Emacsノート

Mac版 GNU Emacs リファレンス

GNU Emacs 31.1（NS 版）　macOS 27.2

mindtools@mac.com

2026 年 11 月　第 1 版

目 次

はじめに **1**
対象の環境 1
読み方 2
章の構成 3

第 1 章　Emacs の基本 **5**
1.1　画面の構成 5
1.2　キーの表記 6
1.3　⌘のキー 7
1.4　起動と終了 8
1.5　コマンドの実行と中断 8
1.6　ミニバッファと補完 9
1.7　モード 10
1.8　チュートリアル 12

第 2 章　ファイルの編集 **13**
2.1　ファイルを開く・保存する 13
2.2　カーソルの移動 14
2.3　改行と挿入 16
2.4　削除 17
2.5　マークとリージョン 18
2.6　キルとヤンク 18
2.7　入れ替え 19
2.8　取り消し 20

第 3 章　検索と置換 **23**
3.1　コマンドの選び方 23

3.2 インクリメンタルサーチ . . . 24
3.3 正規表現による検索 . . . 26
3.4 置換 . . . 28
3.5 一致した行の一覧 . . . 30
3.6 複数のファイルの検索 . . . 31

第 4 章 バッファとウィンドウとタブの使用 33
4.1 4 つの単位 . . . 33
4.2 バッファの操作 . . . 34
4.3 ウィンドウの操作 . . . 36
4.4 フレームの操作 . . . 38
4.5 タブの操作 . . . 39

第 5 章 作業環境としての Emacs 41
5.1 Dired . . . 42
5.2 vterm . . . 47
5.3 外部コマンド . . . 51
5.4 チェックリスト . . . 53

第 6 章 Emacs とインターネット 55
6.1 w3m . . . 55
6.2 navi2ch . . . 58
6.3 チェックリスト . . . 69

第 7 章 シンプルなテキストフォーマッティングと特殊な編集 71
7.1 段落の整形 . . . 71
7.2 長い行の表示 . . . 73
7.3 インデント . . . 73
7.4 矩形の編集 . . . 74
7.5 文字の種類の変換 . . . 76
7.6 行の並べ替えと整列 . . . 77
7.7 特殊な文字の入力 . . . 78
7.8 キーボードマクロ . . . 79
7.9 チェックリスト . . . 82

第 8 章　テキストのマークアップ **83**
8.1　markdown-mode の導入 84
8.2　書式を入れる 85
8.3　リスト 87
8.4　表 87
8.5　リンク 88
8.6　プレビューと書き出し 88
8.7　アウトライン 89
8.8　チェックリスト 91

第 9 章　Emacs のカスタマイズ **93**
9.1　設定ファイル 95
9.2　Emacs Lisp の基本 96
9.3　パッケージの導入 98
9.4　キー割り当ての変更 101
9.5　見た目の設定 103
9.6　起動のトラブル 104
9.7　チェックリスト 105

第 10 章 プログラミングコマンド **107**
10.1　プログラミング用のモード 108
10.2　コードの移動と参照 110
10.3　コンパイル 113
10.4　チェックリスト 114

第 11 章 git と diff と git-bug **117**
11.1　Magit 118
11.2　diff と ediff 121
11.3　git-bug 123
11.4　チェックリスト 126

第 12 章 Claude Code との連携 **127**
12.1　導入 128
12.2　起動と終了 129
12.3　Claude とのやり取り 130

12.4 変更の確認 131
12.5 Emacs のツール 131
12.6 チェックリスト 132

第 13 章 オンラインヘルプ **133**
13.1 ヘルプのキー 134
13.2 Info 135
13.3 日本語のマニュアル 138
13.4 チェックリスト 140

用語集 **141**

索引 **145**

はじめに

この資料は、Mac で GNU Emacs を使うための手元のリファレンスです。チートシートより詳しく、マニュアルより短くまとめています。網羅することより、よく使う操作をすぐ引けることを優先しています。

対象の環境

この資料は、次の環境で確かめた内容だけを書いています。

項目	内容
Emacs	GNU Emacs 31.1。macOS 用の NS 版（`/Applications/Emacs.app`）
OS	macOS 27.2
設定ファイル	`~/.emacs.d/` の `early-init.el` と `init.el`。git で管理している
パッケージの設定	主に leaf。vterm と claude-code-ide は use-package
シェル	zsh

- Linux と Windows は扱いません。ターミナルの中で動かす Emacs も扱いません。
- キーやコマンドは、この init.el を読み込んだ状態のものです。Emacs の初期設定と違うところは、「init.el で割り当てています」などと書いています。

読み方

キーの表記

修飾キーは、Mac のキーボードの記号で書きます。`M-x` は ⌥x 、`C-x C-f` は `^x ^f` です。詳しくは、第 1 章の「キーの表記」で説明しています。

キーの表

キーは、次の形の表にまとめています。左から、キー、呼び出す関数、動作です。

`^x ^s`	`save-buffer`	ファイルを保存する
⌥x	`delete-trailing-whitespace`	行末の空白を消す

- キーの列が ⌥x だけの行は、キーが割り当てられていないコマンドです。⌥x を押し、2 列目の名前を入れて実行します。
- 関数の名前が分かれば、`F1 f` で説明を引けます。第 13 章を参照してください。

そのほかの表

設定や用語の表には、1 行目に太字の見出しを付けています。作業の手順は、流れ図で示しています。第 5 章から第 13 章の最後には、「やるべきこと」と「やってはいけないこと」のチェックリストがあります。

章の構成

章	題名	主な内容
1	Emacs の基本	画面の名前、キーの表記、起動と終了、補完、モード
2	ファイルの編集	移動、削除、キルとヤンク、取り消し、ファイルの保存
3	検索と置換	インクリメンタルサーチ、置換、正規表現、grep
4	バッファとウィンドウとタブの使用	バッファの切り替え、ウィンドウの分割、タブ
5	作業環境としての Emacs	Dired、vterm、外部コマンド
6	Emacs とインターネット	w3m、navi2ch
7	シンプルなテキストフォーマッティングと特殊な編集	詰め込み、インデント、矩形、キーボードマクロ
8	テキストのマークアップ	Markdown、アウトライン
9	Emacs のカスタマイズ	設定ファイル、Emacs Lisp、パッケージ、キーの割り当て
10	プログラミングコマンド	言語のモード、xref と eglot、コンパイル
11	git と diff と git-bug	Magit、diff と ediff、git-bug
12	Claude Code との連携	claude-code-ide、変更の確認
13	オンラインヘルプ	F1 のヘルプ、Info、日本語のマニュアル

巻末の用語集には、Emacs の用語の意味をまとめています。

第1章　Emacsの基本

Emacs の操作は、キーでコマンドを呼び出すことが基本です。この章では、画面の名前、キーの表記、起動と終了、補完、モードを説明します。

1.1 画面の構成

Emacs の画面は、次の部品でできています。

部品	説明
フレーム	macOS のウィンドウ 1 つ分です。 ⌘n で増やせます。
ウィンドウ	フレームを分割した表示領域です。1 つのウィンドウに 1 つのバッファを表示します。
バッファ	編集中のテキストを保持する領域です。ファイルを開くと、内容がバッファに読み込まれます。
モードライン	ウィンドウの下端にある 1 行です。バッファの状態を表示します。
ミニバッファ	フレームの下端にある入力欄です。コマンド名やファイル名を入力します。
エコーエリア	ミニバッファと同じ場所です。入力中でないときはメッセージを表示します。
メニューバー	macOS の画面上端に表示します。ツールバーは init.el で消しています。

1.1.1 モードラインの読み方

モードラインは、左から順に次の情報を表示します。

表示	意味
`U`	文字コード。 `U` は UTF-8 です。
`--`	変更していません。
`**`	変更しています。まだ保存していません。
`%%`	読み出し専用です。
バッファ名	編集中のバッファの名前です。
`All`、`Top`、`Bot`、`40%`	表示している範囲の、バッファの中での位置です。
`(12,5)`	カーソルの行と桁です。桁は init.el の `column-number-mode` で表示しています。
`(Text ...)`	メジャーモードとマイナーモードの名前です。

1.2 キーの表記

この資料では、修飾キーを Mac のキーボードの記号で表記します。

キー	表記	Emacs での役割
command	⌘	Super
option	⌥	Meta
shift	⇧	Shift
control	^	Control

- Emacs の文書の `M-x` は ⌥x 、 `C-x C-f` は `^x ^f` と書きます。
- 修飾キー以外のキーは Emacs の表記で書きます。DEL は delete、RET は return のキーです。
- `^x ^f` は、^x を押して離し、続けて^f を押す操作です。

- ESC を押して離してから文字キーを押しても、⌥と同じ働きをします。

1.2.1 プレフィックスキー

^x のように、続けて別のキーを待つキーをプレフィックスキーと呼びます。

- 主なプレフィックスキーは ^x と ^c です。
- プレフィックスキーを押して少し待つと、続けて押せるキーの一覧が出ます。init.el で有効にした which-key-mode の働きです。
- 途中でやめるときは ^g を押します。

1.3 ⌘のキー

Mac 版の Emacs は、⌘のキーに macOS の一般的な操作を割り当てています。

⌘s	save-buffer	ファイルを保存する
⌘o	ns-open-file-using-panel	ダイアログでファイルを開く
⌘z	undo-tree-undo	直前の操作を取り消す
⇧⌘z	undo-tree-redo	取り消しをやり直す
⌘x	kill-region	リージョンをキルする
⌘c	ns-copy-including-secondary	リージョンをコピーする
⌘v	yank	ヤンクする
⌘a	mark-whole-buffer	バッファ全体を選択する
⌘f	isearch-forward	前方へインクリメンタルサーチする
⌘l	goto-line	指定した行へ移動する
⌘n	make-frame	新しいフレームを開く
⌘w	delete-frame	フレームを閉じる
⌘m	iconify-frame	フレームを Dock にしまう
⌘h	ns-do-hide-emacs	Emacs を隠す

- ⌘z と ⇧⌘z は、init.el で undo-tree のコマンドに置き換えています。

- ⌘w はフレームを閉じるだけです。バッファは残ります。
- ⌘のキーは便利ですが、^と⌥のキーを覚えると、手の移動が減ります。

1.4 起動と終了

1.4.1 起動

- Dock や Launchpad から Emacs.app を開きます。
- ターミナルからファイルを開くときは、 open -a Emacs ファイル名 を実行します。
- init.el で起動画面を消しています。起動すると *scratch* バッファを表示します。
- フレームは画面の左上に、幅 120 桁、高さ 50 行で開きます。init.el の設定です。

1.4.2 終了

^x ^c	save-buffers-kill-terminal	保存を確かめてから終了する
⌘q	save-buffers-kill-emacs	保存を確かめてから終了する
^z	suspend-frame	フレームを Dock にしまう

- 保存していないバッファがあると、終了の前に保存するかを尋ねます。
- ^z は、Mac 版では Emacs を止めません。フレームを Dock にしまうだけです。

1.5 コマンドの実行と中断

Emacs のすべての操作はコマンドです。キーに割り当てのないコマンドは、 ⌥x で名前を入力して実行します。

⌥x	execute-extended-command	コマンド名を入力して実行する
^g	keyboard-quit	入力中のコマンドを中断する
ESC ESC ESC	keyboard-escape-quit	ミニバッファなどから抜ける
^u	universal-argument	次のコマンドに数引数を渡す

- 操作に迷ったら、まず ^g を押します。
- ^g で抜けられないときは、 ESC ESC ESC を押します。

1.5.1 数引数

数引数は、コマンドを繰り返す回数などを指定する数です。

- ^u 8 ^f は、カーソルを 8 文字進めます。
- 数を入れずに ^u だけを押すと、4 を渡します。
- ⌥8 ^f のように、⌥と数字でも指定できます。

1.6 ミニバッファと補完

ミニバッファでは、ファイル名やコマンド名を途中まで入力すると、候補を絞り込めます。これを補完と呼びます。

1.6.1 この環境の補完

init.el で ido を有効にしています。ファイル名、バッファ名、コマンド名の入力で、候補を縦に一覧します。

パッケージ	働き
ido	候補を一覧し、入力に応じて絞り込みます。
ido-completing-read+	⌥x などの入力にも ido を使います。
flx-ido	文字が離れていても一致する候補を出します。 例： bsc で basics.rst 。
ido-flex-with-migemo	ローマ字で日本語の候補を絞り込みます。
ido-vertical-mode	候補を縦に並べます。

1.6.2 ido のキー

^n	ido-next-match	次の候補を選ぶ
^p	ido-prev-match	前の候補を選ぶ
TAB	ido-complete	共通する部分まで補完する
RET	ido-exit-minibuffer	選んだ候補で決定する
^j	ido-select-text	候補を使わず、入力した文字列で決定する
^f	ido-magic-forward-char	入力の末尾で押すと、ido を使わない入力に切り替える

- ^f で切り替わるのは、 ^x ^f などファイルを開く入力のときです。
- 新しいファイルを作るときは、名前を入力して ^j を押します。RET では、似た名前の既存のファイルを開くことがあります。

1.7 モード

Emacs の動作は、バッファごとのモードで変わります。モードには、メジャーモードとマイナーモードの 2 種類があります。

種類	数	決まり方
メジャーモード	1 つのバッファに 1 つ	ファイルの拡張子で自動的に決まります。
マイナーモード	1 つのバッファにいくつでも	個別に入/切を切り替えます。

- モードを切り替えるには、⌥x でモードの名前を実行します。例えば、⌥x `text-mode` でテキストモードになります。
- マイナーモードは、同じコマンドをもう一度実行すると切になります。
- 今のモードの説明とキーの一覧は、 `F1 m` （ `describe-mode` ）で表示します。

1.7.1 主なメジャーモード

モード	用途
`fundamental-mode`	特別な機能のない基本のモードです。
`text-mode`	文章を書くためのモードです。
`lisp-interaction-mode`	*scratch* バッファのモードです。Emacs Lisp の式を評価できます。
`emacs-lisp-mode`	init.el など Emacs Lisp のファイルを編集します。
`c-mode`	C 言語のファイルを編集します。init.el で BSD スタイルのインデントにしています。
`dired-mode`	ディレクトリの中身を一覧し、ファイルを操作します。

1.7.2 主なマイナーモード

モード	働き
auto-fill-mode	決まった幅で自動的に改行します。
overwrite-mode	文字を挿入せず、上書きします。
auto-save-mode	編集中の内容を一定の間隔で自動保存します。
column-number-mode	モードラインに桁を表示します。init.el で有効にしています。
global-hl-line-mode	カーソルのある行に色を付けます。init.el で有効にしています。
global-auto-revert-mode	ほかのアプリでファイルが変わると、バッファを読み直します。init.el で有効にしています。
global-undo-tree-mode	取り消しの履歴を木の形で扱います。init.el で有効にしています。
which-key-mode	プレフィックスキーの後に続くキーを一覧します。init.el で有効にしています。

1.8 チュートリアル

⌥x	help-with-tutorial	チュートリアルを開く

言語を選んで開くには、 ⌥x help-with-tutorial-spec-language を実行し、Japanese を入力します。

第2章　ファイルの編集

ファイルを開き、カーソルを動かし、テキストを書き換えて保存します。この章では、編集の基本の操作と、この環境の設定による違いを説明します。

2.1 ファイルを開く・保存する

^x ^f	find-file	ファイルを開く
^x ^v	find-alternate-file	今のバッファを閉じて別のファイルを開く
^x i	insert-file	カーソル位置にファイルの内容を挿入する
^x ^s	save-buffer	バッファをファイルに保存する
^x s	save-some-buffers	未保存のバッファを 1 つずつ確かめて保存する
^x ^w	write-file	名前を付けて保存する
⌘u	revert-buffer	ファイルを読み直し、未保存の変更を捨てる
^x ^q	read-only-mode	読み出し専用を入/切する

- ^x ^f では ido で候補を選びます。キーは第 1 章の「ido のキー」を見てください。
- 新しいファイルは、^x ^f で存在しない名前を指定して作ります。保存するまで、ファイルはディスクにできません。

- ほかのアプリでファイルが変わると、バッファを自動で読み直します。init.el で有効にした `global-auto-revert-mode` の働きです。
- ファイルを開き直すと、前回のカーソル位置に戻ります。init.el で有効にした `save-place-mode` の働きです。

2.1.1 バックアップと自動保存

Emacs は、編集中のファイルを 2 種類の方法で守ります。init.el で、どちらも `~/.emacs.d/var/` の下に置くように設定しています。

種類	置き場所	内容
バックアップ	`var/backups/`	ファイルを開いてから最初に保存するときに作ります。開いた時点の内容です。
自動保存	`var/auto-saves/`	編集中の内容を一定の間隔で保存します。名前はパスの / を ! に替えた形です。

- 編集するディレクトリには、バックアップや自動保存のファイルを作りません。
- Emacs が異常終了したときは、`⌥x recover-file` で自動保存から復元します。復元する前に、自動保存のファイルと元のファイルの差分を確かめます。

2.2 カーソルの移動

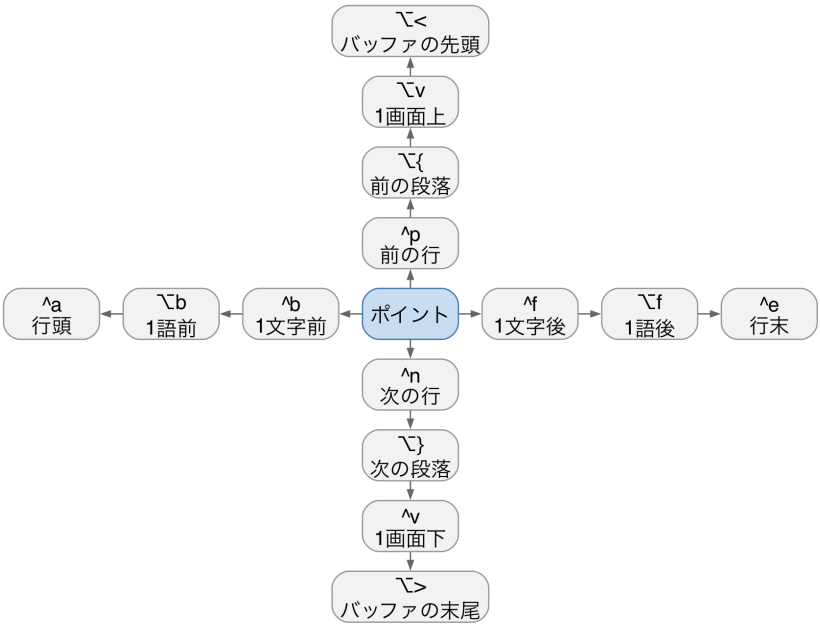
カーソルの位置をポイントと呼びます。

^f	forward-char	1 文字進む
^b	backward-char	1 文字戻る
^n	next-line	次の行へ
^p	previous-line	前の行へ
^a	seq-home	行頭へ。続けて押すとバッファの先頭へ
^e	seq-end	行末へ。続けて押すとバッファの末尾へ
⌥f	forward-word	1 語進む
⌥b	backward-word	1 語戻る
⌥e	forward-sentence	文末へ
⌥a	backward-sentence	文頭へ
⌥}	forward-paragraph	次の段落へ
⌥{	backward-paragraph	前の段落へ
^v	scroll-up-command	1 画面下へ
⌥v	scroll-down-command	1 画面上へ
⌥<	beginning-of-buffer	バッファの先頭へ
⌥>	end-of-buffer	バッファの末尾へ
⌥g g	goto-line	指定した行へ
⌥m	back-to-indentation	行頭の空白の後へ
^l	recenter-top-bottom	カーソル行を中央・上・下へ
⌥r	move-to-window-line-top-bottom	カーソルを画面の中央・上・下へ
⌥[	point-stack-pop	大きく移動する前の位置へ戻る
⌥]	point-stack-forward-stack-pop	⌥[で戻る前の位置へ進む

