとんどのアクションをあなたに尋ねることなく実行できます。Claude Code は、あなたの明示的な ask ルールに一致するアクションの前にはあなたに尋ねます。Claude Code v2.1.283 以降では、auto mode はインタラクティブターミナルと VS Code セッションの built-in starting permission mode であり、以前のバージョンでは Pro、Max、Team プランのみです。分類器はスコープエスカレーション、信頼されていないインフラストラクチャ、および *prompt injection* をブロックします。ツール結果はそれが見るものから削除されるため、ファイルまたは Web ページ内の悪意のあるコンテンツはそれを直接操作することはできません。

詳細情報: Eliminate prompts with auto mode

B

Bare mode

`--bare` を使用すると、Claude Code は hooks、skills、カスタムコマンド、subagents、インストール済み plugins、MCP servers、auto memory、CLAUDE.md を読み込まずに起動します。ただし、`--add-dir` で渡したディレクトリ内の skills は除きます。CI とスクリプト呼び出しで、すべてのマシンで同じ結果が必要な場合に推奨されます。

詳細情報: *Start faster with bare mode* (ページ 696)

Bundled skills
Claude Code に含まれるプロンプトベースのプレイブック。`/batch`、`/code-review`、`/debug`、`/loop` など。固定ロジックを実行する組み込みコマンドとは異なり、bundled skills は Claude に詳細なプロンプトを与え、作業をオーケストレーションさせるため、エージェントを生成し、ファイルを読み取り、コードベースに適応できます。

詳細情報: *Bundled skills* (ページ 275)

C

Channel
イベントを実行中のセッションにプッシュする *MCP* **サーバー** で、ターミナルから離れている間に発生したことに Claude が反応できるようにします。チャネルは双方向にすることができます。Claude は受信イベントを読み取り、同じチャネルを通じて返信します。Telegram、Discord、iMessage は研究プレビューに含まれています。

詳細情報：Channels

Checkpoint
送信するプロンプトごとにターンを開始する復元ポイント。Claude Code はすべての編集の前にファイルをスナップショットするため、チェックポイントはそれらを復元できます。`Esc` キーを 2 回押すか `/rewind` を実行して、コード、会話、またはその両方を以前のポイントに復元するか、選択したメッセージから会話の一部を要約します。チェックポイントは会話とともに保存されるため、再開されたセッションでも `/rewind` でそれらに戻ることができます。これらは git とは別で、Bash ツールを通じて行われた変更は追跡しません。

詳細情報：*Checkpointing* (ページ 1019)

`.claude` ディレクトリ
Claude Code がプロジェクトスコープの設定を読み取るディレクトリ。設定、フック、スキル、サブエージェント、ルール、自動メモリが含まれます。プロジェクトはそのルートに `.claude/` を持ち、ユーザーレベルのデフォルトは `~/.claude/` にあります。

詳細情報：The .claude directory

CLAUDE.md
Claude 用に作成する永続的な指示のマークダウンファイル。システムプロンプトの後、ユーザーメッセージとしてすべてのセッションの開始時に読み込まれます。プロジェクト規約、アーキテクチャノート、「常に X を行う」ルールをここに記述します。プロジェクトルート CLAUDE.md は *compaction* を通じて保存され、その後ディスクから新たに読み込まれます。

CLAUDE.md は `./CLAUDE.md` または `./.claude/CLAUDE.md` でプロジェクトスコープに、`~/.claude/CLAUDE.md` でユーザースコープに、または組織の *managed policy* として配置できます。検出されたすべてのファイルは相互にオーバーライドするのではなく、最も広いスコープから最も具体的なスコープの順に、コンテキストに連結されます。Claude Code は、プロジェクトの *AGENTS.md* ファイルも読み込むことができます。これは単独で、または CLAUDE.md と一緒に読み込まれます。

詳細情報：*CLAUDE.md files* (ページ 998)

Cloud session

claude.ai/code、Claude モバイルアプリ、**Cloud** が選択された Desktop アプリ、`claude --cloud`、または routine から開始する Claude Code セッションで、ラップトップを閉じた後も実行を続けます。これはクラウドインフラストラクチャで実行されるためです。デフォルトでは Anthropic が管理するか、組織が運用する self-hosted environment です。ターミナル、IDE、または **Local** が選択された Desktop アプリ内のセッションはローカルセッションです。別のデバイスからローカルセッションに到達するには、*Remote Control* を使用します。

詳細情報：*Use Claude Code in the cloud* (ページ 223)

