Neovim Lua-scriptable modal editor

START HERE

:checkhealth
Diagnose setup problems

g › **c** › **c**
Toggle line comment

K
Hover docs (LSP)

Ctrl + **** › **Ctrl** + **N**
Leave terminal mode

g › **r** › **n**
Rename symbol (LSP)

g › **r** › **a**
Code action (LSP)

g › **r** › **r**
Find references (LSP)

Ctrl + **W** › **d**
Diagnostics under cursor

CONCEPTS

Neovim keeps Vim's modal editing (see the vim sheet). This sheet covers what Neovim adds: Lua config, built-in LSP, terminal, health checks.

Config lives in ~/.config/nvim/init.lua (init.vim also works). Lua modules go in lua/ next to it and load with require('name').

Server binary installed separately
↓ defines
vim.lsp.config cmd, filetypes, roots
↓ defines
vim.lsp.enable starts on matching buffer
↓ attaches
Defaults + LspAttach gr* keys, K, diagnostics
↓ gives you
LSP in Neovim: no plugin needed for the client

Neovim ships only the LSP client. Language servers (lua-language-server, pyright, gopls...) are separate programs on your PATH.

Run :checkhealth first when something misbehaves: it reports missing providers, tools and LSP problems.

DIFFERENCES FROM VIM

Y Yank to end of line
Neovim maps Y to y$, unlike Vim where it equals yy.

g › **c** › **c**
Toggle line comment

g › **c** › *<motion>*
Toggle comment over motion

g › **O**
Outline of symbols / headings

] › **d** / **[** › **d**
Next / previous diagnostic

] › **q** / **[** › **q**
Next / previous quickfix item

] › **b** / **[** › **b**
Next / previous buffer

***** / **#**
Search selected text (visual mode)
In visual mode these search the selection instead of the word.

a › **n** / **i** › **n**
Grow visual selection by node

LUA CONFIG

:lua › *<code>*
Run a Lua expression

:= › *<expr>* Print a Lua value

:source › **%**
Reload the current file

:messages
Show earlier messages and errors

LUA CONFIG (cont.)

:set › **<opt>?**
Show an option's value

:verbose › **set** › **<opt>?**
Where an option was set

:map / **:nmap** List mappings

:verbose › **map** › *<lhs>*
Who defined a mapping

:Inspect
Highlight groups under cursor

:help › *<topic>*
Open help for a topic

:help › **lua-guide**
Lua in Neovim tutorial

:Tutor
Start the interactive tutorial

Lua equivalents

Vimscript	Lua
`set number`	`vim.o.number = true`
`let g:x = 1`	`vim.g.x = 1`
`nnoremap a b`	`vim.keymap.set()`
`autocmd ...`	`nvim_create_autocmd()`
`command! ...`	`nvim_create_user_command`

Basic init.lua

```
vim.g.mapleader = ' '
vim.opt.number = true
vim.opt.ignorecase = true
vim.opt.smartcase = true
vim.keymap.set('n',
'<leader>w', '<cmd>write<cr>',
{ desc = 'Save' })
```

LSP

K Hover documentation

g › **r** › **n** Rename symbol

LSP (cont.)

g › **r** › **a** Code action

g › **r** › **r** List references

g › **r** › **i**
Go to implementation

g › **r** › **t**
Go to type definition

Ctrl + **]** Go to definition
Works via tagfunc once a server is attached; Ctrl+T jumps back.

Ctrl + **S**
Signature help (insert mode)

Ctrl + **X** › **Ctrl** + **O**
LSP completion (insert mode)

:lsp › **enable** › *<name>*
Start a configured server

:checkhealth › **vim.lsp**
Check LSP clients and logs

:lua › **vim.lsp.buf.format()**
Format the buffer

:lua › **vim.lsp.buf.definition()**
Go to definition

The gr* keys exist only after a server attaches. Without one, g r just waits for the next key.

Minimal LSP setup

```
vim.lsp.config('lua_ls', {
  cmd = { 'lua-language-
server' },
  filetypes = { 'lua' },
  root_markers =
{ '.luarc.json', '.git' },
})
vim.lsp.enable('lua_ls')
```

DIAGNOSTICS

Ctrl + **W** › **d**
Show diagnostic in float

[› **D** / **]** › **D**
First / last diagnostic

TERMINAL

:terminal
Open a shell in a buffer

:terminal › *<cmd>*
Run a command in a terminal buffer

i / **a**
Enter terminal mode from normal

Ctrl + **** › **Ctrl** + **N**
Back to normal mode

Ctrl + **** › **Ctrl** + **O**
One normal command, then back

:split › **|** › **terminal**
Terminal in a split

:tnoremap › *<Esc>* › *<C-\><C-n>*
Make Esc leave terminal mode

In terminal mode every key goes to the program except Ctrl+\. Use :terminal instead of :! for interactive commands.

WINDOWS & BUFFERS

Ctrl + **W** › **s** / **Ctrl** + **W** › **v**
Split horizontally / vertically

WINDOWS & BUFFERS (cont.)

Ctrl + **W** › **h** / **j** / **k** / **l**
Move between windows

Ctrl + **W** › **q** Close window

Ctrl + **W** › **o**
Keep only this window

:ls List buffers

:bnext / **:bprev**
Next / previous buffer

:b › *<name>*
Switch to buffer by name

:bd Delete (close) buffer

PLUGINS & HEALTH

:checkhealth
Run all health checks

:checkhealth › *<name>*
Check one component

:lua › **vim.pack.add(<specs>)**
Install plugins (built-in manager)
vim.pack is new in recent releases; older versions need a plugin manager such as lazy.nvim.

:echo › **stdpath('config')**
Print the config directory

nvim › **--clean**
Start without config or plugins

Bisecting a bug: start with nvim --clean; if it vanishes, the cause is in your config or plugins.