Neovim / Vim Mini-Guide

A practical 80% reference for everyday editing

`vi -> vim -> neovim | modes, motions, operators, text objects, Ex commands`

Purpose: To serve as a compact personal reference that is more structured than a cheat sheet and much shorter than a full tutorial.
Scope: Common commands plus the advanced, everyday commands that support most real work in Vim and Neovim.
Conventions: <leader> is user-defined. If not remapped, it is commonly \ in Vim and Neovim.

Version 1.0.1 - March 17, 2026
Michael Gardner / A Bit of Help, Inc.

Copyright and License

The full license text appears in Appendix B.

Table of Contents

This edition uses a static table of contents, so the structure is visible immediately in any viewer.

1. What Neovim, Vim, and vi are

The editing model is older than modern IDEs, but the core language is still compact and powerful.

- vi is the original modal editor.
- Vim extends vi and popularized the modern command set most people learn today.
- Neovim keeps the core editing language while expanding plugins, scripting, UI integration, and LSP-oriented workflows.
- Most commands in this guide work across Vim and Neovim.
- Ex commands are the colon commands such as :w, :q, :set number, and :%s/old/new/g.

2. Special keys and notation

This is the most important reference block to keep visually prominent.

Special keys

Command	Meaning	Notes
`Esc`	Return to Normal mode or cancel an operation.	When lost, press Esc once or twice.
`Enter`	Confirm a command or move to the next line.	Used after : commands and searches.
`Tab`	Insert a tab or move through completion choices.	Behavior varies by settings and plugins.
`Backspace`	Delete backward in Insert mode.	Often used to fix recent input.
`Space`	A normal character or a user mapping key.	Many users map it as <leader>.
`:`	Open the Ex command line.	Used for file, setting, range, and substitute commands.
`/`	Search forward.	Press Enter to run the search.
`?`	Search backward.	Useful when the target is above the cursor.
`Ctrl-x`	Hold Control and press x.	Examples: Ctrl-r redo, Ctrl-w window commands.
`<leader>`	User-defined mapping prefix.	Often \ unless remapped.
`<localleader>`	Filetype-local mapping prefix.	Used by plugin or ftplugin mappings.

Notation patterns

Command	Meaning	Notes
`dw`	Operator + motion.	Delete to the start of the next word.
`ci"`	Operator + text object.	Change inside double quotes.
`<C-w>v`	Control-key notation.	Open a vertical split.
`:set nu`	Ex command example.	Turn on line numbers.
`5j`	Count + command.	Move down five lines.

3. Modes

Normal mode is home base. Most friction disappears once the mode model feels natural.

Core modes

Command	Meaning	Notes
`Normal`	Navigate, edit, delete, change, search, paste, and issue commands.	Press Esc to return here.
`Insert`	Type text into the buffer.	Entered with i, a, o, O, I, A.
`Visual`	Select text before acting on it.	v characterwise, V linewise, Ctrl-v blockwise.
`Command-line`	Enter Ex commands, searches, and some history-driven input.	Started with :, /, or ?.

Entering and leaving modes

Command	Meaning	Notes
`i`	Insert before the cursor.	Basic text entry.
`a`	Append after the cursor.	One of the most common entry points.
`I`	Insert at the first nonblank character of the line.	Useful for code indentation.
`A`	Append at end of line.	Fast line-end editing.

`o`	Open a new line below and enter Insert mode.	Very common.
`O`	Open a new line above and enter Insert mode.	Useful when adding a missing line.
`v / V / Ctrl-v`	Enter Visual mode.	Character, line, or block selection.
`Esc`	Return to Normal mode.	The universal escape hatch.

4. Movement and navigation

Movement is the real engine of Vim. Operators become powerful only after motions feel natural.