- ^a と ^e は、init.el の sequential-command で置き換えています。3 回目を押すと、最初の位置に戻ります。
- ⌥[と ⌥] は、init.el の point-stack で割り当てています。⌥< 、 ⌥> 、検索などで移動する前の位置を記録します。

主な移動のキーを図にまとめます。中央のポイントから外側へ向かうほど、大きく

移動します。



2.3 改行と挿入

RET	newline-and-indent	改行し、次の行をインデントする
^o	open-line	カーソルの後ろに空行を挿入する
^q	quoted-insert	次に押した文字をそのまま挿入する

- RET は、init.el で newline-and-indent に割り当てています。
- ^q は、タブ文字や制御文字を入力するときに使います。例： ^q TAB 。

2.4 削除

DEL	delete-backward-char	カーソルの前の 1 文字を削除する
^h	delete-backward-char	DEL と同じ。init.el で置き換えている
^d	delete-char	カーソル位置の 1 文字を削除する
⌥DEL	backward-kill-word	カーソルの前の 1 語をキルする
⌥h	backward-kill-word	⌥DEL と同じ。init.el で割り当てている
⌥d	kill-word	カーソル位置から語末までキルする
^k	kill-line	カーソル位置から行末までキルする
⌥k	kill-sentence	カーソル位置から文末までキルする
⌥z	zap-to-char	指定した文字までキルする
⌥\	delete-horizontal-space	カーソルの前後の空白を削除する
^x ^o	delete-blank-lines	カーソルの前後の空行を 1 行に詰める

- 削除は、テキストを捨てます。キルは、テキストをキルリングに保存します。
- ^k を行頭で押すと、行の中身を消します。改行は残ります。もう一度押すと、改行も消します。

2.5 マークとリージョン

マークはリージョンの一端です。もう一端はポイントです。^SPC でマークを置き、カーソルを動かすと、その間がリージョンになります。

^SPC	set-mark-command	マークを設定する
^u ^SPC	set-mark-command	前のマークの位置へ戻る
^x ^x	exchange-point-and-mark	ポイントとマークを入れ替える
⌥@	mark-word	次の語の末尾にマークを置く
^x h	mark-whole-buffer	バッファ全体をリージョンにする

- リージョンを解除するには ^g を押します。
- 段落を選ぶ mark-paragraph の ⌥h は、init.el で backward-kill-word に置き換えています。段落を選ぶときは、 ⌥x mark-paragraph を実行します。

2.6 キルとヤンク

キルしたテキストは、キルリングに順に溜まります。ヤンクで、キルリングのテキストを挿入します。

^w	kill-region	リージョンをキルする
⌥w	kill-ring-save	リージョンをキルリングにコピーする
^y	yank	最後にキルしたテキストを挿入する
⌥y	browse-kill-ring	キルリングを一覧し、選んで挿入する

- 続けてキルしたテキストは、1 つにまとまります。例： ^k を 3 回押すと、3 回分を 1 回でヤンクできます。

- キルリングは、macOS のクリップボードとつながっています。ほかのアプリでコピーしたテキストも ^y で挿入できます。
- ⌥y は、init.el で browse-kill-ring に割り当てています。標準の yank-pop は ⌥x yank-pop で実行します。

2.6.1 browse-kill-ring のキー

n	browse-kill-ring-forward	次の項目へ
p	browse-kill-ring-previous	前の項目へ
RET	browse-kill-ring-insert-and-quit	選んだ項目を挿入して閉じる
y	browse-kill-ring-insert	選んだ項目を挿入する。一覧は閉じない
s	browse-kill-ring-search-forward	項目を検索する
d	browse-kill-ring-delete	選んだ項目をキルリングから削除する
q	browse-kill-ring-quit	何も挿入せずに閉じる

2.7 入れ替え

^t	transpose-chars	カーソルの前後の文字を入れ替える
⌥t	transpose-words	カーソルの前後の語を入れ替える
^x ^t	transpose-lines	今の行と前の行を入れ替える

2.8 取り消し

init.el で undo-tree を有効にしています。取り消しの履歴を木の形で持つので、取り消した変更もやり直せます。

^/	undo-tree-undo	直前の変更を取り消す
⌥_	undo-tree-redo	取り消した変更をやり直す
^x u	undo-tree-visualize	取り消しの履歴を木の形で表示する

- ⌘z と ⇧⌘z も使えます。第 1 章の「⌘のキー」を見てください。
- 取り消しの履歴は、保存するときにファイルにも残します。ファイルを閉じて開き直しても、前の変更を取り消せます。

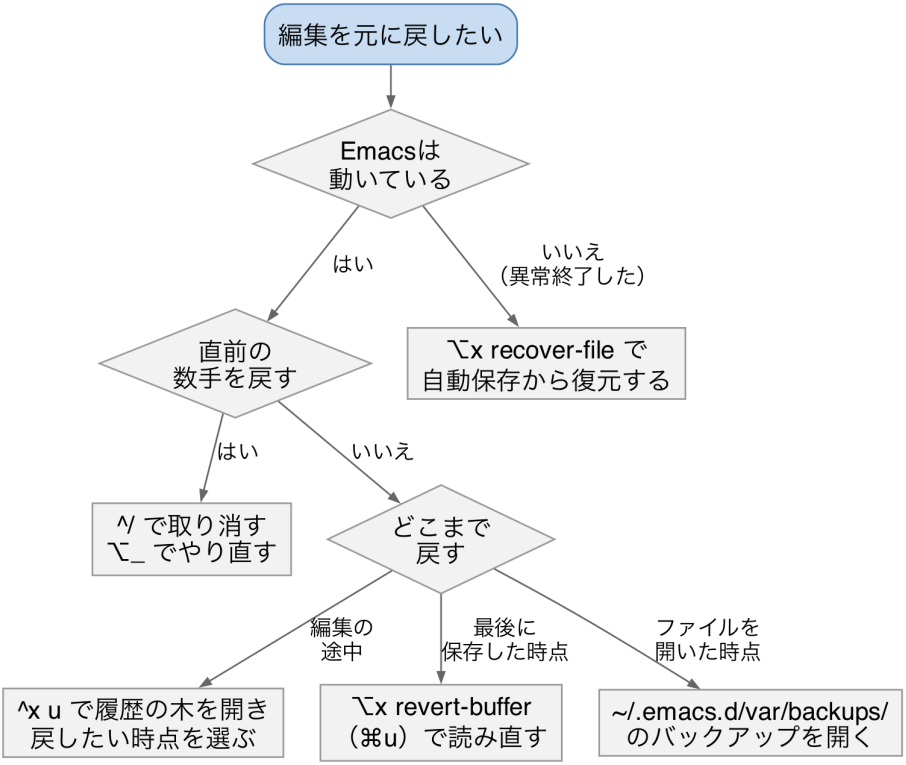
2.8.1 履歴の木のキー

^x u で開いた履歴の木では、次のキーを使います。

p	undo-tree-visualize-undo	1 つ前の状態へ戻る
n	undo-tree-visualize-redo	1 つ後の状態へ進む
b	undo-tree-visualize-switch-branch-left	左の枝へ移る
f	undo-tree-visualize-switch-branch-right	右の枝へ移る
d	undo-tree-visualizer-toggle-diff	差分の表示を入/切する
q	undo-tree-visualizer-quit	選んだ状態で閉じる
^q	undo-tree-visualizer-abort	開く前の状態に戻して閉じる

2.8.2 編集を元に戻す手順

どこまで戻すかで、使う方法が変わります。

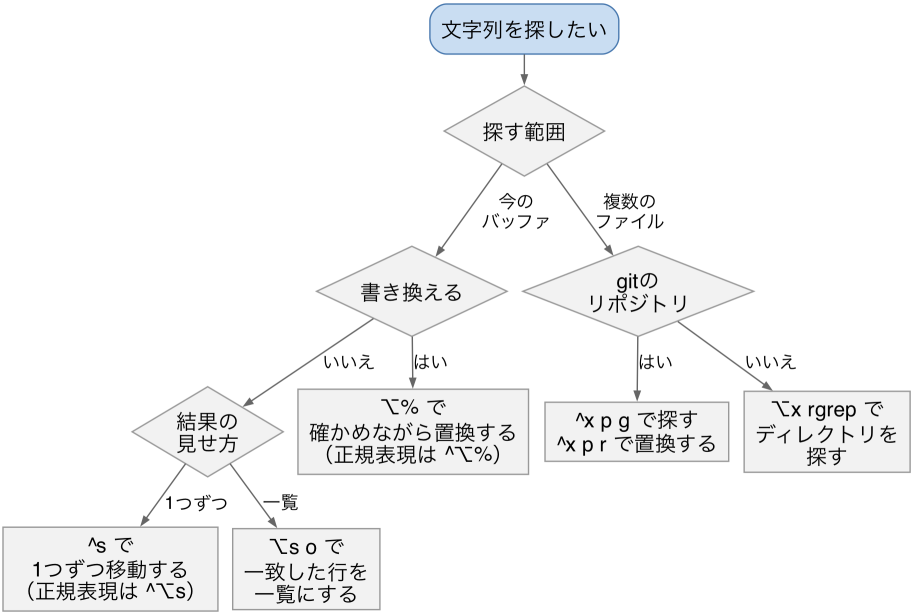


第3章　検索と置換

1 つのバッファの中では ^s で検索し、 ⌥% で置換します。複数のファイルは rgrep か ^x p g で検索します。この章では、検索と置換のコマンドと、この環境の migemo の設定を説明します。

3.1 コマンドの選び方

探す範囲と、結果をどう使うかで、コマンドを選びます。



3.2 インクリメンタルサーチ

文字を入力するたびに、一致する位置へカーソルが動く検索です。

^s	isearch-forward	前方へ検索する
^r	isearch-backward	後方へ検索する
⌥s .	isearch-forward-symbol-at-point	カーソル位置のシンボルを検索する
⌥s _	isearch-forward-symbol	シンボル単位で検索する
⌥s w	isearch-forward-word	単語単位で検索する

検索の流れは次のとおりです。

1. ^s を押し、探す文字列を入力します。一致した箇所はすべて色が付きます。
2. ^s を押すたびに、次の一致へ移動します。 ^r で前に戻ります。
3. RET で検索を終えます。カーソルは見つけた位置に残ります。
4. ^g で検索をやめると、検索を始めた位置に戻ります。

- 検索を始めた位置にはマークが残ります。 ^u ^SPC で戻れます。
- 文字列を入力せずに ^s ^s と押すと、前回の文字列で検索します。
- ⌘f でも検索を始められます。

3.2.1 検索中のキー

^s	isearch-repeat-forward	次の一致へ移動する
^r	isearch-repeat-backward	前の一致へ移動する
DEL	isearch-delete-char	入力した文字を 1 つ消す
^w	migemo-isearch-yank-word	カーソル位置の単語を検索文字列に足す
^y	migemo-isearch-yank-line	行末までを検索文字列に足す
⌥y	isearch-yank-pop-only	キルリングの文字列を検索文字列に足す
⌥e	isearch-edit-string	検索文字列をミニバッファで直す
⌥p	isearch-ring-retreat	前の検索文字列を呼び出す
⌥n	isearch-ring-advance	次の検索文字列を呼び出す
⌥c	isearch-toggle-case-fold	大文字と小文字の区別を入/切する
⌥r	isearch-toggle-regexp	正規表現の検索に切り替える
⌥m	migemo-isearch-toggle-migemo	migemo を入/切する
⌥%	isearch-query-replace	検索文字列で置換を始める
⌥s o	isearch-occur	検索文字列で一覧を作る
RET	isearch-exit	検索を終える
^g	isearch-abort	検索をやめ、元の位置に戻る

- 一致しない文字があるときの ^g は、一致しない部分だけを消します。もう一度 ^g を押すと、検索をやめます。
- ^h は DEL と同じ働きです。

3.2.2 大文字と小文字

検索文字列に大文字があるかどうかで、区別するかが変わります。

- 小文字だけで入力すると、大文字と小文字を区別しません。
- 大文字を 1 つでも入力すると、区別します。

- 検索中に ⌥c を押すと、区別するかを切り替えます。

3.2.3 migemo

migemo は、ローマ字の入力で日本語を検索する仕組みです。init.el で有効にしているため、 ^s と ^r で常に働きます。

- 例えば kensaku と入力すると、「検索」「けんさく」「ケンサク」に一致します。
- 英字の kensaku にも一致します。日本語を入力するために IME を切り替える必要はありません。
- 正規表現の検索では働きません。
- 一致が多すぎるときは、 ⌥m で migemo を切ります。
- 変換には cmigemo コマンドを使います。Homebrew で入れたものです。

3.3 正規表現による検索

^⌥s	isearch-forward-regexp	正規表現で前方へ検索する
^⌥r	isearch-backward-regexp	正規表現で後方へ検索する
⌥x	re-builder	正規表現を試しながら組み立てる

検索中のキーは、インクリメンタルサーチと同じです。

3.3.1 Emacs の正規表現

Emacs の正規表現は、Perl や Swift の正規表現と書き方が違います。グループや選択には \ を付けます。

Emacs	意味	Perl などの書き方
\(\)	グループ	( )
|	または	|
\{2,3\}	2 回から 3 回の繰り返し	{2,3}
[[:digit:]]	数字	\d
[[:space:]]	空白文字	\s
\w	単語を作る文字	\w
\b	単語の境目	\b
_< _>	シンボルの先頭と末尾	なし
^ $	行頭と行末	^ $
*? +?	最短一致の繰り返し	*? +?

- \d は使えません。 \d は文字の d に一致します。
- ( ) | { } は、 \ を付けないと文字そのものに一致します。
- シンボルは、 foo-bar のように - を含む名前を 1 つのまとまりとして扱います。
- ミニバッファでは \ を 1 つで書きます。init.el などの Lisp の文字列の中では、\\(のように 2 つ重ねます。

3.3.2 re-builder

⌥x re-builder は、入力した正規表現に一致する箇所を、今のバッファで色付けします。

- 入力欄は Lisp の文字列の形式です。 \ は 2 つ重ねて書きます。
- ^c TAB で形式を変えると、 \ を 1 つで書けます。
- ^c ^w で正規表現をコピーし、 ^c ^q で終えます。

3.4 置換

⌥%	query-replace	1 つずつ確かめながら置換する
^⌥%	query-replace-regexp	正規表現で、確かめながら置換する
⌥x	replace-string	確かめずにすべて置換する
⌥x	replace-regexp	正規表現で、確かめずにすべて置換する

置換の流れは次のとおりです。

1. ⌥% を押し、置換前の文字列を入力して RET を押します。
2. 置換後の文字列を入力して RET を押します。
3. 一致した箇所ごとに、次の表のキーで置換するかを答えます。

- 置換はカーソル位置からバッファの末尾までが対象です。
- リージョンを選んでいるときは、リージョンの中だけが対象です。
- 小文字だけで指定すると、元の大文字と小文字の形に合わせて置換します。例えば hello を bye に置換すると、 Hello は Bye になります。

3.4.1 置換中のキー

キー	動作
SPC 、 y	置換して次へ進む
DEL 、 n	置換せずに次へ進む
!	残りをすべて置換する
.	置換して終える
,	置換し、次へ進まずに結果を見せる
^	前の一致へ戻る
u	直前の置換を取り消す
U	すべての置換を取り消す
e	置換後の文字列を直す
d	置換の結果を差分で見る
RET 、 q	置換を終える
?	キーの一覧を見る

3.4.2 置換後の文字列の書き方

正規表現の置換では、置換後の文字列に次の記号を使えます。

記号	意味
\&	一致した文字列全体
\1 、 \2	1 番目、2 番目の \(\) に一致した文字列
\#	何番目の置換か（0 から数える）
\,(式)	Lisp の式の値

例を示します。

- \(foo\)-\([0-9]+\) を \2_\1 に置換すると、foo-12 は 12_foo になります。
- ^item を \,(1+ \#). \& に置換すると、行頭に 1. から番号が付きます。

3.5 一致した行の一覧

⌥s o は、一致した行を *Occur* バッファに一覧にします。

⌥s o	occur	正規表現に一致した行を一覧にする

Occur バッファでは、次のキーを使います。

RET	occur-mode-goto-occurrence	一致した行へ移動する
o	occur-mode-goto-occurrence-other-window	別のウィンドウで一致した行へ移動する
^o	occur-mode-display-occurrence	一致した行を表示し、一覧に留まる
n	next-error-no-select	次の行を表示する
p	previous-error-no-select	前の行を表示する
e	occur-edit-mode	一覧を編集できるようにする
^c ^c	occur-cease-edit	編集を終える
g	revert-buffer	一覧を作り直す
q	quit-window	一覧を閉じる

- 一覧を編集すると、元のバッファにすぐ反映します。ファイルへの保存は別に行います。

3.6 複数のファイルの検索

⌥x	rgrep	ディレクトリの下のファイルを再帰的に検索する
⌥x	grep	grep コマンドを直接入力して検索する
^x p g	project-find-regexp	プロジェクトのファイルを正規表現で検索する
^x p r	project-query-replace-regexp	プロジェクトのファイルを正規表現で置換する

- rgrep は、検索する文字列、ファイル名の形（ *.swift など)、ディレクトリの順に尋ねます。
- プロジェクトは、git のリポジトリの全体を指します。

3.6.1 grep の結果

rgrep と grep は、結果を *grep* バッファに表示します。

RET	compile-goto-error	一致した箇所へ移動する
^o	compilation-display-error	一致した箇所を表示し、一覧に留まる
n	next-error-no-select	次の箇所を表示する
p	previous-error-no-select	前の箇所を表示する
⌥g n	next-error	ほかのバッファから次の箇所へ移動する
⌥g p	previous-error	ほかのバッファから前の箇所へ移動する
e	grep-change-to-grep-edit-mode	結果を編集できるようにする
^c ^c	grep-edit-save-changes	編集を終える
g	recompile	もう一度検索する
q	quit-window	一覧を閉じる

- 結果の編集は Emacs 31 の機能です。編集は、ファイルを開いたバッファに反映します。ファイルへの保存は ^x s で行います。

3.6.2 プロジェクトの検索の結果

^x p g は、結果を *xref* バッファに表示します。

RET	xref-goto-xref	一致した箇所へ移動する
^o	xref-show-location-at-point	一致した箇所を表示し、一覧に留まる
n	xref-next-line	次の箇所を表示する
p	xref-prev-line	前の箇所を表示する
r	xref-query-replace-in-results	一覧の箇所だけを置換する
e	xref-change-to-xref-edit-mode	結果を編集できるようにする
^c ^c	xref-edit-save-changes	編集を終える
q	quit-window	一覧を閉じる

