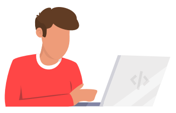


How I Setup Neovim On My Mac To Make it AMAZING in 2024

Published: April 3, 2024

You can find the source code for my config here.

Open a terminal window

Open a terminal window on your mac. You will need a true color terminal for the colorscheme to work properly.

I'm using *iTerm2*

Install Homebrew

Run the following command:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/H    eb
```

If necessary, when prompted, enter your password here and press enter. If you haven't installed the XCode Command Line Tools, when prompted, press enter and homebrew will install this as well.

Add To Path (Only Apple Silicon Macbooks)

After installing, add it to the path. This step shouldn't be necessary on Intel macs.

Run the following two commands to do so:

```
echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> ~/.zprof
eval "$(/opt/homebrew/bin/brew shellenv)"
```

Install iTerm2 If Necessary

If you don't have a true color terminal, install iTerm2 with homebrew:

```
brew install --cask iterm2
```

Then switch to this terminal.

Install A Nerd Font

I use Meslo Nerd Font. To install it do:

```
brew tap homebrew/cask-fonts
```

And then do:

```
brew install font-meslo-lg-nerd-font
```

Then open iTerm2 settings with `CMD+,` and under **Profiles > Text** change the font to **MesloLGS Nerd Font Mono**

Install Neovim

Run:

```
brew install neovim
```

Install Ripgrep

Run:

```
brew install ripgrep
```

Install Node

Run:

```
brew install node
```

Setup Initial File Structure

Your config will be located in `~/.config/nvim`.

Let's setup the initial file structure with the following commands:

Make the nvim config directory.

```
mkdir -p ~/.config/nvim
```

`-p` is used to also create parent directories if they don't already exist

Move to this directory:

```
cd ~/.config/nvim
```

Create main `init.lua` file:

```
touch init.lua
```

Create `lua/josean/core` directories:

Any time I use "josean" you can replace this with your name

```
mkdir -p lua/josean/core
```

Create plugins directory (will have all of the plugin configs/specs):

```
mkdir -p lua/josean/plugins
```

Create `lazy.lua` file (will be used to setup/configure lazy.nvim plugin manager):

```
touch lua/josean/lazy.lua
```

Setup core options

Make sure you're in `~/.config/nvim` and open the config:

```
nvim .
```

Navigate to the core folder and press `%` to create a file and call it: "options.lua"

In this file add:

```
vim.cmd("let g:netrw_liststyle = 3")
```

Open the explorer with `:Explore` and navigate to the main `init.lua` file.

Add the following to load the basic options on startup:

```
require("josean.core.options")
```

Close Neovim with `:w` and reopen it with `nvim .`

Go back to "options.lua" and add the following to setup the rest of the options:

```
local opt = vim.opt -- for conciseness

-- line numbers
opt.relativenumber = true -- show relative line numbers
opt.number = true -- shows absolute line number on cursor line (

-- tabs & indentation
```

```lua
opt.tabstop = 2 -- 2 spaces for tabs (prettier default)
opt.shiftwidth = 2 -- 2 spaces for indent width
opt.expandtab = true -- expand tab to spaces
opt.autoindent = true -- copy indent from current line when star

-- line wrapping
opt.wrap = false -- disable line wrapping

-- search settings
opt.ignorecase = true -- ignore case when searching
opt.smartcase = true -- if you include mixed case in your search

-- cursor line
opt.cursorline = true -- highlight the current cursor line

-- appearance

-- turn on termguicolors for nightfly colorscheme to work
-- (have to use iterm2 or any other true color terminal)
opt.termguicolors = true
opt.background = "dark" -- colorschemes that can be light or dar
opt.signcolumn = "yes" -- show sign column so that text doesn't

-- backspace
opt.backspace = "indent,eol,start" -- allow backspace on indent,

-- clipboard
opt.clipboard:append("unnamedplus") -- use system clipboard as d

-- split windows
```

```lua
opt.splitright = true -- split vertical window to the right
opt.splitbelow = true -- split horizontal window to the bottom

-- turn off swapfile
opt.swapfile = false
```

Do `:e lua/josean/core/init.lua`

Add the following:

```lua
require("josean.core.options")
```

Open the explorer with `:Explore` and go to the main init.lua file and change the require to:

```lua
require("josean.core")
```

Setup core keymaps

Do `:e lua/josean/core/keymaps.lua`

And add the following to this file:

```lua
-- set leader key to space
vim.g.mapleader = " "
```

```

local keymap = vim.keymap -- for conciseness

---------------------
-- General Keymaps -------------------

-- use jk to exit insert mode
keymap.set("i", "jk", "<ESC>", { desc = "Exit insert mode with j

-- clear search highlights
keymap.set("n", "<leader>nh", ":nohl<CR>", { desc = "Clear searc

-- delete single character without copying into register
-- keymap.set("n", "x", '"_x')

-- increment/decrement numbers
keymap.set("n", "<leader>+", "<C-a>", { desc = "Increment number
keymap.set("n", "<leader>-", "<C-x>", { desc = "Decrement number

-- window management
keymap.set("n", "<leader>sv", "<C-w>v", { desc = "Split window v
keymap.set("n", "<leader>sh", "<C-w>s", { desc = "Split window h
keymap.set("n", "<leader>se", "<C-w>=", { desc = "Make splits eq
keymap.set("n", "<leader>sx", "<cmd>close<CR>", { desc = "Close 

keymap.set("n", "<leader>to", "<cmd>tabnew<CR>", { desc = "Open 
keymap.set("n", "<leader>tx", "<cmd>tabclose<CR>", { desc = "Clo
keymap.set("n", "<leader>tn", "<cmd>tabn<CR>", { desc = "Go to n
keymap.set("n", "<leader>tp", "<cmd>tabp<CR>", { desc = "Go to p
keymap.set("n", "<leader>tf", "<cmd>tabnew %<CR>", { desc = "Ope
```

Open the explorer with `:Explore`, open `lua/josean/core/init.lua` and add the following:

```lua
require("josean.core.options")
require("josean.core.keymaps")
```

Exit with `:q` and reenter Neovim with `nvim .`

Setup lazy.nvim

Go to “lazy.lua” and add the following to bootstrap lazy.nvim

```lua
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim
if not vim.loop.fs_stat(lazypath) then
  vim.fn.system({
    "git",
    "clone",
    "--filter=blob:none",
    "https://github.com/folke/lazy.nvim.git",
    "--branch=stable", -- latest stable release
    lazypath,
  })
end
vim.opt.rtp:prepend(lazypath)
```

Then configure lazy.nvim with the following:

```lua
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim"
if not vim.loop.fs_stat(lazypath) then
  vim.fn.system({
    "git",
    "clone",
    "--filter=blob:none",
    "https://github.com/folke/lazy.nvim.git",
    "--branch=stable", -- latest stable release
    lazypath,
  })
end
vim.opt.rtp:prepend(lazypath)

require("lazy").setup("josean.plugins")
```

If you're using your name instead of "josean", change that to your name here as well

Then open the explorer with `:Explore` and navigate to main `init.lua` file.

Add the following to it:

```lua
require("josean.core")
require("josean.lazy")
```

Exit with `:q` and reenter Neovim with `nvim`

You can see the lazy.nvim UI now with `:Lazy` and you can close the UI with `q`

Install plenary & vim-tmux-navigator

Do `:e lua/josean/plugins/init.lua`

Add the following to install plenary and vim-tmux-navigator:

```lua
return {
  "nvim-lua/plenary.nvim", -- lua functions that many plugins us
  "christoomey/vim-tmux-navigator", -- tmux & split window navig
}
```

After adding this, save the file and you can install manually by doing `:Lazy`, then typing `I`.

After install, close the UI with `q` and you can manually load a plugin with `:Lazy reload vim-tmux-navigator` for example.

Otherwise, you can also exit with `:q` and reenter Neovim with `nvim .` and it'll happen automatically.

Install & configure tokyonight colorscheme

Do `:e lua/josean/plugins/colorscheme.lua`

In this file add the following:

```lua
return {
  {
    "folke/tokyonight.nvim",
    priority = 1000, -- make sure to load this before all the ot
    config = function()
      local bg = "#011628"
      local bg_dark = "#011423"
      local bg_highlight = "#143652"
      local bg_search = "#0A64AC"
      local bg_visual = "#275378"
      local fg = "#CBE0F0"
      local fg_dark = "#B4D0E9"
      local fg_gutter = "#627E97"
      local border = "#547998"

      require("tokyonight").setup({
        style = "night",
        on_colors = function(colors)
          colors.bg = bg
          colors.bg_dark = bg_dark
          colors.bg_float = bg_dark
          colors.bg_highlight = bg_highlight
          colors.bg_popup = bg_dark
          colors.bg_search = bg_search
          colors.bg_sidebar = bg_dark
          colors.bg_statusline = bg_dark
          colors.bg_visual = bg_visual
          colors.border = border
          colors.fg = fg
          colors.fg_dark = fg_dark
          colors.fg_float = fg
          colors.fg_gutter = fg_gutter
          colors.fg_sidebar = fg_dark
        end,
      })
      -- load the colorscheme here
      vim.cmd([[colorscheme tokyonight]])
    end,
  },
}
```

This will setup **tokyonight** as the colorscheme and modify some of its colors according to my preference.

Exit with `:q` and reenter Neovim with `nvim .`

Setup nvim-tree file explorer

Do `:e lua/josean/plugins/nvim-tree.lua`

Add the following to this file:

```
return {
  "nvim-tree/nvim-tree.lua",
  dependencies = "nvim-tree/nvim-web-devicons",
  config = function()
    local nvimtree = require("nvim-tree")
```

```lua

-- recommended settings from nvim-tree documentation
vim.g.loaded_netrw = 1
vim.g.loaded_netrwPlugin = 1

nvimtree.setup({
  view = {
    width = 35,
    relativenumber = true,
  },
  -- change folder arrow icons
  renderer = {
    indent_markers = {
      enable = true,
    },
    icons = {
      glyphs = {
        folder = {
          arrow_closed = " ", -- arrow when folder is closed
          arrow_open = " ", -- arrow when folder is open
        },
      },
    },
  },
  -- disable window_picker for
  -- explorer to work well with
  -- window splits
  actions = {
    open_file = {
      window_picker = {
              enable = false,
            },
          },
        },
        filters = {
          custom = { ".DS_Store" },
        },
        git = {
          ignore = false,
        },
      })

      -- set keymaps
      local keymap = vim.keymap -- for conciseness

      keymap.set("n", "<leader>ee", "<cmd>NvimTreeToggle<CR>", { des
      keymap.set("n", "<leader>ef", "<cmd>NvimTreeFindFileToggle<CR>
      keymap.set("n", "<leader>ec", "<cmd>NvimTreeCollapse<CR>", { d
      keymap.set("n", "<leader>er", "<cmd>NvimTreeRefresh<CR>", { de
    end
  }
```

Exit with `:q` and reenter Neovim with `nvim`

Setup which-key

Which-key is great for seeing what keymaps you can use.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `which-key.lua`

Add this to the file:

```lua
return {
  "folke/which-key.nvim",
  event = "VeryLazy",
  init = function()
    vim.o.timeout = true
    vim.o.timeoutlen = 500
  end,
  opts = {
    -- your configuration comes here
    -- or leave it empty to use the default settings
    -- refer to the configuration section below
  },
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup telescope fuzzy finder

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `telescope.lua`

Add this to the file:

```
return {
  "nvim-telescope/telescope.nvim",
  branch = "0.1.x",
  dependencies = {
    "nvim-lua/plenary.nvim",
    { "nvim-telescope/telescope-fzf-native.nvim", build = "make"
    "nvim-tree/nvim-web-devicons",
  },
  config = function()
    local telescope = require("telescope")
    local actions = require("telescope.actions")

    telescope.setup({
      defaults = {
        path_display = { "smart" },
        mappings = {
          i = {
            ["<C-k>"] = actions.move_selection_previous, -- move
            ["<C-j>"] = actions.move_selection_next, -- move to
            ["<C-q>"] = actions.send_selected_to_qflist + action
          },
        },
      },
    })

    telescope.load_extension("fzf")

    -- set keymaps
```

```
      local keymap = vim.keymap -- for conciseness

      keymap.set("n", "<leader>ff", "<cmd>Telescope find_files<cr>
      keymap.set("n", "<leader>fr", "<cmd>Telescope oldfiles<cr>",
      keymap.set("n", "<leader>fs", "<cmd>Telescope live_grep<cr>"
      keymap.set("n", "<leader>fc", "<cmd>Telescope grep_string<cr
    end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup a greeter

We're gonna setup a greeter for Neovim startup with alpha-nvim

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `alpha.lua`

Add the following code:

```
return {
  "goolord/alpha-nvim",
  event = "VimEnter",
  config = function()
    local alpha = require("alpha")
    local dashboard = require("alpha.themes.dashboard")
```

```lua

    -- Set header
    dashboard.section.header.val = {
      "                                                     ",
      "  ███╗   ██╗███████╗ ██████╗ ██╗   ██╗██╗███╗   ███╗ ",
      "  ████╗  ██║██╔════╝██╔═══██╗██║   ██║██║████╗ ████║ ",
      "  ██╔██╗ ██║█████╗  ██║   ██║██║   ██║██║██╔████╔██║ ",
      "  ██║╚██╗██║██╔══╝  ██║   ██║╚██╗ ██╔╝██║██║╚██╔╝██║ ",
      "  ██║ ╚████║███████╗╚██████╔╝ ╚████╔╝ ██║██║ ╚═╝ ██║ ",
      "  ╚═╝  ╚═══╝╚══════╝ ╚═════╝   ╚═══╝  ╚═╝╚═╝     ╚═╝ ",
      "                                                     ",
    }

    -- Set menu
    dashboard.section.buttons.val = {
      dashboard.button("e", "   > New File", "<cmd>ene<CR>"),
      dashboard.button("SPC ee", "   > Toggle file explorer", "<
      dashboard.button("SPC ff", "  > Find File", "<cmd>Telescop
      dashboard.button("SPC fs", "   > Find Word", "<cmd>Telesco
      dashboard.button("SPC wr", "   > Restore Session For Curre
      dashboard.button("q", "  > Quit NVIM", "<cmd>qa<CR>"),
    }

    -- Send config to alpha
    alpha.setup(dashboard.opts)

    -- Disable folding on alpha buffer
    vim.cmd([[autocmd FileType alpha setlocal nofoldenable]])
  end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup automated session manager

Automatic session management is great for auto saving sessions before exiting Neovim and getting back to work when you come back.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `auto-session.lua`

Add the following to this file:

```lua
return {
  "rmagatti/auto-session",
  config = function()
    local auto_session = require("auto-session")

    auto_session.setup({
      auto_restore_enabled = false,
      auto_session_suppress_dirs = { "~/", "~/Dev/", "~/Download
    })

    local keymap = vim.keymap

    keymap.set("n", "<leader>wr", "<cmd>SessionRestore<CR>", { d
    keymap.set("n", "<leader>ws", "<cmd>SessionSave<CR>", { desc
  end,
}
```

Exit with `:q` and reenter Neovim with `nvim` .

When working in a project, you can now close everything with `:qa` and when you open Neovim again in this directory you can use `<leader>wr` to restore your workspace/session.

Disable lazy.nvim change_detection notification

Let's disable the lazy.nvim change_detection notification which I find a bit annoying.

Navigate to `lazy.lua` and modify the code like so:

```lua
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim
if not vim.loop.fs_stat(lazypath) then
  vim.fn.system({
    "git",
    "clone",
    "--filter=blob:none",
    "https://github.com/folke/lazy.nvim.git",
    "--branch=stable", -- latest stable release
    lazypath,
  })
end
vim.opt.rtp:prepend(lazypath)

```

```
require("lazy").setup("josean.plugins", {
  change_detection = {
    notify = false,
  },
})
```

Exit with `:q` and reenter Neovim with `nvim`

Setup bufferline for better looking tabs

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `bufferline.lua`

Add the following code:

```
return {
  "akinsho/bufferline.nvim",
  dependencies = { "nvim-tree/nvim-web-devicons" },
  version = "*",
  opts = {
    options = {
      mode = "tabs",
      separator_style = "slant",
    },
  },
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup lualine for a better statusline

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `lualine.lua`

Add the following code:

```lua
return {
  "nvim-lualine/lualine.nvim",
  dependencies = { "nvim-tree/nvim-web-devicons" },
  config = function()
    local lualine = require("lualine")
    local lazy_status = require("lazy.status") -- to configure l

    local colors = {
      blue = "#65D1FF",
      green = "#3EFFDC",
      violet = "#FF61EF",
      yellow = "#FFDA7B",
      red = "#FF4A4A",
      fg = "#c3ccdc",
      bg = "#112638",
      inactive_bg = "#2c3043",
    }

    local my_lualine_theme = {
      normal = {
        a = { bg = colors.blue, fg = colors.bg, gui = "bold" },
        b = { bg = colors.bg, fg = colors.fg },
        c = { bg = colors.bg, fg = colors.fg },
      },
      insert = {
        a = { bg = colors.green, fg = colors.bg, gui = "bold" },
        b = { bg = colors.bg, fg = colors.fg },
        c = { bg = colors.bg, fg = colors.fg },
      },
      visual = {
        a = { bg = colors.violet, fg = colors.bg, gui = "bold" }
        b = { bg = colors.bg, fg = colors.fg },
        c = { bg = colors.bg, fg = colors.fg },
      },
      command = {
        a = { bg = colors.yellow, fg = colors.bg, gui = "bold" }
        b = { bg = colors.bg, fg = colors.fg },
        c = { bg = colors.bg, fg = colors.fg },
      },
      replace = {
        a = { bg = colors.red, fg = colors.bg, gui = "bold" },
        b = { bg = colors.bg, fg = colors.fg },
        c = { bg = colors.bg, fg = colors.fg },
      },
      inactive = {
        a = { bg = colors.inactive_bg, fg = colors.semilightgray
        b = { bg = colors.inactive_bg, fg = colors.semilightgray
        c = { bg = colors.inactive_bg, fg = colors.semilightgray
```