Command

プロンプトに `/name` と入力して呼び出す再利用可能な指示。`/clear`、`/model`、`/compact` などの組み込みコマンドはセッションを制御します。`.claude/commands/` のファイルとして独自のコマンドを定義するか、*plugin* からインストールできます。*Skills* は複数ステップのコマンドをパッケージ化するための推奨される方法です。

この単語の他の 2 つの用途は関連がありません。`claude` CLI サブコマンド（`claude mcp add` など）は *CLI reference* (ページ 45) に記載されており、stdio *MCP server* エントリの `command` フィールドは、Claude Code が起動するために起動する実行可能ファイルを指定します。

詳細情報：Commands · *Skills* (ページ 275)

Compaction

context window がその制限に近づくときの会話の自動要約。古いツール出力が最初にクリアされ、その後会話が要約されます。プロジェクトルート CLAUDE.md と自動メモリは compaction を通じて保存され、ディスクから再度読み込まれます。会話でのみ与えられた指示は失われる可能性があります。`/compact` を手動でトリガーするか、オプションで `/compact focus on the API changes` のようなフォーカスを指定します。

詳細情報：What survives compaction · When context fills up

Connector

Claude Code ではなく claude.ai アカウントに追加される *MCP server*。そのアカウントで Claude Code にサインインすると、コネクタはローカルに追加したサーバーと一緒に `/mcp` に表示されます。組織はコネクタをプロビジョニングし、それらに対するツール単位の制御を設定することもできます。

詳細情報：*Use MCP servers from claude.ai* (ページ 396)

Context window

セッションの作業メモリ。会話履歴、ファイルコンテンツ、コマンド出力、CLAUDE.md、自動メモリ、読み込まれたスキル、システム指示を保持します。作業を進めると、*compaction* がそれを要約するまでコンテキストが満杯になります。`/context` を実行してスペースを使用しているものを確認します。基盤となるモデルの概念については、platform glossary を参照してください。

詳細情報：Explore the context window

D

Dispatch

電話で開始されたタスクルーター。Claude モバイルアプリからコーディングタスクを送信すると、

Desktop アプリで Claude Code セッションを生成します。プロンプトは自動的に正しいツールにルーティングされます。Pro および Max プランで利用可能です。

詳細情報: *Sessions from Dispatch* (ページ 151)

E

Effort level
: 各ステップで適応的推論を行うかどうか、またどの程度行うかをモデルが決定できるようにする設定です。より高い努力はより多くの思考トークンとより深い推論を意味し、より低い努力はより速く、より安価です。Effort は Fable モデル、Opus 4.6 以降、および Sonnet 4.6 以降でサポートされています。

  詳細情報: *Adjust effort level* (ページ 747)

Extended thinking
: モデルが応答する前に実行する可視的なステップバイステップの推論。*effort level* で調整するか、固定思考予算を持つモデルで `MAX_THINKING_TOKENS` で思考トークンをキャップできます。思考はターミナルのグレーイタリックテキストで表示されます。

  詳細情報: *Use extended thinking* (ページ 751)

F

Frontmatter
: Markdown ファイルの最上部にある YAML 設定ブロックで、開始の `---` 行と終了の `---` 行の間に配置されます。Skills、subagents、output styles、および rules は、それぞれ frontmatter から設定を読み込みます。例えば、skill の `description` や subagent の `tools` などであり、終了の `---` の後のすべてを指示として扱います。開始の `---` はファイルの最初の行である必要があります。各ファイルタイプは独自のフィールドセットを受け入れます。

  詳細情報：*Skill frontmatter* (ページ 288)、*Subagent frontmatter* (ページ 326)、*Output style frontmatter* (ページ 825)、*Rule frontmatter* (ページ 1005)

H

Hook
: Claude Code のライフサイクルの特定のポイント（ツール実行前、ファイル編集後、セッション開始時など）で自動的に実行されるユーザー定義ハンドラー。ハンドラーはシェルコマンド、HTTP エンドポイント、MCP ツール、LLM プロンプト、または subagent にできます。Hooks は決定論的です。モデルの裁量ではなく、固定ライフサイクルポイントで発火します。

  フック設定には 3 つのレベルがあります:

  - **Hook event**: ライフサイクルポイント
  - **Matcher**: どのイベントがそれを発火させるかをフィルタリング
  - **Hook handler**: 実行内容

詳細情報: *Get started with hooks* (ページ 411)・ *Hooks reference* (ページ 443)

