

# Criando um ambiente de desenvolvimento com Vim/Neovim

---

[fonte](#)

Durante lives na Twitch, Rattones e outras pessoas me sugeriram fazer um artigo ensinando como transformar um Vim puro á um bom ambiente de desenvolvimento generalista, e será isso que irei mostrar.

## Neovim

---

Primeiramente, neste artigo não irei usar o Vim, mas sim o Neovim. Neovim é um fork do Vim só que com mais contribuições open-source, mais atualizado com recursos como API's para várias linguagens e janelas flutuantes. Por isso ele será usado ao invés do Vim.

## Instalação

---

Precisaremos do Neovim e Node/NPM (Para o autocomplete). As instruções para instalação estão abaixo:

### Ubuntu, Mint e derivados

---

```
sudo apt install neovim nodejs npm git
```

### Fedora

---

```
sudo dnf install neovim nodejs git
```

### Arch

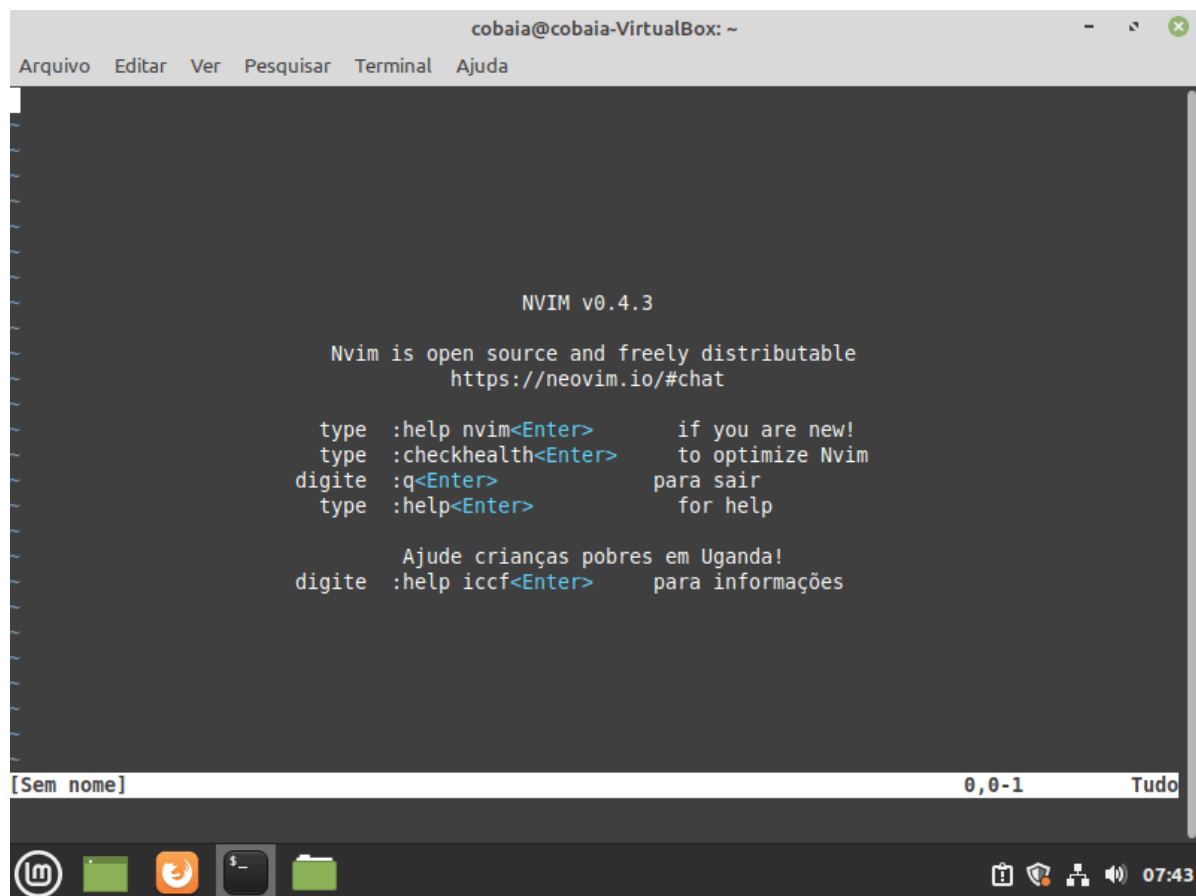
---

```
sudo pacman -S neovim nodejs npm git
```

