Emacs Cheatsheet

Emacs Cheatsheet

GNU Emacs is a Lisp-based, infinitely extensible editor. It's not just a text editor — it's a complete computing environment. Key convention: `C-` = Ctrl, `M-` = Alt (Meta).

Basic Navigation

Key	Action	Mnemonic
`C-f`	Forward one character	Forward
`C-b`	Back one character	Back
`C-n`	Next line	Next
`C-p`	Previous line	Previous
`C-a`	Beginning of line	Beginning of alphabet
`C-e`	End of line	End
`M-f`	Forward one word	Meta Forward
`M-b`	Back one word	Meta Back
`M-a`	Beginning of sentence	Meta Beginning
`M-e`	End of sentence	Meta End
`C-v`	Page down	View
`M-v`	Page up	Meta View
`C-l`	Re-center cursor on screen	Line
`M-<`	Beginning of buffer	Meta Less-than
`M->`	End of buffer	Meta Greater-than
`C-g`	Cancel/quit current command	Go away

File & Buffer Management

```
# Files
C-x C-f       # find (open) file
C-x C-s       # save file
C-x C-w       # write as (save as)
```

```
C-x C-c       # exit Emacs
C-x k         # kill (close) current buffer
C-x C-j       # open Dired (file manager)

# Buffers
C-x b         # switch buffer (with auto-complete)
C-x C-b       # list all buffers
C-x <left>    # previous buffer
C-x <right>   # next buffer
C-x k         # kill buffer
```

Editing

Key	Action
`Backspace`	Delete previous character
`C-d`	Delete next character
`M-Backspace`	Delete previous word
`M-d`	Delete next word
`C-k`	Kill to end of line (cut)
`M-k`	Kill to end of sentence
`C-w`	Kill region (cut selection)
`M-w`	Copy region
`C-y`	Yank (paste)
`M-y`	Cycle through kill ring after yank
`C-/` or `C-x u`	Undo
`C-Shift-/` or `C-x C-Shift-u`	Redo (Emacs 28+)
`C-Space`	Set mark (begin selection)
`C-x h`	Select entire buffer
`C-t`	Transpose two characters
`M-t`	Transpose two words
`C-x C-t`	Transpose two lines
`M-c`	Capitalize word
`M-u`	Uppercase word
`M-l`	Lowercase word
`M-q`	Fill (reflow) paragraph

Search & Replace

```
# Incremental search
C-s           # search forward (incremental)
C-r           # search backward (incremental)
# While searching:
#   C-s again = jump to next match
#   C-r again = jump to previous match
```

```
#   C-g = cancel search

# Non-incremental search
M-C-s         # search forward with regex
M-C-r         # search backward with regex

# Replace
M-%           # query-replace (prompted replace)
#   y = replace, n = skip, ! = replace all, q = quit

# Replace with regex
C-M-%         # query-replace-regexp
```

Kill & Yank Ring

```
# The kill ring is Emacs's multi-item clipboard
C-k           # kill (cut) line from cursor to end
C-k C-k       # kill entire line (including newline)
C-w           # kill region (cut selection)
M-w           # copy region (without killing)
C-y           # yank (paste most recent kill)
M-y           # cycle kill ring — press repeatedly after C-y

# Example: paste something from 3 cuts ago
C-y M-y M-y M-y
```

Window Management

```
# Split windows
C-x 2         # split horizontally
C-x 3         # split vertically
C-x 1         # close all other windows
C-x 0         # close current window

# Navigate between windows
C-x o         # switch to next window

# Resize windows
C-x ^         # grow window height
C-x }         # grow window width
C-x {         # shrink window width
C-x +         # balance all windows
```