M

Managed settings

IT または DevOps によって組織全体で実施される設定。Anthropic のサーバーから管理コンソール経由で配信されるか、`~/.claude` の外の OS レベルパスにデバイスにデプロイされます。ユーザーおよびプロジェクト設定は managed settings をオーバーライドすることはできません。サーバー管理配信は対象となる構成に適用されます。セキュリティに関する考慮事項を参照してください。セキュリティポリシー、コンプライアンス要件、またはフロート全体の標準化されたツールに使用します。

詳細情報: Server-managed settings・ *Settings files* (ページ 711)

MCP (Model Context Protocol)

AI ツールを外部データソースとサービスに接続するためのオープン標準。MCP servers は Claude に Slack、Jira、データベース、ブラウザ、および数百の他の統合用の新しいツールを提供します。`/mcp` を使用するか、`.mcp.json` に追加してサーバーを接続します。プロトコル自体については、プラットフォーム用語集を参照してください。

詳細情報: *Model Context Protocol* (ページ 363)

MCP server

Claude に *MCP* 経由でツール、プロンプト、またはリソースを提供するプログラム。`claude mcp add` を使用して、`.mcp.json` で、*plugin* を通じて、または claude.ai *connector* としてサーバーを追加します。ローカル stdio サーバーは Claude Code がその構成の `command` および `args` フィールドから開始するプロセスとして実行されます。これはプロンプトで入力する *commands* とは関係ありません。

詳細情報: *Model Context Protocol* (ページ 363)

MCP Tool Search

コンテキスト節約メカニズム。MCP ツールスキーマを必要になるまで遅延させます。スタートアップ時にはツール名とサーバー命令のみがロードされます。Claude は特定のツールを使用することを決定したときにオンデマンドで完全なスキーマを取得します。これにより、アイドル MCP servers がコンテキストをあまり消費しないようにします。

詳細情報: *Scale with MCP Tool Search* (ページ 406)

N

Non-interactive mode

単一のプロンプトを実行して会話セッションなしで終了するモード。`-p` または `--print` で呼び出されます。CI、スクリプト、パイピングに使用されます。`--no-session-persistence` を渡さない限り、実行は再開可能なセッションとして保存されます。Agent SDK は Python および TypeScript の同等物です。以前は headless mode と呼ばれていました。

詳細情報: *Run Claude Code programmatically* (ページ 695)

O

Output style

Claude Code が Claude に与える指示を変更して、応答動作、トーン、または形式を設定する設定です。プロジェクトコンテキストを Claude Code のデフォルト指示と一緒に追加する *CLAUDE.md* とは異なり、カスタム output style はデフォルトのソフトウェアエンジニアリング指示を置き換えることができます。

詳細情報: *Output styles* (ページ 821)

P

Permission mode

セッションのベースライン承認動作。CLI で `Shift+Tab` でサイクルするか、VS Code、Desktop、claude.ai のモードセレクターを使用します。利用可能なモードは `default`、`acceptEdits`、`plan`、`auto`、`dontAsk`、`bypassPermissions` です。

`default` モードは CLI および VS Code と JetBrains 拡張機能では Manual とラベル付けされており、Claude Code は値の `manual` をエイリアスとして受け入れます。

詳細情報: Permission mode を選択する

Permission rule

ツール名と引数パターンに基づいてツール呼び出しを許可、質問、または拒否する設定エントリ。ルールは deny→ask→allow で評価され、最初にマッチしたものが優先されます。Permission rules は、より広い *permission mode* の上に層状化された細粒度制御です。

詳細情報: **権限を設定する** (ページ 1039)

Plan mode

permission mode の一種。Claude はソースファイルを編集せずに変更を研究および提案します。読み取り、検索、探索コマンドを実行でき、その後、何かに触れる前に承認用の計画を提示します。`/plan` を入力するか、`Shift+Tab` を押して plan mode に入ります。

詳細情報: 編集前に分析する（plan mode）

Plugin

skills、hooks、subagents、MCP servers のバンドル。単一のインストール可能なユニットとしてパッケージ化されます。Plugin skills は `plugin-name:skill-name` として名前空間化されるため、複数のプラグインが共存できます。marketplace を通じてチーム全体にプラグインを配布します。

詳細情報: Plugins

Project trust

Claude Code がその設定をロードする前に、ディレクトリを受け入れるダイアログ。受け入れはプロジェクトディレクトリごとに保存されます。ただし、ホームディレクトリの場合は、信頼は現在のセッションのみ保持され、起動するたびにプロンプトが再度表示されます。ディレクトリを信頼するまで、Claude Code はそのリポジトリが提供するコンテンツの一部（`.claude/settings.json` のプロジェクト許可ルールや marketplace など）を保留します。**フォルダーを信頼する前に実行されるもの** (ページ 1066) には、各種類のコンテンツが記載されており、`-p` セッションがダイアログなしで実行するものも含まれます。