## Primeiras noções no Neovim

---

Podemos abrir um arquivo vazio novo no Neovim executando `nvim` apenas. Com isso, uma janela como essa irá se abrir:



```
cobaia@cobaia-VirtualBox: ~
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda

NVIM v0.4.3

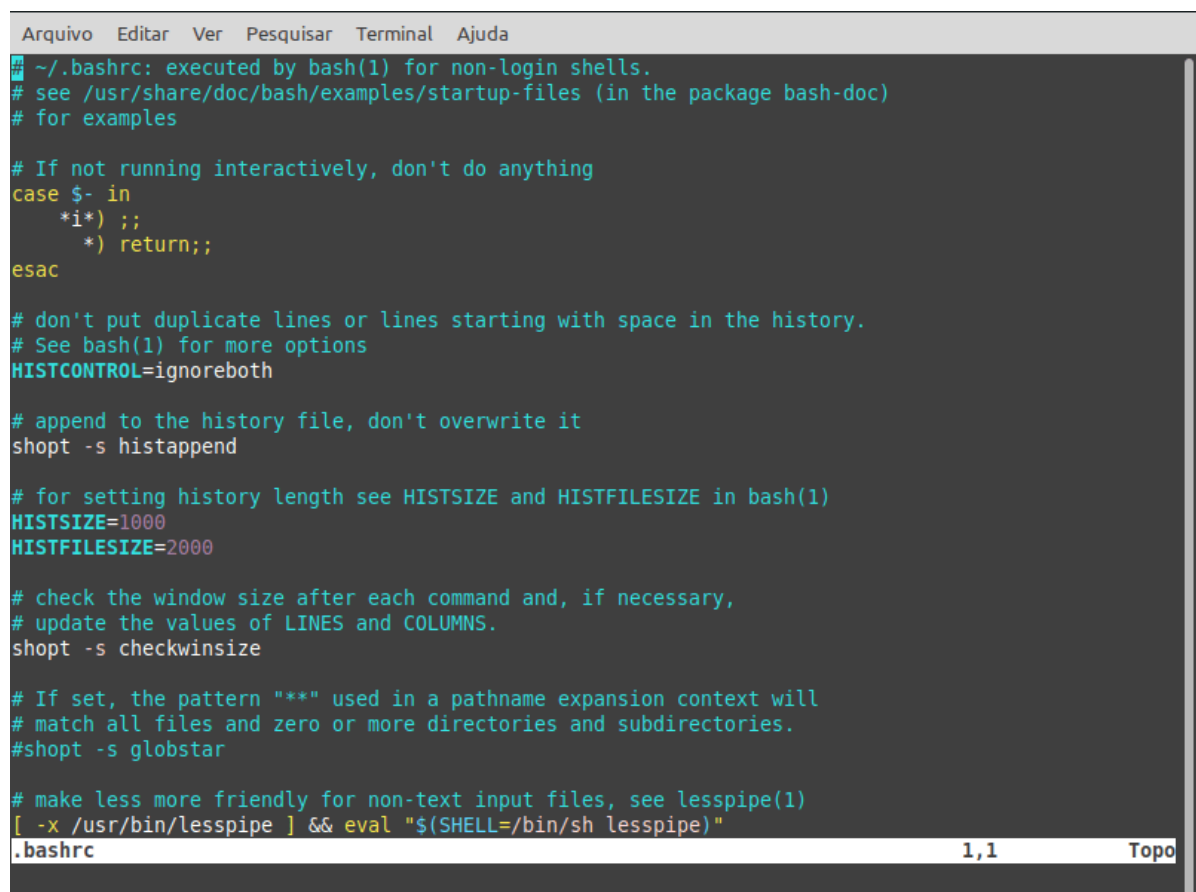
Nvim is open source and freely distributable
https://neovim.io/#chat

type  :help nvim<Enter>      if you are new!
type  :checkhealth<Enter>   to optimize Nvim
digite :q<Enter>             para sair
type  :help<Enter>          for help

Ajude crianças pobres em Uganda!
digite :help iccf<Enter>    para informações

[Sem nome] 0,0-1 Tudo
```

Também podemos especificar um arquivo, abrindo esse arquivo dentro do editor, como por exemplo: `nvim meuArquivo.txt`, uma janela como essa irá se abrir:



```
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda

~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
  *i*) ;;
  *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "*" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

.bashrc 1,1 Topo
```

A maioria dos plugins são compatíveis tanto com Vim e com Neovim. Além das configurações e teclas. Logo toda vez que estiver escrito "Vim" neste artigo, a dica também valerá para o Neovim.