第4章　バッファとウィンドウとタブの使用

1 つのフレームで複数のファイルを扱うには、ウィンドウを分割するか、タブを使います。この章では、バッファ、ウィンドウ、フレーム、タブの操作を説明します。

4.1 4 つの単位

外側から順に、フレーム、タブ、ウィンドウ、バッファの関係です。

単位	内容	macOS での例え
フレーム	macOS のウィンドウ 1 つ分です。	アプリのウィンドウ
タブ	フレームの中で、ウィンドウの配置を切り替えます。	Safari のタブ
ウィンドウ	フレームを分割した表示領域です。	Xcode のエディタ分割
バッファ	ウィンドウに表示する中身です。	開いた書類

- バッファは、ウィンドウを閉じても残ります。閉じるには `^x k` を使います。
- 1 つのバッファを、複数のウィンドウに同時に表示できます。

4.2 バッファの操作

^x b	ido-switch-buffer	バッファを切り替える
^x 4 b	ido-switch-buffer-other-window	別のウィンドウでバッファを切り替える
^x k	ido-kill-buffer	バッファを閉じる
^x →	next-buffer	次のバッファを表示する
^x ←	previous-buffer	前のバッファを表示する
^x x r	rename-buffer	バッファの名前を変える
^x ^b	list-buffers	バッファの一覧を表示する

- ^x b と ^x k では ido で候補を選びます。キーは第 1 章の「ido のキー」を見てください。
- 同じ名前のファイルを開くと、バッファ名にディレクトリ名を足して区別します。例えば a/init.el と b/init.el は、init.el<a> と init.el<b> になります。

4.2.1 バッファの一覧のキー

^x ^b で開く *Buffer List* では、次のキーを使います。d などで印を付け、 x でまとめて実行します。

RET	Buffer-menu-this-window	このウィンドウでバッファを開く
o	Buffer-menu-other-window	別のウィンドウでバッファを開く
^o	Buffer-menu-switch-other-window	別のウィンドウに表示し、一覧に留まる
d	Buffer-menu-delete	閉じる印を付ける
s	Buffer-menu-save	保存する印を付ける
u	Buffer-menu-unmark	印を消す
x	Buffer-menu-execute	印の付いた操作を実行する
%	Buffer-menu-toggle-read-only	読み出し専用を入/切する
T	Buffer-menu-toggle-files-only	ファイルのバッファだけを表示する
S	tabulated-list-sort	カーソルのある列で並べ替える
g	revert-buffer	一覧を更新する
q	quit-window	一覧を閉じる

4.3 ウィンドウの操作

^x 2	split-window-below	上下に分割する
^x 3	split-window-right	左右に分割する
^x 0	delete-window	今のウィンドウを閉じる
^x 1	delete-other-windows	今のウィンドウ以外を閉じる
^x 4 0	kill-buffer-and-window	バッファとウィンドウを閉じる
^x o	ace-window	番号を選んでウィンドウへ移動する
⌥TAB	other-window	次のウィンドウへ移動する
^⌥v	scroll-other-window	次のウィンドウを下へスクロールする
^⌥⇧v	scroll-other-window-down	次のウィンドウを上へスクロールする
^x 4 f	ido-find-file-other-window	別のウィンドウでファイルを開く

- ^x o と ⌥TAB は、init.el で割り当てています。
- ^x 4 で始まるキーは、結果を別のウィンドウに表示します。

4.3.1 ace-window

^x o は、標準の other-window の代わりに ace-window を呼びます。

- ウィンドウが 2 つのときは、もう一方へすぐ移動します。
- 3 つ以上のときは、各ウィンドウに番号が出ます。 1 から 9 で選びます。
- ほかのフレームのウィンドウも候補に入ります。
- ^u ^x o は、選んだウィンドウと今のウィンドウのバッファを入れ替えます。
- ^u ^u ^x o は、選んだウィンドウを閉じます。

番号が出ている間に次のキーを押すと、移動の代わりに別の操作をします。

キー	動作
x	次に選んだウィンドウを閉じる
m	次に選んだウィンドウとバッファを入れ替える
o	次に選んだウィンドウ以外を閉じる
?	操作の一覧を表示する

4.3.2 ウィンドウの大きさ

^x ^	enlarge-window	高さを 1 行増やす
^x }	enlarge-window-horizontally	幅を 1 桁増やす
^x {	shrink-window-horizontally	幅を 1 桁減らす
^x +	balance-windows	ウィンドウの大きさをそろえる
^x -	shrink-window-if-larger-than-buffer	中身に合わせてウィンドウを縮める

- 数引数で量を指定できます。例えば ^u 10 ^x } は幅を 10 桁増やします。
- ウィンドウの境目をマウスでドラッグしても、大きさを変えられます。

4.3.3 配置の入れ替え

Emacs 31 では、分割したウィンドウの配置をまとめて変えるコマンドが加わりました。

^x w t	window-layout-transpose	上下の分割と左右の分割を入れ替える
^x w r →	window-layout-rotate-clockwise	配置を右に 90 度回す
^x w f ↓	window-layout-flip-topdown	配置を上下に反転する
^x w f →	window-layout-flip-leftright	配置を左右に反転する
^x w o →	rotate-windows	各ウィンドウのバッファを順に送る

4.4 フレームの操作

⌘n	make-frame	新しいフレームを開く
^x 5 2	make-frame-command	新しいフレームを開く
⌘w	delete-frame	今のフレームを閉じる
^x 5 0	delete-frame	今のフレームを閉じる
^x 5 1	delete-other-frames	今のフレーム以外を閉じる
⌘`	other-frame	次のフレームへ切り替える
⌘~	ns-prev-frame	前のフレームへ切り替える
^x 5 o	other-frame	次のフレームへ切り替える
^x 5 f	ido-find-file-other-frame	新しいフレームでファイルを開く
^x 5 b	ido-switch-buffer-other-frame	新しいフレームでバッファを切り替える

- ⌘~ は ⇧⌘` で押します。
- ^x 5 で始まるキーは、結果を新しいフレームに表示します。
- 最後のフレームは ⌘w で閉じられません。Emacs を終えるには ⌘q を使います。
- フレームを複数開いても、macOS のタブにはまとまりません。NS 版の Emacs は、macOS のタブ機能を無効にしています。タブは、次の節の Emacs のタブを使います。

4.5 タブの操作

Emacs のタブは、フレームの上端のタブバーに並びます。タブごとに、ウィンドウの分割と表示するバッファを覚えています。

^x t 2	tab-new	新しいタブを開く
^x t 0	tab-close	今のタブを閉じる
^x t 1	tab-close-other	今のタブ以外を閉じる
^x t u	tab-undo	閉じたタブを開き直す
^x t o	tab-next	次のタブへ切り替える
^x t O	tab-previous	前のタブへ切り替える
^TAB	tab-next	次のタブへ切り替える
^⇧TAB	tab-previous	前のタブへ切り替える
^x t RET	tab-switch	名前を選んでタブを切り替える
^x t r	tab-rename	タブの名前を変える
^x t m	tab-move	タブを右へ動かす
^x t f	find-file-other-tab	新しいタブでファイルを開く
^x t b	switch-to-buffer-other-tab	新しいタブでバッファを開く
^x t d	dired-other-tab	新しいタブで Dired を開く

- init.el では tab-bar-mode を有効にしていません。^x t 2 などで 2 つ目のタブを開くと、自動で有効になります。
- ^TAB と ^⇧TAB は、 tab-bar-mode が有効なときだけ使えます。
- タブの名前は、初期設定ではそのタブで選んでいるバッファの名前です。
- タブバーのタブは、マウスでクリックしても切り替えられます。

4.5.1 フレームとタブの使い分け

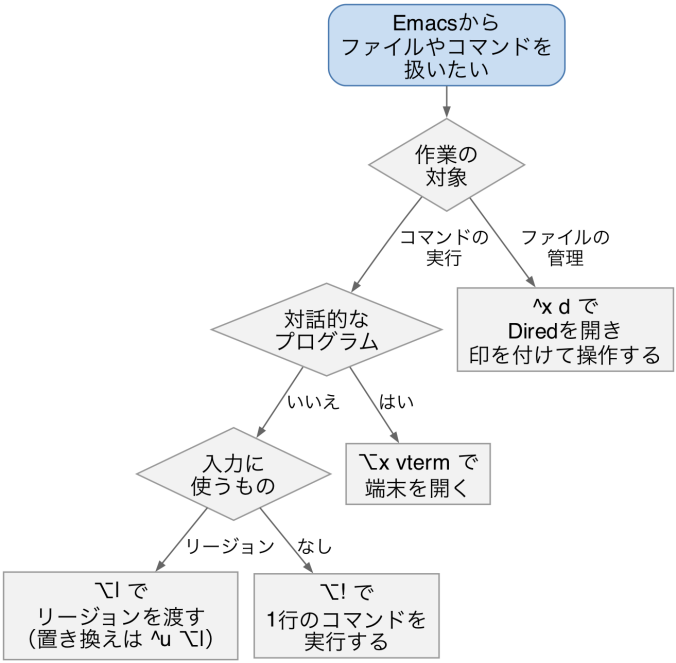
	フレーム	タブ
向いている使い方	2 つの画面を並べて見る	1 つの画面で作業を切り替える
切り替えのキー	⌘`	^TAB
開くキー	⌘n	^x t 2
Mission Control	別のウィンドウとして出る	出ない

第5章　作業環境としてのEmacs

Emacs を離れずに、ファイルの管理とシェルの作業ができます。この章では、Dired、vterm、外部コマンドの 3 つを説明します。

機能	向いている作業	macOS での例え
Dired	ファイルのコピー、移動、削除、名前の変更	Finder
vterm	対話的なプログラムやシェルでの作業	ターミナル
外部コマンド	1 行のコマンドの実行、テキストの加工	ターミナルで 1 行だけ実行する

作業の対象と、コマンドの入力に何を使うかで選びます。



5.1 Dired

Dired は、ディレクトリの中身を一覧で表示し、ファイルを操作するモードです。名前は Directory Editor の略です。

5.1.1 Dired を開く

^x d	ido-dired	ディレクトリを選んで Dired を開く
^x 4 d	ido-dired-other-window	別のウィンドウで Dired を開く
^x ^j	dired-jump	今のファイルのディレクトリを開く
^x 4 ^j	dired-jump-other-window	別のウィンドウで ^x ^j をする

- init.el で ido を有効にしているので、 ^x d は ido-dired を呼びます。
- RET で候補のディレクトリに入り、 ^j でそのディレクトリを開きます。
- ^x ^j は、カーソルを今のファイルの行に置いて開きます。

5.1.2 Dired の画面

1 行目はディレクトリのパスです。2 行目からは ls -l と同じ形で並びます。

```
  /Users/user/Project:
  total used in directory 24 available 120 GiB
  drwxr-xr-x  5 user  staff  160 11 月 10 10:36 .
  drwxr-xr-x 12 user  staff  384 11 月  9 23:26 ..
* -rw-r--r--  1 user  staff 2048 11 月 10 10:30 README.md
D -rw-r--r--  1 user  staff 1024 11 月  9 21:15 old.txt
```

- 行頭の * はマーク、 D は削除フラグです。
- 開いたときに「ls does not support --dired」と出ます。macOS の ls に --dired オプションがないためで、表示と操作に支障はありません。

5.1.3 移動と表示

n	dired-next-line	次の行へ移動する
p	dired-previous-line	前の行へ移動する
>	dired-next-dirline	次のディレクトリの行へ移動する
<	dired-prev-dirline	前のディレクトリの行へ移動する
j	dired-goto-file	名前を入力してその行へ移動する
RET	dired-find-file	ファイルを開く
o	dired-find-file-other-window	別のウィンドウでファイルを開く
^o	dired-display-file	別のウィンドウに表示し、Dired に留まる
v	dired-view-file	読むだけのモードで開く
^	dired-up-directory	親ディレクトリへ移動する
i	dired-maybe-insert-subdir	サブディレクトリを同じバッファに足す
s	dired-sort-toggle-or-edit	名前順と日付順を切り替える
(	dired-hide-details-mode	名前以外の情報を隠す/出す
g	revert-buffer	表示を更新する
q	quit-window	Dired のウィンドウを閉じる

- ^n と ^p でも移動できます。
- v で開いたファイルは、 q で閉じて Dired へ戻ります。

5.1.4 印を付ける

Dired では、ファイルに印を付けてから、まとめて操作します。印にはマーク（ * ）と削除フラグ（ D ）の 2 種類があります。

m	dired-mark	マークを付ける
u	dired-unmark	印を外す
DEL	dired-unmark-backward	前の行の印を外す
U	dired-unmark-all-marks	すべての印を外す
t	dired-toggle-marks	マークの有無を反転する
% m	dired-mark-files-regexp	正規表現に合うファイルにマークを付ける
d	dired-flag-file-deletion	削除フラグを付ける
~	dired-flag-backup-files	バックアップファイルに削除フラグを付ける
#	dired-flag-auto-save-files	自動保存ファイルに削除フラグを付ける
x	dired-do-flagged-delete	削除フラグの付いたファイルを削除する

- 印を付けたファイルがないときは、カーソルのある行のファイルが対象です。
- 数引数を付けると、カーソルから下の指定した数のファイルが対象です。

5.1.5 ファイルの操作

次のキーは、マークの付いたファイルに働きます。

C	dired-do-copy	コピーする
R	dired-do-rename	名前を変える・移動する
D	dired-do-delete	すぐに削除する
+	dired-create-directory	ディレクトリを作る
S	dired-do-symlink	シンボリックリンクを作る
M	dired-do-chmod	アクセス権を変える
T	dired-do-touch	更新日時を今にする
Z	dired-do-compress	圧縮する/展開する
c	dired-do-compress-to	1 つのアーカイブにまとめる
=	dired-diff	別のファイルとの差分を表示する
E	dired-do-open	macOS の既定のアプリで開く
w	dired-copy-filename-as-kill	ファイル名をキルリングに入れる
A	dired-do-find-regexp	正規表現でファイルの中を検索する
Q	dired-do-find-regexp-and-replace	正規表現でファイルの中を置換する

- E は open コマンドを使います。Finder でダブルクリックしたのと同じです。
- ^u 0 w は、ファイル名を絶対パスで入れます。
- A と Q の使い方は、第 3 章の「複数のファイルの検索」と同じです。

5.1.6 削除とゴミ箱

初期設定では、 D と x はファイルを完全に削除します。macOS のゴミ箱に移すには、init.el に次の 1 行を足します。

```
(setq delete-by-moving-to-trash t)
```

- 今の init.el には、この設定はありません。

- 削除は確認を求めます。 `yes` と入力するまで実行しません。

5.1.7 2 つの Dired の間でコピーする

コピー元とコピー先の Dired を左右のウィンドウに並べると、作業が楽になります。

1. `^x 3` でウィンドウを分け、両方で Dired を開きます。
2. コピー元のウィンドウでファイルにマークを付けます。
3. `C` を押し、コピー先を入力します。

- `dired-dwim-target` を `t` にすると、コピー先の初期値が隣の Dired のディレクトリになります。
- `R` での移動も同じです。今の init.el には、この設定はありません。

5.1.8 ファイル名をまとめて変える（wdired）

`^x ^q` で、Dired のバッファを普通のテキストのように編集できます。ファイル名を書き換えて確定すると、実際のファイル名が変わります。

`^x ^q`	`dired-toggle-read-only`	ファイル名の編集を始める
`^c ^c`	`wdired-finish-edit`	編集を確定し、名前を変える
`^c ^k`	`wdired-abort-changes`	編集を取り消す

- 第 3 章の置換（`⌥%`）や矩形の編集と組み合わせると、一括で名前を変えられます。
- `^x ^s` でも確定できます。

5.2 vterm

vterm は、Emacs の中で動く端末です。ターミナルと同じように、zsh や対話的なプログラムを使えます。

5.2.1 vterm の導入

vterm は MELPA のパッケージです。init.el では次のように設定しています。

```
(use-package vterm
  :ensure t
  :config
  (setq vterm-shell "zsh")
  (setq vterm-max-scrollback 10000))
```

変数	値	意味
vterm-shell	"zsh"	起動するシェル
vterm-max-scrollback	10000	さかのぼって残す行数

vterm は、C 言語で書いたモジュール vterm-module を使います。初めて起動すると、「Compile it now?」とコンパイルを求めます。y で答えます。コンパイルには、次のものが必要です。

- cmake：コンパイルの手順を作るツールです。
- libtool：vterm が使うライブラリ libvterm を作るときに使います。

どちらも Homebrew で brew install cmake libtool として入れます。

5.2.2 zsh との連携

zsh の設定に vterm 付属のスクリプトを読み込むと、次のことができるようになります。

- シェルで cd すると、Emacs 側の作業ディレクトリも変わります。vterm で ^x ^f を押すと、シェルのディレクトリから探せます。
- ^c ^n と ^c ^p で、プロンプトの位置を正確に移動できます。
- clear コマンドで、さかのぼる分の行も消えます。

スクリプトは、パッケージのディレクトリの etc/emacs-vterm-zsh.sh です。~/.zshrc の末尾に、次のように書きます。

```
if [[ "$INSIDE_EMACS" = 'vterm' ]]; then
    vterm_sh=(~/.emacs.d/elpa/vterm-*/etc/emacs-vterm-zsh.sh(N))
    (( $#vterm_sh )) && source $vterm_sh[-1]
    unset vterm_sh
fi
```

- INSIDE_EMACS は、vterm がシェルに渡す環境変数です。値は vterm です。
- vterm を更新すると、ディレクトリ名の日付の部分が変わります。そのため * で探し、見つかったうちの最後の 1 つを読み込みます。
- (N) は、見つからないときに空にする zsh の指定です。
- プロンプトに印を足すので、 PROMPT を設定した後に書きます。

5.2.3 vterm を開く

⌥x	vterm	vterm を開く
⌥x	vterm-other-window	別のウィンドウで vterm を開く

- バッファ名は *vterm* です。すでにあれば、そのバッファへ切り替えます。
- ^u ⌥x vterm は、別の vterm を新しく開きます。名前は *vterm*<2> などになります。
- シェルで exit を実行すると、バッファも閉じます。

5.2.4 vterm のキー

vterm では、ほとんどのキーをそのまま端末へ送ります。次のキーだけは、Emacs が受け取ります。

キー	理由
^c ^x ^u	Emacs のプレフィックスキーとして使う
^g ^l	中断と画面の消去に使う
^h	ヘルプに使う（init.el では DEL に変わる）
⌥x ⌥o	コマンドの実行などに使う
^y ⌥y	キルリングから貼り付けに使う