```lua
      },
    }

    -- configure lualine with modified theme
    lualine.setup({
      options = {
        theme = my_lualine_theme,
      },
      sections = {
        lualine_x = {
          {
            lazy_status.updates,
            cond = lazy_status.has_updates,
            color = { fg = "#ff9e64" },
          },
          { "encoding" },
          { "fileformat" },
          { "filetype" },
        },
      },
    })
  end,
}
```

So that lualine can show pending plugin updates through lazy.nvim, open "lazy.lua" and modify it like so:

```lua
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim
if not vim.loop.fs_stat(lazypath) then
```

```lua
  vim.fn.system({
    "git",
    "clone",
    "--filter=blob:none",
    "https://github.com/folke/lazy.nvim.git",
    "--branch=stable", -- latest stable release
    lazypath,
  })
end
vim.opt.rtp:prepend(lazypath)

require("lazy").setup("josean.plugins", {
  checker = {
    enabled = true,
    notify = false,
  },
  change_detection = {
    notify = false,
  },
})
```

Exit with `:q` and reenter Neovim with `nvim`

Setup dressing.nvim

Dressing.nvim improves the ui for `vim.ui.select` and `vim.ui.input`

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `dressing.lua`

Add the following code:

```lua
return {
  "stevearc/dressing.nvim",
  event = "VeryLazy",
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup vim-maximizer

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `vim-maximizer.lua`

Add the following code:

```lua
return {
  "szw/vim-maximizer",
  keys = {
    { "<leader>sm", "<cmd>MaximizerToggle<CR>", desc = "Maximize
  },
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup treesitter

Treesitter is an awesome Neovim feature that provides better syntax highlighting, indentation, autotagging, incremental selection and many other cool features.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `treesitter.lua`

Add the following code:

```lua
return {
  "nvim-treesitter/nvim-treesitter",
  event = { "BufReadPre", "BufNewFile" },
  build = ":TSUpdate",
  dependencies = {
    "windwp/nvim-ts-autotag",
  },
  config = function()
    -- import nvim-treesitter plugin
    local treesitter = require("nvim-treesitter.configs")

    -- configure treesitter
    treesitter.setup({ -- enable syntax highlighting
      highlight = {
        enable = true,
      },
      -- enable indentation
      indent = { enable = true },
      -- enable autotagging (w/ nvim-ts-autotag plugin)
      autotag = {
        enable = true,
      },
      -- ensure these language parsers are installed
      ensure_installed = {
        "json",
        "javascript",
        "typescript",
        "tsx",
        "yaml",
        "html",
        "css",
        "prisma",
        "markdown",
        "markdown_inline",
        "svelte",
        "graphql",
        "bash",
        "lua",
        "vim",
        "dockerfile",
        "gitignore",
        "query",
        "vimdoc",
        "c",
      },
      incremental_selection = {
        enable = true,
        keymaps = {
          init_selection = "<C-space>",
          node_incremental = "<C-space>",
          scope_incremental = false,
          node_decremental = "<bs>",
        },
      },
    })
  end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup indent guides

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `indent-blankline.lua`

Add the following code:

```
return {
  "lukas-reineke/indent-blankline.nvim",
  event = { "BufReadPre", "BufNewFile" },
  main = "ibl",
  opts = {
    indent = { char = "┆" },
  },
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup autocompletion

We’re going to setup completion with “nvim-cmp” to get suggestions as we type.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `nvim-cmp.lua`

Add the following code:

```
return {
  "hrsh7th/nvim-cmp",
  event = "InsertEnter",
  dependencies = {
    "hrsh7th/cmp-buffer", -- source for text in buffer
    "hrsh7th/cmp-path", -- source for file system paths
    {
      "L3MON4D3/LuaSnip",
      -- follow latest release.
      version = "v2.*", -- Replace <CurrentMajor> by the latest
```

```
        -- install jsregexp (optional!).
        build = "make install_jsregexp",
      },
      "saadparwaiz1/cmp_luasnip", -- for autocompletion
      "rafamadriz/friendly-snippets", -- useful snippets
      "onsails/lspkind.nvim", -- vs-code like pictograms
    },
    config = function()
      local cmp = require("cmp")

      local luasnip = require("luasnip")

      local lspkind = require("lspkind")

      -- loads vscode style snippets from installed plugins (e.g.
      require("luasnip.loaders.from_vscode").lazy_load()

      cmp.setup({
        completion = {
          completeopt = "menu,menuone,preview,noselect",
        },
        snippet = { -- configure how nvim-cmp interacts with snipp
          expand = function(args)
            luasnip.lsp_expand(args.body)
          end,
        },
        mapping = cmp.mapping.preset.insert({
          ["<C-k>"] = cmp.mapping.select_prev_item(), -- previous
          ["<C-j>"] = cmp.mapping.select_next_item(), -- next sugg
          ["<C-b>"] = cmp.mapping.scroll_docs(-4),
          ["<C-f>"] = cmp.mapping.scroll_docs(4),
          ["<C-Space>"] = cmp.mapping.complete(), -- show completi
          ["<C-e>"] = cmp.mapping.abort(), -- close completion win
          ["<CR>"] = cmp.mapping.confirm({ select = false }),
        }),
        -- sources for autocompletion
        sources = cmp.config.sources({
          { name = "luasnip" }, -- snippets
          { name = "buffer" }, -- text within current buffer
          { name = "path" }, -- file system paths
        }),

        -- configure lspkind for vs-code like pictograms in comple
        formatting = {
          format = lspkind.cmp_format({
            maxwidth = 50,
            ellipsis_char = "...",
          }),
        },
      })
    end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup auto closing pairs

This plugin will help us auto close surrounding characters like parens, brackets, curly braces, quotes, single quotes and tags

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `autopairs.lua`

Add the following code:

```lua
return {
  "windwp/nvim-autopairs",
  event = { "InsertEnter" },
  dependencies = {
    "hrsh7th/nvim-cmp",
  },
  config = function()
    -- import nvim-autopairs
    local autopairs = require("nvim-autopairs")

    -- configure autopairs
    autopairs.setup({
      check_ts = true, -- enable treesitter
      ts_config = {
        lua = { "string" }, -- don't add pairs in lua string tre
        javascript = { "template_string" }, -- don't add pairs i
        java = false, -- don't check treesitter on java
      },
    })

    -- import nvim-autopairs completion functionality
    local cmp_autopairs = require("nvim-autopairs.completion.cmp

    -- import nvim-cmp plugin (completions plugin)
    local cmp = require("cmp")

    -- make autopairs and completion work together
    cmp.event:on("confirm_done", cmp_autopairs.on_confirm_done()
  end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup commenting plugin

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `comment.lua`

Add the following code:

```
return {
  "numToStr/Comment.nvim",
  event = { "BufReadPre", "BufNewFile" },
  dependencies = {
    "JoosepAlviste/nvim-ts-context-commentstring",
  },
  config = function()
    -- import comment plugin safely
```

```lua
      local comment = require("Comment")

      local ts_context_commentstring = require("ts_context_commentstr

      -- enable comment
      comment.setup({
        -- for commenting tsx, jsx, svelte, html files
        pre_hook = ts_context_commentstring.create_pre_hook(),
      })
    end,
  }
```

Exit with `:q` and reenter Neovim with `nvim`

Setup todo comments

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `todo-comments.lua`

Add the following code:

```lua
return {
  "folke/todo-comments.nvim",
  event = { "BufReadPre", "BufNewFile" },
  dependencies = { "nvim-lua/plenary.nvim" },
  config = function()
```

```lua
    local todo_comments = require("todo-comments")

    -- set keymaps
    local keymap = vim.keymap -- for conciseness

    keymap.set("n", "]t", function()
      todo_comments.jump_next()
    end, { desc = "Next todo comment" })

    keymap.set("n", "[t", function()
      todo_comments.jump_prev()
    end, { desc = "Previous todo comment" })

    todo_comments.setup()
  end,
}
```

Look for `telescope.lua` with telescope with `<leader>ff`

Open this file and add the following to be able to look for todos with telescope:

```lua
return {
  "nvim-telescope/telescope.nvim",
  branch = "0.1.x",
  dependencies = {
    "nvim-lua/plenary.nvim",
    { "nvim-telescope/telescope-fzf-native.nvim", build = "make"
    "nvim-tree/nvim-web-devicons",
    "folke/todo-comments.nvim",
    },
    config = function()
      local telescope = require("telescope")
      local actions = require("telescope.actions")

      telescope.setup({
        defaults = {
          path_display = { "smart" },
          mappings = {
            i = {
              ["<C-k>"] = actions.move_selection_previous, -- move
              ["<C-j>"] = actions.move_selection_next, -- move to
              ["<C-q>"] = actions.send_selected_to_qflist + action
            },
          },
        },
      })

      telescope.load_extension("fzf")

      -- set keymaps
      local keymap = vim.keymap -- for conciseness

      keymap.set("n", "<leader>ff", "<cmd>Telescope find_files<cr>
      keymap.set("n", "<leader>fr", "<cmd>Telescope oldfiles<cr>",
      keymap.set("n", "<leader>fs", "<cmd>Telescope live_grep<cr>"
      keymap.set("n", "<leader>fc", "<cmd>Telescope grep_string<cr
      keymap.set("n", "<leader>ft", "<cmd>TodoTelescope<cr>", { de
    end,
  }
```

Exit with `:q` and reenter Neovim with `nvim`

Setup substitution plugin

This plugin allows us to use `s` followed by a `motion` to substitute text that was previously copied.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `substitute.lua`

Add the following code:

```lua
return {
  "gbprod/substitute.nvim",
  event = { "BufReadPre", "BufNewFile" },
  config = function()
    local substitute = require("substitute")

    substitute.setup()

    -- set keymaps
    local keymap = vim.keymap -- for conciseness

    vim.keymap.set("n", "s", substitute.operator, { desc = "Subs
    vim.keymap.set("n", "ss", substitute.line, { desc = "Substit
    vim.keymap.set("n", "S", substitute.eol, { desc = "Substitut
    vim.keymap.set("x", "s", substitute.visual, { desc = "Substi
    end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup nvim-surround

This plugin is great for adding, deleting and modifying surrounding symbols and tags.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `surround.lua`

Add the following code:

```lua
return {
  "kylechui/nvim-surround",
  event = { "BufReadPre", "BufNewFile" },
  version = "*", -- Use for stability; omit to use `main` branch
  config = true,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup LSP

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `lua/josean/plugins` add a new directory with `a`, calling it `lsp/`

Navigate to `lazy.lua` and modify it so that `lazy.nvim` knows about the new `lsp` directory like so:

```lua
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim
if not vim.loop.fs_stat(lazypath) then
  vim.fn.system({
    "git",
    "clone",
    "--filter=blob:none",
    "https://github.com/folke/lazy.nvim.git",
    "--branch=stable", -- latest stable release
    lazypath,
  })
end
vim.opt.rtp:prepend(lazypath)

require("lazy").setup({ { import = "josean.plugins" }, { import
  checker = {
    enabled = true,
    notify = false,
  },
  change_detection = {
    notify = false,
  },
})
```

Setup mason.nvim

Mason.nvim is used to install and manage all of the language servers that you need for the languages you work for.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins/lsp` add a new file with `a` and call it `mason.lua`

Add the following code:

```lua
return {
  "williamboman/mason.nvim",
  dependencies = {
    "williamboman/mason-lspconfig.nvim",
  },
  config = function()
    -- import mason
    local mason = require("mason")

    -- import mason-lspconfig
    local mason_lspconfig = require("mason-lspconfig")

    -- enable mason and configure icons
    mason.setup({
      ui = {
        icons = {
          package_installed = "✓",
          package_pending = "➜",
          package_uninstalled = "✗",
        },
      },
    })

    mason_lspconfig.setup({
      -- list of servers for mason to install
      ensure_installed = {
        "tsserver",
        "html",
        "cssls",
        "tailwindcss",
        "svelte",
        "lua_ls",
        "graphql",
        "emmet_ls",
        "prismals",
        "pyright",
      },
    })
  end,
}
```

Setup nvim-lspconfig

Nvim-lspconfig is used to configure your language servers.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins/lsp` add a new file with `a` and call it `lspconfig.lua`

Add the following code:

```lua
return {
  "neovim/nvim-lspconfig",
  event = { "BufReadPre", "BufNewFile" },
  dependencies = {
    "hrsh7th/cmp-nvim-lsp",
    { "antosha417/nvim-lsp-file-operations", config = true },
    { "folke/neodev.nvim", opts = {} },
  },
  config = function()
    -- import lspconfig plugin
    local lspconfig = require("lspconfig")

    -- import mason_lspconfig plugin
    local mason_lspconfig = require("mason-lspconfig")

    -- import cmp-nvim-lsp plugin
    local cmp_nvim_lsp = require("cmp_nvim_lsp")

    local keymap = vim.keymap -- for conciseness

    vim.api.nvim_create_autocmd("LspAttach", {
      group = vim.api.nvim_create_augroup("UserLspConfig", {}),
      callback = function(ev)
        -- Buffer local mappings.
        -- See `:help vim.lsp.*` for documentation on any of the
        local opts = { buffer = ev.buf, silent = true }

        -- set keybinds
        opts.desc = "Show LSP references"
        keymap.set("n", "gR", "<cmd>Telescope lsp_references<CR>

        opts.desc = "Go to declaration"
        keymap.set("n", "gD", vim.lsp.buf.declaration, opts) --

        opts.desc = "Show LSP definitions"
        keymap.set("n", "gd", "<cmd>Telescope lsp_definitions<CR

        opts.desc = "Show LSP implementations"
        keymap.set("n", "gi", "<cmd>Telescope lsp_implementation

        opts.desc = "Show LSP type definitions"
        keymap.set("n", "gt", "<cmd>Telescope lsp_type_definitio

        opts.desc = "See available code actions"
        keymap.set({ "n", "v" }, "<leader>ca", vim.lsp.buf.code_

        opts.desc = "Smart rename"
        keymap.set("n", "<leader>rn", vim.lsp.buf.rename, opts)

        opts.desc = "Show buffer diagnostics"
        keymap.set("n", "<leader>D", "<cmd>Telescope diagnostics

        opts.desc = "Show line diagnostics"
        keymap.set("n", "<leader>d", vim.diagnostic.open_float,

        opts.desc = "Go to previous diagnostic"
        keymap.set("n", "[d", vim.diagnostic.goto_prev, opts) --

```

```
        opts.desc = "Go to next diagnostic"
        keymap.set("n", "]d", vim.diagnostic.goto_next, opts) --

        opts.desc = "Show documentation for what is under cursor
        keymap.set("n", "K", vim.lsp.buf.hover, opts) -- show do

        opts.desc = "Restart LSP"
        keymap.set("n", "<leader>rs", ":LspRestart<CR>", opts) -
      end,
    })

    -- used to enable autocompletion (assign to every lsp server
    local capabilities = cmp_nvim_lsp.default_capabilities()

    -- Change the Diagnostic symbols in the sign column (gutter)
    -- (not in youtube nvim video)
    local signs = { Error = "  ", Warn = "  ", Hint = "  ", Info
    for type, icon in pairs(signs) do
      local hl = "DiagnosticSign" .. type
      vim.fn.sign_define(hl, { text = icon, texthl = hl, numhl =
    end

    mason_lspconfig.setup_handlers({
      -- default handler for installed servers
      function(server_name)
        lspconfig[server_name].setup({
          capabilities = capabilities,
        })
      end,
      ["svelte"] = function()
```

```
        -- configure svelte server
        lspconfig["svelte"].setup({
          capabilities = capabilities,
          on_attach = function(client, bufnr)
            vim.api.nvim_create_autocmd("BufWritePost", {
              pattern = { "*.js", "*.ts" },
              callback = function(ctx)
                -- Here use ctx.match instead of ctx.file
                client.notify("$/onDidChangeTsOrJsFile", { uri =
              end,
            })
          end,
        })
      end,
      ["graphql"] = function()
        -- configure graphql language server
        lspconfig["graphql"].setup({
          capabilities = capabilities,
          filetypes = { "graphql", "gql", "svelte", "typescriptr
        })
      end,
      ["emmet_ls"] = function()
        -- configure emmet language server
        lspconfig["emmet_ls"].setup({
          capabilities = capabilities,
          filetypes = { "html", "typescriptreact", "javascriptre
        })
      end,
      ["lua_ls"] = function()
        -- configure lua server (with special settings)
```

```
          lspconfig["lua_ls"].setup({
            capabilities = capabilities,
            settings = {
              Lua = {
                -- make the language server recognize "vim" global
                diagnostics = {
                  globals = { "vim" },
                },
                completion = {
                  callSnippet = "Replace",
                },
              },
            },
          })
        end,
      })
    end,
}
```

In the code under `mason_lspconfig.setup_handlers` I setup a default for my language servers and some custom configurations for `svelte`, `graphql`, `emmet_ls`, and `lua_ls`. This can vary depending on the languages that you're gonna be using.

Navigate to `nvim-cmp.lua` and make the following change to add the lsp as a completion source:

```
return {
  "hrsh7th/nvim-cmp",
  event = "InsertEnter",
  dependencies = {
    "hrsh7th/cmp-buffer", -- source for text in buffer
    "hrsh7th/cmp-path", -- source for file system paths
    {
      "L3MON4D3/LuaSnip",
      -- follow latest release.
      version = "v2.*", -- Replace <CurrentMajor> by the latest
      -- install jsregexp (optional!).
      build = "make install_jsregexp",
    },
    "saadparwaiz1/cmp_luasnip", -- for autocompletion
    "rafamadriz/friendly-snippets", -- useful snippets
    "onsails/lspkind.nvim", -- vs-code like pictograms
  },
  config = function()
    local cmp = require("cmp")

    local luasnip = require("luasnip")

    local lspkind = require("lspkind")

    -- loads vscode style snippets from installed plugins (e.g.
    require("luasnip.loaders.from_vscode").lazy_load()

    cmp.setup({
      completion = {
        completeopt = "menu,menuone,preview,noselect",
      },
      snippet = { -- configure how nvim-cmp interacts with snipp
```

```
          expand = function(args)
            luasnip.lsp_expand(args.body)
          end,
        },
        mapping = cmp.mapping.preset.insert({
          ["<C-k>"] = cmp.mapping.select_prev_item(), -- previous
          ["<C-j>"] = cmp.mapping.select_next_item(), -- next sugg
          ["<C-b>"] = cmp.mapping.scroll_docs(-4),
          ["<C-f>"] = cmp.mapping.scroll_docs(4),
          ["<C-Space>"] = cmp.mapping.complete(), -- show completi
          ["<C-e>"] = cmp.mapping.abort(), -- close completion win
          ["<CR>"] = cmp.mapping.confirm({ select = false }),
        }),
        -- sources for autocompletion
        sources = cmp.config.sources({
          { name = "nvim_lsp"},
          { name = "luasnip" }, -- snippets
          { name = "buffer" }, -- text within current buffer
          { name = "path" }, -- file system paths
        }),