Dentro do Vim (e dentro do Neovim também) temos o conceito de modos. Há 4 principais modos dentro do Vim:

## Normal

---

O modo normal é o principal modo do Vim. Dentro desse modo a maioria das teclas tem uma função específica, como deletar um caractere, ir para outro modo, mover o cursor, etc. Alguns exemplos são:

- `h` ou seta para esquerda | Mover o cursor um caractere á esquerda
- `j` ou seta para baixo | Mover o cursor um caractere para baixo
- `k` ou seta para cima | Mover o cursor um caractere para cima
- `l` ou seta para direita | Mover o cursor um caractere á direita
- `x` | Deletar o caractere atual
- `dd` | Remove uma linha
- `u` | Retroceder uma ação
- `<CTRL>+R` | Refazer uma ação

Entre outras várias teclas.

## Inserção

---

O modo de inserção será o modo onde iremos inserir texto. Dentro deste modo toda tecla comum insere texto dentro do arquivo. Podemos entrar dentro do modo de inserção normalmente apertando `i` e começando a escrever. Podemos sair do modo de inserção apertando a tecla `<ESC>`. Voltando ao modo normal.

Uma dica é usar o `O` para entrar no modo de inserção inserindo uma linha abaixo, e com o `o` para inserir uma linha acima. Isso é útil para adicionar linhas mais rapidamente.

## Visual

---

O modo visual é o modo de seleção. Dentro desse modo poderemos selecionar texto para copiar, remover, editar, etc. Para acessar o modo visual, esteja no modo normal e aperte a tecla `v`. Para apagar todo o texto selecionado por exemplo, selecione o texto e aperte `x`.

## Comando

---

O modo de comando é o modo onde podemos usar comandos do Vim ou dos plugins que iremos inserir. Esse modo pode ser acessado usando a tecla `:` no modo normal. Experimente digitar `:echo "Hello World"` no modo de comando para ver um `Hello World` na barra do seu Vim.

Alguns comandos são:

- `:w` | Salvar arquivo
- `:q` | Sair do arquivo
- `:q!` | Forçar saída do arquivo (caso o arquivo não esteja salvo)
- `:wq` | Salvar e sair do arquivo

## Vim Plug

---

Agora vamos começar a parte dos Plugins. Primeiramente iremos precisar de um gerenciador de Plugins. Há vários ótimos disponíveis, mas neste tutorial iremos usar o Vim Plug. O Vim Plug é bem simples de adicionar, remover e gerenciar os plugins.

Para instalar, use este comando em qualquer distribuição Linux:

```
sh -c 'curl -fLo
"${XDG_DATA_HOME:-$HOME/.local/share}"/nvim/site/autoload/plug.vim --
create-dirs \
    https://raw.githubusercontent.com/junegunn/vim-
plug/master/plug.vim'
```

## Instalando Plugins

---

Para instalar um plugin, primeiramente precisaremos criar o nosso arquivo de configuração do Neovim. Primeiramente, crie um diretório `nvim` dentro de `~/.config`, com por exemplo:

```
mkdir ~/.config/nvim
```

Agora, dentro do diretório `nvim`, iremos criar um arquivo `init.vim` com as configurações, logo, você pode fazer isso com:

```
nvim ~/.config/nvim/init.vim
```