Org-Mode Basics

Org-Mode is Emacs's killer app — an outline-based document and task management system built in.

```
# Open an Org file
C-x C-f todo.org

# Outline navigation
Tab           # cycle visibility (fold/unfold subtree)
S-Tab         # cycle global visibility
C-c C-n       # next heading
C-c C-p       # previous heading
```

```
C-c C-f       # forward heading (same level)
C-c C-b       # backward heading (same level)
C-c C-u       # go to parent heading

# Structure editing
M-Enter       # new heading/item below
M-S-Enter     # new heading (same level as parent)
M-Left/Right  # promote/demote subtree
M-S-Left/Right # promote/demote subtree with children
M-Up/Down     # move subtree up/down

# Todo lists
C-c C-t       # cycle todo state (TODO → DONE → TODO)
S-Right/Left  # cycle through todo keywords

# Tables (built-in spreadsheet)
C-c |         # create or convert to table
Tab           # move to next cell / re-align
S-Tab         # move to previous cell
M-Left/Right  # move column
M-Up/Down     # move row
C-c -         # insert horizontal line
C-c Enter     # insert row below
C-c C-c       # recalculate table formulas

# Links
C-c C-l       # insert/edit link
C-c C-o       # open link at point

# Export
C-c C-e       # export dispatcher
#   h h = export to HTML
#   l p = export to PDF (via LaTeX)
#   l o = export to ODT
#   m m = export to Markdown
```

Practical Org-Mode Example

```
# Create a task list
* Project Alpha
** TODO Research competitor features
   SCHEDULED: <2026-04-25>
** TODO Design API schema
   SCHEDULED: <2026-04-28>
** DONE Write initial documentation
   CLOSED: [2026-04-18 Sat]

# A simple table with formula
| Item     | Qty | Price | Total |
|----------+-----+-------+-------|
| Widget   |   3 |    10 |    30 |
| Gadget   |   2 |    25 |    50 |
| Doohickey|   1 |    15 |    15 |
|----------+-----+-------+-------|
| Total    |   6 |       |    95 |
#+TBLFM: @5$4=vsum(@2$4..@4$4)
```

.emacs / emacs.d Init

```
# ~/.emacs.d/init.el — modern Emacs configuration

;; Package management
(require 'package)
(setq package-archives '(("melpa" . "https://melpa.org/packages/")
                         ("gnu"   . "https://elpa.gnu.org/packages/")))
(package-initialize)
(unless package-archive-contents
  (package-refresh-contents))

;; Use-package for clean config (built-in since Emacs 29)
(setq use-package-always-ensure t)

;; UI
(tool-bar-mode -1)          ; hide toolbar
(menu-bar-mode -1)          ; hide menubar
(scroll-bar-mode -1)        ; hide scrollbar
(column-number-mode 1)      ; show column number
(global-display-line-numbers-mode 1)  ; show line numbers
(show-paren-mode 1)         ; highlight matching parens
(electric-pair-mode 1)      ; auto-close brackets

;; Theme
(use-package doom-themes
  :config
  (setq doom-themes-enable-bold t
        doom-themes-enable-italic t)
  (load-theme 'doom-one t))

;; Editing
(setq-default tab-width 4)
(setq-default indent-tabs-mode nil)
(setq-default fill-column 80)
(global-auto-revert-mode t)  ; auto-reload changed files
(setq backup-directory-alist '(("." . "~/.emacs.d/backups")))
(setq auto-save-file-name-transforms '((".*" "~/.emacs.d/auto-save/" t)))

;; Key bindings
(global-set-key (kbd "C-<tab>") 'other-window)
(global-set-key (kbd "M-/") 'company-complete)
(global-set-key (kbd "C-c g") 'magit-status)
(global-set-key (kbd "C-x k") 'kill-current-buffer)
```

Common Packages

Magit (Git Interface)

```
# Install
M-x package-install RET magit RET

# Usage
C-x g         # open Magit status buffer
# In Magit status:
#   s = stage file
```

```
#   u = unstage file
#   c c = commit (opens commit message buffer)
#   C-c C-c = finish commit
#   P p = push
#   F p = pull
#   l l = log
#   d d = diff
#   z s = stash
#   b b = checkout branch
#   Tab = expand/collapse section
#   q = quit Magit
```

Company (Auto-Completion)

```
# Install
M-x package-install RET company RET

# Configuration (add to init.el)
(use-package company
  :hook (prog-mode . company-mode)
  :config
  (setq company-minimum-prefix-length 1
        company-idle-delay 0.0)
  (global-company-mode 1))

# Usage (in prog-mode, completions appear automatically):
M-/           # trigger completion manually
C-n / C-p     # navigate completion candidates
Tab           # accept completion
```