- init.el の設定で、vterm でも ^h で 1 文字消えます。
- ⌘v と ^y は、キルリングの文字列を端末へ送ります。

^c ^c	vterm--self-insert	端末へ ^c を送り、実行中のプログラムを止める
^c ^t	vterm-copy-mode	コピーモードを入/切する
^c ^n	vterm-next-prompt	次のプロンプトへ移動する
^c ^p	vterm-previous-prompt	前のプロンプトへ移動する
^l	vterm-clear	画面を消す
^c ^l	vterm-clear-scrollback	さかのぼる分の行も消す

5.2.5 コピーモード

vterm の出力をコピーするには、コピーモードを使います。コピーモードでは、普通のバッファと同じキーでカーソルを動かせます。

1. ^c ^t でコピーモードに入ります。
2. ^SPC でマークを置き、コピーする範囲を選びます。
3. RET を押します。範囲をキルリングに入れ、コピーモードを抜けます。

- 範囲を選ばずに RET を押すと、カーソルのある行をコピーします。
- コピーせずに抜けるときは、もう一度 ^c ^t を押します。
- コピーモードの間は、端末の出力が止まって見えます。抜けると最新の状態に戻ります。

5.3 外部コマンド

Emacs からシェルのコマンドを 1 回だけ実行できます。実行するディレクトリは、今のバッファのディレクトリです。

⌥!	shell-command	コマンドを実行し、出力を表示する
^u ⌥!	shell-command	出力をカーソルの位置に挿入する
⌥&	async-shell-command	コマンドを終わりを待たずに実行する
⌥|	shell-command-on-region	リージョンをコマンドの入力に渡す
^u ⌥|	shell-command-on-region	リージョンをコマンドの出力で置き換える

- 出力が短いときはエコーエリアに出ます。長いときは *Shell Command Output* に出ます。
- ⌥& の出力は *Async Shell Command* に出ます。時間のかかるビルドなどに使います。
- シェルは zsh です。 ~/.zshrc は読み込みません。

5.3.1 使用例

入力	結果
⌥! git status	作業ツリーの状態を表示する
^u ⌥! date	今の日時をカーソルの位置に入れる
⌥| wc -l	リージョンの行数を表示する
^u ⌥| sort	リージョンの行を並べ替える
^u ⌥| jq .	リージョンの JSON を整形する
⌥& make	ビルドを裏で実行する

5.3.2 Dired から実行する

Dired では、マークの付いたファイルにコマンドを実行できます。

!	dired-do-shell-command	ファイルにコマンドを実行する
&	dired-do-async-shell-command	ファイルに終わりを待たずに実行する

コマンドの中の * と ? で、ファイル名を入れる位置と回数を決めます。どちらも前後に空白が必要です。

書き方	例	動作
* を含む	tar czf a.tgz *	全ファイルを並べて 1 回実行する
? を含む	cp ? ?.bak	ファイルごとに ? を置き換えて実行する
どちらもない	wc -l	ファイルごとに末尾に名前を足して実行する

5.3.3 コマンドが見つからないとき

⌥! で「command not found」と出るときは、Emacs の PATH を確かめます。

- ⌥x getenv RET PATH RET で、Emacs が使う PATH を表示します。
- exec-path は、Emacs がプログラムを探すディレクトリの一覧です。F1 v exec-path RET で表示します。

5.4 チェックリスト

5.4.1 やるべきこと

- ファイルの削除や名前の変更は、印を付けてから実行する。
- 大事なファイルを消す前に、 delete-by-moving-to-trash を設定する。
- ~/.zshrc に、vterm 付属のスクリプトを読み込む設定を書く。
- 時間のかかるコマンドは ⌥& か vterm で実行する。

5.4.2 やってはいけないこと

- 対話的なプログラム（ ssh や top など）を ⌥! で実行する。入力を受け付けず、止まります。vterm を使います。
- wdired で編集したまま、確定も取り消しもせずにバッファを閉じる。
- ^u ⌥| の結果を確かめずに保存する。失敗しても ^/ で元に戻せます。

第6章　Emacsとインターネット

Emacs の中で Web ページと掲示板を読めます。この章では、Web ブラウザの w3m と、5 ちゃんねる用のブラウザ navi2ch を説明します。

機能	向いている作業	起動
w3m	文字中心の Web ページやマニュアルを読む	`⌥x w3m`
navi2ch	5 ちゃんねるのスレを読み、書き込む	`⌥x navi2ch`
macOS のブラウザ	JavaScript を使うページを見る	w3m で `M`

6.1 w3m

w3m は、端末で動くテキストの Web ブラウザです。emacs-w3m は、w3m を使って Emacs のバッファにページを表示するパッケージです。

- JavaScript は動きません。動きのあるページは macOS のブラウザで開きます。
- 画像は、ページの中に表示できます。

6.1.1 w3m の導入

w3m の本体と emacs-w3m の両方が必要です。

部品	入れ方	置き場所
w3m	`brew install w3m`	`/opt/homebrew/bin/w3m`
emacs-w3m	MELPA の `w3m`	`~/.emacs.d/elpa/w3m-*`

init.el では次のように設定しています。

```
(leaf w3m-load
  :ensure w3m
  :require t
  :config
  (setq w3m-default-display-inline-images t))
```

- `w3m-default-display-inline-images` を `t` にすると、最初から画像を表示します。
- ブックマークや履歴は `~/.w3m/` に保存します。

6.1.2 ページを開く

⌥x	`w3m`	w3m を開く
⌥x	`w3m-browse-url`	URL を入力して開く
⌥x	`w3m-search`	検索エンジンで検索する
⌥x	`w3m-find-file`	手元の HTML ファイルを開く

- `w3m` は、すでに w3m のバッファがあれば、そのバッファへ切り替えます。
- `w3m-browse-url` は、カーソルの位置にある URL を初期値にします。
- 検索エンジンの初期値は Google です。 `^u` を付けると、エンジンを選べます。

6.1.3 w3m のキー

g	w3m-goto-url	URL を入力して開く
G	w3m-goto-url-new-session	URL を新しいバッファで開く
RET	w3m-view-this-url	リンクをたどる
⇧RET	w3m-view-this-url-new-session	リンクを新しいバッファで開く
TAB	w3m-next-anchor	次のリンクへ移動する
⇧TAB	w3m-previous-anchor	前のリンクへ移動する
SPC	w3m-scroll-up-or-next-url	1 画面先へ進む
DEL	w3m-scroll-down-or-previous-url	1 画面前へ戻る
B	w3m-view-previous-page	前のページへ戻る
N	w3m-view-next-page	次のページへ進む
^	w3m-view-parent-page	上の階層のページを開く
R	w3m-reload-this-page	ページを読み込み直す
s	w3m-search	検索する
M	w3m-view-url-with-browse-url	macOS の既定のブラウザで開く
c	w3m-print-current-url	今のページの URL をキルリングに入れる
u	w3m-print-this-url	リンク先の URL をキルリングに入れる
d	w3m-download-this-url	リンク先をダウンロードする
T	w3m-toggle-inline-images	画像を表示する/隠す
a	w3m-bookmark-add-current-url	今のページをブックマークに入れる
v	w3m-bookmark-view	ブックマークを表示する
q	w3m-close-window	w3m のウィンドウを閉じる
Q	w3m-quit	w3m を終える

- 矢印キーも使えます。← は前のページ、→ はリンクをたどります。↑と↓は

前後のリンクへ移動します。

- q はバッファを残します。 ⌥x w3m で同じページに戻れます。
- Q は w3m のバッファをすべて閉じます。

6.1.4 複数のページ

G や ⇧RET で開いたページは、別のバッファになります。

^c ^n	w3m-next-buffer	次の w3m のバッファへ切り替える
^c ^p	w3m-previous-buffer	前の w3m のバッファへ切り替える
^c ^s	w3m-select-buffer	w3m のバッファの一覧から選ぶ
^c ^w	w3m-delete-buffer	今の w3m のバッファを閉じる

6.2 navi2ch

navi2ch は、5 ちゃんねる（旧 2 ちゃんねる）の専用ブラウザです。板の一覧、スレの一覧、スレの本文を、Emacs のバッファで読みます。

6.2.1 用語

用語	意味
板	話題ごとの掲示板です。「Mac」「UNIX」などがあります。
スレ	板の中の、1 つの話題のまとまりです。スレッドの略です。
レス	スレへの 1 件の書き込みです。番号が付きます。
sage	書き込んでもスレを一覧の上に上げない指定です。
age	書き込んだスレを一覧の上に上げることです。

6.2.2 navi2ch の導入

navi2ch は、開発版を ~/.emacs.d/lisp/navi2ch/ に置いています。ELPA のパッケージではないので、init.el で読み込む場所を指定します。

- 版は 2.0.0-DEV です。 ⌥x navi2ch-version で確かめます。
- 本体は、 ⌥x navi2ch を実行したときに初めて読み込みます。
- 読んだスレや設定は ~/.navi2ch/ に保存します。
- info 形式のマニュアルが付属しています。^u F1 i で ~/.emacs.d/lisp/navi2ch/navi2ch.info.gz を指定して読みます。

6.2.3 init.el の設定

init.el の (leaf navi2ch ...) で、次の変数を設定しています。

変数	設定の内容
navi2ch-list-bbstable-url	板の一覧を 5 ちゃんねるの bbsmenu.json から取る
navi2ch-list-valid-host-regexp	板として扱うホスト（ 5ch.io 、 5ch.net など）
navi2ch-list-invalid-host-regexp	板として扱わないホスト（案内、検索、課金など）
navi2ch-list-stay-list-window	板を開いた後も、板の一覧を表示したままにする
navi2ch-article-new-message-range	初めて読むスレは、先頭の 1000 件と末尾の 1 件を表示する
navi2ch-article-exist-message-range	読んだことのあるスレは、先頭の 1 件と末尾の 1000 件を表示する
navi2ch-mona-enable	モナーフォントでスレを表示する
navi2ch-mona-use-ipa-mona	フォントを名前で指定する方式にする
navi2ch-mona-ipa-mona-font-family-name	モナーフォントに Mona を使う
navi2ch-net-save-old-file-when-aborn	レスが削除されたとき、元のファイルを残さない
navi2ch-net-user-agent	通信に使うブラウザ名
navi2ch-list-ignore-category-list	「BBSPINK」のカテゴリを一覧に出さない

- 表示範囲の初期値は 100 件です。init.el で 1000 件に広げています。
- 表示範囲は、スレを開いた後に r で変えられます。

6.2.4 モナーフォント

モナーフォントは、アスキーアート（文字で描いた絵）を崩さずに表示するフォントです。~/Library/Fonts/mona-mac.ttf に入っています。フォント名は Mona です。

navi2ch が探すフォント名の初期値は mona-gothic です。macOS には無いので、そ

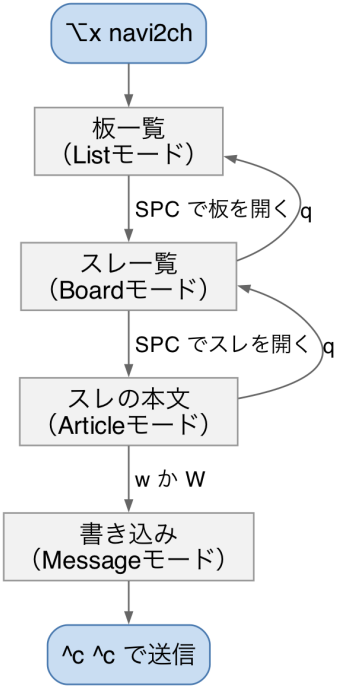
のままでは Menlo で表示されます。init.el では次の 2 行で、Mona を使うようにしています。

```
(navi2ch-mona-use-ipa-mona . t)
(navi2ch-mona-ipa-mona-font-family-name . "Mona")
```

- navi2ch-mona-use-ipa-mona を t にすると、フォントを名前で直接指定する方式になります。
- このとき使うフォント名は navi2ch-mona-ipa-mona-font-family-name です。navi2ch-mona-font-family-name ではありません。
- 設定は ⌥x navi2ch で起動したときに効きます。

6.2.5 画面の移り方

navi2ch は、板の一覧からスレの本文へと順に開きます。q で 1 つ前の画面に戻ります。


⌥x navi2ch
板一覧
（Listモード）
SPC で板を開く
q
スレ一覧
（Boardモード）
SPC でスレを開く
q
スレの本文
（Articleモード）
w か W
書き込み
（Messageモード）
^c ^c で送信

6.2.6 どの画面でも使うキー

1	navi2ch-one-pane	今の画面だけを表示する
2	navi2ch-two-pane	2 つの画面を上下に並べる
3	navi2ch-three-pane	板、スレ、本文の 3 つを並べる
n	next-line など	次の行やレスへ移動する
p	previous-line など	前の行やレスへ移動する
B	navi2ch-bookmark-goto-bookmark	グローバルブックマークを表示する
t	navi2ch-toggle-offline	オフラインとオンラインを切り替える
^c ^u	navi2ch-goto-url	URL を入力して板やスレを開く

- オフラインでは通信せず、保存済みのスレだけを読みます。
- 書き込み中の画面では、 1 などの 1 文字のキーは文字の入力になります。

6.2.7 板の一覧

起動すると、板の一覧がカテゴリごとに出ます。

SPC	navi2ch-list-select-current-board	カテゴリを開く/閉じる、板を開く
[	navi2ch-list-open-all-category	すべてのカテゴリを開く
]	navi2ch-list-close-all-category	すべてのカテゴリを閉じる
a	navi2ch-list-add-bookmark	板をブックマークに入れる
b	navi2ch-list-toggle-bookmark	一覧とブックマークを切り替える
?	navi2ch-list-search	スレを検索する
s	navi2ch-list-sync	板の一覧を取り直す
z	navi2ch-suspend	navi2ch を中断する
q	navi2ch-exit	navi2ch を終える

- z はバッファを残します。 ⌥x navi2ch ですぐ戻れます。
- q は確認の後、navi2ch のバッファをすべて閉じます。

6.2.8 スレの一覧

板を開くと、スレの一覧が出ます。行頭の記号で、スレの状態が分かります。

```
16 +C くだらない質問はここに書き込め！          ( 636)
17 =C*navi2ch for emacs (part 4)                (  50)
18  U Unix の質問スレ                           ( 101)
```

左から、スレ番号、更新フラグ、状態フラグ、マーク、題名、レス数です。

記号	種類	意味
%	更新フラグ	前に板を開いたときには無かった新しいスレ
+	更新フラグ	前に板を開いた後に、レスが増えたスレ
=	更新フラグ	前に板を開いた後に、レスが増えていないスレ
V	状態フラグ	今バッファに開いているスレ
C	状態フラグ	手元に保存してあるスレ
U	状態フラグ	i で読み込んだスレ
*	マーク	* で印を付けたスレ

SPC	navi2ch-bm-select-article-or-scroll-up	スレを開く、本文を先へ送る
DEL	navi2ch-bm-select-article-or-scroll-down	スレを開く、本文を前へ戻す
i	navi2ch-bm-fetch-article	スレを開かずに読み込む
s	navi2ch-board-sync	スレの一覧を取り直す
+	navi2ch-board-toggle-updated	更新されたスレだけを表示する/戻す
S	navi2ch-bm-sort	スレを並べ替える
?	navi2ch-bm-search	スレを検索する
*	navi2ch-bm-mark	スレに印を付ける
u	navi2ch-bm-unmark	スレの印を外す
a	navi2ch-board-add-bookmark	スレを板のブックマークに入れる
b	navi2ch-board-toggle-bookmark	一覧と板のブックマークを切り替える
A	navi2ch-bm-add-global-bookmark	スレをグローバルブックマークに入れる
d	navi2ch-board-hide-article	スレを隠す
h	navi2ch-board-toggle-hide	隠したスレを表示する/戻す
q	navi2ch-bm-exit	板を閉じる

- 板のブックマークはその板の中だけで使います。グローバルブックマークは、複数の板のスレをまとめます。 `B` で開きます。
- `m` で始まるキーは、印を付けたスレにまとめて働きます。例えば `m i` は、印を付けたスレを読み込みます。

6.2.9 スレを読む

スレを開くと、上にスレの一覧、下に本文が出ます。

SPC	navi2ch-article-scroll-up	1 画面先へ進む
DEL	navi2ch-article-scroll-down	1 画面前へ戻る
n	navi2ch-article-next-message	次のレスへ移動する
p	navi2ch-article-previous-message	前のレスへ移動する
<	navi2ch-article-goto-first-message	最初のレスへ移動する
>	navi2ch-article-goto-last-message	最後のレスへ移動する
g	navi2ch-article-goto-number-or-board	番号を入力してそのレスへ移動する
TAB	navi2ch-article-next-link	次のリンクへ移動する
RET	navi2ch-article-select-current-link	リンク先のレスや URL を開く
l	navi2ch-article-pop-point	リンクをたどる前のレスへ戻る
L	navi2ch-article-pop-poped-point	l で戻る前の位置へ進む
s	navi2ch-article-sync	新しいレスを読み込む
r	navi2ch-article-redraw-range	表示するレスの範囲を変える
?	navi2ch-article-search	レスを検索する
a	navi2ch-article-add-important-message	レスをスレのブックマークに入れる
$	navi2ch-article-toggle-important	スレのブックマークを表示する/戻す
d	navi2ch-article-hide-message	レスを隠す
h	navi2ch-article-toggle-hide	隠したレスを表示する/戻す
A	navi2ch-article-add-global-bookmark	スレをグローバルブックマークに入れる
U	navi2ch-article-show-url	スレやレスの URL をコピーする
q	navi2ch-article-exit	スレを閉じる

- RET で >>5 のようなレスへの参照を開くと、そのレスへ移動します。
- RET で https:// のリンクを開くと、 browse-url でブラウザを呼びます。
- ? は、名前、ID、本文などから検索する項目を選んでから入力します。

6.2.10 書き込む

w	navi2ch-article-write-message	レスを書く
W	navi2ch-article-write-sage-message	レスを sage で書く
^c ^y	navi2ch-message-cite-original	見ていたレスを引用する
^c ^c	navi2ch-message-send-message	書き込みを送る
^c ^k	navi2ch-message-exit	書き込みをやめる
^c ^o	navi2ch-message-jump-to-message-buffer	書きかけの画面に戻る

書き込みの画面は次の形です。区切りの線の下に本文を書きます。

```
From:
Mail: sage
----------------
本文
```

- From: は名前です。空のままでも書き込めます。
- Mail: に sage と書くと、sage で書き込みます。 W では最初から入っています。
- ^u ^c ^y は、レスの番号だけを >>5 の形で入れます。

6.3 チェックリスト

6.3.1 やるべきこと

- JavaScript が必要なページは、w3m で `M` を押して macOS のブラウザで開く。
- navi2ch をしばらく使わないときは `z` で中断し、すぐ戻れるようにする。
- 書き込む前に、 `Mail:` 欄と本文を確かめる。

6.3.2 やってはいけないこと

- w3m で、ログインや支払いの必要なページを使う。表示が崩れ、操作を誤りやすくなります。
- navi2ch で、 `I` や `m i` で大量のスレを一度に読み込む。サーバーに負荷をかけます。
- 読み直さずに `^c ^c` で送る。送ったレスは消せません。