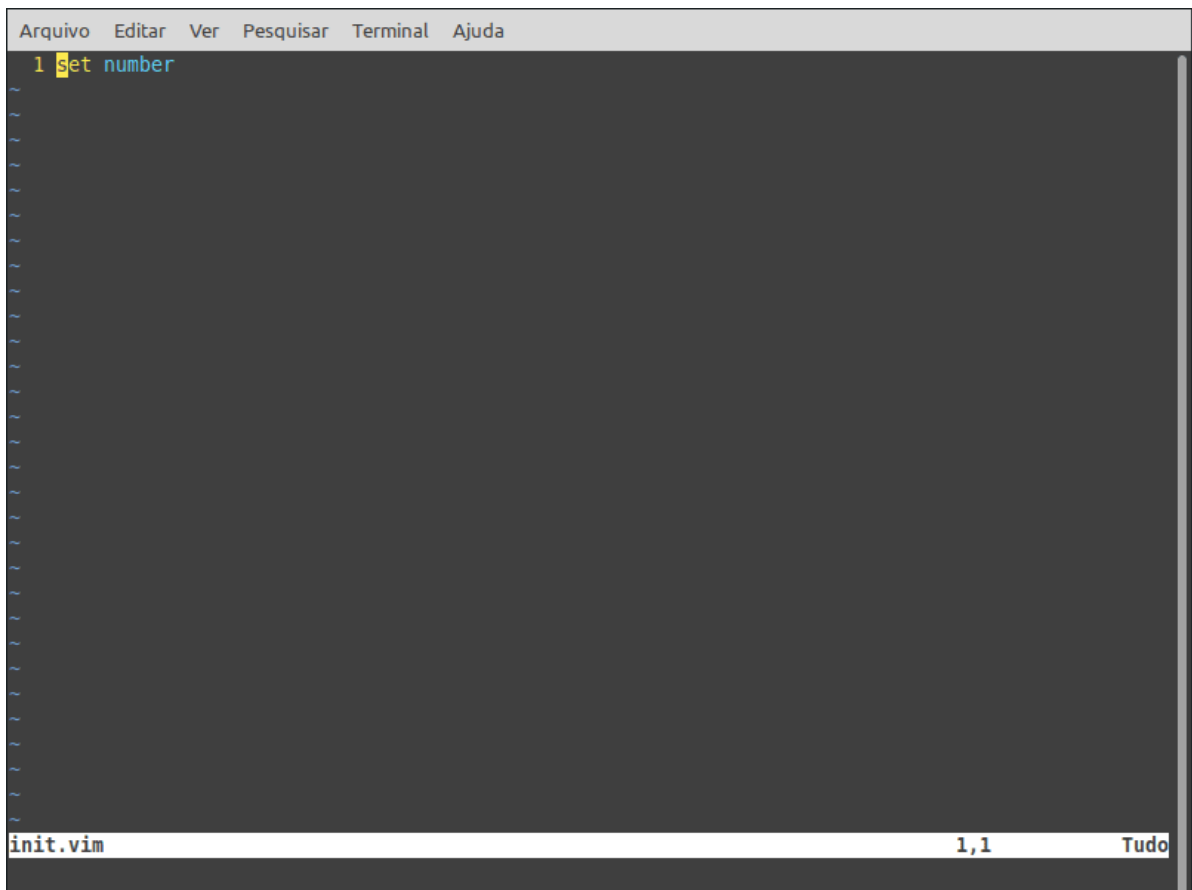
Apenas por teste, abra esse arquivo, aperte `i` para ir ao modo de inserção, digite:

```
set number  
set termguicolors
```

Agora aperte `<ESC>`, e `:wq` para salvar o arquivo.

Agora, caso você entre no arquivo novamente, você verá números que representam cada linha do arquivo!

A outra configuração que fizemos `set termguicolors` habilita melhor compatibilidade com temas. Isso será útil para instalar um tema personalizado ao seu Neovim.



Agora, iremos inserir abaixo dessa configuração dos números, este código:

```
set number  
  
call plug#begin()  
  
call plug#end()
```

Agora, iremos inserir dentro destes dois `call`, os plugins. Vamos supor que você queria usar o tema de cores Dalton:



Para instalar qualquer plugin, precisaremos colocar `Plug` seguido do usuário/repostório. Para instalar especificamente o Dalton será por exemplo assim:

```
set number

call plug#begin()

Plug 'lissaferreira/dalton-vim'

call plug#end()
```

Agora, salve e feche o arquivo com `:wq`, entre novamente e digite `:PlugInstall` no modo de comando para instalar o tema.

Toda vez que você adicionar um novo plugin, lembre de rodar `:PlugInstall` para fazer a instalação.

Após instalar o tema, vá na linha abaixo de `call plug#end()` e digite `color dalton` para definir o Dalton como o tema padrão do seu Neovim. Ao final, o arquivo ficará assim:

```
set number

call plug#begin()

Plug 'lissaferreira/dalton-vim'

call plug#end()

color dalton
```

Após isso, o Dalton já será o tema padrão do seu Neovim!

## Plugins Funcionais

Agora iremos instalar alguns plugins funcionais para o Neovim. Que trarão novas funcionalidades e facilidades pra transformar um Neovim puro em um bom ambiente de desenvolvimento.

