Vim quick reference

by Marc Penninga *version* 2024-06-05

Moving around

`gg`, `G`	Go to top/end of file (with prefix *n:* go to line *n*)
`^e`, `^d`, `^f`	Scroll down one line/half a screen/full screen
`^y`, `^u`, `^b`	Scroll up one line/half a screen/full screen
`H`, `M`, `L`	Go to top/middle/bottom of screen
`zt`, `z<Enter>`	Scroll current line to top of screen
`zz`, `z.`	Scroll current line to center of screen
`zb`, `z-`	Scroll current line to bottom of screen
	(prefix *n:* scroll line *n* to top/center/bottom)
`^o`, `^i`	Go to location of last/next change or jump

Marks

`mx`	Set mark `x` (`a` – `z` per buffer, `A` – `Z` across buffers)
`:marks`	List all current marks
`'x`	Jump to (begin of line containing) mark `x`
`` `x ``	Jump to exact location of mark `x`
`''`	Jump back to start of last jump

Insert mode

An optional prefixed count indicates how many characters or lines to insert or replace.

`i`, `a`	Insert at/after cursor position
`gI`, `I`, `A`	Insert at start/begin/end of line
`o`, `O`	Insert new line after/before current line
`s`	Delete char(s) and start Insert mode
`C`	Delete rest of line, start Insert mode
`S`	Delete entire line, start Insert mode
`<esc>`, `^[`	Return to Normal mode

Insert mode commands

`^d`, `^t`	Shift left/right (by shiftwidth)
`^k xy`	Insert accented characters (see `:digraphs` for full list)
	`a`` = à, `a'` = á, `a:` = ä, `c,` = ç, `n~` = ñ, `ss` = ß, `>>` = »
`^v k`	Key *k* (e.g., <Esc>, <Enter>) is included verbatim.
	If *k* = `o777`, `999`, `xFF`, `uFFFF`, `UFFFFFFFF`: Unicode char
`^o`	Execute single command, then return to Insert mode
`^r r`	Insert contents of register `r` (see *Registers*)
`^a`, `^@`	Repeat previous insert (`^@` terminates Insert mode)

Text editing

General editing actions are of the form:

```
[register] operator [count] operand
```

"Operand" may be a motion, text object, visual selection, search or mark. As a shortcut, operators can be doubled to work on full lines: `cc`, `dd`, `yy`, `<<`, `>>`, `==`, `guu`, `gUU`, `g~~` etc. Naming a specific register with operators `d`, `y`, `p` yanks to or pastes from that register.

Operators

`d`	Delete text
`y`	Copy ("yank") text
`c`	Change text
`g~`	Swap case
`gu`, `gU`	Make lower/uppercase
`<`, `>`, `=`	Shift left/right (by `shiftwidth`), autoindent
`gq`, `gw`	Format text; `gq` uses `formatexpr`/`formatprg`

Motions

Motions move the cursor when used without operator.

`h`, `j`, `k`, `l`	Left, down, up, right (`gj`, `gk` move by *screen* line)
`0`, `^`, `$`	Go to start/begin/end of line
	("begin" is first non-space character of line)
`w`, `W`	Go to next word/WORD ("WORDS" delimited only by whitespace; "words" also by punctuation)
`b`, `B`	Go to previous word/WORD
`e`, `ge`, `E`, `gE`	Go to end of next/previous word/WORD
`%`	Go to matching parenthesis/brace
`(`, `)`	Go to beginning of current/next sentence
`{`, `}`	Go to beginning of current/next paragraph

Text objects

These `a` commands include surrounding whitespace; alternative `i` forms don't. As a general rule, use `a` forms with delete, `i` forms with copy and change (mnemonic: *around* versus *inside*).

`aw`, `aW`, `as`, `ap`	Word, WORD, sentence, paragraph
`a)`, `a]`, `a}`, `a>`	Block in `()`, `[]`, `{}`, `<>` (`ab`, `aB` same as `a)`, `a}`)
`a"`, `a'`, `a``, `at`	Various strings; HTML/XML tag

Visual mode

Use motions, search, text objects or marks to select text.

