How to use Neovim LSP

Diego Quiroz · Follow
4 min read · May 13, 2021

10 3

```
 _   _                 _             _     ____  ____
| \ | | ___  _____   _(_)_ __ ___   | |   / ___||  _ \
|  \| |/ _ \/ _ \ \ / / | '_ ` _ \  | |   \___ \| |_) |
| |\  |  __/ (_) \ V /| | | | | | | | |___ ___) |  __/
|_| \_|\___|\___/ \_/ |_|_| |_| |_| |_____|____/|_|
```

Neovim is a powerful editor used all around the world mainly by software developers. But the software development process is not that peasant if you don't have tools that can make the process easier and faster. Two of those tools are code linting and completion.

Modern code editors like VS Code uses something called a Language Server, this is the most advanced technology that helps code editors to understand errors in the code while they are writing it. In terms of Microsoft:

> *A* Language Server *is meant to provide the language-specific smarts and communicate with development tools over a protocol that enables inter-process communication.*

So Editors connect to Language Servers via a protocol commonly known as LSP (Language Server Protocol) to complete tasks like linting and code autocompletion.

Configure LSP with Neovim 0.5.0

Neovim 0.5.0 (currently in development) has native LSP support, so plugins like Conquer of Completion (CoC) or ALE are not needed anymore for some of the most common uses of an LSP: completion and linting.

To begin with this tutorial, we need to install the LspConfig plugin (developed and mantained by Neovim devs). This is not a plugin to implement LSP but the collection of common configs that will allow you to connect easily your Language Servers. With this you'll be able to use *:LspInfo*, *:LspRestart*, *:LspStop* and *:LspStart* commands. All LSP Client features are implemented on Neovim Core, this plugin is just for easy configuration.

For this you can use your favorite Neovim plugin manager, in my case is Dein (any other like Plug or Packer will work too).

Inside your *init.vim* file you will add:

```
# Using Plug:
:Plug 'neovim/nvim-lspconfig'

# Using Dein:
call dein#add('neovim/nvim-lspconfig')
```

Now we have acces to the power of LSP inside our editor, but we now need to add a Language Server. For this, there are many options, but this are the recommended by developers of Neovim.

In this tutorial we will follow the process for tsserver. So let's install it.

```
npm install -g typescript typescript-language-server
```

Now Neovim can have access to tsserver, we just need to add the folowing line to our *init.vim*

```
require('lspconfig').tsserver.setup{}
```

And with this, Neovim will have access to the Typescript Server Protocol.

Adding Completion

Unfortunately for us, there is something extra we need to add to our Neovim config if we want autocompletion, the easiest way is a plugin called nvim-compe.

So we will need to install it like we saw earlier:

```
# Using Plug:
:Plug 'hrsh7th/nvim-compe'

# Using Dein:
call dein#add('hrsh7th/nvim-compe')
```

Now we just need to add the following configurations to have Autocompletion of installed servers up and running.

Using Vimscript

```
set completeopt=menuone,noselect

let g:compe = {}
let g:compe.enabled = v:true
let g:compe.autocomplete = v:true
let g:compe.debug = v:false
let g:compe.min_length = 1
let g:compe.preselect = 'enable'
let g:compe.throttle_time = 80
let g:compe.source_timeout = 200
let g:compe.incomplete_delay = 400
let g:compe.max_abbr_width = 100
let g:compe.max_kind_width = 100
let g:compe.max_menu_width = 100
let g:compe.documentation = v:true

let g:compe.source = {}
let g:compe.source.path = v:true
let g:compe.source.buffer = v:true
let g:compe.source.calc = v:true
let g:compe.source.nvim_lsp = v:true
let g:compe.source.nvim_lua = v:true
let g:compe.source.vsnip = v:true
let g:compe.source.ultisnips = v:true
```