## Vim Polyglot

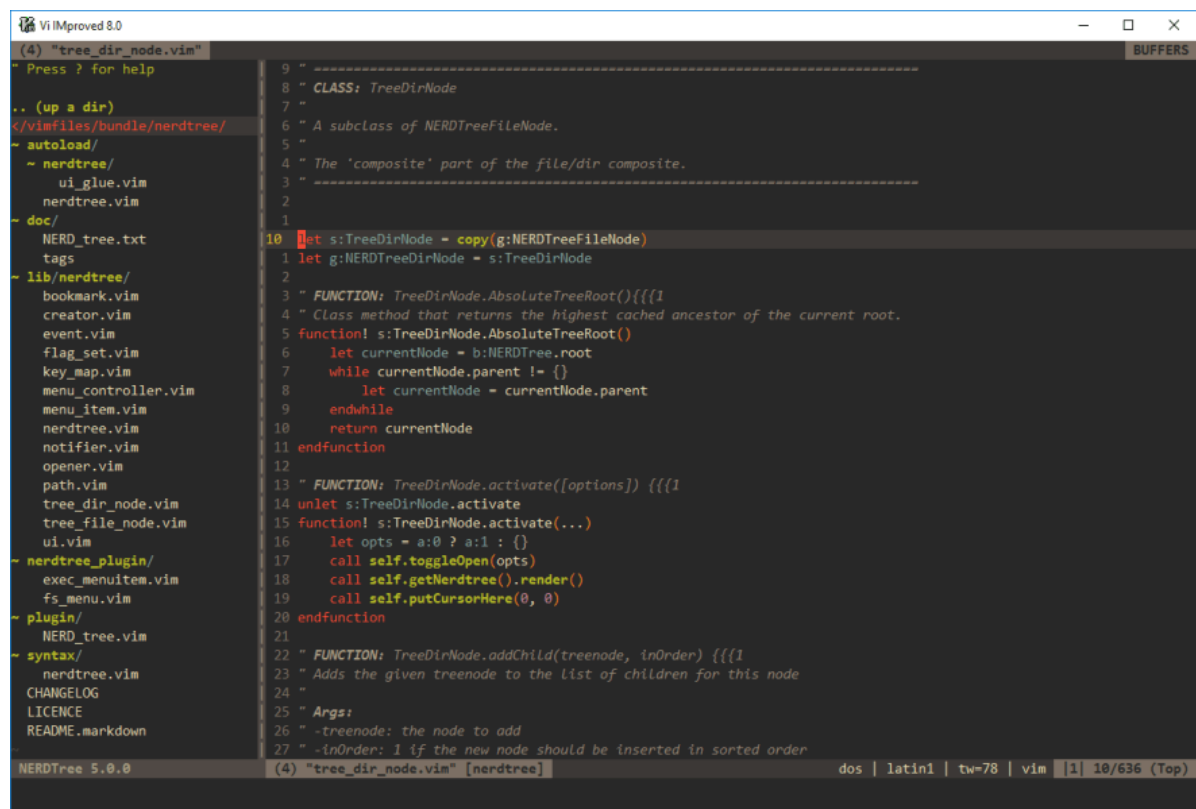
Vim Polyglot é um plugin que possibilita *syntax highlight* para várias linguagens no Vim. Para instalar, adicione

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

Entre os dois `call` dentro do seu `init.vim`

## NERDtree

NERDtree É uma barra de navegação por arquivos e diretórios. Facilita na edição de múltiplos arquivos.



Para instalar, coloque esse `Plug` no seu `init.vim`

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

Agora devemos criar um **map**, que são atalhos para comandos do Vim. Normalmente, após o `:PlugInstall`, você teria que executar `:NERDTree` no modo de comando, mas podemos criar um map para facilitar isso:

Insira abaixo do `call plug#end()` este conteúdo:

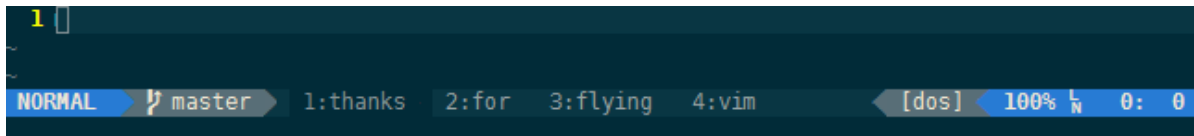
```
nnoremap <C-n> :NERDTreeToggle<CR>
```

Com isso, após salvar e entrar novamente em algum arquivo, usando **CTRL + N** A barra lateral do NERDTree irá aparecer.

## Vim Airline

Vim Airline é uma barra personalizada para melhorar a beleza, e possibilitar que você veja os buffers (arquivos abertos) de forma gráfica e facil.