詳細情報: .claude ディレクトリ

Prompt injection
ファイル、ウェブページ、またはツール結果に埋め込まれた敵対的な指示。Claude を、あなたが決して求めなかったアクションにリダイレクトしようとします。Claude Code の防御には、権限システム、コマンドインジェクション検出、信頼検証が含まれます。*Auto mode* は、ツール結果の疑わしいコンテンツをスキャンするサーバー側プローブと、ツール結果を見ない分類器を追加します。そのため、注入されたテキストが直接それを操作することはできません。

詳細情報: *Prompt injection* **から保護する** (ページ 1034)

R

Remote Control
ローカル Claude Code セッションを電話またはブラウザから claude.ai 経由で続行する方法です。コード実行とファイルはマシンに留まります。インターフェースはリモートです。クラウドサンドボックスで実行される *cloud session* (ページ 223) とは異なります。

詳細情報: Remote Control

Rules
`.claude/rules/` 内のモジュール化された指示ファイルで、CLAUDE.md と一緒にロードされます。ルールは YAML フロントマターの `paths:` でパススコープを設定できます。これにより、Claude が一致するファイルを読み取る、書き込む、または編集するときにのみロードされ、関連するまではコンテキストを軽量に保てます。

詳細情報: *Organize rules with .claude/rules/* (ページ 1003)

S

Sandboxing
Bash ツールの OS レベルのファイルシステムおよびネットワーク分離。コマンドは事前に定義した境界内で実行されるため、Claude はコマンドごとの承認プロンプトなしで自由に作業できます。Sandboxing は *permission rules* とは別のレイヤーです。

詳細情報: *Sandboxing* (ページ 1069)

Session
現在のディレクトリに関連付けられた会話。独自の独立した *context window* を持ちます。セッションは `claude -c` で再開でき、`--fork-session` でフォークして履歴を新しいセッション ID の下に保存でき、またはターミナル全体で並列実行できます。`/clear` を実行すると新しいセッションが開始されます。前のセッションは保存されたままで、`/resume` を通じて利用可能です。各セッションのトランスクリプトは `~/.claude/projects/` に保存されます。

詳細情報: Work with sessions

Settings layers
Claude Code が設定を読み取る階層。優先順位の高い順から低い順: *managed policy*、コマンドライン引数、`.claude/settings.local.json` のローカル設定、`.claude/settings.json` のプロジェクト設定、`~/.claude/settings.json` のユーザー設定。配列はレイヤー全体でマージされます。スカ

ラーは高いレイヤーで低いレイヤーをオーバーライドします。*Settings precedence* (ページ 719) を参照してください。

詳細情報: *Settings files* (ページ 711)

Skill

指示、知識、またはワークフローを含む `SKILL.md` ファイル。Claude はそれをツールキットに追加します。Claude は関連する場合に自動的にスキルをロードするか、`/skill-name` で直接呼び出します。Skills は Agent Skills オープン標準に従います。Claude Code はそれを呼び出し制御と subagent 実行で拡張します。

Skills は custom commands の推奨される後継者です。`.claude/commands/deploy.md` のファイルと `.claude/skills/deploy/SKILL.md` のファイルの両方が `/deploy` を作成し、同じように機能します。既存のコマンドファイルは引き続き機能します。

詳細情報: *Extend Claude with skills* (ページ 275)

Subagent

独自のコンテキストウィンドウ、カスタムシステムプロンプト、特定のツールアクセス、独立した権限で実行される特化した AI アシスタント。委任されたタスクで機能し、メイン会話に要約を返します。大規模な探索をプライマリコンテキストから除外するか、並列研究を実行するために subagents を使用します。subagent は、それを生成したセッション内に留まります。別々のセッション間で調査結果を渡すには、自分で実行する場合は cross-session messaging を使用してください。

組み込み subagents には Explore、Plan、汎用があります。

詳細情報: *Create custom subagents* (ページ 317)

Surface

Claude Code にアクセスする任意の場所: CLI、VS Code、JetBrains、Desktop、または claude.ai。すべてのサーフェスは同じエンジンを共有します。マシン上のセッションはローカルの CLAUDE.md、設定、スキルを読み取ります。cloud sessions はリポジトリの新しいクローンから開始され、マシン上の `~/.claude/` を読み取りません。Slack と Chrome 拡張機能は、サーフェス自体ではなくサーフェスに接続する統合です。