第7章　シンプルなテキストフォーマッティングと特殊な編集

文章の形を整える操作と、まとめて編集する操作を説明します。

節	できること
段落の整形	段落を決まった幅で折り返す
長い行の表示	画面の幅を超える行の見せ方を変える
インデント	行の先頭の空白をそろえる、ずらす
矩形の編集	四角い範囲を切り取る、文字を差し込む
文字の種類の変換	大文字と小文字、全角と半角を変える
行の並べ替えと整列	行を並べ替える、桁をそろえる
特殊な文字の入力	名前や絵文字の一覧から文字を入れる
キーボードマクロ	一連の操作を記録して繰り返す

7.1 段落の整形

段落を、決まった幅（`fill-column`）で折り返すことを「フィル」と言います。段落は、空行で区切られたひとまとまりの行です。

⌥q	fill-paragraph	段落を折り返す
^x f	set-fill-column	折り返す幅を決める
⌥x	auto-fill-mode	入力中に自動で折り返す/やめる
⌥x	center-line	行を中央にそろえる
⌥x	center-region	リージョンの各行を中央にそろえる

- fill-column の初期値は 70 桁です。全角文字は 1 文字で 2 桁と数えます。
- ^x f は、新しい幅を尋ねます。 ^u ^x f は、カーソルのある桁を幅にします。
- リージョンを選んで ⌥q を押すと、リージョンの中の段落をすべて折り返します。
- ^u ⌥q は、空白を足して行の右端もそろえます。

7.1.1 日本語の折り返し

日本語は、語の切れ目に関係なく、幅に合わせて折り返します。句読点が行頭に来ないように調整します（禁則処理）。

```
日本語の文章を折り返
します。句読点は行頭
に来ないように調整さ
れます。
```

- 折り返した改行は、ファイルにそのまま残ります。
- Markdown などで 1 段落を 1 行で書くときは、フィルを使わずに次の節の表示の切り替えを使います。

7.2 長い行の表示

画面の幅を超える行は、初期設定では次の行に続けて表示します。ファイルの中身は変わりません。表示だけを切り替えます。

^c l	toggle-truncate-lines	右端で切る/次の行に続ける
⌥x	visual-line-mode	語の切れ目で折り返して表示する

- ^c l は、init.el で割り当てています。
- 右端で切ると、行の続きは画面の外に隠れます。カーソルを動かすと横にスクロールします。
- visual-line-mode では、 ^k が表示上の 1 行をキルします。init.el で置き換えた ^a と ^e は、ファイル上の行の先頭と末尾へ移動します。

7.3 インデント

init.el では、インデントに次の設定をしています。

変数	値	意味
tab-width	4	タブ文字を 4 桁で表示する
indent-tabs-mode	nil	インデントにタブ文字を使わず、空白を使う
tab-always-indent	complete	TAB でインデントし、インデント済みなら補完する

TAB	indent-for-tab-command	行をインデントする。インデント済みなら補完する
^⌥\	indent-region	リージョンの各行をインデントする
^x TAB	indent-rigidly	リージョンをまとめて左右にずらす
⌥i	tab-to-tab-stop	次のタブ位置まで空白を入れる
⌥m	back-to-indentation	行の最初の空白でない文字へ移動する
⌥^	delete-indentation	今の行を前の行につなげる
⌥x	untabify	リージョンのタブ文字を空白に変える
⌥x	tabify	リージョンの空白をタブ文字に変える

^x TAB の後は、次のキーでずらす量を調整します。ほかのキーで終わります。

キー	動作
← →	1 桁ずつ左右にずらす
⇧← ⇧→	タブ位置まで左右にずらす

- ^u 4 ^x TAB のように数引数を付けると、その桁数だけ右にずらします。負の数で左にずらします。

7.4 矩形の編集

矩形は、リージョンの始めと終わりを角とする四角い範囲です。表の列を消したり、各行の先頭に同じ文字を入れたりするときに使います。

7.4.1 矩形を選ぶ

^x SPC で矩形の選択を始め、カーソルを動かして範囲を広げます。選んだ範囲は、四角く色が付きます。

^x SPC	rectangle-mark-mode	矩形の選択を始める
^w	kill-region	選んだ矩形をキルする
⌥w	kill-ring-save	選んだ矩形をキルリングにコピーする
^y	yank	キルした矩形を矩形のまま貼り付ける
^o	open-rectangle	矩形の範囲に空白を差し込む
^t	string-rectangle	矩形の各行を入力した文字列に置き換える

- ^o と ^t は、矩形を選んでいる間だけ働きます。
- 矩形の選択は ^g でやめます。

7.4.2 矩形のコマンド

普通のリージョンを選んでから使うと、そのリージョンを矩形として扱います。

^x r k	kill-rectangle	矩形をキルする
^x r ⌥w	copy-rectangle-as-kill	矩形をコピーする
^x r y	yank-rectangle	最後にキルした矩形を貼り付ける
^x r d	delete-rectangle	矩形を削除する（保存しない）
^x r c	clear-rectangle	矩形を空白で塗りつぶす
^x r o	open-rectangle	矩形の範囲に空白を差し込む
^x r t	string-rectangle	矩形の各行を入力した文字列に置き換える
^x r N	rectangle-number-lines	矩形の各行に連番を入れる

7.4.3 使用例

5 行の先頭に、まとめて // と空白 1 つを入れる手順です。

1. 1 行目の先頭で ^x SPC を押します。
2. ^n で 5 行目まで下がります。幅 0 の矩形になります。
3. ^t を押し、 // と空白を入力して RET を押します。

- 幅 0 の矩形に ^t を使うと、文字列を差し込みます。
- 列を消すときは、消す列を矩形で選んで ^w を押します。

7.5 文字の種類の変換

7.5.1 大文字と小文字

⌥u	seq-upcase-backward-word	カーソルの前の語を大文字にする
⌥l	seq-downcase-backward-word	カーソルの前の語を小文字にする
⌥c	seq-capitalize-backward-word	カーソルの前の語の頭を大文字にする
^x ^u	upcase-region	リージョンを大文字にする
^x ^l	downcase-region	リージョンを小文字にする
⌥x	capitalize-region	リージョンの各語の頭を大文字にする

- ⌥u 、 ⌥l 、 ⌥c は、init.el の sequential-command で置き換えています。
- 標準の upcase-word などは、カーソルの後ろの語を変えます。置き換えたコマンドは、入力し終えた直前の語を変えます。カーソルは動きません。
- 続けて押すと、1 つ前の語まで範囲を広げます。2 回で 2 語、3 回で 3 語です。

- ^x ^u と ^x ^l は、初期設定で使用禁止になっています。使うと確認が出ます。答えは次のとおりです。

キー	動作
SPC	今回だけ実行する
y	実行し、禁止を解く。続けて、init.el に書くかを尋ねる
!	実行し、このセッションの間は禁止のコマンドをすべて許す
n	実行しない

- init.el に書くと、(put 'upcase-region 'disabled nil) が init.el の末尾に足されます。

7.5.2 全角と半角

⌥x	japanese-hankaku-region	リージョンの全角文字を半角にする
⌥x	japanese-zenkaku-region	リージョンの半角文字を全角にする
⌥x	japanese-katakana-region	ひらがなをカタカナにする
⌥x	japanese-hiragana-region	カタカナをひらがなにする

- japanese-hankaku-region は、カタカナも半角カタカナに変えます。^u を付けると、英数字と記号だけを半角にし、カタカナは全角のまま残します。

7.6 行の並べ替えと整列

どれもリージョンの中の行に働きます。

⌥x	sort-lines	行を文字の順に並べ替える
⌥x	sort-numeric-fields	指定した欄の数で並べ替える
⌥x	reverse-region	行の順を逆にする
⌥x	delete-duplicate-lines	重複する行を 1 つだけ残して消す
⌥x	align-regexp	指定した文字の位置で桁をそろえる
⌥=	count-words-region	行数、語数、文字数を表示する

- ^u ⌥x sort-lines は、逆の順に並べ替えます。
- sort-numeric-fields の欄は、空白で区切った何番目かです。数引数で指定します。
- ⌥= は、マークとカーソルの間を数えます。 ^u ⌥= はバッファ全体を数えます。

7.6.1 align-regexp の例

= の位置をそろえるときは、リージョンを選んで ⌥x align-regexp RET = RET と入力します。

```
変更前                  変更後
width = 10              width  = 10
height = 20             height = 20
x = 0                   x      = 0
```

7.7 特殊な文字の入力

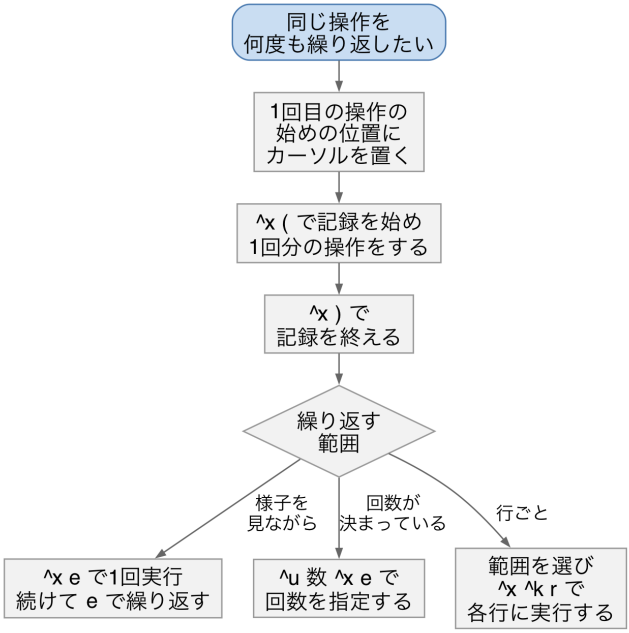
キーボードにない文字は、名前や一覧から選んで入力します。

^x 8 RET	insert-char	名前か文字コードで文字を入れる
^x 8 e e	emoji-insert	分類から絵文字を選んで入れる
^x 8 e s	emoji-search	名前で絵文字を探して入れる

- ^x 8 RET では、Unicode の英語の名前を補完して選びます。例えば RIGHTWARDS ARROW で → を入れます。
- 文字コードは 16 進数で入力します。例えば 2318 で ⌘ を入れます。
- タブ文字や制御文字は、第 2 章の ^q で入れます。

7.8 キーボードマクロ

キーボードマクロは、キー操作を記録して、何度でも再生する機能です。置換では書けない、行ごとの決まった編集に使います。



7.8.1 記録と実行

^x (	kmacro-start-macro	記録を始める
^x)	kmacro-end-macro	記録を終える
^x e	kmacro-end-and-call-macro	最後のマクロを実行する
F3	kmacro-start-macro-or-insert-counter	記録を始める
F4	kmacro-end-or-call-macro	記録を終える、マクロを実行する
^x ^k r	apply-macro-to-region-lines	リージョンの各行でマクロを実行する
^x ^k n	kmacro-name-last-macro	最後のマクロに名前を付ける
^x ^k b	kmacro-bind-to-key	最後のマクロをキーに割り当てる
^x ^k e	edit-kbd-macro	マクロのキー操作を編集する

- ^x e の後に続けて e を押すと、もう一度実行します。
- ^u 10 ^x e は 10 回実行します。 ^u 0 ^x e は、エラーで止まるまで繰り返します。
- 記録中に ^g を押すと、記録をやめます。
- Mac で F3 と F4 を押すときは、 fn と一緒に押します。

7.8.2 マクロを作るこつ

- 行の先頭で始め、次の行の先頭で終えます。続けて実行すると、1 行ずつ進みます。
- カーソルの移動には、語や行の単位で動くキー（⌥f 、 ^e など）を使います。矢印キーで文字数を数えて動くと、行の長さが違うときに失敗します。
- 探す位置へは ^s で移動します。見つからないとエラーになり、繰り返しが止まります。
- ^x ^k r で使うマクロは、次の行へ移動する操作を含めません。各行の先頭から 1 回ずつ実行するためです。

7.8.3 カウンター

記録中に F3 を押すと、カウンターの値を入れ、値を 1 増やします。連番を振るときに使います。

- カウンターは 0 から始まります。 ^u 1 F3 で記録を始めると、1 から始まります。
- ^x ^k ^c で、カウンターの値を設定し直せます。

7.9 チェックリスト

7.9.1 やるべきこと

- 並べ替えや置き換えの前に、範囲が正しいかを確かめる。間違えたら `^/` で戻す。
- キーボードマクロは、1 回実行して結果を確かめてから繰り返す。
- 1 段落を 1 行で書く文書では、`⌥q` の代わりに `visual-line-mode` を使う。

7.9.2 やってはいけないこと

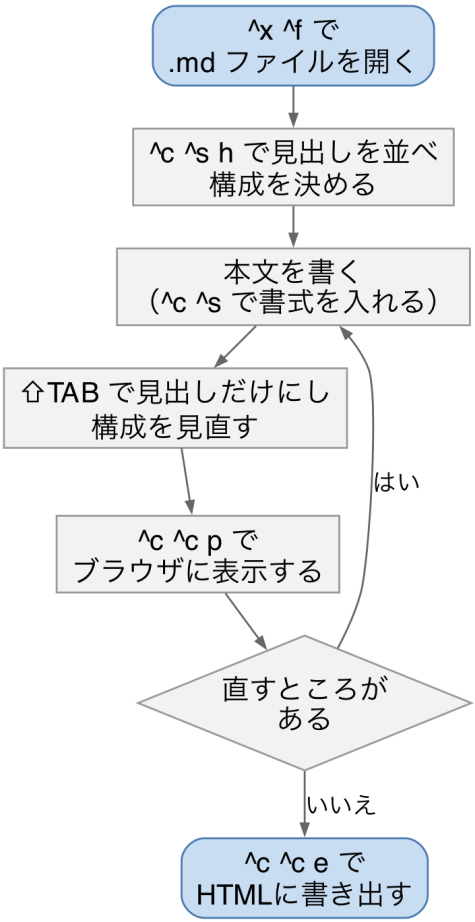
- プログラムのコードに `⌥q` を使う。コメント以外の行もつながることがあります。
- Makefile など、タブ文字が必要なファイルで `untabify` を使う。動かなくなります。
- `^u 0 ^x e` を、止まる条件のないマクロで使う。終わらなくなります。止めるには `^g` を押します。

第8章　テキストのマークアップ

Markdown の文書は、markdown-mode で書きます。見出しで折り畳むアウトラインの機能を使うと、長い文書の構成を見渡せます。

機能	できること
markdown-mode	Markdown の書式を色分けし、キーで書式を入れる
pandoc	Markdown を HTML に変換する。プレビューと書き出しに使う
アウトライン	見出しだけを表示し、見出しの単位で移動、入れ替えをする

Markdown の文書は、次の流れで書きます。



8.1 markdown-mode の導入

Emacs 31.1 には、Markdown 用のモードが組み込まれていません。そのままでは、`.md` のファイルは基本のモード（`fundamental-mode`）で開きます。

部品	入れ方	役割
markdown-mode	MELPA の `markdown-mode`	Markdown の編集
pandoc	`brew install pandoc`	HTML への変換

init.el では次のように設定しています。

```
(leaf markdown-mode
  :ensure t
  :custom
  (markdown-command . '("pandoc" "-f" "gfm")))
```

- `markdown-command` は、HTML に変換するコマンドです。
- `-f gfm` は、GitHub と同じ書き方（GitHub Flavored Markdown）として読む指定です。表やチェックボックスも変換します。
- `.md` や `.markdown` のファイルを開くと、自動で markdown-mode になります。

8.2 書式を入れる

`^c ^s` で始まるキーで、書式の記号を入れます。リージョンを選んでいるときは、リージョンを記号で囲みます。

^c ^s h	markdown-insert-header-dwim	見出しを入れる
^c ^s 1	markdown-insert-header-atx-1	レベル 1 の見出し（ # ）を入れる
^c ^s 2	markdown-insert-header-atx-2	レベル 2 の見出し（ ## ）を入れる
^c ^s b	markdown-insert-bold	太字にする
^c ^s i	markdown-insert-italic	斜体にする
^c ^s c	markdown-insert-code	行内のコードにする
^c ^s C	markdown-insert-gfm-code-block	コードブロックを入れる
^c ^s q	markdown-insert-blockquote	引用にする
^c ^s l	markdown-insert-link	リンクを入れる
^c ^s t	markdown-insert-table	表を入れる
^c ^s [	markdown-insert-gfm-checkbox	チェックボックスを入れる
^c -	markdown-insert-hr	区切り線を入れる

- ^c ^s h は、前の見出しと同じレベルの見出しを入れます。
- ^c ^s 3 から ^c ^s 6 で、レベル 3 から 6 の見出しを入れます。
- 太字の語の上で ^c ^s b を押すと、太字を外します。斜体なども同じです。
- ^c ^s C は、コードの言語名を尋ねます。 swift などと入力します。

8.2.1 画面の表示

^c ^x RET	markdown-toggle-markup-hiding	書式の記号を隠す/出す
^c ^x ^l	markdown-toggle-url-hiding	リンクの URL を隠す/出す
^c ^x TAB	markdown-toggle-inline-images	画像をバッファの中に表示する/やめる
^c ^x ^f	markdown-toggle-fontify-code-blocks-natively	コードブロックを言語に合わせて色分けする/やめる

8.3 リスト

⌥RET	markdown-insert-list-item	次のリストの項目を入れる
^c ^d	markdown-do	チェックボックスの印を入/切する
^c ↑	markdown-move-up	項目を 1 つ上へ動かす
^c ↓	markdown-move-down	項目を 1 つ下へ動かす
^c →	markdown-demote	項目を 1 段下げる
^c ←	markdown-promote	項目を 1 段上げる
^c ^c n	markdown-cleanup-list-numbers	番号付きリストの番号を振り直す

- ⌥RET は、前の項目に合わせて - や次の番号を入れます。
- ^c ^d は、カーソルの位置に合わせて動作を変えます。リンクの上ではリンクを開き、表の中では表をそろえます。

8.4 表

^c ^s t で表を入れ、 TAB でセルを移動しながら書きます。TAB を押すたびに、列の幅をそろえ直します。

入力した表です。

```
| name | value |
|---|---|
| width | 10 |
```

TAB でそろえた後です。

```
| name  | value |
|-------|-------|
| width | 10    |
```

TAB	markdown-cycle	列をそろえ、次のセルへ移動する
⇧TAB	markdown-shifttab	前のセルへ移動する
RET	markdown-enter-key	次の行の同じ列へ移動する
^c ⇧↓	markdown-table-insert-row	行を足す
^c ⇧↑	markdown-table-delete-row	行を消す
^c ⇧→	markdown-table-insert-column	列を足す
^c ⇧←	markdown-table-delete-column	列を消す
^c ^c ^	markdown-table-sort-lines	今の列で行を並べ替える
^c ^c |	markdown-table-convert-region	リージョンの文字列を表にする

- 表の中では、 ^c ↑ と ^c ↓ で行を、 ^c ← と ^c → で列を動かします。
- ^c ^c | は、タブ文字、カンマ、空白の順に区切りを判断して表にします。

8.5 リンク

^c ^l	markdown-insert-link	リンクを入れる、直す
^c ^o	markdown-follow-thing-at-point	リンク先を開く
⌥n	markdown-next-link	次のリンクへ移動する
⌥p	markdown-previous-link	前のリンクへ移動する