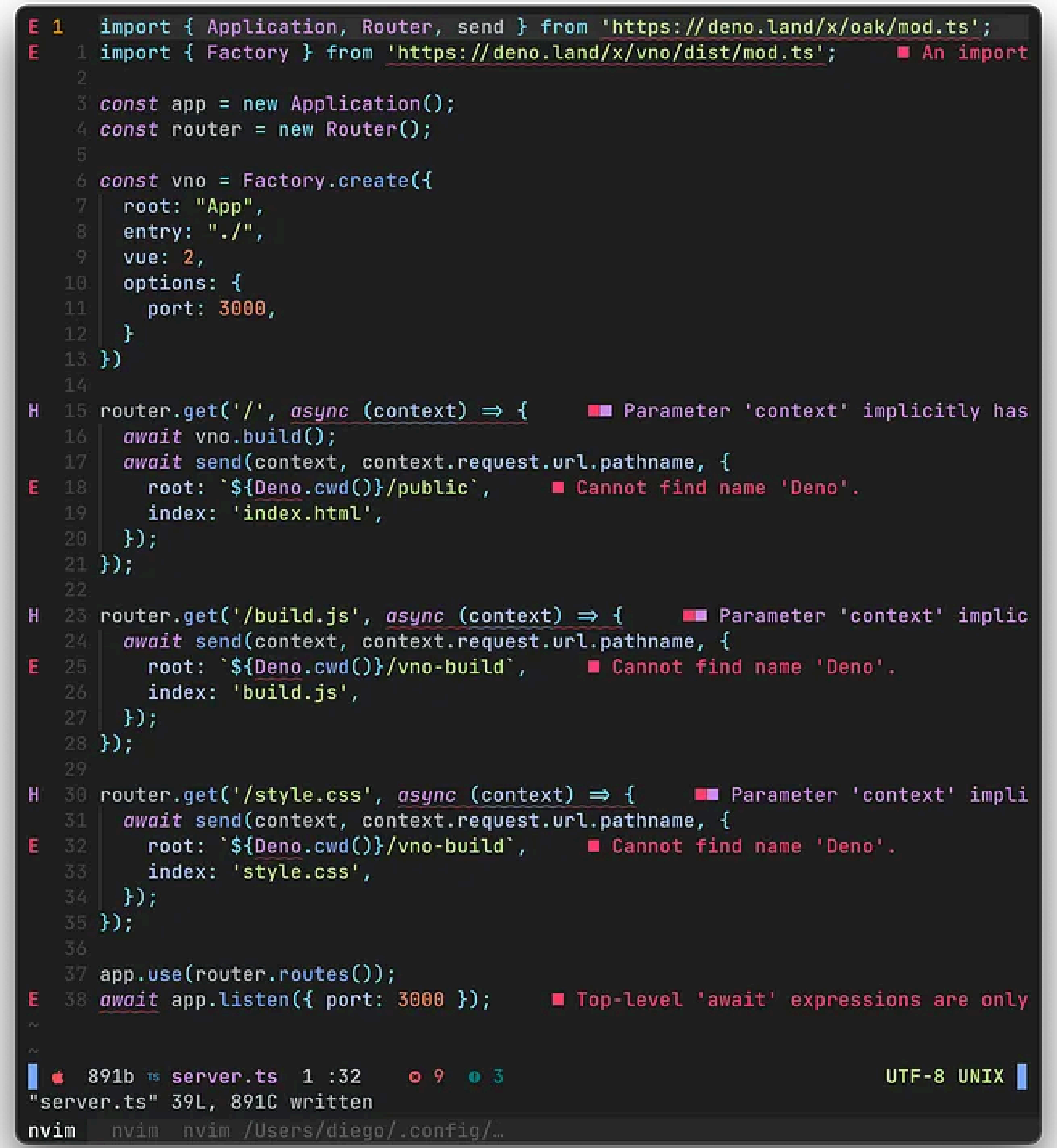
Using Lua:

```
vim.o.completeopt = “menuone,noselect”

require('compe').setup {
  enabled = true;
  autocomplete = true;
  debug = false;
  min_length = 1;
  preselect = 'enable';
  throttle_time = 80;
  source_timeout = 200;
  incomplete_delay = 400;
  max_abbr_width = 100;
  max_kind_width = 100;
  max_menu_width = 100;
  documentation = true;

  source = {
    path = true;
    buffer = true;
    calc = true;
    nvim_lsp = true;
    nvim_lua = true;
    vsnip = true;
    ultisnips = true;
  };
}
```

And we are done. We can access our Typescript buffers having Linting and completion:



```
E  import { Application, Router, send } from 'https://deno.land/x/oak/mod.ts';
E  import { Factory } from 'https://deno.land/x/vno/dist/mod.ts';     ■ An import

   const app = new Application();
   const router = new Router();

   const vno = Factory.create({
     root: "App",
     entry: "./",
     vue: 2,
     options: {
       port: 3000,
     }
   })

H  router.get('/', async (context) => {     ■■ Parameter 'context' implicitly has
     await vno.build();
     await send(context, context.request.url.pathname, {
E      root: `${Deno.cwd()}/public`,     ■ Cannot find name 'Deno'.
       index: 'index.html',
     });
   });

H  router.get('/build.js', async (context) => {     ■■ Parameter 'context' implic
     await send(context, context.request.url.pathname, {
E      root: `${Deno.cwd()}/vno-build`,     ■ Cannot find name 'Deno'.
       index: 'build.js',
     });
   });

H  router.get('/style.css', async (context) => {     ■■ Parameter 'context' impli
     await send(context, context.request.url.pathname, {
E      root: `${Deno.cwd()}/vno-build`,     ■ Cannot find name 'Deno'.
       index: 'style.css',
     });
   });

   app.use(router.routes());
E  await app.listen({ port: 3000 });     ■ Top-level 'await' expressions are only
~
~
▌  891b  server.ts  1 :32    ⊘ 9   ⊙ 3                        UTF-8 UNIX ▌
"server.ts" 39L, 891C written
nvim   nvim  nvim /Users/diego/.config/…
```