GIF



Para instalar o Vim Airline, coloque este **Plug** no seu **init.vim**

```
Plug 'vim-airline/vim-airline'
```

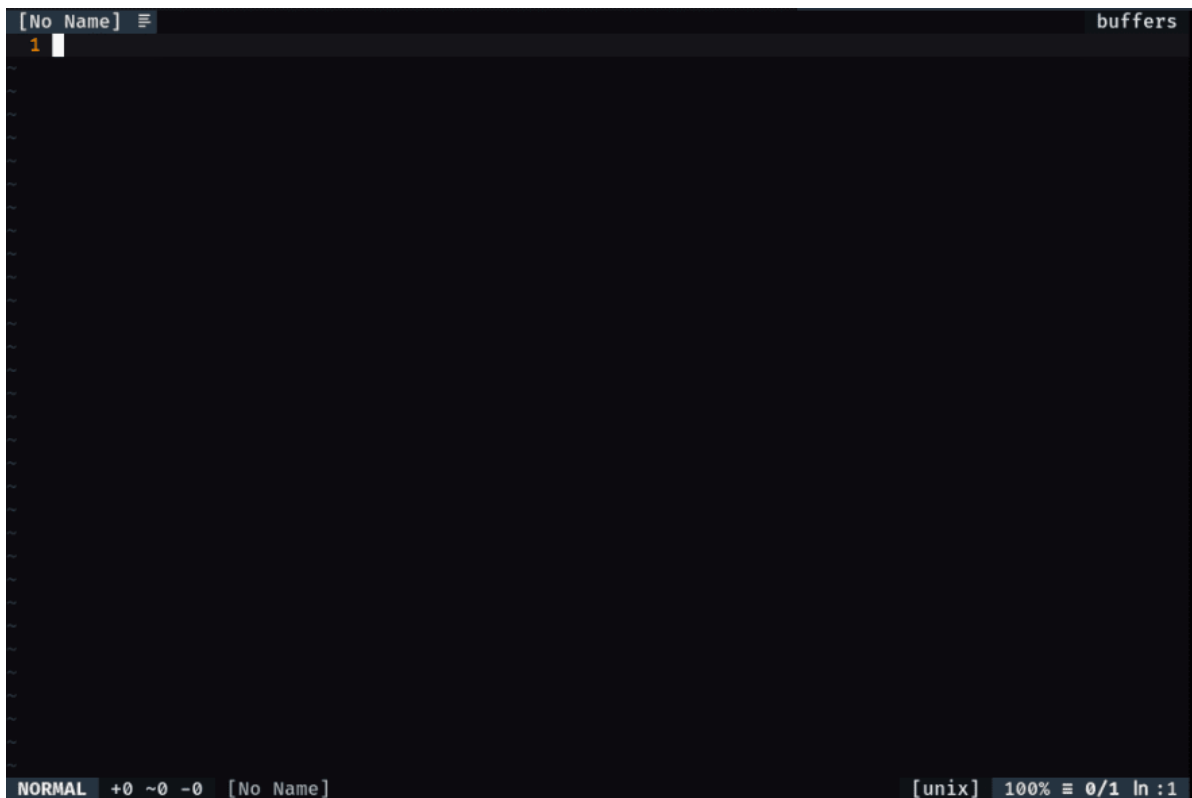
Agora, precisaremos configurar algumas configurações do Airline, deixando mais funcional e bonito. Um exemplo de configuração é essa:

Sempre quando é algo não relacionado á instalação de plugins, copie fora dos dois **call**, para separar as instalações dos plugins das configurações.

```
let g:airline#extensions#tabline#enabled = 1
let g:airline#extensions#tabline#show_buffers = 1
let g:airline#extensions#tabline#switch_buffers_and_tabs = 1
let g:airline#extensions#tabline#tab_nr_type = 1
let g:airline_theme='dalton'
```

Agora você terá todos os arquivos que você abriu em uma barra cima, e com o tema Dalton que instalamos anteriormente.





## CoC.nvim

CoC.nvim é um autocomplete para várias linguagens. O CoC funciona apenas no Neovim pois é usado internamente a API do Neovim, para acessar os dados sobre o arquivo e aparecer as janelas que sugerem as palavras.