Basic cursor and line movement

Command	Meaning	Notes
`h j k l`	Left, down, up, right.	The classic home-row movement keys.
`0`	Go to the first column of the line.	Absolute line start.
`^`	Go to the first nonblank character.	Often better than 0 in code.
`$`	Go to end of line.	Pairs well with d, c, and y.
`gg`	Go to top of file.	Count form: 1gg.
`G`	Go to bottom of file.	Count form: 42G.

Word, block, screen, and jump movement

Command	Meaning	Notes
`w`	Jump to next word start.	A core motion.
`b`	Jump to previous word start.	The backward partner to w.
`e`	Jump to end of word.	Useful when targeting the current word end.
`{ / }`	Move to previous or next blank-line-separated block.	Great in prose and code.
`%`	Jump to matching bracket, brace, or parenthesis.	Excellent in code.

`Ctrl-d`	Half-page down.	Keeps context while moving faster.
`Ctrl-u`	Half-page up.	The reverse of Ctrl-d.
`Ctrl-f / Ctrl-b`	Page forward or backward.	Larger movement than half-page.
`Ctrl-o / Ctrl-i`	Jump backward or forward in the jump list.	Very useful after big jumps.

5. Insert, append, open, and replace

These are the main ways to start typing or overwrite text directly.

Insert and append

Command	Meaning	Notes
`i`	Insert before cursor.	The default starting point.
`a`	Append after cursor.	Slightly more natural at the end of words.
`I`	Insert at first nonblank on the line.	Keeps indentation.
`A`	Append at end of line.	Fast line editing.
`o`	Open line below.	Creates a new line and enters Insert mode.
`O`	Open line above.	Creates a new line above.

Replace

Command	Meaning	Notes
`r<char>`	Replace one character.	Example: rx replaces with x.
`R`	Enter replace mode.	Overtypes until Esc.

6. Edit text: delete, change, yank, paste, undo, repeat

This is the everyday editing core. The single most important idea is operator + motion.

Single-purpose essentials

Command	Meaning	Notes
`x`	Delete the character under the cursor.	Small corrective deletion.
`dd`	Delete the current line.	Very common.
`yy`	Yank the current line.	Copy a whole line.
`p`	Paste after the cursor or below the current line.	Standard paste.
`P`	Paste before the cursor or above the current line.	Useful when order matters.
`u`	Undo.	Your recovery key.
`Ctrl-r`	Redo.	Reverse an undo.
`.`	Repeat the last change.	One of Vim's highest-value commands.

Operator patterns

Command	Meaning	Notes
`dw`	Delete a word forward.	Delete to the next word start.
`cw`	Change a word forward.	Delete then enter Insert mode.
`d$`	Delete to end of line.	Same family as D.
`c$`	Change to end of line.	Same family as C.
`yw`	Yank forward by word.	A smaller copy operation.
`D`	Delete to end of line.	Equivalent to d$.
`C`	Change to end of line.	Equivalent to c$.
`cc`	Change the whole line.	Delete line content and enter Insert mode.

7. Visual mode and selection

Visual mode is often the easiest bridge from ordinary editors into Vim-style editing.

Visual selection

Command	Meaning	Notes
`v`	Start characterwise selection.	Select exact characters.
`V`	Start linewise selection.	Select whole lines quickly.
`Ctrl-v`	Start blockwise selection.	Rectangular selections in columns.
`Esc`	Cancel selection and return to Normal mode.	As usual, Esc is your reset.

Common actions on a selection

Command	Meaning	Notes
`d`	Delete the selected text.	Works after any Visual selection.
`y`	Yank the selected text.	Copy selection.
`c`	Change the selected text.	Delete and enter Insert mode.
`>`	Indent selection to the right.	Common in code blocks.
`<`	Indent selection to the left.	Outdent code or prose.

8. Search, substitute, and repeatable editing

Search is not only for finding text; it is also a navigation tool.

Searching

Command	Meaning	Notes
`/pattern`	Search forward for pattern.	Press Enter to run.
`?pattern`	Search backward for pattern.	Reverse direction.
`n`	Repeat search in the same direction.	Keeps current search direction.
`N`	Repeat search in the opposite direction.	Useful when overshooting.
`*`	Search forward for the word under the cursor.	A huge time-saver.