Adding more languages:

To add more languages you just need to search for your Language Server and install it like pyright:

```
npm i -g pyright
```

And add support for it inside your *init.vim*

```
require(‘lspconfig’).pyright.setup{}
```

If you want to change configs, just add them to the **setup**:

```
require('lspconfig').pyright.setup{
  settings = {
    python = {
      analysis = {
        autoSearchPaths = true,
        useLibraryCodeForTypes = true
      }
    }
  }
}
```

If this was any help for you, please share it.❤️

Neovim   Lsp   Language Server Protocol

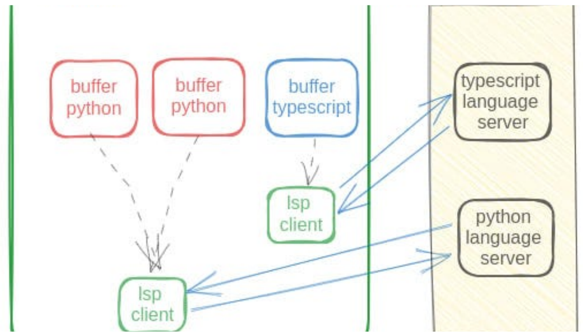
Written by Diego Quiroz

48 Followers

Software Developer

Follow

Recommended from Medium



Rishav Raj

How to setup lsp in neovim

In this tutorial we will learn how to setup lsp for auto-completion and understand role of...

7 min read · Oct 4, 2023

4

Anthony Garo

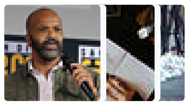
Neovim config from scratch [on OSX]

I've seen several posts recently of people that have a hard time customizing their Neovim...

3 min read · Sep 12, 2023

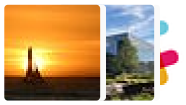
2

Lists



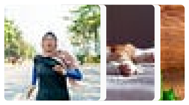
Staff Picks

587 stories · 776 saves



Stories to Help You Level-Up at Work

19 stories · 491 saves



Self-Improvement 101

20 stories · 1383 saves

Productivity 101

20 stories · 1272 saves

Sean Au Jong

Setting up Neovim in WSL

In my personal time, I develop on WSL because I only own Windows laptops. I'm a...

2 min read · Sep 4, 2023

8

Rajdeep Singh, M.Sc. & Technic... in The Li...

The correct way to install the Neovim

Installation of neovim in your Debian or Linux distro.

6 min read · Oct 3, 2023

56 2

David Carr
dcblog.dev

VIM shortcuts

4 min read

David Carr

```
source(n)                      Tcl Built-In Commands                      source(n)

NAME
       source - Evaluate a file or resource as a Tcl script

SYNOPSIS
       source fileName

       source [illegible] encodingName fileNam[illegible]

DESCRIPTION
       This [illegible] it to the Tcl interpreter as a text script.  The return
       value [illegible] If an error occurs in evaluating the contents of  the
       script  then the source command will return that error.  [illegible] return command is invoked from within the script then the remainder of the
       file will be skipped and the source command will return normally with the result from the return command.
man://source(n)                                                          1,1           Top

pkgname=dr[illegible]ck
pkgver=1.0
pkgrel=1
pkgdesc="A[illegible]"
arch=('any[illegible]
url="https[illegible]
license=([illegible])
source=("https://github.com/bouteillerAlan/${pkgname}/releases/download/${pkgname}-v${pkgver}/${pkgname}-v${pkgver}.tar.gz")
depends=("sh")
sha256sums=('74e9b5e4d58acff8835c4fd4503618afc035f5b29b02794be2463f8018eed39f')

package() {
  cd "${pkgname}-v${pkgver}"
  install -Dm 644 50numlock/module-setup.sh "${pkgdir}"/usr/lib/dracut/modules.d/50numlock/module-setup.sh
  install -Dm 644 50numlock/numlock.sh "${pkgdir}"/usr/lib/dracut/modules.d/50numlock/numlock.sh
  install -D LICENSE "${pkgdir}"/usr/share/licenses/"${pkgname}"/LICENSE
}
```

Bouteiller < A2N > Alan

VIM shortcuts

There are multiple modes in vim, normal mode is loaded by default.

✦ · 4 min read · Sep 20, 2023

27

Neovim config from scratch

Today we are going to mega shad your neovim—from scratch ! Youtube video...

10 min read · Oct 25, 2023

58

See more recommendations