`v`, `V`, `^v`	Select characters/lines/block

Search and replace

`fx`, `tx`, `Fx`, `Tx`	Find/move to next/previous `x` on current line
`;` `,`	Repeat find in same/oposite direction
`/pat`, `?pat`	Search next/prev occurrence of pattern
`*`, `#`	Search next/prev occurrence of current word
`n`, `N`	Repeat search in same/opposite direction
`:s/old/new/`	Change first `old` on current line to `new`
	Flags: `g` = change all; `c` = confirm;
	`e` = no error if `old` not found
`:%s/old/new/g`	Change all `old` in current buffer to `new`

Miscellaneous other editing commands

These command may also be prefixed with a count.

`C`, `D`	Change/delete rest of current line (same as `c$`, `d$`)
`S`, `Y`	Substitute (change)/yank *entire* current line
`p`, `P`	Paste yanked text after/before cursor
`]p`, `[P`	Same as `p`/`P`, but match current indentation
`x`, `X`	Delete character under/before cursor
`rx`	Replace character under cursor with `x`
`R`	Replace mode (overwrite existing text until `<Esc>`)
`~`	Change case of character under cursor
`J`, `gJ`	Join current and next lines, with/without space
`^a`, `^x`	Increment/decrement number under cursor

Repeat and undo

`.`	Repeat last command
`u`	Undo last operation
`^r`	Redo "undone" operation
`U`	Undo all edits on current line (when cursor still on line)

Registers and macros

`""`	"Unnamed" (default) register
`"a` – `"z`	Registers; use with `d`, `y`, `p` (`"A` – `"Z` *append* to `a` – `z`)
`"0`	Contains last yanked text
`"1` – `"9`	Stack containing "big" deletes (one or more lines)
`"-`	Last "small" delete (only part of line)
`"+`	System clipboard (if supported; check `:version`)
`qr`, `qR`	Record operations into register `r` (`qR`: *append* to `r`)
`q`	Quit recording
`@r`	Replay operations from register `r`
`@@`	Repeat last `@r` (with optional prefixed repeat count)
`:reg`	Show all current registers

Editor commands

`:h subject`	Show help text on *subject*
`:w`, `:saveas`	Save file; takes optional filename
`:w >> filename`	Append text to file
`:q`	Quit editor
`:wq`, `:x`, `ZZ`	Save file and quit editor
`:q!`, `ZQ`	Quit editor without saving changes
`:e filename`	Edit another file (in current buffer)
`:n`, `:N`, `:prev`	Edit next/previous file in arg list
`:wn`, `:wN`, `:wprev`	Save file and edit next/prev in arg list
`:! command`	Execute shell command
`:r file`	Insert file contents after current line
`:r !command`	Insert command output after current line
`:0r file/!cmd`	Insert file or command at start of buffer
`:$r file/!cmd`	Insert file or command at end of buffer
`^r^w`	Copy word under cursor to command
`^f`	Edit command line in normal mode
`^d`	Completion suggestions

Windows and tabs

In all commands below, `^w x` may also be typed as `^w^x`.

`:split`, `^w s`	Split window horizontally
`:vsplit`, `^w v`	Split window vertically (`:split` and `:vsplit` optionally take filename or `+command`, e.g. `split +term`)
`:new`, `^w n`	Split horizontally; new buffer is empty
`:vnew`	Split vertically; new buffer is empty
`^w hjkl`	Move to window left/below/above/right (with *tmux-navigator.vim:* also `^hjkl`)
`^w w`	Cycle through open windows
`^w HJKL`	Move window left/down/up/right
`^w rR`	Rotate windows clockwise/countercw.
`^w +-<>`	Increase/decrease window height/width
`^w =`	Resize windows to equal size
`^w q`, `:close`	Quit window (`:close` fails if only buffer)
`:only`, `^w o`	Close all other windows
`:term [cmd]`	Open terminal and run `cmd` (default: shell)
`^\ n`	Switch to Terminal-Normal mode
`:tabnew`	Open new tab; takes optional filename
`^w T`	Move current window to new tab
`:tabclose`	Close current tab
`:tabonly`	Close all other tabs
`^<PageDown>`, `gt`	Move to next tab
`^<PageUp>`, `gT`	Move to previous tab