Flycheck (Syntax Checking)

```
# Install
M-x package-install RET flycheck RET

# Configuration (add to init.el)
(use-package flycheck
  :init (global-flycheck-mode))

# Usage:
# Flycheck runs automatically in supported modes
C-c ! l       # list errors
C-c ! n       # jump to next error
C-c ! p       # jump to previous error
C-c ! v       # verify Flycheck setup
C-c ! s       # set checker
```

Key Chord Reference

Essential Survival Keys

Key	Action
`C-g`	Cancel any command

Key	Action
`C-x C-c`	Quit Emacs
`C-x u` or `C-/`	Undo
`C-x C-f`	Open file
`C-x C-s`	Save file
`C-x C-b`	List buffers
`C-x b`	Switch buffer

Movement

Key	Action
`C-f / C-b`	Char forward / back
`C-n / C-p`	Line down / up
`C-a / C-e`	Line start / end
`M-f / M-b`	Word forward / back
`M-< / M->`	Buffer start / end
`C-v / M-v`	Page down / up

Killing & Yanking

Key	Action
`C-k`	Kill line (cut to end)
`C-w`	Kill region (cut selection)
`M-w`	Copy region
`C-y`	Yank (paste)
`M-y`	Cycle kill ring

Window & Buffer

Key	Action
`C-x 2`	Split horizontal
`C-x 3`	Split vertical
`C-x 1`	Single window
`C-x o`	Other window
`C-x b`	Switch buffer
`C-x k`	Kill buffer

Search & Replace

Key	Action
`C-s / C-r`	Search forward / backward

Key	Action
`M-%`	Query replace
`C-M-%`	Query replace regex

Help System

```
C-h f         # describe function
C-h v         # describe variable
C-h k         # describe key binding
C-h m         # describe current mode's bindings
C-h a         # apropos (search all commands by keyword)
C-h t         # built-in Emacs tutorial
C-h i         # Info manual browser
```

Dired (File Manager)

Dired is Emacs's built-in file manager — a powerful directory editor for navigating, viewing, and manipulating files.

```
# Open Dired
C-x C-j         # open Dired in current directory (dired-jump)
C-x d           # open Dired (prompts for directory)
C-u C-x d       # open Dired in other window

# Navigation
n               # next file
p               # previous file
C-n / C-p       # same as n/p
SPC             # scroll down (next page)
DEL             # scroll up (previous page)
g               # refresh directory listing
s               # sort by name (toggle date/name)
C-u s           # sort by date

# Marking files
m               # mark file
u               # unmark file
U               # unmark all files
d               # flag for deletion
D               # delete flagged files
x               # execute all deletions
t               # toggle mark (invert selection)
* .             # mark all files with extension
* @             # mark all symlinks
* /             # mark all directories
M-DEL           # unmark all files

# File operations on marked files
C               # copy files (prompts for destination)
R               # rename file
M               # change file mode (chmod)
G               # change group (chgrp)
O               # change owner (chown)
Z               # compress/decompress files
!               # run shell command on marked file
```

```
# Search in files
A               # search marked files for regex
Q               # query replace in marked files

# Writable Dired (wdired) — edit filenames like text
C-x C-q         # enter wdired mode (make directory editable)
C-c C-c         # commit changes and exit wdired
C-c C-k         # abort wdired changes

# Find files
M-x find-name-dired     # find files by name in directory
M-x find-grep-dired     # find files matching grep pattern
```

Shell & Eshell

```
# Shell mode (interactive system shell)
M-x shell        # open shell buffer (runs $SHELL)
M-x eshell       # open Eshell (Emacs Lisp shell)

# Run shell commands from any buffer
M-!              # run shell command, show output
M-|              # pipe region through shell command (replace region)
C-u M-!          # run command and insert output at point

# Eshell-specific features
# Eshell is written in Emacs Lisp — works the same on all platforms
# Supports Emacs Lisp expressions alongside regular commands
# Tab completion works with Emacs's completion system

# Useful Eshell aliases (add to ~/.eshell-aliases)
# alias ll ls -l $*
# alias gs git status
# alias gl git log --oneline -20

# Async shell commands
M-& or M-x async-shell-command   # run command in background
M-x shell-command-on-region      # pipe selection to shell command

# Compilation mode (run builds and capture output)
M-x compile      # run make (or custom compile command)
M-x recompile    # re-run last compile command
# After compilation:
#   M-n / M-p = next/previous error
#   Enter / g = jump to error
#   C-c C-k = kill compilation
```