        -- configure lspkind for vs-code like pictograms in comple
        formatting = {
          format = lspkind.cmp_format({
            maxwidth = 50,
            ellipsis_char = "...",
          }),
        },
      })
    end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup trouble.nvim

This is another plugin that adds some nice functionality for interacting with the lsp and some other things like todo comments.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `trouble.lua`

Add the following code:

```lua
return {
  "folke/trouble.nvim",
  dependencies = { "nvim-tree/nvim-web-devicons", "folke/todo-co
  opts = {
    focus = true,
  },
  cmd = "Trouble",
  keys = {
    { "<leader>xw", "<cmd>Trouble diagnostics toggle<CR>", desc
    { "<leader>xd", "<cmd>Trouble diagnostics toggle filter.buf=
    { "<leader>xq", "<cmd>Trouble quickfix toggle<CR>", desc = "
    { "<leader>xl", "<cmd>Trouble loclist toggle<CR>", desc = "O
```

```
      { "<leader>xt", "<cmd>Trouble todo toggle<CR>", desc = "Open
    },
  }
```

The code above has been refactored to work with trouble version 3. This is different from the code in the video

Exit with `:q` and reenter Neovim with `nvim`

Setup formatting

We're gonna use `conform.nvim` to setup formatting in Neovim.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `formatting.lua`

Add the following code:

```
return {
  "stevearc/conform.nvim",
  event = { "BufReadPre", "BufNewFile" },
  config = function()
    local conform = require("conform")

    conform.setup({
      formatters_by_ft = {
        javascript = { "prettier" },
        typescript = { "prettier" },
        javascriptreact = { "prettier" },
        typescriptreact = { "prettier" },
        svelte = { "prettier" },
        css = { "prettier" },
        html = { "prettier" },
        json = { "prettier" },
        yaml = { "prettier" },
        markdown = { "prettier" },
        graphql = { "prettier" },
        liquid = { "prettier" },
        lua = { "stylua" },
        python = { "isort", "black" },
      },
      format_on_save = {
        lsp_fallback = true,
        async = false,
        timeout_ms = 1000,
      },
    })

    vim.keymap.set({ "n", "v" }, "<leader>mp", function()
      conform.format({
        lsp_fallback = true,
        async = false,
        timeout_ms = 1000,
      })
    end, { desc = "Format file or range (in visual mode)" })
  end,
}
```

Navigate to `mason.lua` and add the following to auto install formatters:

```lua
return {
  "williamboman/mason.nvim",
  dependencies = {
    "williamboman/mason-lspconfig.nvim",
    "WhoIsSethDaniel/mason-tool-installer.nvim",
  },
  config = function()
    -- import mason
    local mason = require("mason")

    -- import mason-lspconfig
    local mason_lspconfig = require("mason-lspconfig")

    local mason_tool_installer = require("mason-tool-installer")

    -- enable mason and configure icons
    mason.setup({
      ui = {
        icons = {
          package_installed = "✓",
          package_pending = "➜",
          package_uninstalled = "✗",
        },
      },
    })

    mason_lspconfig.setup({
      -- list of servers for mason to install
      ensure_installed = {
        "tsserver",
        "html",
        "cssls",
        "tailwindcss",
        "svelte",
        "lua_ls",
        "graphql",
        "emmet_ls",
        "prismals",
        "pyright",
      },
    })

    mason_tool_installer.setup({
      ensure_installed = {
        "prettier", -- prettier formatter
        "stylua", -- lua formatter
        "isort", -- python formatter
        "black", -- python formatter
      },
    })
  end,
}
```

Exit with `:q` and reenter Neovim with `nvim`

Setup linting

We're gonna be using nvim-lint to setup linting in Neovim.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `linting.lua`

Add the following code:

```lua
return {
  "mfussenegger/nvim-lint",
  event = { "BufReadPre", "BufNewFile" },
  config = function()
    local lint = require("lint")

    lint.linters_by_ft = {
      javascript = { "eslint_d" },
      typescript = { "eslint_d" },
      javascriptreact = { "eslint_d" },
      typescriptreact = { "eslint_d" },
      svelte = { "eslint_d" },
      python = { "pylint" },
    }

    local lint_augroup = vim.api.nvim_create_augroup("lint", { c

    vim.api.nvim_create_autocmd({ "BufEnter", "BufWritePost", "I
      group = lint_augroup,
      callback = function()
        lint.try_lint()
      end,
    })

    vim.keymap.set("n", "<leader>l", function()
      lint.try_lint()
    end, { desc = "Trigger linting for current file" })
  end,
}
```

Navigate to `mason.lua` and add the following to auto install linters:

```
return {
  "williamboman/mason.nvim",
  dependencies = {
    "williamboman/mason-lspconfig.nvim",
    "WhoIsSethDaniel/mason-tool-installer.nvim",
  },
  config = function()
    -- import mason
    local mason = require("mason")

    -- import mason-lspconfig
    local mason_lspconfig = require("mason-lspconfig")

    local mason_tool_installer = require("mason-tool-installer")

    -- enable mason and configure icons
```

```lua
    mason.setup({
      ui = {
        icons = {
          package_installed = "✓",
          package_pending = "➜",
          package_uninstalled = "✗",
        },
      },
    })

    mason_lspconfig.setup({
      -- list of servers for mason to install
      ensure_installed = {
        "tsserver",
        "html",
        "cssls",
        "tailwindcss",
        "svelte",
        "lua_ls",
        "graphql",
        "emmet_ls",
        "prismals",
        "pyright",
      },
    })

    mason_tool_installer.setup({
      ensure_installed = {
        "prettier", -- prettier formatter
        "stylua", -- lua formatter
```

```lua
          "isort", -- python formatter
          "black", -- python formatter
          "pylint", -- python linter
          "eslint_d", -- js linter
        },
      })
    end,
  }
```

Exit with `:q` and reenter Neovim with `nvim`

Setup git functionality

Setup gitsigns plugin

Gitsigns is a great plugin for interacting with git hunks in Neovim.

Open the file explorer with `<leader>ee` (in my config the `<leader>` key is `space`).

Under `plugins` add a new file with `a` and call it `gitsigns.lua`

Add the following code:

```lua
return {
  "lewis6991/gitsigns.nvim",
  event = { "BufReadPre", "BufNewFile" },
  opts = {
    on_attach = function(bufnr)
```

```
      local gs = package.loaded.gitsigns

      local function map(mode, l, r, desc)
        vim.keymap.set(mode, l, r, { buffer = bufnr, desc = desc
      end

      -- Navigation
      map("n", "]h", gs.next_hunk, "Next Hunk")
      map("n", "[h", gs.prev_hunk, "Prev Hunk")

      -- Actions
      map("n", "<leader>hs", gs.stage_hunk, "Stage hunk")
      map("n", "<leader>hr", gs.reset_hunk, "Reset hunk")
      map("v", "<leader>hs", function()
        gs.stage_hunk({ vim.fn.line("."), vim.fn.line("v") })
      end, "Stage hunk")
      map("v", "<leader>hr", function()
        gs.reset_hunk({ vim.fn.line("."), vim.fn.line("v") })
      end, "Reset hunk")

      map("n", "<leader>hS", gs.stage_buffer, "Stage buffer")
      map("n", "<leader>hR", gs.reset_buffer, "Reset buffer")

      map("n", "<leader>hu", gs.undo_stage_hunk, "Undo stage hun

      map("n", "<leader>hp", gs.preview_hunk, "Preview hunk")

      map("n", "<leader>hb", function()
        gs.blame_line({ full = true })
      end, "Blame line")
```

```
        map("n", "<leader>hB", gs.toggle_current_line_blame, "Togg

        map("n", "<leader>hd", gs.diffthis, "Diff this")
        map("n", "<leader>hD", function()
          gs.diffthis("~")
        end, "Diff this ~")

        -- Text object
        map({ "o", "x" }, "ih", ":<C-U>Gitsigns select_hunk<CR>",
      end,
    },
  }
```

Exit with `:q`

Setup lazygit integration

Make sure you have lazygit installed.

Install with homebrew:

```
brew install jesseduffield/lazygit/lazygit
```

Open Neovim with `nvim .`

Under `plugins` add a new file with `a` and call it `lazygit.lua`

Add the following code:

```lua
return {
  "kdheepak/lazygit.nvim",
  cmd = {
    "LazyGit",
    "LazyGitConfig",
    "LazyGitCurrentFile",
    "LazyGitFilter",
    "LazyGitFilterCurrentFile",
  },
  -- optional for floating window border decoration
  dependencies = {
    "nvim-lua/plenary.nvim",
  },
  -- setting the keybinding for LazyGit with 'keys' is recommend
  -- order to load the plugin when the command is run for the fi
  keys = {
    { "<leader>lg", "<cmd>LazyGit<cr>", desc = "Open lazy git" }
  },
}
```

Exit with `:q` and reenter Neovim with `nvim`

YOU'RE DONE! 🚀

26 reactions

☺ 👍 4 🎉 2 ❤️ 14 🚀 6

Sign in to add your reaction

55 comments · 33+ replies

Oldest Newest

👍 👎 😄 🎉
😕 ❤️ 🚀 👀

alecthegeek Apr 3, 2024

Noice as we say in Australia.