From my *.vimrc*

```
set nocompatible    " allow non-vi-compatible stuff
set encoding=utf-8
set fileformat=unix
filetype plugin on
syntax enable
set autoindent      " copy indent from prev line
filetype indent on  " auto-indent based on filetype
set tabstop=4       " for showing existing tabs
set softtabstop=4   " used when editing
set expandtab       " expand tabs to spaces ...
au BufRead,BufFilePre,BufNewFile Makefile* \
  set noexpandtab   " ... except in Makefiles
set shiftwidth=4    " used when shifting text
set shiftround      " shift to multiple of sw
set smarttab        " auto-indent by sw
set number          " show line numbers
set laststatus=2    " show status line
set showmatch       " show matching quote/paren
set incsearch       " show matches while typing
set hlsearch        " highlight all matches
set virtualedit=block  " select beyond EOL
set splitbelow      " open new windows below ...
set splitright      " ... or to the right

fun! RemoveTrailingWhitespace()
    let l:save = winsaveview()
    keeppatterns %s/\s\+$//e
    call winrestview(l:save)
endfun
au BufWritePre * :call RemoveTrailingWhitespace()

set ttimeoutlen=10          " fix timeout after Esc
tnoremap <Esc> <C-\><C-n>   " Terminal-Normal mode

" Use 24-bit colours and true italics
if exists('+termguicolors')
    let &t_8f = "\<Esc>[38;2;%lu;%lu;%lum"
    let &t_8b = "\<Esc>[48;2;%lu;%lu;%lum"
    set termguicolors
endif
highlight Comment cterm=italic
```

My favorite plugins

Plugins go in subdirs of `$HOME/.vim/pack/packages/start/`; run `:helptags ALL` to generate help texts.

autopairs.vim	Add/delete quotes, parens in pairs
commentary.vim	`gc` operator to (un)comment stuff
lightline.vim	Configurable statusline
gitbranch.vim	Show branchname in status line
python-pep8-indent.vim	Fix auto-indentation for Python
tmux-navigator.vim	Switch between vim and tmux
nord.vim	My favourite dark color scheme

surround.vim

`ys arg1 arg2`	Quote or parenthesize something.
	`arg1` is a motion, text object etc.,
	`arg2` chooses quotes/parens: `" ] ) } > <`
	Args `[ ( {` add space inside parens.
	`ysM>` wraps `M` in `<` and `>`
	`ysM<` wraps `M` in `<...>` and `</...>` tags (prompts for tag name); same as `t`
	`ysMf` wraps `M` in parentheses and prepends function call (prompts for name); `F` adds extra space inside parentheses
	`ysM\` wraps `M` in LaTeX `\begin-\end` block (prompts for environment name)
	`ysM*` wraps `M` in `*` (like `*M*`); also `_`, `|`, `#` etc.

`S arg2`	Use in Visual mode; `arg1` is implicit
`cs old new`	Change surrouding quotes or parens: `cs'"` changes single to double quotes `cs)}` changes parentheses to curly braces `cs)(` adds space inside parentheses
`ds arg`	Delete surrounding quotes, parens

fugitive.vim

`:G`	Show Git status. Key maps (see also `g?`): `gu`, `gU`, `gs` to (*n*th) untracked/unstaged/staged file `<`, `>`, `=` show/hide/toggle in-line diff `(`, `)` to previous/next file or chunk `s`, `u` stage/unstage current file or chunk `cc` commit `ca`, `ce`, `cw` amend/--no-edit/reword last commit `gq` quit status buffer
`:G cmd`	Run `git cmd`