Regex & Isearch

```
# Incremental regex search
C-M-s           # isearch-forward with regex
C-M-r           # isearch-backward with regex

# Regex replace
C-M-%           # query-replace-regexp

# Emacs regex syntax
```

```
\( \)           # capture group (use \1, \2 to reference)
\|              # alternation (OR)
\`              # beginning of buffer
\'              # end of buffer
\b              # word boundary
\<              # beginning of word
\>              # end of word
\[ \]           # literal brackets (unlike BRE/ERE)
\{m,n\}         # repetition (use \\{ in string literals)

# re-builder — interactive regex tester
M-x re-builder          # test regex against buffer content
#   C-c C-c = send regex to replace
#   C-c C-w = copy regex
#   Tab = toggle between readable and string syntax
```

Bookmarks

```
# Set bookmarks
C-x r m         # set bookmark at current position
C-x r m <name>  # set named bookmark

# Jump to bookmarks
C-x r b         # jump to bookmark (prompts which one)
C-x r b <name>  # jump to named bookmark directly

# List bookmarks
C-x r l         # list all bookmarks in a buffer
#   In bookmark list:
#   Enter = jump to bookmark
#   d = mark for deletion
#   x = delete marked bookmarks
#   r = rename bookmark
#   o = jump in other window

# Bookmark file (bookmarks persist across sessions)
# Stored in ~/.emacs.d/bookmarks by default
# Automatically saved on exit
```

Registers

```
# Point-to-register (save cursor position)
C-x r SPC <r>   # save current position to register r
C-x r j <r>     # jump to position stored in register r

# Copy to register
C-x r s <r>     # copy region to register r
C-x r x <r>     # copy region to register r (same as above)

# Insert from register
C-x r i <r>     # insert contents of register r at point

# Frame/window configuration registers
C-x r w <r>     # save window configuration to register r
C-x r f <r>     # save frame configuration to register r
C-x r j <r>     # restore window/frame configuration
```

```
# Number register
C-u 100 C-x r i 1   # insert register 1 contents 100 times

# View registers
M-x view-register     # view contents of a register
C-x r j <register>    # jump to register (works for all types)

# Keyboard macro register
C-x (           # start defining keyboard macro
C-x )           # end defining keyboard macro
C-x e           # execute last defined macro
C-u 10 C-x e    # execute macro 10 times
```

Macros

```
# Keyboard macros (F3/F4 keys in modern Emacs)
F3 or C-x (     # start recording macro
F4 or C-x )     # stop recording macro
F4 or C-x e     # replay last macro
C-u 10 F4       # replay 10 times
C-u 0 F4        # replay until error (macro stops when it fails)

# Name a macro
M-x name-last-kbd-macro    # give the last macro a name
# It becomes an interactive command you can M-x invoke

# Save macros to init file
M-x insert-kbd-macro        # insert macro definition as Lisp code
# Add the output to your init.el to persist across sessions

# Macro ring (if you record multiple macros)
C-x e then e then e         # each e plays the next macro in the ring
```

Rectangle Editing

Rectangle editing works on column-aligned blocks of text — essential for editing tables, structured data, and code.

```
# Kill and yank rectangles
C-x r k         # kill rectangle (cut rectangular region)
C-x r y         # yank rectangle (paste)
C-x r d         # delete rectangle (don't copy to kill ring)
C-x r M-w       # copy rectangle

# Rectangle operations
C-x r o         # open rectangle (insert blank space, shifting text right)
C-x r c         # clear rectangle (replace with spaces)
C-x r t         # string rectangle (insert string on left side of rectangle)
C-x r N         # insert line numbers along left side of rectangle

# Rectangle mark mode
C-x SPC         # enter rectangle mark mode
# Then use movement keys to define the rectangle
# Use C-x r k to kill it

# Practical example: insert line numbers
```

```
# 1. Select lines with C-SPC, move down
# 2. C-x r N to number them
# 3. C-x r t "prefix: " to prepend text to each line
```

