Configurando o Vim e Neovim do Zero em 2022

fonte

Introdução

Nesta página vou mostrar como instalar e configurar do zero o Vim e o Neovim.

Se já tiver as aplicações instaladas, vou supor que o Vim é a ***versão 8.2*** pra cima, e para o Neovim a ***versão 0.5.1*** pra cima.

Antes de começar a instalar o Vim e o Neovim, é necessário instalar o *Python* e o *Nodejs*, pois o Vim e o Neovim utilizam em certos plugins. Isso é explicado nas próximas seções.

Também é necessário ter um Terminal já configurado para rodar o Vim. No Windows recomendo o ***Windows Terminal***, de preferência já utilizando uma *Nerd Font*. Caso queira aprender como configurar do zero o *Windows Terminal*, já fiz um vídeo sobre isso. Acesse pelo link:

- Windows Terminal

Instalação do Python

Windows

No Windows não tem muito segredo. É só entrar no link abaixo e baixar a última versão do instalador.

Não esqueça e confirmar que o Python está sendo instalado no *Path* do Windows, pois o Vim e o Neovim precisam acessar o Python.

- https://www.python.org/downloads/

Ubuntu/Wsl

É muito provável que o Python já esteja instalado no *Wsl* e *Ubuntu*. Se o ***pip*** não estiver instalado é, instale com os seguintes comandos:

```
sudo apt update
sudo apt install python3-pip
```

Se quiser instalar uma versão mais recente do Python, use os comandos:

```
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt install python3.9
```

O Pip 3.9 pode ser instalado assim:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
sudo python3.9 get-pip.py
```

No exemplo acima está o Python 3.9, porém coloque a versão mais nova atual, como 3.10, etc.

Biblioteca pynvim

A biblioteca pynvim é necessária para o Neovim conseguir utilizar o Python da forma que ele precisa. Para instalar a biblioteca pynvim, use o comando:

```
  pip install pynvim
```

Caso você estiver usando um ambiente virtual, sempre vai ter que instalar a biblioteca pynvim.

Instalação do Nodejs

Windows

No Windows basta acessar o site abaixo e baixar a última versão do instalador.

- https://nodejs.org/

Lembre de deixar o Nodejs instalado no *Path* do Windows, pois o Vim e o Neovim precisam acessar o Nodejs.

Ubuntu/Wsl

Recomendo instalar a última versão *LTS* do Nodejs.

Para isso acesse o link abaixo e siga as instruções de instalação do ***nvm***.

- https://github.com/nvm-sh/nvm#installing-and-updating

Em seguida reinicie o terminal e execute o comando:

```
nvm install --lts
```

Instalação do Vim no Windows

Baixar a versão estável em:

- https://www.vim.org/download.php

Instalar e escolher uma pasta. Pode ser o padrão, ou ***c:/vim***, que é a pasta em que eu instalo.

Linux / Wsl

```
sudo add-apt-repository ppa:jonathonf/vim
sudo apt update
sudo apt install vim
```

Instalação do nvim.

Windows

Para o Neovim é possível instalar a versão estável ou a versão mais recente instável.

Para a versão estável, baixar a versão mais recente em:

- https://github.com/neovim/neovim/releases/tag/stable

Porém recomendo a versão instável, que possue bem mais recursos. O link é:

- https://github.com/neovim/neovim/releases/tag/nightly

Baixar o arquivo *nvim-win64.zip* e descompactar.

Pasta de instalação

Eu sempre instalo na pasta *c:/nvim* ou *c:\Neovim*. Apenas descompacte o arquivo baixado e copie a pasta para a raiz do *c:*. O binário do Neovim fica instalado na pasta *c:/nvim/bin*.

Em seguida é necessário adicionar esta pasta ao *Path* do Windows. Procure por *variáveis de ambiente* na busca do Windows e adicione a pasta *c:/nvim/bin* ao *Path*.

Veja o vídeo se tiver dúvidas.

Linux / Wsl

No Linux recomendo instalar a versão unstable. Use as seguintes linhas para realizar a instalação:

```
$ sudo add-apt-repository ppa:neovim-ppa/unstable
$ sudo apt-get update
$ sudo apt-get install neovim
```

Adicionar o link *v* para o neovim

Uma vez que temos o neovim instalado, iniciamos a aplicação com o comando *nvim*, porém eu gosto de iniciar apenas chamando o comando *v*.

Para isso, no Windows temos que copiar o arquivo *nvim.exe* e renomear para *v.exe* (a cópia fica na mesma pasta *bin* que já está no *Path*). Desta forma podemos iniciar o Neovim com o comando *v*.

No Linux basta criar um link. Fica de lição de casa.

Criação dos arquivos de configuração

Quando instalamos o Vim e o Neovim, temos que criar os arquivos de configuração. Os arquivos não são criados automaticamente, então é necessário criá-los manualmente nas pastas corretas.

Vim

Para o Vim temos que criar um arquivo chamado ***.vimrc*** na pasta *HOME*.

No *Windows* temos que colocar em *C:/Users/NomeDoUsuário/.vimrc*.

No Linux temos que colocar em *~/.vimrc*.

Neovim

Para o Neovim temos que criar um arquivo chamado ***init.vim*** e a localização varia no Windows e no Linux.

No *Windows* temos que colocar em *C:/Users/NomeDoUsuário/AppData/Local/nvim/init.vim*. Temos então que criar uma pasta chamada ***nvim*** na pasta *AppData* e dentro dela criar um arquivo chamado ***init.vim***.

No Linux temos que colocar em *~/.config/nvim/init.vim*. Temos então que criar uma pasta chamada ***nvim*** na pasta *~/.config* e dentro dela criar um arquivo chamado ***init.vim***.

Configurações globais do Vim e estrutura

A primeira coisa a se fazer é realizar as configurações globais. Isso varia com o gosto pessoal e existem muitas opções e variáveis. Abaixo estão as minhas escolhas, que é o satisfatório para a maior parte dos casos:

```
" Global Sets
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
syntax on            " Enable syntax highlight
set nu               " Enable line numbers
set tabstop=4        " Show existing tab with 4 spaces width
set softtabstop=4    " Show existing tab with 4 spaces width
set shiftwidth=4     " When indenting with '>', use 4 spaces width
set expandtab        " On pressing tab, insert 4 spaces
set smarttab         " insert tabs on the start of a line according to
shiftwidth
set smartindent      " Automatically inserts one extra level of
indentation in some cases
set hidden           " Hides the current buffer when a new file is
openned
set incsearch        " Incremental search
set ignorecase       " Ingore case in search
set smartcase        " Consider case if there is a upper case character
set scrolloff=8      " Minimum number of lines to keep above and below
the cursor
set colorcolumn=100  " Draws a line at the given line to keep aware of
the line size
set signcolumn=yes   " Add a column on the left. Useful for linting
set cmdheight=2      " Give more space for displaying messages
set updatetime=100   " Time in miliseconds to consider the changes
set encoding=utf-8   " The encoding should be utf-8 to activate the font
icons
set nobackup         " No backup files
set nowritebackup    " No backup files
set splitright       " Create the vertical splits to the right
set splitbelow       " Create the horizontal splits below
set autoread         " Update vim after file update from outside
set mouse=a          " Enable mouse support
filetype on          " Detect and set the filetype option and trigger
the FileType Event
```

```
filetype plugin on   " Load the plugin file for the file type, if any
filetype indent on   " Load the indent file for the file type, if any
```

Cada linha possui um comentário explicando. Fique livre para mudar alguma configuração. Se quiser ver mais detalhes assista o vídeo.

Remaps

Ao longo da configuração vamos adicionar vários remaps, então já vou deixar um seção pronta no arquivo específica para Remaps. A linha abaixo vai organizar os remaps no arquivo:

```
" Remaps """""""""""
" remaps aqui
```

Autocmd

É muito comum adicionarmos auto comandos no vim, que são tarefas que são automaticamente ativadas de acordo com as opções configuradas. Também já vou deixar uma seção pronta para irmos preenchendo ao longo do vídeo:

```
" autocmd """""""""""
" autocmds aqui
```

Plugin - Vim Plug

O vim trabalha com plugins e existem vários gerenciadores disponíveis. Vamos usar o Plug, que é um dos gerenciadores de Plugins mais usados. Então colocaremos no topo do arquivo do vim são as seguintes linhas:

```
call plug#begin()
" Plugins aqui
call plug#end()
```

Porém para começarmos a instalar os Plugins, precisamos primeiramente instalar o Plug, para que o vim consiga entender a sintaxe e instalar os plugins. O site do Plug é:

- https://github.com/junegunn/vim-plug

Nesta página conseguimos ver os comandos de instalação para os vários tipos de sistemas operacionais. Existem comandos específicos para o Vim e para o Neovim. No caso do Windows os comandos são:

Para o vim:

```
iwr -useb https://raw.githubusercontent.com/junegunn/vim-
plug/master/plug.vim |`
    ni $HOME/vimfiles/autoload/plug.vim -Force
```

Para o Neovim:

```
iwr -useb https://raw.githubusercontent.com/junegunn/vim-
plug/master/plug.vim |`
    ni "$(@($env:XDG_DATA_HOME, $env:LOCALAPPDATA)[$null -eq
$env:XDG_DATA_HOME])/nvim-data/site/autoload/plug.vim" -Force
```

Com o *Plug* instalado, agora é só instalar os plugins que queremos e instalá-los com o comando:

```
:PlugInstall
```

Sempre que adicionar um novo plugin no arquivo de configuração do vim, reinicie o vim/neovim e rode o comando acima para que o plugin seja instalado.

Temas

Para começar vamos colocar um tema de cores. Existem muitos e é até difícil escolher. Eu vou escolher o tema Sonokai:

- https://github.com/sainnhe/sonokai

A documentação do Sonokai fica nesta página:

- https://github.com/sainnhe/sonokai/blob/master/doc/sonokai.txt

Primeiramente temos que instalar o Sonokai. Instalamos como um plugin, então adicionamos a seguinte linha na seção de Plugins:

```
  Plug 'sainnhe/sonokai'
```

Após adicionar esta linha no arquivo de configuração do vim, reinicie o vim/neovim e rode o comando *PlugInstall* para que o Sonokai seja instalado. Esse padrão se aplica para todos os outros plugins.

Existem várias configurações para o Sonokai, eu vou usar as configurações abaixo, com o estilo Andromeda:

```

```

```
" Themes
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""
if exists('+termguicolors')
  let &t_8f = "\<Esc>[38;2;%lu;%lu;%lum"
  let &t_8b = "\<Esc>[48;2;%lu;%lu;%lum"
  set termguicolors
endif

let g:sonokai_style = 'andromeda'
let g:sonokai_enable_italic = 1
let g:sonokai_disable_italic_comment = 0
let g:sonokai_diagnostic_line_highlight = 1
let g:sonokai_current_word = 'bold'
colorscheme sonokai

if (has("nvim")) "Transparent background. Only for nvim
    highlight Normal guibg=NONE ctermbg=NONE
    highlight EndOfBuffer guibg=NONE ctermbg=NONE
endif
```

A parte de termguicolors é para ativa cores de 24 bits. Isso não está disponível em todos os terminais, por isso temos que verificar se o terminal suporta. No final coloco a configuração para deixar o background transparente, com isso conseguimos ver a imagem de fundo do terminal. Isso só funciona corretamente no NeoVim, por isso colocamos a checagem para ver se é o NeoVim antes de configurar.

AirLine

Seguindo nos plugins visuais, vamos agora instalar o Airline, que adiciona uma barra na parte de baixo com informações úteis. O site do Airline é:

- https://github.com/vim-airline/vim-airline

Para instalar, adicionamos os seguintes plugins:

```
Plugin 'vim-airline/vim-airline'
Plugin 'vim-airline/vim-airline-themes'
```

O tema Sonokai que usamos também tem um tema para o airline. Para ativar colocamos isso na seção de temas do arquivo:

```
let g:airline_theme = 'sonokai'
```

Lembrando de rodar o “*:PlugInstall*” sempre que adicionar um novo plugin, para que o vim/neovim o instale.

Para a configuração do Ariline, temos as seguintes linhas:

```
" AirLine
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"""""""""""""""""
let g:airline#extensions#tabline#enabled = 1
let g:airline_powerline_fonts = 1
```

Devicons

Este plugins ativa vários ícones na interface do vim. Para isso o seu terminal tem que estar usando alguma das fontes to NerdFonts que tem os ícones. Esta é uma configuração do terminal. No meu caso eu uso a fonte Hack, que pode ser baixada no seguinte endereço:

- https://www.nerdfonts.com/
- https://github.com/ryanoasis/nerd-fonts/tree/master/patched-fonts/Hack

Instale essa fonte e configure o seu terminal para usar a fonte Hack. No caso do Windows recomendo usar o Windows Terminal, que pode ser baixado na Store da Microsoft.

Veja o link na introdução para ver o vídeo onde mostro como configurar o *Windows Terminal*.

Uma vez que o seu terminal esteja usando uma NedFont, adicione o plugin do Devicon conforme a linha abaixo:

```
Plug 'ryanoasis/vim-devicons'
```

Para mais informações acesse o site do devicons:

- https://github.com/ryanoasis/vim-devicons

Polyglot

O Polyglot adiciona syntax highlighting pra várias linguagens. Basta adiciona o plugin:

```
Plug 'sheerun/vim-polyglot'
```

NerdTree

O NerdTree é um plugin bem legal para podermos manipular arquivos. Teremos mais formas de fazer esse tipo de coisas, mas recomendo ter o NerdTree como mais uma opção. O site do NerdTree é:

- https://github.com/preservim/nerdtree

Para instalar, adicionamos os seguinte plugin:

```
Plug 'preservim/nerdtree'
```

No site do NerdTree tem muitos exemplos de configurações e autocmds disponíveis. Aqui vou usar a seguinte linha:

```
nmap <C-a> :NERDTreeToggle<CR>
```

Isso vai mapear o **** para ativar ou desativar o NerdTree. Como uso o NerdTree apenas para navegar nos arquivos do projeto, apenas essa configuração basta. Fique livre para adicionar outras que achar interessante para o seu caso.

Somado a essa configuração, adicionarei as seguintes linhas na seção de Remap:

```
" Shortcuts for split navigation
map <C-h> <C-w>h
map <C-j> <C-w>j
map <C-k> <C-w>k
map <C-l> <C-w>l
```

Essas linhas são úteis não apenas com o NerdTree mas com todos os outros splits no geral. Com essas linhas apertamos o Ctrl e navegamos entre os splits com o *"hjkl"*, o que agiliza muito a navegação entre os splits.

Para finalizar com o NerdTree, vou adicionar mais dois plugins para adicionar elementos visuais no NerdTree. O primeiro colocar cores diferentes nos ícones de acordo com a linguagem e o segundo adiciona ícones específicos para o Git.

```
Plug 'tiagofumo/vim-nerdtree-syntax-highlight'
Plug 'Xuyuanp/nerdtree-git-plugin'
```

ALE

Com o *ALE* (Asynchronous Lint Engine) conseguimos configurar linters e fixers para certas linguagens. Várias linguagens usam o *COC*, porém em alguns casos onde a linguagem não tem suporte no COC, podemos facilmente adicionar no ALE. O site é:

- https://github.com/dense-analysis/ale

Para instalar o plugin, usamos a seguinte linha:

```
Plug 'dense-analysis/ale'
```

Para as configurações temos as linhas abaixo:

```
" ALE
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""""""""""""
let g:ale_linters = {
\}

let g:ale_fixers = {
\   '*': ['trim_whitespace'],
\}

let g:ale_fix_on_save = 1
```

No momento está tudo vazio, exceto pela funcionalidade *trim_whitespace*, que elimina todos os espaços extras do arquivo quando salvamos. Em vídeos futuros vou utilizar o ALE para fazer o linting de algumas linguagens, então já estou deixando tudo preparado.

COC (Conquer of Completion)

Agora vamos para o *Coc* (Conquer Of Completion), para termos code completion, language servers e várias outras funcionalidades que temos em IDEs como o Vscode. O site é:

- https://github.com/neoclide/coc.nvim

O Coc tem muitas configurações e é inviável falar de tudo. O que temos que fazer inicialmente é instalar o plugin e copiar do site as configurações padrão. Para instalar usamos a linha:

```
Plug 'neoclide/coc.nvim' , { 'branch' : 'release' }
```

Para as configurações, aconselho copiar a configuração padrão do site acima, pois ela é grande e vai mudando com o tempo. As configurações do Coc são as que mais poluem o arquivo do vim, então aconselho deixar essas configurações por último no final do arquivo. Veja o vídeo para mais informações de como eu configurei o Coc.

O Coc aceita os seus próprios plugins, e existem muitos plugins para várias linguagens e funcionalidades. Podemos instalar manualmente os plugins usando o comando *:CocInstall*, mas o mais fácil é adicionar a linha abaixo no topo da configuração do Coc:

```
let g:coc_global_extensions = [ ]
```

Com isso podemos colocar os nomes dos plugins nesta lista e ao abrir o vim os novos plugins já são instalados automaticamente, caso ainda não foram instalados.

Muitos plugins tem configurações específicas do Coc, e pra isso temos que criar o arquivo ***"coc-settings.json"*** que contém as configurações das extensões do Coc. No começo este arquivo não existe. Podemos criar manualmente ou criarmos dentro do vim com o comando:

```
:CocConfig
```

Isso vai abrir o arquivo e é possível ver o endereço na parte de baixo da tela, caso queira abrir manualmente. No momento colocaremos a seguinte configuração:

```
{
    "languageserver": {

    }
}
```

Em vídeos futuros colocaremos as configurações das linguagens e dos language servers neste aqui, que já está com o esqueleto pronto.

Um outro detalhe é que talver no seu terminal a sequência **** não funcione, devidos aos terminais do Windows. Essa sequência é usada para aparecer a caixinha de sugestões do auto-complete. Uma possibilidade é trocar para o ****, caso essa sequência faça muita falta pra você.

Coc Snippets

A primeira extensão do Coc que iremos utilizar é o *Coc-Snippets*, que serve para adicionar snippets para várias linguagens. Desta forma quando aparecer o menu de auto complete, vai aparecer algumas opções com um ”~” no final, que indica que são widgets que começam com aquele padrão sendo digitado. Alguns widgets tem vários campos, que podem ser passados usando o ******, conforme mapeamento indicado mais abaixo. O site do Coc-snippets é:

- https://github.com/neoclide/coc-snippets

Para instalar, adicionamos o ‘coc-snippets’ na lista de extensões do Coc (“coc_global_extensions”), conforme linha abaixo:

```
let g:coc_global_extensions = [ 'coc-snippets', ]
```

A instalação é feita utomaticamente quando reiniciarmos o vim. Isso vai apenas adicionar a engine de snippets, porém os snippets em si temos que adicionar como um plugin, conforme linha abaixo:

```
Plug 'honza/vim-snippets'
```

As configurações são as mesmas do site do Coc-snippets:

```
" Coc Snippets
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""


" Use <C-l> for trigger snippet expand.
imap <C-l> <Plug>(coc-snippets-expand)

" Use <C-j> for select text for visual placeholder of snippet.
vmap <C-j> <Plug>(coc-snippets-select)

" Use <C-j> for jump to next placeholder, it's default of coc.nvim
let g:coc_snippet_next = '<c-j>'

" Use <C-k> for jump to previous placeholder, it's default of coc.nvim
let g:coc_snippet_prev = '<c-k>'

" Use <C-j> for both expand and jump (make expand higher priority.)
imap <C-j> <Plug>(coc-snippets-expand-jump)

" Use <leader>x for convert visual selected code to snippet
xmap <leader>x  <Plug>(coc-convert-snippet)

inoremap <silent><expr> <TAB>
```

```
      \ pumvisible() ? coc#_select_confirm() :
      \ coc#expandableOrJumpable() ? "\<C-r>=coc#rpc#request('doKeymap',
['snippets-expand-jump',''])\<CR>" :
      \ <SID>check_back_space() ? "\<TAB>" :
      \ coc#refresh()

function! s:check_back_space() abort
  let col = col('.') - 1
  return !col || getline('.')[col - 1]  =~# '\s'
endfunction

let g:coc_snippet_next = '<tab>'
```

Colocamos isso no final do arquivo junto com as outras configurações do Coc.

Coc-explorer

A próxima extensão do Coc é a Coc-explorer, que é uma alternativa ao NerdTree. Eu utilizo ambas as extensões no meu dia a dia. O site do coc-explorer é:

- https://github.com/weirongxu/coc-explorer

Para instalar colocamos a seguintes string na lista de extensões do coc:

```
'coc-explorer'
```

No final do arquivo colocamos as configurações do coc-explorer, que é apenas uma cópia da configuração básica mostrada no site:

```
" Coc Explorer
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""
:nnoremap <space>e :CocCommand explorer<CR>

let g:coc_explorer_global_presets = {
\   '.vim': {
\     'root-uri': '~/.vim',
\   },
\   'cocConfig': {
\      'root-uri': '~/.config/coc',
\   },
\   'tab': {
\     'position': 'tab',
\     'quit-on-open': v:true,
```

```
\   },
\   'tab:$': {
\     'position': 'tab:$',
\     'quit-on-open': v:true,
\   },
\   'floating': {
\     'position': 'floating',
\     'open-action-strategy': 'sourceWindow',
\   },
\   'floatingTop': {
\     'position': 'floating',
\     'floating-position': 'center-top',
\     'open-action-strategy': 'sourceWindow',
\   },
\   'floatingLeftside': {
\     'position': 'floating',
\     'floating-position': 'left-center',
\     'floating-width': 50,
\     'open-action-strategy': 'sourceWindow',
\   },
\   'floatingRightside': {
\     'position': 'floating',
\     'floating-position': 'right-center',
\     'floating-width': 50,
\     'open-action-strategy': 'sourceWindow',
\   },
\   'simplify': {
\     'file-child-template': '[selection | clip | 1] [indent][icon | 1]
[filename omitCenter 1]'
\   },
\   'buffer': {
\     'sources': [{'name': 'buffer', 'expand': v:true}]
\   },
\ }

" Use preset argument to open it
nnoremap <space>ed :CocCommand explorer --preset .vim<CR>
nnoremap <space>ef :CocCommand explorer --preset floating<CR>
nnoremap <space>ec :CocCommand explorer --preset cocConfig<CR>
nnoremap <space>eb :CocCommand explorer --preset buffer<CR>

" List all presets
nnoremap <space>el :CocList explPresets
```

Temos então que podemos abrir o explorer fazendo , ou pra ir mais rápido, . Dentro do explorer é só dar um “?” para ver a lista de comandos. Dá pra criar arquivos apenas apertando o “a”, e diretórios com o “A”.

Para finalizar a configuração do coc-explorer, adicionamos as seguintes linhas no arquivo ***“coc-settings”***:

```
"explorer.width": 30,
"explorer.icon.enableNerdfont": true,
"explorer.previewAction.onHover": false,
"explorer.keyMappings.global": {
    "<cr>": ["expandable?", "expand", "open"],
    "v": "open:vsplit",
    "h": "open:split"
},
"explorer.git.enable": true
```

Com isso habilitamos a NerdFont, o git e adicionamos os comandos “v” e “h” para abrir os arquivos em modo split vertical ou horizontal, para quem preferir.

Auto Pair

Esta extensão adiciona pares para vários caracteres. O site é:

- https://github.com/jiangmiao/auto-pairs

Já funciona muito bem logo de cara, mas existem várias opções de configuração. Uma alternativa a este plugin é o coc-pairs, mostrado abaixo.

coc-pairs (opcional)

Outra extensão útil do Coc é o coc-pairs, que adiciona automaticamente os pares para vários caracteres. A página é:

- https://github.com/neoclide/coc-pairs

Para adicionar basta inserir a string abaixo na lista de extensões do Coc:

```
'coc-pairs'
```

Telescope

Agora vamos adicionar uma extensão específica do NeoVim, que é a Telescope. O Telescope é um fuzzy finder, que serve para buscar arquivos e strings dentro do projeto. A página é:

- https://github.com/nvim-telescope/telescope.nvim

Para instalar o Telescope, adicionamos os plugins:

```
if (has("nvim"))
    Plug 'nvim-lua/plenary.nvim'
    Plug 'nvim-telescope/telescope.nvim'
endif
```

Como este plugin é específico do NeoVim, estou criando uma seção apenas para o NeoVim usando o "if" acima. Portanto o Vim normal não vai as linhas que estão dentro do "if".

Porém também precisamos instalar o "ripgrep" e o "fd", para conseguirmos fazer as buscas por palavras. A página destes projetos são:

- https://github.com/BurntSushi/ripgrep
- https://github.com/sharkdp/fd

No Windows podemos instalar usando o "scoop", de acordo com as instruções no site acima:

```
scoop install ripgrep
scoop install fd
```

Para instalar o scoop é só seguir as instruções no site:

```
https://scoop.sh/
```

No caso temos que rodar no powershell o comando:

```
iwr -useb get.scoop.sh | iex
```

Outra opção é baixar o binário no site do github acima e colocar o executável em alguma pasta do Path.

Para as configurações, adicionamos as seguintes linhas:

```
if (has("nvim"))

    " Telescope
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
""""""""""""
    nnoremap <leader>ff <cmd>Telescope find_files<cr>
    nnoremap <leader>fg <cmd>Telescope live_grep<cr>
    nnoremap <leader>fb <cmd>Telescope buffers<cr>
    nnoremap <leader>fh <cmd>Telescope help_tags<cr>
endif
```

Remaps

Agora vamos focar nos remaps que iremos adicionar para facilitar mais a nossa vida. Todos os remaps serão colocados na seção de Remap que criamos lá no início, pra deixar tudo organizado. A maioria dos novos remaps trabalham sob o comando “t”, que peguei como base para a maioria dos meus remaps pessoais.

Criação de novas linhas

Aqui vou remapear o “o”, que serve para adicionar uma linha abaixo e entrar no modo insert, para adicionar outros comandos que tem a ver com adicionar novas linhas.

```
" Adding an empty line below, above and below with insert
nmap op o<Esc>k
nmap oi O<Esc>j
nmap oo A<CR>
```

Então o antigo “o” agora ficou “oo”. O “op” apenas adiciona uma linha abaixo, ficando no mesmo lugar e continuando no modo normal. O “oi” apenas adiciona uma linha acima, ficando no mesmo lugar e continuando no modo normal.

Criação de Tabs

Agora o “te” agiliza a criação de uma tab nova.

```
" Create a tab
nmap te :tabe<CR>
```

Navegar nos buffers

Com o “ty” e o “tr” conseguimos navegar facilmente nos buffers.

```
" Navigate between buffers
nmap ty :bn<CR>
nmap tr :bp<CR>
```

Deletar buffers

```
" Delete a buffer
nmap td :bd<CR>
```

Manipulação de splits

Os remaps abaixo agilizam a criação de splits verticais e horizontais e o fechamento dos splits.

```
" Create splits
nmap th :split<CR>
nmap tv :vsplit<CR>

" Close splits and others
nmap tt :q<CR>
```

Atalho para comandos.

Este remap serve para agilizar a chamada de comandos do terminal. Em vez de digitar “:!” antes do comando, apenas aperto “tc” e isso já adiciona o “:!” agilizando o processo.

```
" Call command shortcut
nmap tc :!
```

Autocmd

Os autocomandos servem para executar comandos automaticamente quando alguma coisa acontece.

Nesta configuração vou adicionar um autocomando para fazer o highlight de todas as palavras que se encontrar sob o cursor, o que é algo normal em vários editores.

```
function! HighlightWordUnderCursor()
    if getline(".")[col(".")-1] !~# '[[:punct:][:blank:]]'
        exec 'match' 'Search' '/\V\<'.expand('<cword>').'\>/'
    else
        match none
    endif
endfunction

autocmd! CursorHold,CursorHoldI * call HighlightWordUnderCursor()
```

Adicionar estas linhas na seção de *autocmd* do arquivo de configuração.

Arquivos no Linux

No início da página mostrei como instalar o vim e o neovim no linux. Mostrei também onde ficam localizados os arquivos de configuração de cada editor.

Então agora é só copiar os arquivos que já estão feitos para o seu sistema. No caso copie os arquivos do vim e o neovim e os do *Coc*. Aí é só abrir os editores e rodar o comando *:PlugInstall* para instalar os plugins.

É comum muitos programadores colocarem os seus arquivos de configuração no Github, pois aí fica disponível para usar em qualquer computador posteriormente.

Agora que você já tem um arquivo de configuração, coloque no seu GitHub. É comum criar um repositório chamado *dotfiles* para colocar os arquivos de configuração, porém fique a vontade para escolher o nome que quiser.

Considerações finais

As configurações que mostrei aqui servem de base para um programador começar a usar o Vim e o Neovim de forma profissional.

A partir daqui é ganhar familiaridade com esses editores e começar a estudar sobre novas coisas que eles oferecem. É só dar uma pesquisada que vocês encontrarão muitas coisas legais que aumentarão a produtividade na criação de código.