Just a personal preference thing, but I like to use "pure" lua. So for example

```
vim.g.netrw_liststyle = 3
```

instead of the Vimscript API call.

Also, I am curious as to why you touch files before editing them?

↑ 1 1 reply

amolsleekbill Dec 16, 2024

With touch command it creates empty file with the name we provide

highend Apr 5, 2024

Hi Josean,
thanks (again) for such a great video & learning resource for nvim!

With your current setup and the usage of nvim-cmp, is it possible to use to either

1. Automatically expand the first entry of the popup menu (if no menu entry is / was selected) or
2. Expand the entry that was previously selected (via /) instead ?

↑ 1 0 replies

azrias789 Apr 6, 2024

Just one word. Amazing period.

Pls update documentation to include require("lazy").setup("josean.plugins") in step Go to “lazy.lua” and add the following to bootstrap lazy.nvim

↑ 1 0 replies

wilbrijo Apr 8, 2024

Awesome videos/tutorials Josean. Thanks for all the help. Question: Any idea why I'm getting italics on my "return" commands after loading tree-sitter? Also in comments, which I don't mind so much but would like to know where it's configured. Mine looks like:
return {
...
}

↑ 1 1 reply

swickrotation 3 days ago

You've probably figured it out by now, but if you look in the repo for the tokyonight colorscheme, in tokyonight.nvim/lua/tokyonight/config.lua, you can see that both keywords and comments are set to italics by default.

mayur01201 Apr 22, 2024

Great video. Thanks for showing all stuffs in details.

If you facing issue in trancperacy, check colorscheme.lua file inside plugins directory.
add transparent = true under require("tokyonight").setup

so colorscheme.lua will look like below

```
return {
  "folke/tokyonight.nvim",
  priority = 1000,
  config = function()
    local bg = "#011628"
    local bg_dark = "#011423"
    local bg_highlight = "#143652"
```

```
      local bg_search = "#0A64AC"
      local bg_visual = "#275378"
      local fg = "#CBE0F0"
      local fg_dark = "#B4D0E9"
      local fg_gutter = "#627E97"
      local border = "#547998"

      require("tokyonight").setup({
       transparent = true,
       style = "night",
        on_colors = function(colors)
          colors.bg = bg
          colors.bg_dark = bg_dark
          colors.bg_float = bg_dark
          colors.bg_highlight = bg_highlight
          colors.bg_popup = bg_dark
          colors.bg_search = bg_search
          colors.bg_sidebar = bg_dark
          colors.bg_statusline = bg_dark
          colors.bg_visual = bg_visual
          colors.border = border
          colors.fg = fg
          colors.fg_dark = fg_dark
          colors.fg_float = fg
          colors.fg_gutter = fg_gutter
          colors.fg_sidebar = fg_dark
        end
      })

      vim.cmd("colorscheme tokyonight")
    end
  }
```

↑ 1 1 reply

josean-dev Apr 27, 2024 Owner

I just made a modification to the repo to add better support for transparency with these two changes: [colorscheme.lua]() change and [bufferline.lua]() change. Hope that helps!

ZebraAlgebra Apr 24, 2024

awesome post and video.

one small thing: in the setting up formatter section, code segments on changes to be made in the `mason.lua` file can also highlight the change made in line 14 (this line looks like:

```
local mason_tool_installer = require("mason-tool-installer")
```

↑ 0 1 reply

josean-dev Apr 27, 2024 Owner

Just fixed this! Thanks.

👍 1

zrengifo Apr 24, 2024

Great video!
Just got my first mac so this helped a lot!

I am getting an issue with the mason config

here is the error
Failed to run `config` for mason.nvim

.../zachrengifo/.config/nvim/lua/zach/plugins/lsp/mason.lua:41: attempt to index global 'mason_tool _installer' (a nil value)

stacktrace:

- lua/zach/plugins/lsp/mason.lua:41 *in* **config**
- lua/zach/lazy.lua:14
- init.lua:2

And in the mason.lua file beside line 40 "mason_tool_installer.setup" I get an error that says undefined global

↑ 1 5 replies

WillScarlettOhara Apr 25, 2024

Look previous comment. ;)

`local mason_tool_installer = require("mason-tool-installer")` is not highlighted but it should be.

josean-dev Apr 27, 2024 Owner

I was missing a highlight for this. Just fixed it, thanks!

zrengifo Apr 27, 2024

I'm sorry but I'm new to this. Is there a previous thread about this? Or do I just add that into the file?

josean-dev Apr 27, 2024 Owner

No worries! Yes, this line is missing in mason.lua. Take a look at the formatters section of the blog post where I add mason-tool-installer to the mason.lua file in order to see where I add this line. I had accidentally forgotten to highlight the addition and if you don't add the line properly, you'll get the error you're talking about.

zrengifo Apr 28, 2024

Got it thank you so much!

WillScarlettOhara Apr 25, 2024

I guess you meant `josean.lsp` not `josean.plugins.lsp` for the second import in lazy.lua file
`require("lazy").setup({ { import = "josean.plugins" }, { import = "josean.plugins.lsp" } }, {`

josean-dev Apr 27, 2024 Owner

It should be “josean.plugins.lsp” as the “lsp”module is inside “plugins”.

tyrellcurry Apr 25, 2024

Thanks for this amazing configuration! Excited to get started working in it.

Quick question, how can I disable auto formatting on save altogether and only rely on the command instead? In my role, we can only use formatters very seldomly. Thanks!

1 1 reply

WillScarlettOhara Apr 25, 2024

https://github.com/stevearc/conform.nvim/blob/master/doc/recipes.md#command-to-toggle-format-on-save

foxjazz Apr 25, 2024

why don't you just zip up your neovim config?

1 1 reply

WillScarlettOhara Apr 25, 2024

He did it. Go on his github and download repo as ZIP but there's no reason doing that when cloning is possible.

👍 1

vasi786 Apr 28, 2024

Hi, I have followed your video entirely and I have errors particularly when I am using linters, language servers. In both the cases the error is as follows from the Mason.log

error=spawn: npm failed with exit code - and signal -.npm is not executable.

Somehow this error only pops for some linters or language servers. For example "prettier" has the error, while "pylint" doesn't.

I am on centos-7. Everything works well except for this. Is there any setting I am missing. let me know. Thank you.

↑ 1 1 reply

vasi786 Apr 30, 2024

Figured it. I do not have node.js installed on my workstation. After that everything worked great.

ImtiazKhanDS Apr 30, 2024

How to configure a debugger ?

↑ 1 0 replies

arkem-gs May 1, 2024

Hi Josean, thank you for the tutorial! Everything works great except for the linting, I cannot figure out whats going on but I have errors in every .js file I open. Please see below for example from opening a JS file in buffer. Ignore the "..."s I just hid my names. Also below that is my mason log which is also showing errors. I have double checked everything from tutorial and it is correct. Thank you.

EXAMPLE FROM BUFFER:

| Could not parse linter output due to: Expected value but found invalid token at character 1 eslint_d [1, 1]
| output: Error: No ESLint configuration found in /.../.../Desktop/prepTest.

MASON LOG:

[ERROR Tue Apr 30 13:32:34 2024] ...m/lazy/mason-lspconfig.nvim/lua/mason-lspconfig/init.lua:33:
Failed to set up lspconfig integration ... on-lspconfig.nvim/lua/mason-

```
Failed to set up lspconfig integration: ...on-lspconfig.nvim/lua/mason-
lspconfig/lspconfig_hook.lua:55: module 'lspconfig.util' not found:
no field package.preload['lspconfig.util']
cache_loader: module lspconfig.util not found
cache_loader_lib: module lspconfig.util not found
no file './lspconfig/util.lua'
no file '/opt/homebrew/share/luajit-2.1/lspconfig/util.lua'
no file '/usr/local/share/lua/5.1/lspconfig/util.lua'
no file '/usr/local/share/lua/5.1/lspconfig/util/init.lua'
no file '/opt/homebrew/share/lua/5.1/lspconfig/util.lua'
no file '/opt/homebrew/share/lua/5.1/lspconfig/util/init.lua'
no file './lspconfig/util.so'
no file '/usr/local/lib/lua/5.1/lspconfig/util.so'
no file '/opt/homebrew/lib/lua/5.1/lspconfig/util.so'
no file '/usr/local/lib/lua/5.1/loadall.so'
no file './lspconfig.so'
no file '/usr/local/lib/lua/5.1/lspconfig.so'
no file '/opt/homebrew/lib/lua/5.1/lspconfig.so'
no file '/usr/local/lib/lua/5.1/loadall.so'
```

↑ 1 11 replies

Show 6 previous replies

josean-dev May 2, 2024 Owner

@WillScarlettOhara thanks for pointing out the typo! You were right, just fixed it. As to the linting error, it is unrelated to the lsp and has to do with a missing config file for eslint. A possible solution is with the code I provided above.

arkem-gs May 2, 2024

@josean-dev thank you for the code snippet that is greatly appreciated sir! You are correct I did not have linting config for the files I was working on.