- ^c ^l は、URL、リンクの文字列、説明（省略可）を順に尋ねます。
- ^c ^o は、URL を macOS の既定のブラウザで開きます。

8.6 プレビューと書き出し

どれも markdown-command （pandoc）で HTML に変換します。

^c ^c p	markdown-preview	ブラウザで見た目を確かめる
^c ^c e	markdown-export	HTML のファイルに書き出す
^c ^c v	markdown-export-and-preview	書き出して、ブラウザで開く
^c ^c l	markdown-live-preview-mode	保存のたびに、隣のウィンドウで見た目を更新する
^c ^c m	markdown-other-window	変換した HTML を隣のウィンドウに表示する

- ^c ^c p は、一時ファイルを macOS の既定のブラウザで開きます。
- ^c ^c e は、同じディレクトリに拡張子を .html に変えた名前で書き出します。
- ^c ^c l は、Emacs 組み込みのブラウザ EWW で表示します。CSS による装飾は出ません。文書の構成の確認に使います。

8.7 アウトライン

アウトラインは、見出しだけを表示して、本文を折り畳む機能です。長い文書の構成を見直すとき、見出しの単位で並べ替えるときに使います。

8.7.1 Markdown のアウトライン

markdown-mode では、 # で始まる行を見出しとして扱います。

TAB	markdown-cycle	見出しの上で、その見出しを折り畳む/開く
⇧TAB	markdown-shifttab	文書全体の表示を切り替える
^c ^n	markdown-outline-next	次の見出しへ移動する
^c ^p	markdown-outline-previous	前の見出しへ移動する
^c ^f	markdown-outline-next-same-level	同じレベルの次の見出しへ移動する
^c ^b	markdown-outline-previous-same-level	同じレベルの前の見出しへ移動する
^c ^u	markdown-outline-up	1 つ上のレベルの見出しへ移動する
^c ↑	markdown-move-up	見出しを中身ごと上へ動かす
^c ↓	markdown-move-down	見出しを中身ごと下へ動かす
^c ←	markdown-promote	見出しのレベルを 1 つ上げる
^c →	markdown-demote	見出しのレベルを 1 つ下げる
^x n s	markdown-narrow-to-subtree	今の見出しの中だけを表示する

- ⇧TAB は、押すたびに「すべての見出し」「レベル 1 の見出しだけ」「全文」の順に切り替えます。
- ^x n s で絞った表示は、 ^x n w で元に戻します。
- 見出しの外や表の外で TAB を押すと、普通のインデントになります。

8.7.2 ほかのファイルのアウトライン

Markdown 以外のファイルでは、 outline-minor-mode を使います。⌥x outline-minor-mode で入/切します。キーは ^c @ で始まります。

^c @ ^t	outline-hide-body	本文を隠し、見出しだけを表示する
^c @ ^a	outline-show-all	すべてを表示する
^c @ ^d	outline-hide-subtree	今の見出しの中を隠す
^c @ ^s	outline-show-subtree	今の見出しの中を表示する
^c @ ^n	outline-next-visible-heading	次の見出しへ移動する
^c @ ^p	outline-previous-visible-heading	前の見出しへ移動する
^c @ ^u	outline-up-heading	1 つ上のレベルの見出しへ移動する

- 何を見出しとするかは、メジャーモードが決めます。
- Emacs Lisp では、 ;;; で始まるコメントの行と、行頭の (で始まる式が見出しです。init.el で ^c @ ^t を使うと、設定の一覧を見渡せます。

8.8 チェックリスト

8.8.1 やるべきこと

- 書き始めに見出しだけを並べ、 ⇧TAB で構成を確かめる。
- 表は TAB でそろえてから保存する。
- 公開する前に ^c ^c p で見た目を確かめる。

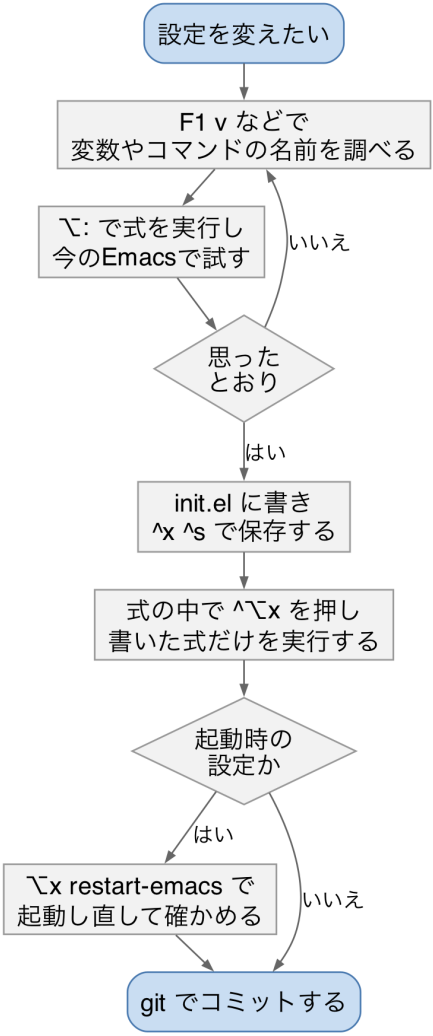
8.8.2 やってはいけないこと

- 1 段落を 1 行で書く文書に ⌥q を使う。段落の途中に改行が入ります。
- 見出しを折り畳んだまま、範囲を選んで消す。隠れた本文も一緒に消えます。

第9章　Emacsのカスタマイズ

Emacs の設定は、 `~/.emacs.d/` の設定ファイルに Emacs Lisp で書きます。この章では、設定ファイルの役割、Emacs Lisp の基本、パッケージの導入、キーの割り当てを説明します。

設定は、次の手順で変えます。


設定を変えたい
F1 v などで
変数やコマンドの名前を調べる
⌥: で式を実行し
今のEmacsで試す
いいえ
思った
とおり
はい
init.el に書き
^x ^s で保存する
式の中で ^⌥x を押し
書いた式だけを実行する
起動時の
設定か
はい
⌥x restart-emacs で
起動し直して確かめる
いいえ
git でコミットする

9.1 設定ファイル

ファイル	役割	書く人
`early-init.el`	画面を作る前に読み込む設定	自分
`init.el`	主な設定。パッケージの導入もここに書く	自分
`custom.el`	カスタマイズの画面で保存した設定	Emacs
`lisp/`	手で置いた Emacs Lisp のファイル（navi2ch など）	自分

- どれも `~/.emacs.d/` に置きます。このディレクトリは git で管理しています。
- `elpa/`（パッケージ）、`eln-cache/`（コンパイル結果）、`var/`（バックアップ）は管理から外しています。`.gitignore` に書いてあります。

9.1.1 読み込む順序

順	読み込むもの	書く内容の例
1	`early-init.el`	起動時の警告の抑止、ネイティブコンパイルの設定
2	入れたパッケージ	（自動で有効になる）
3	`init.el`	キー、表示、パッケージごとの設定
4	`custom.el`	init.el の途中の `(load custom-file)` で読み込む

- `early-init.el` は、フレーム（画面）を作る前に読み込みます。画面の部品やフォントの設定は `init.el` に書きます。
- `custom.el` は、`⌥x customize` の画面で保存したときに Emacs が書き換えます。手で書き換えないようにします。

9.1.2 設定を試す

書いた式は、Emacs を起動し直さなくても実行できます。

⌥:	eval-expression	ミニバッファで式を入力して実行する
^x ^e	eval-last-sexp	カーソルの直前の式を実行する
^⌥x	eval-defun	カーソルのある一番外側の式を実行する
⌥x	eval-buffer	バッファ全体を実行する
⌥x	load-file	ファイルを読み込んで実行する
⌥x	restart-emacs	Emacs を終えて起動し直す

- 実行の結果は、エコーエリアに表示します。
- init.el を書き換えたときは、^⌥x で書いた式だけを実行します。⌥x eval-buffer で init.el 全体を実行すると、フックなどが二重に登録されることがあります。
- 起動時に一度だけ働く設定は、 ⌥x restart-emacs で確かめます。

9.2 Emacs Lisp の基本

設定に必要な最小限の構文です。

構文	書式	意味
関数の呼び出し	(関数名 引数 1 引数 2 ...)	関数を実行する
代入	(setq 変数名 値)	変数に値を入れる
既定値の代入	(setq-default 変数名 値)	バッファごとの変数の既定値を変える
リストへの追加	(add-to-list 'リスト 要素)	リストの先頭に要素を足す
フック	(add-hook 'フック 関数)	あるときに実行する関数を登録する
条件分岐	(if 条件 真の場合 偽の場合)	条件で処理を分ける
条件分岐	(when 条件 真の場合)	条件が真のときだけ実行する

- nil は偽です。 nil 以外はすべて真です。真を表すときは t と書きます。
- '（クォート）を付けた名前は、値ではなく名前そのものを表します。
- ; から行末まではコメントです。

```
;;; 見出しはセミコロン 3 つ
;; 独立した行のコメントはセミコロン 2 つ
(blink-cursor-mode t) ; 行末のコメントはセミコロン 1 つ
```

9.2.1 setq と setq-default

tab-width のように、バッファごとに値を持つ変数があります。このような変数は、setq では今のバッファだけが変わります。

```
;; すべてのバッファの既定値を 4 にする
(setq-default tab-width 4)
;; インデントに空白を使う
(setq-default indent-tabs-mode nil)
```

- バッファごとの変数かどうかは、 F1 v （ describe-variable ）の説明に書いてあります。
- init.el の tab-width と indent-tabs-mode は、 setq-default で設定しています。

9.2.2 変数とコマンドを調べる

init.el で ^h を DEL に変えているので、ヘルプは F1 で始まるキーで呼び出します。

F1 v	describe-variable	変数の値と説明を表示する
F1 f	describe-function	関数の説明を表示する
F1 k	describe-key	押したキーに割り当てたコマンドを表示する
⌥x	customize-variable	変数をカスタマイズの画面で変える

- 説明の画面から、その変数や関数を定義したファイルへ移動できます。
- ヘルプの使い方の詳しいことは、第 13 章で説明します。

9.3 パッケージの導入

パッケージは、Emacs に機能を足す Emacs Lisp のまとまりです。Emacs 組み込みの package.el で、インターネット上の置き場所（アーカイブ）から入れます。

9.3.1 アーカイブ

init.el では、次のアーカイブを使っています。

名前	URL	内容
gnu	`https://elpa.gnu.org/packages/`	GNU の公式のパッケージ
nongnu	`https://elpa.nongnu.org/nongnu/`	GNU 以外の公認のパッケージ
melpa	`https://melpa.org/packages/`	有志のパッケージ（最も多い）
org	`https://orgmode.org/elpa/`	org-mode

- gnu と nongnu は、Emacs の初期設定で使います。init.el では melpa、org、gnu を足しています。
- vterm や markdown-mode など、この資料のパッケージの多くは melpa から入れています。

9.3.2 パッケージの一覧

⌥x `list-packages` で、入れられるパッケージの一覧を開きます。

`/ n`	`package-menu-filter-by-name`	名前で絞り込む
`?`	`package-menu-describe-package`	パッケージの説明を表示する
`i`	`package-menu-mark-install`	入れる印を付ける
`d`	`package-menu-mark-delete`	消す印を付ける
`U`	`package-menu-mark-upgrades`	新しい版のあるものに更新の印を付ける
`u`	`package-menu-mark-unmark`	印を消す
`x`	`package-menu-execute`	印の付いた操作を実行する
`r`	`revert-buffer`	アーカイブから一覧を取り直す

一覧を開かずに使うコマンドです。

⌥x	package-install	名前を指定して入れる
⌥x	package-upgrade-all	入れたパッケージをすべて更新する
⌥x	package-autoremove	使われなくなったパッケージを消す
⌥x	package-vc-install	git のリポジトリから直接入れる

- 入れたパッケージは ~/.emacs.d/elpa/ に置きます。
- 入れたパッケージの名前は、 custom.el の package-selected-packages に記録します。package-autoremove は、ここに無いパッケージを消す対象にします。

9.3.3 leaf と use-package

パッケージの導入と設定は、1 つの式にまとめて書きます。init.el では、主に leaf を使い、一部で use-package を使っています。

項目	leaf	use-package
入手先	MELPA のパッケージ	Emacs 29 から組み込み
init.el での例	migemo、ido、navi2ch など	vterm、claude-code-ide

leaf の書き方の例です。

```
(leaf markdown-mode
  ;; 無ければ入れる
  :ensure t
  ;; 変数を設定する
  :custom (markdown-command . '("pandoc" "-f" "gfm"))
  ;; キーを割り当てる
  :bind (("C-c m" . markdown-mode))
```

(次のページに続く)

(前のページからの続き)

```
;; 読み込んだ後に実行する
:config (setq markdown-hide-urls t))
```

主なキーワードの意味です。leaf と use-package で共通です。

キーワード	意味
:ensure t	パッケージが入っていなければ、アーカイブから入れる
:require t	起動時にすぐ読み込む（leaf のみ）
:custom	変数を設定する。(変数 . 値) の形で書く
:bind	キーにコマンドを割り当てる
:init	パッケージを読み込む前に実行する
:config	パッケージを読み込んだ後に実行する
:commands	指定したコマンドを初めて使うときに読み込む
:vc	git のリポジトリから入れる（use-package）

- :config の中身は、パッケージを読み込むまで実行しません。:require t が無く、コマンドもまだ使っていないときは、設定がまだ効いていません。
- use-package の :custom は (変数 値) の形で書きます。leaf とは書き方が違います。

9.4 キー割り当ての変更

9.4.1 キーの書き方

設定ファイルでは、キーを Emacs の表記で書きます。この資料の表記との対応です。

Emacs の表記	この資料	Mac のキー
C-	^	control
M-	⌥	option
s-	⌘	command（Emacs では super）
S-	⇧	shift
RET TAB SPC	RET TAB SPC	return、tab、space

- 例えば ^⌘? は "C-s-?" 、 ^x ^f は "C-x C-f" と書きます。
- command を super、option を meta として扱うのは、NS 版の Emacs の初期設定です。

9.4.2 割り当ての書き方

書き方	意味
(keymap-global-set "C-c l" #'cmd)	どのモードでも使うキーを割り当てる
(keymap-set モードのマップ "C-c l" #'cmd)	特定のモードだけのキーを割り当てる
(keymap-global-unset "C-c l")	割り当てを外す
(global-set-key (kbd "C-c l") 'cmd)	古い書き方。init.el の多くはこの形

- keymap-global-set は Emacs 29 からの書き方です。キーの書き方の誤りを、その場で知らせます。
- モードのマップは、パッケージを読み込んだ後でないとありません。leaf や use-package の :bind を使うと、読み込みを待って割り当てます。

9.4.3 init.el の割り当て

init.el で変えている主なキーです。

キー	コマンド	書き方
^h	DEL と同じ	key-translation-map でキーを置き換える
⌥h	backward-kill-word	global-set-key
^⌘?	help-for-help	global-set-key
RET	newline-and-indent	global-set-key （"C-m"）
^c l	toggle-truncate-lines	define-key
^x o	ace-window	leaf の :bind
^a ^e ⌥u など	seq-home など	sequential-command の設定関数

- key-translation-map は、キーを別のキーとして読み替えます。^h を押すと、どのモードでも DEL を押したのと同じになります。
- 自分で割り当てるキーは、^c に続けて英字 1 文字にします。このキーはユーザーのために空けてある決まりです。

9.5 見た目の設定

9.5.1 テーマ

テーマは、文字や背景の色の組み合わせです。 load-theme で読み込みます。

⌥x	customize-themes	テーマを一覧から選ぶ
⌥x	load-theme	テーマを読み込む
⌥x	disable-theme	テーマを外す

- init.el では、 rotor を使っています。color-theme-modern パッケージのテーマです。

- リモートのファイル（TRAMP で開いたもの）を表示している間は、organic-green に切り替えます。手元のファイルと見分けるためです。
- Emacs 組み込みのテーマには、modus-operandi（明るい）と modus-vivendi（暗い）があります。

9.5.2 文字の大きさ

init.el では、次の式で文字の大きさを 18 ポイントにしています。

```
(set-face-attribute 'default nil :height 180)
```

- :height の単位は 0.1 ポイントです。180 で 18 ポイントです。

一時的に大きさを変えるときは、次のキーを使います。

⌘+	text-scale-adjust	今のバッファの文字を大きくする
⌘-	text-scale-adjust	今のバッファの文字を小さくする
⌘0	text-scale-adjust	今のバッファの文字を元の大きさに戻す
⌥x	global-text-scale-adjust	すべてのバッファの文字の大きさを変える

- ⌘+ は ⌘= でも同じです。⇧を押さずに使えます。
- 続けて +、 -、 0 を押すと、そのまま調整を続けられます。

9.6 起動のトラブル

init.el に誤りがあると、起動の途中で止まり、設定の一部が効きません。

方法	使いどころ
`Emacs --debug-init`	init.el のどこで止まったかを表示する
`Emacs -Q`	設定ファイルを読まずに起動する。設定と Emacs 本体のどちらの問題かを分ける
`⌥x emacs-init-time`	起動にかかった時間を表示する
`⌥x toggle-debug-on-error`	エラーが起きたときに、呼び出しの経過を表示する

- `Emacs` は、ターミナルから `/Applications/Emacs.app/Contents/MacOS/Emacs` として起動します。
- `--debug-init` で出る画面は、 `q` で閉じます。

9.7 チェックリスト

9.7.1 やるべきこと

- 書く前に `⌥:` で試し、思ったとおりに動くかを確かめる。
- 設定にはコメントで理由を書く。後で読んだときに、消してよいかを判断できます。
- 設定を変えたら、git でコミットする。起動しなくなったときに戻せます。
- 自分のキーは `^c` と英字 1 文字に割り当てる。

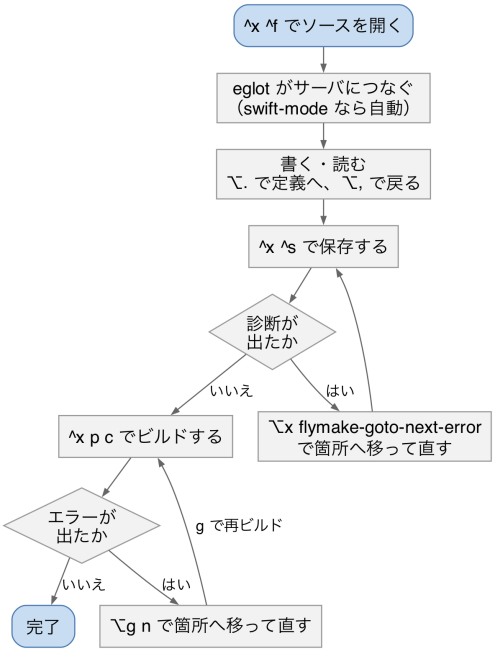
9.7.2 やってはいけないこと

- `custom.el` を手で書き換える。Emacs が保存するときに上書きします。
- 同じ変数を、init.el と `⌥x customize` の両方で設定する。どちらが効くか分かりにくくなります。
- 動いている Emacs で `⌥x eval-buffer` を使って init.el 全体を実行する。フックが二重に登録されます。