`#`	Search backward for the word under the cursor.	Reverse word search.
`:noh`	Clear search highlighting.	Useful after repeated searches.

Substitute

Command	Meaning	Notes
`:s/old/new/`	Replace first match on the current line.	Small, local edit.
`:s/old/new/g`	Replace all matches on the current line.	Global for the current line.
`:%s/old/new/g`	Replace all matches in the whole file.	Classic full-file substitute.
`:%s/old/new/gc`	Replace all matches and confirm each.	Safer interactive form.

9. Text objects and the editing language

This is the point where Vim stops feeling like a list of commands and starts feeling like a language.

Operator + motion examples: dw, cw, y$, d}. Operator + text object examples: ci", di(, ya{. Text objects let you target the structure around text instead of counting characters.

Common text objects

Command	Meaning	Notes
`iw / aw`	Inner word / a word.	iw excludes surrounding space, aw includes it.
`i" / a"`	Inside quotes / around quotes.	Works for double quotes.
`i' / a'`	Inside single quotes / around single quotes.	Often used in prose and code.
`i( / a(`	Inside parentheses / around parentheses.	Very common.
`i[ / a[`	Inside brackets / around brackets.	Useful in arrays and indexing.
`i{ / a{`	Inside braces / around braces.	Very useful in code blocks.

High-value examples

Command	Meaning	Notes
`ci"`	Change inside double quotes.	Leaves the quotes in place.
`di(`	Delete inside parentheses.	Keeps the parentheses.
`ya{`	Yank around braces.	Copies the braces and enclosed text.
`ciw`	Change inner word.	One of the best early commands to learn.

10. Files, save, quit, and reload

These are core Ex commands and belong in the first tier of memory.

File and quit commands

Command	Meaning	Notes
`:w`	Write the file.	Save current changes.
`:q`	Quit.	Works only when no unsaved changes block the quit.
`:q!`	Quit without saving.	Discard changes.
`:wq`	Write and quit.	Classic combined form.
`:x`	Write only if changed, then quit.	A particularly nice everyday habit.
`ZZ`	Normal-mode equivalent of write if changed, then quit.	No colon required.
`:e filename`	Edit a file.	Open or switch to a named file.
`:e!`	Reload file and discard unsaved changes.	Useful when you want a clean reset.

11. Windows, buffers, and tabs

These are not the same thing. A buffer is the file in memory; a window is a view; a tab is a layout container.

Windows and splits

Command	Meaning	Notes
`:sp`	Open a horizontal split.	Alias for :split.

`:vs`	Open a vertical split.	Alias for :vsplit.
`Ctrl-w h/j/k/l`	Move among windows.	Left, down, up, right.
`Ctrl-w w`	Cycle to the next window.	Simple next-window command.
`Ctrl-w q`	Close the current window.	Equivalent to quitting that split.

Buffers and tabs

Command	Meaning	Notes
`:ls`	List buffers.	Shows open buffers.
`:bn`	Go to next buffer.	Short for :bnext.
`:bp`	Go to previous buffer.	Short for :bprevious.
`:bd`	Delete the current buffer.	Close it from the buffer list.
`:b 3`	Go to buffer 3.	Use a specific buffer number.
`:tabnew`	Open a new tab.	New tab page.
`gt / gT`	Next tab / previous tab.	Tab navigation.

12. Useful toggles and view commands

These are practical utility commands that make the editor more readable and easier to inspect.

Line numbers and basic view toggles

Command	Meaning	Notes
`:set nu`	Show line numbers.	Short for :set number.
`:set nonu`	Hide line numbers.	Short for :set nonumber.
`:set number!`	Toggle line numbers.	Great utility command.
`:set number?`	Show whether line numbers are enabled.	State query.
`:set rnu`	Show relative line numbers.	Short for :set relativenumber.
`:set nornu`	Hide relative line numbers.	Short for :set norelativenumber.
`:set wrap`	Enable line wrapping.	Useful for prose.
`:set nowrap`	Disable line wrapping.	Common in code.