@WillScarlettOhara haha I actually followed the video so I had that part right, thanks though!

What would you recommend as the better method? Should I modify the linting.lua file or should I just add linting configs to each file I work on, or perhaps install eslint_d globally?

arkem-gs May 6, 2024

@wilbrijo To anyone who might be getting a similar error. I was able to resolve this by downgrading my eslint version to 8. I made mistake of installing to latest and it broke everything for some reason. It all works now.

Thanks again @josean for the tut and your other content, much appreciated!

juancamilo-dev Sep 13, 2024

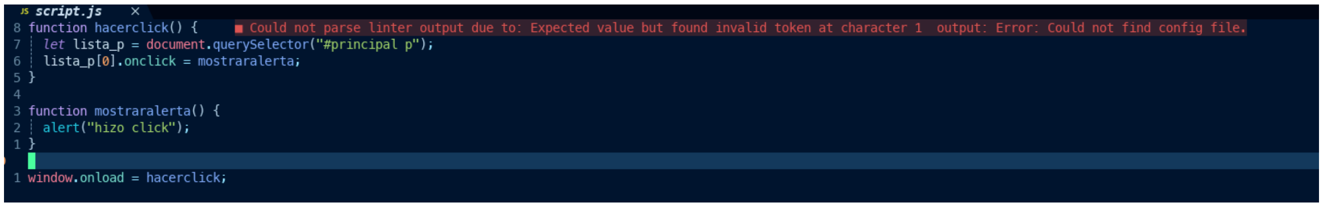
> The issue here is that when you don't have a config file for eslint, it will generate this error. This can happen for standalone js files and projects without linting. I typically have one for all my projects, but I can understand there being cases where you don't have linting setup.
>
> You can do something like this in linting.lua:

```
local function file_in_cwd(file_name)
  return vim.fs.find(file_name, {
    upward = true,
    stop = vim.loop.cwd():match("(.+)/"),
    path = vim.fs.dirname(vim.api.nvim_buf_get_name(0)),
    type = "file",
  })[1]
end

local function remove_linter(linters, linter_name)
  for k, v in pairs(linters) do
    if v == linter_name then
      linters[k] = nil
      break
    end
  end
end

local function linter_in_linters(linters, linter_name)
  for k, v in pairs(linters) do
    if v == linter_name then
      return true
    end
  end
  return false
end

local function remove_linter_if_missing_config_file(linters, linter_name
  if linter_in_linters(linters, linter_name) and not file_in_cwd(config_
    remove_linter(linters, linter_name)
  end
end

local function try_linting()
  local linters = lint.linters_by_ft[vim.bo.filetype]

   if linters then
    remove_linter_if_missing_config_file(linters, "eslint_d", ".eslintrc
  end

  lint.try_lint(linters)
end

vim.api.nvim_create_autocmd({ "BufEnter", "BufWritePost", "InsertLeave"
  group = lint_augroup,
  callback = function()
    try_linting()
  end,
})

vim.keymap.set("n", "<leader>l", function()
  try_linting()
end, { desc = "Trigger linting for current file" })
```

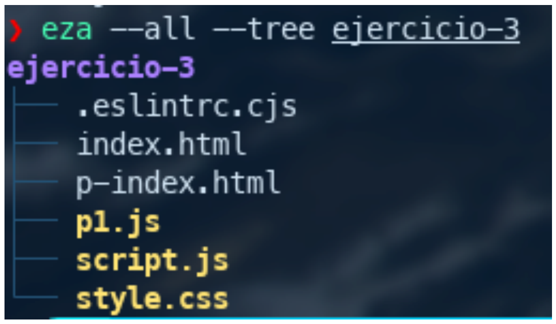
This code essentially checks if you have the config file in the current working directory before executing the linting when you are editing a filetype that has "eslint_d" enabled for it.

For the line that says `remove_linter_if_missing_config_file(linters, "eslint_d", ".eslintrc.cjs")`, you can replace ".eslintrc.cjs" with the file name you typically use for configuring eslint.

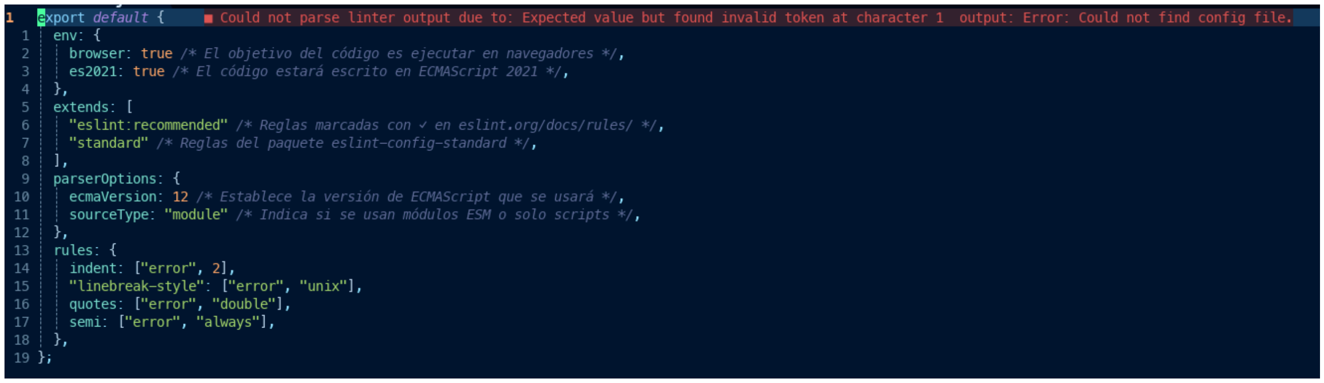
==================================
I have tried adding this to the code that is in the post, but I still have the problem, eslint_d is executed but it does not find the configuration file to be able to perform the linter of the file, when I execute:!eslint_d file.js
It does not find the configuration file, I have tried with the configuration file inside the folder where the file is, and with global files like .eslintrc.cjs
I hope you can help me since this tool is too important.



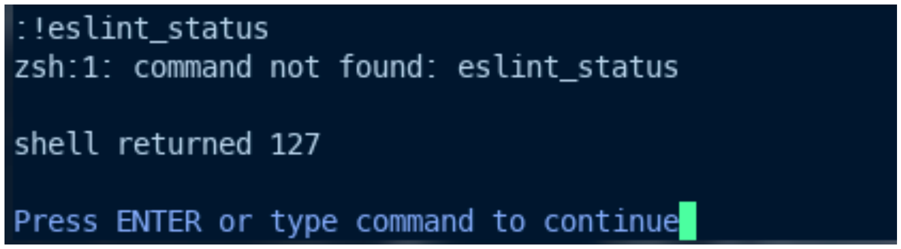
```
script.js  ×
function hacerclick() {     ■ Could not parse linter output due to: Expected value but found invalid token at character 1  output: Error: Could not find config file.
  let lista_p = document.querySelector("#principal p");
  lista_p[0].onclick = mostraralerta;
}

function mostraralerta() {
  alert("hizo click");
}

window.onload = hacerclick;
```



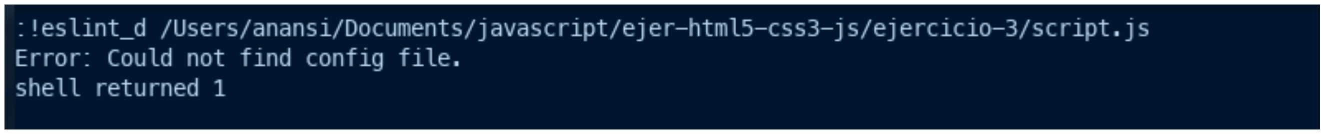
```
❯ eza --all --tree ejercicio-3
ejercicio-3
├── .eslintrc.cjs
├── index.html
├── p-index.html
├── p1.js
├── script.js
└── style.css
```



```
export default {     ■ Could not parse linter output due to: Expected value but found invalid token at character 1  output: Error: Could not find config file.
  env: {
    browser: true /* El objetivo del código es ejecutar en navegadores */,
    es2021: true /* El código estará escrito en ECMAScript 2021 */,
  },
  extends: [
    "eslint:recommended" /* Reglas marcadas con ✓ en eslint.org/docs/rules/ */,
    "standard" /* Reglas del paquete eslint-config-standard */,
  ],
  parserOptions: {
    ecmaVersion: 12 /* Establece la versión de ECMAScript que se usará */,
    sourceType: "module" /* Indica si se usan módulos ESM o solo scripts */,
  },
  rules: {
    indent: ["error", 2],
    "linebreak-style": ["error", "unix"],
    quotes: ["error", "double"],
    semi: ["error", "always"],
  },
};
```



```
:!eslint_status
zsh:1: command not found: eslint_status

shell returned 127

Press ENTER or type command to continue
```



```
:!eslint_d /Users/anansi/Documents/javascript/ejer-html5-css3-js/ejercicio-3/script.js
Error: Could not find config file.
shell returned 1
```

ghost Sep 28, 2024

I had the same problem. Try using eslint.config.js instead of .eslintrc.cjs

SarathLUN May 6, 2024

Hello @josean-dev,
Thank you for the sharing.
I learn a lot from this tutorial.

on top of this, I would like to requests you as below.