第10章　プログラミングコマンド

ソースコードは、言語ごとのメジャーモードで編集します。定義への移動や診断はeglot、ビルドはcompileで行います。eglotは、言語サーバ（LSP）につなぐEmacs標準のパッケージです。

コードを書いて直す流れは次のとおりです。



10.1 プログラミング用のモード

ファイルを開くと、Emacs は拡張子でメジャーモードを選びます。プログラミング用のモードは、どれも prog-mode を元にしています。

10.1.1 ファイルとモード

ファイル	モード	補足
.swift	なし	Emacs に含まれない。swift-mode を入れる
.m	objc-mode	先頭が function なら Octave のモードになる
.h	c-mode	Objective-C の書き方でも c-mode になる
.c	c-mode	
.py	python-mode	
.sh	sh-mode	
Makefile	makefile-bsdmake-mode	macOS の make(BSD)に合わせたモード
.json	js-json-mode	

- モードを変えるには、 ⌥x objc-mode のようにモードの名前を実行します。
- C、Objective-C のインデントは、init.el で bsd の書き方にしています。
- インデントのキーは、第 7 章の「インデント」で説明しています。

10.1.2 Swift を使う設定

Swift には、MELPA の swift-mode を使います。init.el には次の設定を書いています。.swift を開くと swift-mode になり、eglot が自動でつなぎます。

```
(leaf swift-mode
  :ensure t
  :hook (swift-mode-hook . eglot-ensure))

;; eglot に Swift の言語サーバを教える
(leaf eglot
  :defer-config
  (add-to-list 'eglot-server-programs
               '(swift-mode . ("xcrun" "sourcekit-lsp"))))
```

- 言語サーバは、Xcode に入っている `sourcekit-lsp` を `xcrun` で呼び出します。
- eglot の標準の設定には、Swift の言語サーバが入っていません。
- `:defer-config` は、eglot を読み込んだ後に式を実行します。`:config` にすると、eglot の標準の設定が消えます。

10.1.3 共通のキー

⌥;	comment-dwim	リージョンをコメントにする。行末にコメントを入れる
^x ^;	comment-line	今の行をコメントにする。もう一度押すと戻す
^⌥f	forward-sexp	括弧の組や式の 1 つ先へ移動する
^⌥b	backward-sexp	括弧の組や式の 1 つ前へ移動する
^⌥d	down-list	次の括弧の内側へ入る
^⌥k	kill-sexp	括弧の組や式を 1 つ削除する
^⌥a	beginning-of-defun	関数の先頭へ移動する
^⌥e	end-of-defun	関数の末尾へ移動する
^⌥h	mark-defun	関数の全体を選ぶ
⌥g i	imenu	関数や型の名前を選んで移動する

- sexp は、括弧で囲んだ組や、1 つの語や文字列などの式のことです。
- ⌥; は、リージョンがすべてコメントなら、コメントを外します。

10.2 コードの移動と参照

定義への移動と参照の検索は、xref で行います。xref は、検索の手段（バックエンド）を切り替えて使います。eglot がつないでいるバッファでは、言語サーバがバックエンドになります。

10.2.1 xref

⌥.	xref-find-definitions	カーソルの名前の定義へ移動する
⌥,	xref-go-back	移動する前の位置へ戻る
^⌥,	xref-go-forward	戻る前の位置へ進む
⌥?	xref-find-references	カーソルの名前を使っている箇所を一覧にする
^⌥.	xref-find-apropos	名前の一部で定義を探す

- 候補が複数あると、 *xref* バッファに一覧を出します。一覧のキーは、第 3 章の「プロジェクトの検索の結果」と同じです。
- ⌥. を続けて使っても、 ⌥, で順にさかのぼれます。

10.2.2 eglot

eglot は、言語サーバと通信して、定義、補完、診断、名前の変更を行います。言語サーバは、言語ごとにコードを解析する別のプログラムです。

言語	言語サーバ	設定
Swift	sourcekit-lsp	init.el で設定済み。ファイルを開くと自動でつなぐ
C、Objective-C	clangd	不要。 ⌥x eglot でつなぐ

- eglot は、プロジェクトの単位でサーバを起動します。プロジェクトは、git のリポジトリの全体です。
- サーバは、つないだ後にプロジェクトのファイルを解析します。大きなプロジェクトでは、定義へ移れるまでに時間がかかります。

eglot の主なコマンドには、キーが割り当てられていません。 ⌥x で呼び出します。

⌥x	eglot	今のバッファを言語サーバにつなぐ
⌥x	eglot-rename	カーソルの名前を、プロジェクト全体で変える
⌥x	eglot-code-actions	サーバが提案する修正を選んで適用する
⌥x	eglot-format-buffer	バッファ全体を整形する
⌥x	eglot-find-implementation	プロトコルなどの実装へ移動する
⌥x	eldoc-doc-buffer	カーソルの名前の説明を別のバッファに出す
TAB	indent-for-tab-command	インデント済みの行では、サーバの候補で補完する
⌥x	eglot-reconnect	サーバを起動し直す
⌥x	eglot-shutdown	サーバを止める

- カーソルの名前の説明は、エコーエリアにも短く出ます。
- 補完は、TAB で行います。1 回目でインデントし、行がインデント済みなら補完します。設定は第 7 章の「インデント」にあります。
- C、Objective-C のモードは、もとは TAB に独自のコマンドを割り当てています。init.el で indent-for-tab-command に割り当て直しています。
- ^⌥i でも補完できますが、macOS の機能と衝突するので使いません。
- C、Objective-C、Swift のバッファでは、入力の途中で auto-complete の候補も出ます。候補が出ている間の TAB は、auto-complete の候補を展開します。auto-complete は、サーバではなく、バッファの中の語から候補を出します。

補完の候補が複数あると、*Completions* バッファに一覧を出します。一覧は、次のキーで選びます。

⌥↓	minibuffer-next-completion	次の候補を選ぶ
⌥↑	minibuffer-previous-completion	前の候補を選ぶ
⌥RET	minibuffer-choose-completion	選んだ候補を入れる

10.2.3 診断

eglot は、サーバが見つけたエラーと警告を、flymake で表示します。該当の箇所には下線が付きます。

⌥x	flymake-goto-next-error	次の診断の箇所へ移動する
⌥x	flymake-goto-prev-error	前の診断の箇所へ移動する
⌥x	flymake-show-buffer-diagnostics	バッファの診断を一覧にする

- sourcekit-lsp は、ファイルを保存したときに診断を更新します。
- 診断の数は、モードラインに出ます。

10.3 コンパイル

ビルドは、Emacs の中で実行します。出力は *compilation* バッファに出て、エラーの行からソースへ移れます。

^x p c	project-compile	プロジェクトの根でビルドのコマンドを実行する
⌥x	compile	今のディレクトリでビルドのコマンドを実行する
⌥x	recompile	前と同じコマンドを実行する
⌥g n	next-error	次のエラーの箇所へ移動する
⌥g p	previous-error	前のエラーの箇所へ移動する

- コマンドは、ミニバッファで尋ねます。最初は make -k が入っています。

- Swift のパッケージでは `swift build`、Xcode のプロジェクトでは `xcodebuild` に書き換えます。
- 入力したコマンドは、次から最初に入ります。`⌥p` で前のコマンドを選べます。

`*compilation*` バッファの移動のキー（`RET`、`^o`、`n`、`p`）は、第 3 章の「grep の結果」と同じです。そのほかのキーは次のとおりです。

`g`	`recompile`	もう一度ビルドする
`^c ^k`	`kill-compilation`	実行中のビルドを止める
`⌥n`	`compilation-next-error`	次のエラーの行へ移動する（ソースは開かない）
`⌥p`	`compilation-previous-error`	前のエラーの行へ移動する（ソースは開かない）
`⌥}`	`compilation-next-file`	次のファイルのエラーへ移動する
`⌥{`	`compilation-previous-file`	前のファイルのエラーへ移動する

10.4 チェックリスト

10.4.1 やるべきこと

- プロジェクトを git のリポジトリにしてから、eglot を使う。
- 定義へ移ったら、`⌥,` で元の位置へ戻る。
- 名前を変えるときは `eglot-rename` を使う。文字列の置換では、別の名前まで変わります。
- ビルドは `^x p c` で、プロジェクトの根から実行する。

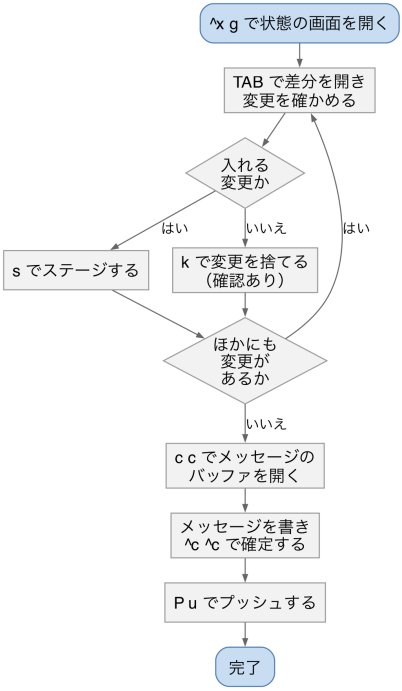
10.4.2 やってはいけないこと

- eglot の設定を `:config` に書く。eglot の標準のサーバの設定が消えます。
- 保存しないまま診断を待つ。sourcekit-lsp は、保存するまで診断を更新しません。

第11章　gitとdiffとgit-bug

git の操作は、Magit で行います。Magit は、git の状態を 1 つの画面に出し、1 文字のキーで操作するパッケージです。2 つのファイルの比較は diff と ediff、課題の管理は git-bug で行います。

変更をコミットする流れは次のとおりです。



11.1 Magit

11.1.1 設定

Magit は MELPA から入れます。init.el には次の設定を書いています。

```
(leaf magit
  :ensure t)
```

入れるだけで、次のキーが使えるようになります。

^x g	magit-status	今のリポジトリの状態の画面を開く
^x ⌥g	magit-dispatch	Magit のコマンドの一覧を出す
^c ⌥g	magit-file-dispatch	今のファイルに対するコマンドの一覧を出す

- Magit は、macOS に入っている /usr/bin/git を使います。
- Magit の画面では、⌥TAB は差分の開閉になります。init.el で割り当てた other-window は、Magit の画面では使えません。

11.1.2 状態の画面

^x g で開く画面には、次の部分（セクション）が並びます。

セクション	内容
Head	今のブランチと、最後のコミット
Untracked files	git で管理していないファイル
Unstaged changes	変更したが、ステージしていないファイル
Staged changes	次のコミットに入れるファイル
Recent commits	最近のコミット

ステージは、次のコミットに入れる変更を選ぶことです。カーソルの位置のファイルや差分のまとまり（ハンク）が、操作の対象になります。

TAB	magit-section-toggle	セクションを開閉する。ファイルでは差分を出す
n	magit-section-forward	次のセクションへ移動する
p	magit-section-backward	前のセクションへ移動する
s	magit-stage	ファイルやハンクをステージする
u	magit-unstage	ステージを取り消す
S	magit-stage-modified	変更したファイルをすべてステージする
U	magit-unstage-all	ステージをすべて取り消す
k	magit-discard	変更を捨てる。確認してから捨てる
RET	magit-diff-visit-file	ファイルを開き、カーソルの行へ移動する
g	magit-refresh	画面を更新する
$	magit-process-buffer	git の実行の記録とエラーを出す
q	magit-mode-bury-buffer	画面を閉じる

- リージョンで差分の行を選んで s を押すと、その行だけをステージします。
- 操作の前に、リポジトリのファイルの未保存の変更を保存するか尋ねます。

11.1.3 メニュー

コミットやプッシュは、まず 1 文字でメニューを開き、次の 1 文字でコマンドを選びます。メニューには、選べるキーと引数（ - で始まるオプション）が出ます。

c c	magit-commit-create	コミットする
c a	magit-commit-amend	直前のコミットを修正する
P u	magit-push-current-to-upstream	上流のブランチへプッシュする
F u	magit-pull-from-upstream	上流のブランチからプルする
f a	magit-fetch-all	すべてのリモートからフェッチする
b b	magit-checkout	ブランチを切り替える
b c	magit-branch-and-checkout	ブランチを作って切り替える
l l	magit-log-current	今のブランチの履歴を出す
d d	magit-diff-dwim	カーソルの位置に合わせた差分を出す
z z	magit-stash-both	変更を一時的に退避する（stash）
m m	magit-merge-plain	ブランチをマージする
?	magit-dispatch	メニューの一覧を出す

- メニューは ^g で閉じます。
- 上流は、ブランチを最初にプッシュしたときに git が記録する、対応するリモートのブランチです。まだ無ければ、 P u が尋ねます。

11.1.4 コミットのメッセージ

c c を押すと、メッセージを書くバッファと、ステージした差分が開きます。

^c ^c	with-editor-finish	メッセージを確定してコミットする
^c ^k	with-editor-cancel	コミットをやめる
⌥p	git-commit-prev-message	前に書いたメッセージを入れる
⌥n	git-commit-next-message	次に書いたメッセージを入れる

- 1 行目に要約を書き、1 行空けて本文を書きます。

11.1.5 ファイルの履歴

ファイルを開いたバッファで ^c ⌥g を押し、次の 1 文字を選びます。

^c ⌥g s	magit-file-stage	今のファイルをステージする
^c ⌥g d	magit-diff-buffer-file	今のファイルの差分を出す
^c ⌥g l	magit-log-buffer-file	今のファイルの履歴を出す
^c ⌥g B	magit-blame	各行を最後に変えたコミットを出す
^c ⌥g p	magit-blob-previous	1 つ前のコミットのファイルを出す

11.2 diff と ediff

diff は差分を 1 つのバッファに出し、ediff は 2 つのファイルを並べて出します。git で管理していないファイルどうしの比較にも使えます。

11.2.1 diff

⌥x	diff	2 つのファイルの差分を出す
⌥x	diff-buffer-with-file	バッファと、保存したファイルの差分を出す

差分は *Diff* バッファに出ます。このバッファでは、次のキーを使います。

n	diff-hunk-next	次のハンクへ移動する
p	diff-hunk-prev	前のハンクへ移動する
}	diff-file-next	次のファイルへ移動する
{	diff-file-prev	前のファイルへ移動する
RET	diff-goto-source	ハンクの元のファイルの行へ移動する
^c ^a	diff-apply-hunk	ハンクをファイルに当てる
^c ^b	diff-refine-hunk	ハンクの中の変わった語を強調する

11.2.2 ediff

⌥x	ediff-files	2 つのファイルを並べて比べる
⌥x	ediff-buffers	2 つのバッファを並べて比べる
⌥x	ediff-current-file	バッファと、保存したファイルを並べて比べる
e	magit-ediff-dwim	Magit の画面で、カーソルの位置の変更を並べて比べる

ediff を始めると、小さな操作用のウィンドウ（ *Ediff Control Panel* ）が開きます。キーは、このウィンドウで押します。

n	ediff-next-difference	次の違いへ移動する
p	ediff-previous-difference	前の違いへ移動する
a	ediff-copy-A-to-B	左（A）の内容を右（B）へ写す
b	ediff-copy-B-to-A	右（B）の内容を左（A）へ写す
|	ediff-toggle-split	左右と上下の並べ方を切り替える
?	ediff-toggle-help	キーの説明を出す
q	ediff-quit	ediff を終える

GUI の ediff は、初期設定では操作用のウィンドウを別のフレームに出します。init.el では、1 つのフレームに収め、左右に並べています。

```
(leaf ediff
  :custom
  (ediff-window-setup-function . 'ediff-setup-windows-plain)
  (ediff-split-window-function . 'split-window-horizontally))
```

11.3 git-bug

git-bug は、課題（バグや作業）を git のリポジトリの中に記録するツールです。課題は、ソースのファイルとは別の git のオブジェクトに入ります。Emacs からは、MELPA の git-bug パッケージで操作します。

11.3.1 準備

git-bug のコマンドは Homebrew で入れます。課題を書く人（利用者）は、リポジトリごとに作ります。

```
brew install git-bug
# リポジトリの中で実行する
git bug user new
```

- `git bug user new` は、名前とメールアドレスを尋ねます。
- 課題の一覧の作成には `jq` を使います。macOS の `/usr/bin/jq` で動きます。

11.3.2 キー

^c b m	git-bug-menu	課題を選び、次に操作を選ぶ
^c b c	git-bug-new-from-line	今の行の `TODO:` から課題を作る
⌥x	git-bug-ls	課題の一覧を org-mode のバッファに出す

- ^c b m と ^c b c は、init.el で割り当てています。
- ^c b c は、 `TODO:` 、 `FIX:` 、 `BUG:` 、 `HACK:` 、 `XXX:` のある行で使います。後ろの文が課題の題名になり、行は `TODO(gb#76c50e2):  題名` のように変わります。

^c b m で課題を選んだ後に選ぶ操作は次のとおりです。

操作	内容
show	題名とコメントをバッファに出す
edit	題名を書き換える
comment	コメントを加える
insert	課題の番号（ gb# と 7 桁）をカーソルの位置に入れる
close	課題を閉じる
open	閉じた課題を開き直す
rm	課題を消す

11.3.3 編集のバッファ

edit と comment は、 BUG_MESSAGE_EDITMSG というバッファを開きます。書き終えたら、 ^x ^s で保存し、 ^x # で確定します。中身を空にして確定すると、操作をやめます。

git-bug は、 GIT_EDITOR に指定したエディタで題名やコメントを書かせます。パッケージのままでは、git に設定したエディタ（vim など）を画面の無い所で起動し、編集できません。init.el では、Magit と一緒に入る with-editor で、Emacs のバッファを開くようにしています。

```
(leaf git-bug
  :ensure t
  :bind
  ("C-c b m" . git-bug-menu)
  ("C-c b c" . git-bug-new-from-line)
  :config
  (require 'with-editor)
  (defun my/gb-editor (fn &rest args)
    (with-editor* "GIT_EDITOR" (apply fn args)))
  (advice-add 'git-bug-edit-bug :around #'my/gb-editor)
  (advice-add 'git-bug-comment-new :around #'my/gb-editor))
```

- advice-add は、パッケージの関数の前後に処理を足す仕組みです。

11.3.4 ほかの人と共有する

課題は、 git push では送られません。ターミナルで次のコマンドを使います。

```
git bug push    # 課題をリモートへ送る
git bug pull    # 課題をリモートから受け取る
```

11.4 チェックリスト

11.4.1 やるべきこと

- コミットの前に、 TAB で差分を開いて中身を確かめる。
- 関係のない変更は、別のコミットに分ける。行の単位でステージできます。
- git のコマンドが失敗したら、 $ で記録を見る。
- 課題を作ったら、 git bug push で送る。