TRAMP (Remote Editing)

TRAMP (Transparent Remote Access, Multiple Protocol) lets you edit files on remote servers as if they were local.

```
# SSH remote files
C-x C-f /ssh:user@host:/path/to/file

# SSH with custom port
C-x C-f /ssh:user@host#2222:/path/to/file

# Sudo (edit as root)
C-x C-f /sudo::/etc/nginx/nginx.conf
C-x C-f /sudo:root@localhost:/etc/hosts

# Docker containers
C-x C-f /docker:container-name:/path/to/file
C-x C-f /docker:user@container:/path/to/file

# Other protocols
C-x C-f /su:               # su/sudo access
C-x C-f /rsync:user@host:/path  # rsync-based
C-x C-f /sftp:user@host:/path   # SFTP

# TRAMP tips
# - First connection is slow (caches for subsequent use)
# - Works with all Emacs features (grep, dired, magit, etc.)
# - M-x find-file-at-point (C-x C-f) completes TRAMP paths
# - M-x tramp-cleanup-connection to reset cached connections
```

Org-Mode Advanced

```
# Org Capture — quick note capture
C-c c           # open capture template
# Configure templates in init.el:
# (setq org-capture-templates
#   '(("t" "Todo" entry (file+headline "~/org/todo.org" "Tasks")
#      "* TODO %?\n  %U\n  %a")
#     ("n" "Note" entry (file+datetree "~/org/notes.org")
#      "* %? :note:\n  %U")))

# Org Agenda
C-c a           # open agenda dispatcher
#   a = agenda (weekly view)
#   t = global todo list
#   s = search
#   m = match tags
#   T = custom todo query

# Org Refile — move subtrees between files
C-c C-w         # refile subtree to another location
# Configure refile targets:
# (setq org-refile-targets
```

```
#   '((org-agenda-files :maxlevel . 3)))

# Clocking time
C-c C-x C-i     # clock in (start timer on current heading)
C-c C-x C-o     # clock out (stop timer)
C-c C-x C-r     # insert clock report (table of time spent)
C-c C-x C-d     # display clock diary

# Properties
C-c C-x p       # set property
C-c C-x p RET   # remove property

# Export backends
C-c C-e         # export dispatcher
#   h h = HTML    h o = HTML and open in browser
#   l p = PDF     l o = PDF and open
#   o o = ODT     m m = Markdown
#   b b = Beamer (slides)

# Org-Babel — execute source code in documents
C-c C-c         # execute code block under cursor
C-c C-v t       # toggle display of source block body
C-c C-v e       # expand code block
C-c C-v n       # jump to next code block
C-c C-v p       # jump to previous code block

# Example code block:
# #+BEGIN_SRC bash
# echo "Hello from Org-Babel"
# #+END_SRC
# #+RESULTS:
# : Hello from Org-Babel

# Tags and priorities
C-c C-q         # set tags for current heading
C-c ,           # set priority (A/B/C)
S-UP/DOWN       # cycle priority
```

Version Control (VC Mode)

```
# VC basic commands (works with git, hg, svn, bzr)
C-x v v         # perform next logical VC action (stage, commit, etc.)
C-x v =         # show diff of current file
C-x v d         # open VC directory (list all files with status)
C-x v l         # show log
C-x v u         # revert file to last committed version
C-x v i         # register file for version control
C-x v ~         # visit previous version of file
C-x v c         # check in (commit) changes
C-x v r         # checkout a branch
C-x v s         # create a snapshot (tag)

# Ediff — side-by-side comparison
M-x ediff           # compare two files
M-x ediff-buffers   # compare two open buffers
M-x ediff-files     # compare files (prompts for both)
M-x ediff3          # 3-way comparison (merge)
M-x ediff-revision  # compare file with a git revision
```

```
# In ediff:
#   n / p = next/previous diff
#   a / b = copy from A/B to other
#   = = highlight differences
#   q = quit
```