1. I like to maintain LSP config file separately for each language server, could you please share how I can archive this?
2. Could you please share what is best way to setup LSP for Go & Rust?
   Looking forward to get your feedback bro!

↑ 1 2 replies

SarathLUN May 6, 2024

1. I like to maintain LSP config file separately for each language server, could you please share how I can *achieve* this?

alecthegeek May 6, 2024

Just split each LSP config into it's own the file. The pattern is already in the examples, use that to help you.

greyilocks2 May 7, 2024

Nice!

A few things:

1. I just copied your git repo across, and git an IBL error. I had to run ':Lazy' and run U (update) to get rid of the error.
2. 'setup' is a thing. 'set up' is the verb, e.g. 'I set up nvim', or 'I set nvim up'. You're using 'setup' the noun everywhere, but you mean the verb (e.g. 'How I Setup Neovim ...' - could you change this, it's difficult to read if you have to mentally translate this each time?
3. Adding Rust to your config would be great. I don't use JS.

Thanks for this, it's excellent!

↑ 1 0 replies

25 hidden items
Load more...

kaiwah Sep 18, 2024 edited

This was a very thorough tutorial and appreciate the effort that went into this. Helped a lot in resetting my entire vim env, these days hard to find what is the new tools to use these days.

One thing I did notice however is that LazyVim is overriding my keybinds, more specifically `<S-h>` and `<S-l>`. After hours of combing through comments, issue tracking, docs, etc. I just decided to remove it directly from the lazyvim defaults. Not a sustainable fix but honestly tried everything from doing `vim.keymap.del` to setting keymaps in a plugin config, no matter what the lazyvim defaults always overwrites.

If anyone has thoughts on this would love to hear them.

↑ 1 0 replies

dgtipon Sep 20, 2024

The nvim lazy setup is fantastic but I did have problems on my manjaro linux operating system. I got errors when starting nvim. The biggest error was the tsserver required by the mason.lua plugin was not acceptable. I am a novice but I did find solutions on the Internet. I ran the command "sudo pacman -S nodejs npm" because my system did not come with nodejs and npm installed. I probably should have run the command without the sudo because nodejs and npm were installed in /user which made them global applications. So I ran the command "npm config get prefix" to change the application location to "/user/local". Then I ran the command "npm install -g typescript-language-server typescript" to install typescript on my computer. Then I replaced tsserver in the plugin mason.lua with typescript. Everything works now. No errors on starting nvim.

↑ 1 1 reply

sithadmin Sep 24, 2024

I would remove node with pacman and just set up nvm or n node manager. It installs any version of node you want but it is in your home directory.

ocitocit Sep 25, 2024

I am facing this Error while opening nvim

```
tsserver is deprecated, use ts_ls instead.
[mason-lspconfig.nvim] Server "tsserver" is not a valid entry in ensure_installed. Mak
sure to only provide lspconfig server names.
```

what should I do to fix this kind off error?

↑ 1 2 replies

ImtiazKhanDS Sep 25, 2024 edited

In mason.lua file replace “tsserver” with “ts_ls” in the ensure_installed section

👍 4

ocitocit Sep 26, 2024

Got it thank you so much!

halfpastfive23 Oct 4, 2024

Hey, I'm having a trouble in lspconfig.lua file where the error show "unused local 'bufnr' in line 92

↑ 1 0 replies

i-AmanRawat Oct 10, 2024

anyidea why my everything in my editor is looking purple.
although I copied all steps

↑ 1 2 replies

halfpastfive23 Oct 10, 2024

You mean by the colorscheme??

i-AmanRawat Oct 10, 2024

https://x.com/i_AmanRawat/status/1844617615853093209 have a look

acidclouds Oct 22, 2024

Hi!
Thanks for the guide, it really helped me to get started with nvim!

I do have a small comment though:

During the setup of lspconfig, you require mason-lspconfig, which in turn requires mason.core in it's init.lua.
Then, when the setup of mason is called, you have a require of mason-lspconfig which creates a loop. This happened to me only sporadically, and I still can't understand why only sporadically and not all the time...

according to mason documentation, you should first finish setting up mason, and only then setup mason-lspconfig, so the dependencies should be reversed, mason-lspconfig should depend on mason, as far as I understand.

After I took out all the mason-lspconfig to a different file, and added a dependency on mason, and called require mason.setup() explicitly, the errors were gone.
Maybe it has to do with me installing nvim-navic and adding it the the lualine for context. Because nvim-navic depends on mason-lspconfig so maybe there was some race condition on who requires mason-lspconfig first...

I am pretty new to this, so maybe I understand this wrong. Would be glad to hear what you think.

https://github.com/williamboman/mason-lspconfig.nvim?tab=readme-ov-file#setup

↑ 1 0 replies

mtlaso Oct 23, 2024

Thank you!

↑ 1 0 replies

dgtipon Oct 30, 2024

As a newbe I need to use ":Whichkey" to show the keymaps for NvimTree and native vim keymaps when editing a file. I decided to add this key map so I don't have to type anything: keymap.set("n", "", ":WhichKey", { desc = "WhichKey", silent = true })

↑ 1 0 replies

dgtipon Oct 30, 2024

In which-key.lua I added this code to set the first level and second level key group names. You will have to make modifications for your specific keymaps. I suggest you start by adding one group name, get that to work and then add more.

```
config = function()
    local status_ok, which_key = pcall(require, "which-key") -- Corrected to which_key
    if not status_ok then
        return
    end

    which_key.add({
        { "<leader>c", group = "Code suggestions", nowait = true, remap = false },
        { "<leader>e", group = "Explorer", nowait = true, remap = false },
        { "<leader>f", group = "Find", nowait = true, remap = false },
        { "<leader>h", group = "Hunk", nowait = true, remap = false },
        {
            "<leader>k",
            group = "Colorschemes",
            nowait = true,
            remap = false,
            { "<leader>kc", name = "Catppuccin", nowait = true, remap = false },
            { "<leader>ke", name = "Everforest", nowait = true, remap = false },
            { "<leader>kk", name = "Kanagawa", nowait = true, remap = false },
            { "<leader>kn", name = "Knightfox", nowait = true, remap = false },
            { "<leader>kt", name = "Tokyonight", nowait = true, remap = false },
        },
        { "<leader>m", group = "Format", nowait = true, remap = false },
        { "<leader>n", group = "Search highlights", nowait = true, remap = false },
        { "<leader>s", group = "Split", nowait = true, remap = false },
        { "<leader>t", group = "Tab", nowait = true, remap = false },
        { "<leader>w", group = "Session", nowait = true, remap = false },
        { "<leader>x", group = "Trouble", nowait = true, remap = false },
    })
  end,
```

↑ 1 0 replies

kalib Nov 1, 2024

The only thing missing is github copilot.. Maybe you could write something about how to include it as well? :D

↑ 1 0 replies

huchukato Nov 8, 2024

Thanks for this guide, now I love my nvim :D Just for instance, tsserver was renamed into ts_ls, this caused an error in mason, easy to fix indeed)

↑ 1 👍 1 0 replies

SirSaliver Nov 16, 2024

Anyone knows how to set font correctly? Got lots of "?" icons, showing faulty render.

↑ 1 1 reply

sithadmin Nov 18, 2024

Sounds like you don't have the font installed properly.

andrewyang17 Nov 24, 2024

Thanks for this awesome guide, can you setup a lspconfig for python? I been tinkering around with it but it doesn't work, not sure what's going wrong, the linter keeps complaining Unable to import. I have tried to activate the venv env before running nvim, I have tried to use venv-selector plugin, still couldn't figure it out, I'm a complete beginner, please help! thank you so much!

↑ 1 0 replies

jilvanx Dec 3, 2024 edited

If anyone had a problem with auto-tag follow those steps:
1 - create a file inside plugins folder called autotag.lua
2 - paste the content bellow:

```
    "windwp/nvim-ts-autotag",
    config = function()

      -- import nvim-ts-autotag plugin
      local autotag = require("nvim-ts-autotag")

      autotag.setup({
        opts = {
          -- Defaults
          enable_close = true, -- Auto close tags
          enable_rename = true, -- Auto rename pairs of tags
          enable_close_on_slash = false -- Auto close on trailing </
        },
      })

    end
  }
```

3 - comment those lines in treesitter.lua

```
  dependencies = {
     "windwp/nvim-ts-autotag",
  },
```

and

```
  autotag = {
    enable = true,
  },
```

4 - quit nvim and enter again.

↑ 1 👍 1 0 replies

anwarahmed Jan 1

Hey @josean-dev, thanks for the amazing tutorial, and especially for going into the explanations behind each setting!
As a first time user of nvim this has been very helpful in getting started and learning about the basics of configuring the plugins.

I am having one problem after setting up. The auto-formatter (*format on save* or **SPACE m p**) is replacing the spaces in my lua files with tabs. This seems to have started after making the changes to `mason.lua` in the ***Setup Linting*** step. I have tried to add the following setting in the `formatting.lua` file, with no luck:

```
conform.setup({
    formatters_by_ft = {
        -- ...
    },
    -- new section
    formatters = {
        stylua = {
            args_append = { "--indent-type", "Spaces", "--indent-width", "2" },
            -- I also tried `prepend_args = {}` and `args = {}`
```