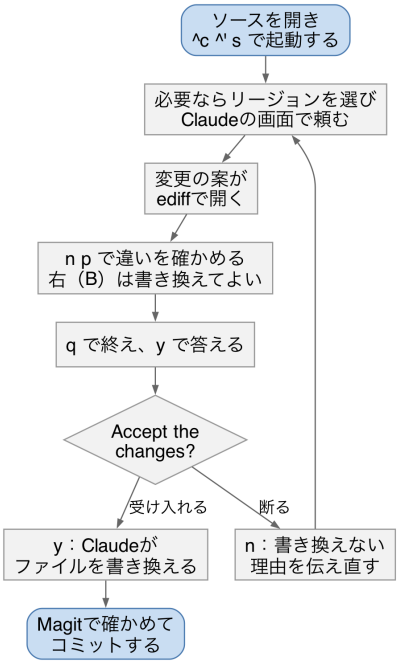
11.4.2 やってはいけないこと

- k の確認に、内容を見ずに答える。捨てた変更は戻せません。
- プッシュした後のコミットを c a で修正する。リモートの履歴と食い違います。

第12章　Claude Codeとの連携

Claude Code は、Emacs の中で claude-code-ide パッケージを通して使います。claude-code-ide は、Claude Code のコマンド（CLI）を vterm で動かし、Emacs とつなぎます。つなぐと、Claude は開いているファイルと選んだ範囲を知り、変更の案を ediff で見せます。

変更を頼む流れは次のとおりです。



12.1 導入

12.1.1 必要なもの

もの	入れ方
Claude Code	`brew install --cask claude-code`。コマンドは `claude`
vterm	第 5 章の「vterm の導入」を参照
claude-code-ide	init.el の設定で、GitHub から入れる

- Claude Code を初めて使うときは、ターミナルで `claude` を実行してログインします。

12.1.2 設定

init.el には次の設定を書いています。

```
(use-package claude-code-ide
  :vc (:url "https://github.com/manzaltu/claude-code-ide.el"
       :rev :newest)
  :init
  (setq claude-code-ide-terminal-backend 'vterm)
  :bind ("C-c C-'" . claude-code-ide-menu)
  :config
  (claude-code-ide-emacs-tools-setup))
```

設定	意味
`:vc`	パッケージを GitHub のリポジトリから入れる
`claude-code-ide-terminal-backend`	Claude Code を動かす端末に vterm を使う
`:bind`	`^c ^'` でメニューを開く
`claude-code-ide-emacs-tools-setup`	Emacs のツールを Claude が使えるようにする

12.2 起動と終了

ソースのファイルを開いて、 ^c ^' でメニューを開き、次の 1 文字を選びます。

^c ^' s	claude-code-ide	プロジェクトで Claude Code を起動する
^c ^' c	claude-code-ide-continue	最後の会話の続きから起動する
^c ^' r	claude-code-ide-resume	前の会話を選んで起動する
^c ^' q	claude-code-ide-stop	Claude Code を終える
^c ^' b	claude-code-ide-switch-to-buffer	Claude のバッファへ移る
^c ^' w	claude-code-ide-toggle	Claude のウィンドウを隠す/出す
^c ^' l	claude-code-ide-list-sessions	動いている Claude Code を一覧にして移る

- Claude Code は、ウィンドウの右側に幅 100 桁で開きます。
- バッファ名は *claude-code[プロジェクト名]* です。
- プロジェクトは、git のリポジトリの全体です。Claude Code はその根で起動します。
- 初めてのディレクトリでは、そのディレクトリを信頼するかを尋ねます。↓で「Yes, I trust this folder」を選び、 RET で決めます。
- ウィンドウを ^x 0 で閉じても、Claude Code は止まりません。

同じプロジェクトで ^c ^' s をもう一度押すと、2 つ目の Claude Code を起動します。名前を尋ねるので、 refactor などと付けます。

12.3 Claude とのやり取り

Claude のバッファは vterm です。文字を打ち、RET で送ります。vterm の基本のキーは、第 5 章の「vterm のキー」で説明しています。

⇧RET	claude-code-ide-insert-newline	送らずに改行する
^ESC	claude-code-ide-send-escape	Claude の作業を止める
^c ^x	claude-code-ide-clear-selection	ファイルと範囲の文脈を外す

- ESC は、Emacs では次のキーと組み合わせるメタキーとして働きます。そのため、作業を止めるときは ^ESC を使います。

ほかのバッファからは、メニューで次の操作ができます。

^c ^' p	claude-code-ide-send-prompt	ミニバッファで書いた文を送る
^c ^' i	claude-code-ide-insert-at-mentioned	選んだ範囲を、Claude の入力欄に入れる
^c ^' x	claude-code-ide-clear-selection	ファイルと範囲の文脈を外す
^c ^' e	claude-code-ide-send-escape	Claude の作業を止める

12.3.1 ファイルと範囲の共有

Claude には、今開いているファイルの名前と、選んでいる範囲が伝わります。

- Claude の入力欄の下に、「In ファイル名」や「3 lines selected」と出ます。
- 次に送る文に、そのファイルや範囲の情報が付きます。ファイルの中身は送りません。
- 「この関数を直して」のように、リージョンで選んでから頼めます。

- 伝えたくないときは、`^c ^x` で外します。別のファイルを開くと、また伝わります。

12.4 変更の確認

Claude がファイルを書き換えるときは、先に変更の案を ediff で開きます。左（A）が今のファイル、右（B）が Claude の案です。

順	操作
1	`n` と `p` で違いを順に見る
2	直したいところは、右（B）のバッファを書き換える
3	操作用のウィンドウで `q` を押し、「Quit this Ediff session?」に `y` で答える
4	「Accept the changes?」に答える。 `y` で受け入れ、 `n` で断る

- `y` で受け入れると、右（B）の内容で Claude がファイルを書き換えます。手で書き換えた分も入ります。
- ediff のキーは、第 11 章の「ediff」で説明しています。

12.5 Emacs のツール

`claude-code-ide-emacs-tools-setup` で、Claude は Emacs の次の機能を使えます。

機能	Claude ができること
xref	名前を使っている箇所や、名前の一部に合う定義を探す
imenu	ファイルの関数や変数の一覧を得る
project	プロジェクトのディレクトリやファイルの情報を得る
tree-sitter	コードの構文の木を調べる

- eglot がつないでいるバッファでは、xref が言語サーバの結果を返します。第 10 章を参照してください。
- Claude に「この関数の呼び出し元を探して」と頼むと、これらの機能を使います。

12.6 チェックリスト

12.6.1 やるべきこと

- Claude Code は、git のリポジトリの中のファイルを開いてから起動する。
- 頼む前に、対象の範囲をリージョンで選ぶ。
- 変更の案は、ediff で違いを 1 つずつ確かめてから受け入れる。
- 受け入れた後は、Magit の差分で全体を確かめてからコミットする。

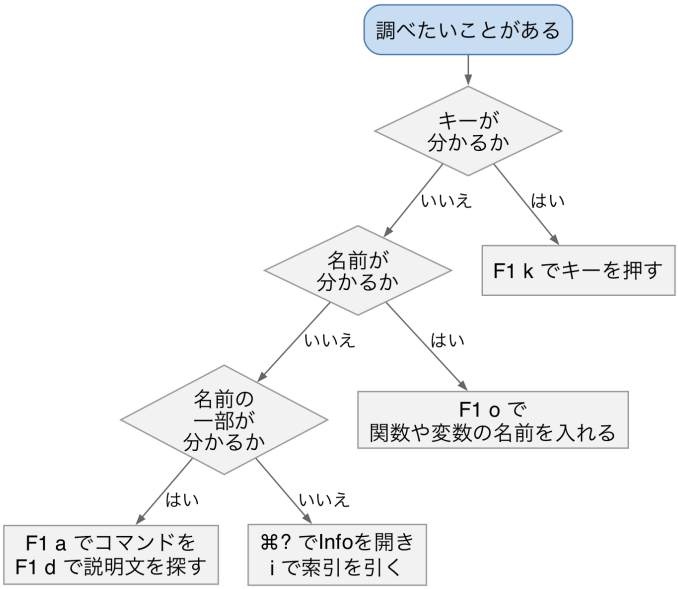
12.6.2 やってはいけないこと

- 中身を確かめずに、信頼の確認に「Yes」と答える。Claude はそのディレクトリのファイルを読み書きします。

第13章　オンラインヘルプ

ヘルプは F1 で始まるキーで呼び出します。Emacs の標準のヘルプのキーは ^h で始まりますが、この環境では ^h を DEL に置き換えています。F1 は ^h と同じ働きをするので、 ^h k は F1 k と読み替えます。

調べたいことから、使うヘルプを次のように選びます。



13.1 ヘルプのキー

Mac のキーボードで F1 を押すには、 fn を押しながら F1 を押します。システム設定で「F1、F2 などのキーを標準のファンクションキーとして使用」を入にすると、fn は要りません。

F1 k	describe-key	押したキーの説明を出す
F1 c	describe-key-briefly	押したキーのコマンド名をエコーエリアに出す
F1 w	where-is	コマンドを割り当てたキーを出す
F1 f	describe-function	関数の説明を出す
F1 v	describe-variable	変数の値と説明を出す
F1 o	describe-symbol	関数と変数のどちらでも説明を出す
F1 m	describe-mode	今のモードの説明とキーを出す
F1 b	describe-bindings	今使えるすべてのキーを出す
F1 a	apropos-command	名前に文字列を含むコマンドを探す
F1 d	apropos-documentation	説明文に文字列を含むものを探す
F1 P	describe-package	パッケージの説明を出す
F1 l	view-lossage	最近押したキーとコマンドを出す
F1 e	view-echo-area-messages	これまでのメッセージ（ *Messages* ）を出す
F1 t	help-with-tutorial	チュートリアルを開く
F1 ?	help-for-help	ヘルプのキーの一覧を出す
^⌘?	help-for-help	ヘルプのキーの一覧を出す

- ^⌘? は、 ^⌘⇧/ で押します。init.el で割り当てています。
- 名前を入れるときは、カーソルの位置の名前が最初の候補になります。

- `F1 l` は、思わぬ動きをしたときに、何を押したかを確かめるのに使います。

13.1.1 ヘルプの画面

説明は `*Help*` バッファに出ます。このバッファでは、次のキーを使います。

TAB	forward-button	次のリンクへ移動する
RET	push-button	リンクをたどる
l	help-go-back	前の説明へ戻る
r	help-go-forward	次の説明へ進む
s	help-view-source	定義したソースを開く
i	help-goto-info	Emacs のマニュアルの該当のノードを開く
I	help-goto-lispref-info	Emacs Lisp のマニュアルの該当のノードを開く
q	quit-window	ヘルプを閉じる

13.2 Info

Info は、Emacs でマニュアルを読む仕組みです。マニュアルは、ノードという単位の画面に分かれ、ノードはメニューと相互参照でつながっています。

⌘?	info	Info を開く
F1 i	info	Info を開く
F1 r	info-emacs-manual	Emacs のマニュアルを開く
F1 R	info-display-manual	名前を入れてマニュアルを開く
F1 F	Info-goto-emacs-command-node	コマンドを説明したノードを開く
F1 K	Info-goto-emacs-key-command-node	キーを説明したノードを開く
F1 S	info-lookup-symbol	今のモードの言語のマニュアルで、名前を引く

- ⌘? は、 ⌘⇧/ で押します。
- ^u F1 i は、info のファイルを指定して開きます。

13.2.1 Info の使い方

Info のバッファでは、次のキーを使います。

SPC	Info-scroll-up	1 画面下へ。末尾では次のノードへ
DEL	Info-scroll-down	1 画面上へ。先頭では前のノードへ
⇧SPC	Info-scroll-down	1 画面上へ。先頭では前のノードへ
b	beginning-of-buffer	ノードの先頭へ
n	Info-next	同じ階層の次のノードへ
p	Info-prev	同じ階層の前のノードへ
]	Info-forward-node	文書の順で次のノードへ
[	Info-backward-node	文書の順で前のノードへ
u	Info-up	1 つ上の階層のノードへ
t	Info-top-node	マニュアルの先頭のノードへ
d	Info-directory	マニュアルの一覧へ
m	Info-menu	メニューの項目を選んで移動する
f	Info-follow-reference	相互参照をたどる
TAB	Info-next-reference	次のリンクへ移動する
l	Info-history-back	履歴を戻る
r	Info-history-forward	履歴を進む
i	Info-index	索引から探す
,	Info-index-next	索引の次の候補へ
I	Info-virtual-index	索引の候補をメニューで表示する
s	Info-search	正規表現で検索する
g	Info-goto-node	ノードの名前を入れて移動する
q	quit-window	Info を閉じる

- 調べものは、まず i で索引を引きます。索引からは、説明のあるノードへ直接移れます。

13.3 日本語のマニュアル

この環境では、Emacs、Emacs Lisp、Org のマニュアルを日本語版で開きます。

マニュアル	ファイル	版	開き方
Emacs	`emacs-ja.info`	29.4	`F1 r`
Emacs Lisp	`elisp-ja.info`	29.4	`F1 R elisp`
Org	`org-ja.info`	7.5	`F1 R org`

- ファイルは `~/.emacs.d/info/` の下にあります。git では管理していません。
- Emacs 本体は 31.1 ですが、マニュアルは 29.4 版です。30 と 31 で増えた機能は載っていません。新しい機能は、`F1` のヘルプで確かめます。
- Org の日本語版は、Org 7.5 のときのものです。Emacs 31.1 に入っている Org とは内容が大きく違います。
- 英語版を読むには、`^u F1 i` で `/Applications/Emacs.app/Contents/Resources/info/emacs.info.gz` を開きます。

13.3.1 日本語版に替える設定

init.el では、マニュアルの名前を日本語版の名前に置き換えています。

```
(add-to-list 'Info-default-directory-list
             "~/.emacs.d/info/emacs-ja.texis/")
(add-to-list 'Info-default-directory-list
             "~/.emacs.d/info/elisp-ja.texis/")
(add-to-list 'Info-default-directory-list
             "~/.emacs.d/info/org-mode.texis/")
(defun Info-find-node--info-ja (orig-fn filename &rest args)
  (apply orig-fn
         (pcase filename
           ("emacs" "emacs-ja")
```

(次のページに続く)

(前のページからの続き)

```
            ("elisp" "elisp-ja")
            ("org" "org-ja")
            (_ filename))
          args))
(advice-add 'Info-find-node :around 'Info-find-node--info-ja)
```

- Info-default-directory-list は、info のファイルを探すディレクトリの一覧です。
- Info-find-node は、マニュアルのノードを開く関数です。emacs を開こうとすると、代わりに emacs-ja を開きます。

13.3.2 新しい版に替える

日本語版は、綾小路さんの翻訳（https://ayatakesi.github.io/）を使っています。2026 年 11 月の時点の最新版（29.4）を入れています。新しい版は、原稿（texi）から info を作って置き換えます。

```
cd ~/Downloads
base=https://ayatakesi.github.io
curl -LO $base/emacs/29.4/emacs-ja.texis.tar.gz
tar xzf emacs-ja.texis.tar.gz
cd emacs/japanese_texis
makeinfo --no-split -o emacs-ja.info emacs-ja.texi
cp emacs-ja.info ~/.emacs.d/info/emacs-ja.texis/
```

- makeinfo は、Homebrew の texinfo に入っています。
- --no-split は、info を 1 つのファイルにまとめる指定です。
- Emacs Lisp のマニュアルは、 $base/lispref/29.4/elisp-ja.texis.tar.gz を同じ手順で処理します。展開先は lispref/japanese_texis 、原稿は elisp-ja.texi です。
- 置き換えた後は、開いている Info を閉じてから開き直します。

13.4 チェックリスト

13.4.1 やるべきこと

- キーの動きが分からないときは、 `F1 k` でそのキーを押す。
- 関数や変数の説明では、 `s` でソースを見る。
- マニュアルは、 `i` で索引から引く。
- 日本語版のマニュアルに無い機能は、英語版で確かめる。

13.4.2 やってはいけないこと

- ヘルプのつもりで `^h` を押す。この環境では 1 文字消えます。
- 日本語版のマニュアルの内容を、Emacs 31 の動きとして信じ込む。29.4 版の内容です。
- Org の日本語版で、今の Org の使い方を調べる。版が古すぎます。

用語集

インクリメンタルサーチ
: 文字を入力するたびに、一致する位置へ移動する検索です。 `^s` で始めます。

Info
: Emacs でマニュアルを読む仕組みです。マニュアルはノードという画面の単位に分かれています。 `⌘?` で開きます。

ウィンドウ
: フレームを分割した表示領域です。1 つのウィンドウに 1 つのバッファを表示します。

エコーエリア
: フレームの下端の、メッセージを表示する場所です。ミニバッファと同じ場所を使います。

キル
: テキストを削除し、キルリングに保存する操作です。macOS のカットに当たります。

キルリング
: キルしたテキストを保存しておく場所です。

言語サーバ
: 言語ごとにコードを解析する別のプログラムです。eglot が、LSP という決まりで通信します。

数引数
: コマンドに渡す数です。繰り返す回数などを指定します。 `^u` で入力します。

ステージ
: git で、次のコミットに入れる変更を選ぶことです。Magit では s で行います。

正規表現
: 文字列の形を記号で表す書き方です。Emacs では \(\) のように \ を付ける記号があります。

タブ
: フレームの上端に並ぶ切り替え口です。タブごとにウィンドウの配置を覚えます。

バッファ
: 編集中のテキストを保持する領域です。ファイルを開くと、内容がバッファに読み込まれます。

ハンク
: 差分の中の、ひとまとまりの変更です。Magit では、ハンクの単位でステージできます。

フレーム
: Emacs が開く macOS のウィンドウ 1 つ分です。

プレフィックスキー
: ^x のように、続けて別のキーを待つキーです。

プロジェクト
: git のリポジトリの全体です。 ^x p で始まるコマンドや eglot が、この単位で動きます。

ポイント
: カーソルの位置です。

補完
: 名前を途中まで入力して、候補から選ぶ機能です。

マーク
: ポイントとともにリージョンの範囲を決める位置です。

マイナーモード
: バッファに機能を足すモードです。1 つのバッファでいくつでも有効にできます。

ミニバッファ
: フレームの下端にある入力欄です。コマンド名やファイル名を入力します。

メジャーモード
: バッファの編集方法を決めるモードです。1 つのバッファに 1 つだけ設定します。

モードライン
: ウィンドウの下端にある 1 行です。バッファ名やモードなどを表示します。

ヤンク
: キルリングのテキストを挿入する操作です。macOS のペーストに当たります。

リージョン
: ポイントとマークの間のテキストです。

索引

I

Info, **141**

イ

インクリメンタルサーチ, **141**

ウ

ウィンドウ, **141**

エ

エコーエリア, **141**

キ

キル, **141**
キルリング, **141**

ケ

言語サーバ, **141**

ス

数引数, **141**
ステージ, **142**

セ

正規表現, **142**

タ

タブ, **142**

ハ

バッファ, **142**
ハンク, **142**

フ

フレーム, **142**
プレフィックスキー, **142**
プロジェクト, **142**

ホ

ポイント, **142**
補完, **142**

マ

マーク, **142**
マイナーモード, **143**

ミ

ミニバッファ, **143**

メ

メジャーモード, **143**

モ

モードライン, **143**

ヤ

ヤンク, **143**

リ

リージョン, **143**