`:set list`	Show normally invisible characters.	Good for tabs and trailing spaces.
`:set nolist`	Hide list characters.	Return to normal display.

13. Help, shell access, and command-line history

The built-in help system is excellent. Do not ignore it.

Help and shell

Command	Meaning	Notes
`:help`	Open help.	Broad entry point.
`:help topic`	Open help on a topic.	Example: :help motion.txt or :help i_CTRL-R.
`:!command`	Run a shell command.	Example: :!ls or :!make.
`q:`	Open command-line history window.	Lets you edit and rerun recent : commands.
`q/`	Open search history window.	History for / searches.

14. Common operations

Short task-oriented patterns that use multiple Vim steps without trying to be exhaustive.

Common operations

Operation	Steps	Notes
Copy a few lines and paste elsewhere	Move to the first line. Press V to start linewise selection. Move with j or k to include the lines you want. Press y to yank. Navigate to the destination. Press p to paste below or P to paste above.	For exact characters instead of full lines, use v instead of V.
Cut a few lines and move them elsewhere	Move to the first line. Press V. Extend the selection with j or k. Press d to delete the selection into the unnamed register. Navigate to the destination. Press p or P to paste.	If you already know it is exactly one line, dd then p is faster.

Copy or cut part of a line	Place the cursor at the start. Press v for characterwise Visual mode. Move to expand the selection. Press y to copy or d to cut. Move to the destination and paste with p or P.	This is the closest equivalent to ordinary editor selection.
Replace text inside quotes or parentheses	Put the cursor anywhere inside the quoted text or parentheses. Use ci" to replace inside double quotes, or di(to delete inside parentheses.	Text objects are one of the biggest productivity gains in Vim.
Search for a word and step through matches	Place the cursor on the word and press *. Use n for the next match and N for the previous match. Press :noh when the highlight becomes distracting.	This is often faster than typing a full search pattern.
Save and exit cleanly	Use :x when you want to write the file only if it changed, then quit.	ZZ is the Normal-mode equivalent.

Appendix A. Quick lookup index

Alphabetized quick lookup by command or concept.

Lookup table

Command	Meaning	Notes
`:e`	Edit a file	Section 10
`:e!`	Reload and discard unsaved changes	Section 10
`:help`	Open help	Section 13
`:ls`	List buffers	Section 11
`:noh`	Clear search highlight	Section 8
`:set nu`	Turn on line numbers	Section 12
`:set number!`	Toggle line numbers	Section 12
`:set rnu`	Turn on relative numbers	Section 12
`:sp / :vs`	Horizontal / vertical split	Section 11
`:w / :x / :wq`	Save and quit family	Section 10
`*`	Search word under cursor forward	Section 8
`.`	Repeat last change	Section 6
`Ctrl-o / Ctrl-i`	Jump list navigation	Section 4
`ci"`	Change inside quotes	Section 9
`ciw`	Change inner word	Section 9
`dd`	Delete line	Section 6
`dw / cw`	Delete or change by word motion	Section 6
`gg / G`	Top or bottom of file	Section 4
`gt / gT`	Next or previous tab	Section 11
`i / a / o / O`	Insert and open-line commands	Section 5
`n / N`	Repeat search	Section 8
`p / P`	Paste after or before	Section 6
`u / Ctrl-r`	Undo and redo	Section 6
`v / V / Ctrl-v`	Visual modes	Section 7
`w / b / e`	Word movement	Section 4
`x / yy`	Delete character / yank line	Section 6

Appendix B. BSD 3-Clause License

Full license text included for book-style front matter and redistribution clarity.

BSD 3-Clause License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS 'AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.