詳細情報: Platforms and integrations

System prompt

Claude Code がすべてのリクエストで会話の前に送信する指示。Claude がツールをどのように使用するか、安全に動作する方法、および応答をどのようにフォーマットするかについて説明します。`--append-system-prompt` でシステムプロンプトに追加するか、`--system-prompt` で置き換えることができます。システムプロンプトは prompt cache の最初のレイヤーです。

CLAUDE.md ファイルと *output style* の指示はシステムプロンプトの一部ではありません。Claude Code はそれらを会話で *system reminder* として配信します。

詳細情報: *System prompt flags* (ページ 72)

System reminder

Claude Code が *harness* として会話に追加するメッセージ。Claude に文脈を提供します。システムリマインダーは自分で送信しません。Claude Code はセッションの実行中にそれらを挿入します。たとえば、セッションが開始されるとき、フックがテキストを返すとき、またはファイルがディスク上で

変更されるときです。Claude はそれらをメッセージと一緒に読みます。以下のすべてがシステムリマインダーとして Claude に到達します:

- *CLAUDE.md* ファイル
- *output style* の指示
- *hook* が `additionalContext` として返すテキスト
- 利用可能な *skills* のリスト
- Claude が以前に読んだファイルがディスク上で変更されたというメモ
- コミットおよびプルリクエストの属性行

ログに記録された API リクエストでは、システムリマインダーはユーザーメッセージ内の `<system-reminder>` タグでラップされるか、一部のモデルでは `system` ロールを持つ別のメッセージとして表示されます。

詳細情報: Context Claude Code adds outside the system prompt

T

Teleport

コマンド `/teleport`。クラウド Claude Code セッションをローカルターミナルにプルします。Claude はブランチをフェッチし、会話履歴をロードし、クラウドセッションの最後の状態から再開します。逆方向は `--cloud` です。ローカルタスクをクラウドで実行するために送信します。

詳細情報: **クラウドからターミナルへ** (ページ 229)

Tool

Claude が実行できるアクション: ファイルを読み取る、コードを編集する、シェルコマンドを実行する、web を検索する、subagent を生成する。Tools は Claude Code を agentic にするものです。それらなしでは、Claude はテキストのみで応答できます。各ツール使用は、*agentic loop* での Claude の次の決定に情報を提供する結果を返します。

詳細情報: Tools available to Claude

Transcript

session の保存されたレコード。会話はあなたと Claude が交換するもの。トランスクリプトはその会話をファイルとして保存したもので、デフォルトでは `~/.claude/projects/<project>/<session-id>.jsonl` に保存されます。Claude Code は再開時にファイルを読み込み直します。これにより、セッション終了後も会話が続きます。同じ会話の画面上のビューについては、*transcript viewer* (ページ 82) を参照してください。

詳細情報: トランスクリプトの保存場所

Turn

Claude が *session* 内で行う 1 つの完全な応答です。Turn は、メッセージを送信するときに開始され、Claude が応答を終了するときに終了します。その間に任意の数の *tool* 呼び出しがあります。*Stop hooks* は各 turn の終了時に発火します。セッションは多くの turn で構成され、*agentic loop* は 1 つの内部で何が起こるかを説明します。

詳細情報: How Claude Code works

V

Verification loop

セッションが作業が実際に完了したことを知る方法。テストスイート、ビルド、またはスクリーンショット比較など、Claude が実行できるチェックを提供します。Claude はチェックが成功するまで反復します。1 回の試行後に停止するのではなく。Verification loop は `/goal`、無人実行、および dynamic workflows の前提条件です。それなしでは、エージェントが完了したことを決定する唯一のものはエージェント自体です。

詳細情報: *Claude* **に作業を検証する方法を提供する** (ページ 115)

W

Worktree isolation

Claude を `.claude/worktrees/` の別の git worktree で実行する分離モード。`-w` フラグまたは subagent 設定の `isolation: worktree` で有効にされます。変更は別のブランチの別のディレクトリに留まるため、並列エージェントはお互いのファイルを上書きしません。

詳細情報: git worktrees を使用した並列セッションの実行

非推奨および名前変更された用語

これらの用語は古いドキュメント、ブログ投稿、コミュニティコンテンツに表示されます。このサイトを検索するときは現在の名前を使用してください。

古い用語	現在の呼び方	注記
Headless mode	*Non-interactive mode*	同じ `-p` フラグ、同じ動作
Web session; "Claude Code on the web" as the name for any cloud session	*Cloud session*	「Claude Code on the web」は現在、claude.ai/code のブラウザサーフェスのみを指します
Custom commands	*Skills*	`.claude/commands/` ファイルは引き続き機能
Slash commands	Commands	製品コピーから「Slash」を削除