

# Neovim Plugin Project Structure

Modern Lua-based Neovim plugin with lazy loading support, telescope integration, and proper module structure.

#neovim #lua #vim #plugin #editor

PNG PDF Copy Prompt

## Project Directory

my-plugin.nvim/

>

lua/

Lua source code

>

my-plugin/

Plugin namespace

init.lua

Main entry, set...

config.lua

Default configu...

commands.lua

User commands

keymaps.lua

Default keybind...

>

core/

Core functional...

init.lua

utils.lua

actions.lua

>

ui/

UI components

init.lua

window.lua

Float windows

picker.lua

Telescope picker

health.lua

:checkhealth su...

>

plugin/

Auto-loaded on ...

my-plugin.lua

Commands and au...

>

after/

Loaded after pl...

>

plugin/

my-plugin.lua

>

doc/

Vim help docs

my-plugin.txt

:help my-plugin

tags

Generated by :h...

>

tests/

Plenary tests

minimal\_init.lua

Test config

>

my-plugin/

core\_spec.lua

config\_spec.lua

README.md

LICENSE

.gitignore

.luacheckrc

Linter config

stylua.toml

Formatter config

## Why This Structure?

Modern Neovim plugins use Lua for performance and access to the full Neovim API. The `lua/plugin-name/` structure enables `require('plugin-name')`. The `plugin/` folder auto-loads commands, while lazy.nvim users can defer loading until needed.

## Key Directories

**lua/my-plugin/** - Main plugin code, loaded via `require()`  
**init.lua** - Entry point with `setup()` function  
**plugin/** - Auto-loaded commands and autocommands  
**doc/** - Help documentation (`:help my-plugin`)

## Setup Pattern

```
-- lua/my-plugin/init.lua
local M = {}

M.config = {
  option_one = true,
  keymaps = { toggle = "mp" },
}

function M.setup(opts)
  M.config = vim.tbl_deep_extend("force", M.config, opts or {})
  require("my-plugin.commands").setup()
  require("my-plugin.keymaps").setup(M.config.keymaps)
end

function M.toggle()
  -- Plugin functionality
end

return M
```

## Getting Started

- Create plugin directory with `-nvim` suffix
- Add to Neovim with lazy.nvim or packer
- Implement `setup()` function in `lua/plugin/init.lua`
- Write help docs in `doc/plugin.txt`
- Run `:helptags doc/` to generate tags

## When To Use This

- Adding new functionality to Neovim
- Creating custom UI components
- Integrating external tools with Neovim
- Building workflow automation
- Language-specific tooling

## Plugin Manager Integration

**Lazy loading** - Use `cmd`, `keys`, or `event` in lazy.nvim spec  
**Dependencies** - Declare plenary, telescope as dependencies  
**Config function** - Users call `require('plugin').setup({})`

## Best Practices

- Use `vim.tbl_deep_extend` for config merging
- Prefix autocommand groups with plugin name
- Support `:checkhealth` for dependency checking
- Use `vim.notify` for user messages
- Provide sensible defaults, make everything optional

## Trade-offs

**Lua only** - No Vimscript, requires Neovim 0.5+  
**API changes** - Neovim API evolves, check compatibility  
**Testing setup** - Plenary test framework requires configuration

## Naming Conventions

**Repository** - plugin-name.nvim suffix  
**Lua modules** - lowercase with hyphens  
**Functions** - snake\_case (setup, toggle\_feature)  
**Commands** - PascalCase (MyPluginToggle)