Multiple Cursors

```
# mc/mark-more-like-this (install: mc/multiple-cursors package)
C-S-c C-S-c     # edit lines of the same region at once
M-x mc/edit-lines   # edit multiple lines simultaneously

# Mark next/previous like current selection
C-> or M-x mc/mark-next-like-this    # mark next occurrence
C-< or M-x mc/mark-previous-like-this
C-M-> or M-x mc/mark-all-like-this    # mark all occurrences

# Exit multiple cursors
C-g             # exit multiple-cursor mode (single cursor remains)

# Practical example: rename a variable
# 1. Place cursor on variable name
# 2. C-S-c C-S-c to mark all lines
# 3. Edit — changes apply to all marked lines
```

Projectile (Project Management)

```
# Install: M-x package-install RET projectile RET

# Core commands (prefix: C-c p)
C-c p p         # switch project
C-c p f         # find file in project
C-c p F         # find file in project with fd/rg
C-c p b         # switch to project buffer
C-c p k         # kill all project buffers
C-c p d         # find directory in project
C-c p g         # run grep in project
C-c p s         # run ag/rg in project
C-c p o         # run multi-occur in project
C-c p r         # replace string in project
C-c p c         # run compile in project
C-c p t         # open project terminal
C-c p D         # open project root in dired
C-c p 4 f       # find file in other window
C-c p 4 b       # switch buffer in other window

# Configuration (add to init.el)
# (projectile-mode 1)
# (setq projectile-completion-system 'ivy)  ; or 'helm or 'default
# (define-key projectile-mode-map (kbd "C-c p") 'projectile-command-map)
```

Elisp Basics

```
# Evaluate expressions
C-x C-e         # eval-last-sexp (evaluate expression before cursor)
M-x eval-region # evaluate selected region
M-x eval-buffer # evaluate entire buffer
M-:             # eval a single expression in minibuffer

# Help and documentation
C-h f           # describe-function (show function documentation)
C-h v           # describe-variable
C-h k           # describe-key (show what a keybinding does)
C-h m           # describe-mode (show all bindings for current mode)
C-h a           # apropos (search commands by keyword)

# Write a simple function
(defun insert-timestamp ()
  "Insert current timestamp at point."
  (interactive)
  (insert (format-time-string "%Y-%m-%d %H:%M:%S")))

# Bind it to a key
(global-set-key (kbd "C-c t") 'insert-timestamp)

# Interactive functions can be called with M-x
# The (interactive) special form makes it callable

# Common Elisp patterns
(message "Debug: value is %s" my-variable)   ; print to *Messages* buffer
(setq my-var "hello")                        ; set variable
(let ((x 10) (y 20)) (+ x y))               ; local bindings (= 30)
(if (> x 5) "big" "small")                   ; conditional
(when (buffer-modified-p) (save-buffer))     ; when (like if with one branch)
(unless (file-exists-p path) (error "Not found"))  ; unless
(dolist (item my-list) (message "%s" item))  ; iterate list
```

Frame & Tab Management

```
# Frames (OS-level windows)
C-x 5 2         # new frame
C-x 5 0         # close current frame
C-x 5 o         # switch to other frame
C-x 5 f         # open file in new frame
C-x 5 d         # open dired in new frame

# Tab bar mode (built-in since Emacs 27)
M-x tab-bar-mode          # enable/disable tab bar
C-x t 2 or C-x T 2        # new tab
C-x t 0                   # close current tab
C-x t k                   # close tab (prompts which)
C-x t o                   # switch to next tab
C-x t p                   # switch to previous tab
C-x t 1                   # close all other tabs
C-x t f                   # open file in new tab
C-x t d                   # open dired in new tab
C-x t RET or C-x t C-f    # open file and switch to its tab
```

Tips

Install Packages from MELPA

```
# Add MELPA to init.el (if not already present)
(add-to-list 'package-archives
             '("melpa" . "https://melpa.org/packages/") t)

# Then in Emacs:
M-x package-refresh-contents RET
M-x package-install RET <package-name> RET
```

Next Steps

- Bash CLI Tools Cheat Sheet — Unix command-line utilities for power users
- Git CLI Cheat Sheet — Version control commands and workflows
- Vim Cheat Sheet — Powerful modal editor for rapid editing
- Nano Cheat Sheet — Simple terminal editor for quick edits