GIF

```
import snippetManager from './snippets/manager'
import sources from './sources'
import { Autocmd, OutputChannel } from './types'
import clean from './util/clean'
import workspace from './workspace'
const logger = require('./util/logger')('plugin')

export default class Plugin extends EventEmitter {
  private ready = false
  private handler: Handler
  private infoChannel: OutputChannel

  constructor(public nvim: Neovim) {
    super()
    Object.defineProperty(workspace, 'nvim', {
      get: () => this.nvim
    })
    this.addMethod('hasSelected', () => {
      return completion.hasSelected()
    })
    this.addMethod('listNames', () => {
```

Para instalar, adicione este `Plug`:

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

Agora, depois do CoC instalado, use o comando `CocInstall` seguido do pacote da linguagem que você deseja. Para por exemplo, instalar o pacote do Javascript, TypeScript e JSON, use:

```
:CocInstall coc-json coc-tsserver
```

E além disso precisaremos configurar os atalhos do CoC. Para assim usar `<TAB>` para dar foco nas opções, poder usar as setas para mover e `<ENTER>` para selecionar. Coloque isso no final do arquivo.

```
" Configurações do CoC.nvim

inoremap <silent><expr> <TAB>
  \ pumvisible() ? "\<C-n>" :
  \ <SID>check_back_space() ? "\<TAB>" :
  \ coc#refresh()
inoremap <expr><S-TAB> pumvisible() ? "\<C-p>" : "\<C-h>"
```

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

inoremap <silent><expr> <cr> pumvisible() ? coc#_select_confirm()
  \: "\<C-g>u\<CR>\<c-r>=coc#on_enter()\<CR>"

" Fim das configurações do CoC.nvim
```

Pronto, agora você já tem um bom autocomplete instalado, Você só vai precisar ver quais são os nomes dos pacotes de autocomplete do CoC das linguagens que você usa, e instalar com `:CocInstall`.

## ALE

ALE é um analisador de código assíncrono. Com o ALE, você conseguirá ver os erros que há no seu código diretamente no editor.

GIF

```
test.js
class Foo {
  bar() {
    var x = 3

    x = x + 2;
  }
}
~
~
~
~
N ~/js-test/test.js 3:13
```

Para instalar, adicione este `Plug`:

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

## Vim Coloresque

Vim Coloresque é um plugin que adiciona destaque de cores para nomes de cores, hexadecimais, rgba,etc. Facilitando pra editar por exemplo, arquivos CSS.

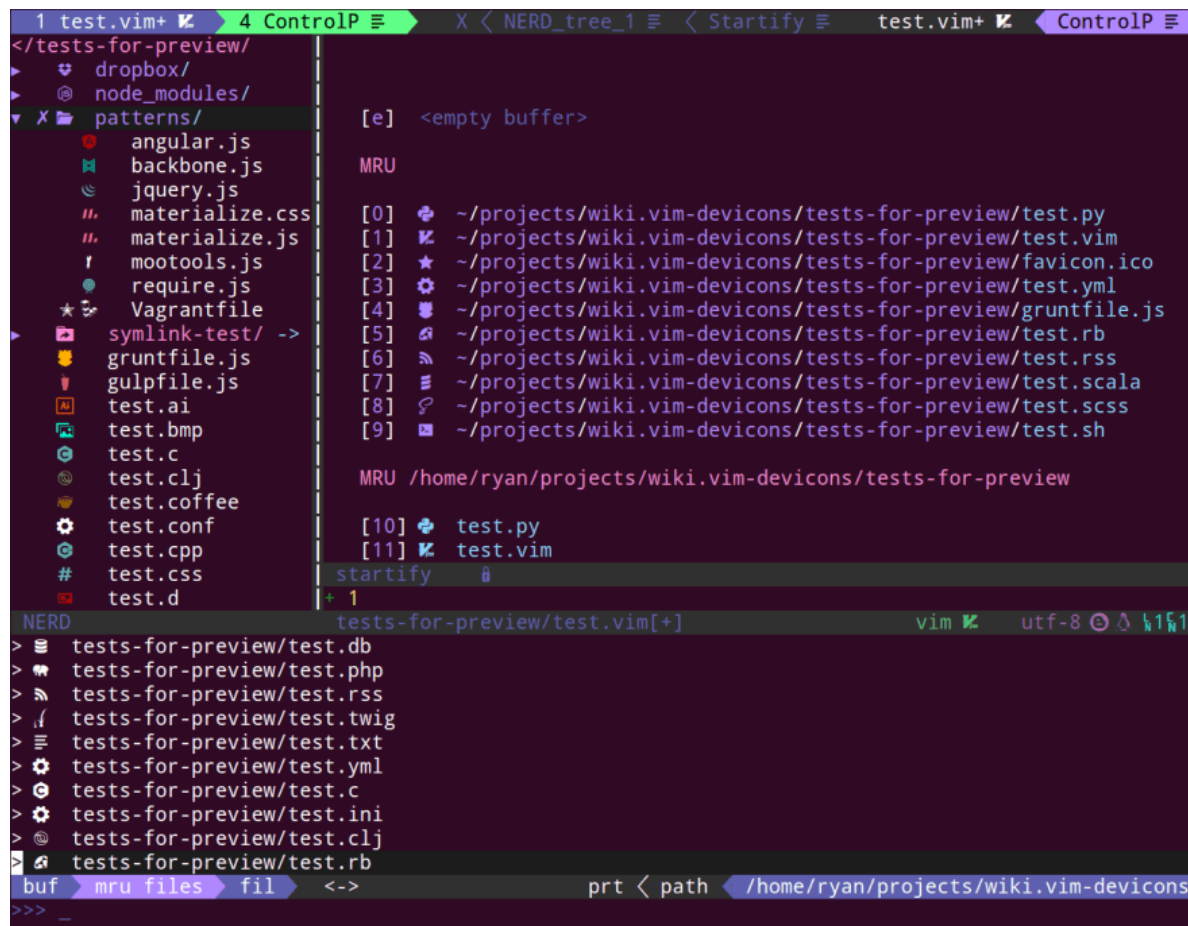
```
style.css (~) - GVIM
1 body {
2   color: yE110w;
3   color: rEd;
4   color: #000;
5   color: #fF0000;
6   color: rgb(10, 129, 10);
7   color: rgb(110, 107, 51, 0.5);
8   color: Aqua;
9   color: sIlver;
10  color: hsl(5, 50%, 40%);
11  color: hsla(5, 50%, 67%, 0.1);
12  color: RoyalBlue;
13 }
~
~
~
~
~
NORMAL style.css  unix | utf-8 | css  23%  LN  3:15
```

Para instalar o Vim Coloresque adicione este `Plug`:

```
Plug 'gko/vim-coloresque'
```

## Vim Devicons

Vim Devicons é um plugin para adicionar ícones para certas linguagens, frameworks, tipos de arquivos, etc. ao Vim.



Para instalar, adicione este `Plug`:

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

E também rode este comando para adicionar as fontes:

```
cd ~/.local/share/fonts && curl -fLo "Droid Sans Mono for Powerline Nerd  
Font Complete.otf" https://github.com/ryanoasis/nerd-  
fonts/raw/master/patched-  
fonts/DroidSansMono/complete/Droid%20Sans%20Mono%20Nerd%20Font%20Comple  
te.otf
```

Também é bom usar junto com o NERDTree Syntax Highlight



O NERDTree Syntax Highlight pode ser instalado adicionando esse `Plug`:

```
Plug 'tiagofumo/vim-nerdtree-syntax-highlight'
```

## Vim IndentGuides

Vim IndentGuides é um plugin que destaca a indentação do arquivo que você está editando.

```

9 function! s:Setl
10 " space indents
11 |   if index(g:ind
12 |       if !a:user_
13 |           silent! sy
14 |           silent! sy
15 |       endif
16 |       execute "hi
17 |       execute "hi
18 " tab indents
19 |   |   if g:indentg
20 |   |   |   execute pr
21 |   |   endif
22 |   |   execute 'syn
23 |   |   execute pri
24 |   endif
25 endfunction

```

Para instalar o Vim IndentGuides adicione este `Plug`:

```
Plug 'thaerkh/vim-indentguides'
```

E no final do seu `init.vim`, adicionar os caracteres que irão aparecer na guia da indentação. Tanto para espaço quanto para TAB:

```

" Configurações do Vim IndentGuides

let g:indentguides_spacechar = '|'
let g:indentguides_tabchar = '|'

" Fim das configurações do Vim IndentGuides

```

## Lexima.vim

Levxima.vim é um plugin de auto-pairs para o Neovim. Com ele, sempre que você digitar `{`, `'`, `"`, `[` ou outras teclas, já será autocompletado o fechamento, como `}`, `'`, `"`, `]`.

Para instalar esse Plugin, adicione este `Plug`:

```
Plug 'cohama/lexima.vim'
```

Agora, no final do arquivo você deve adicionar qual será o caractere que irá adicionar as linhas de indentação, uma sugestão é essa:

```
" Configurações do lexima.vim

let g:indentguides_spacechar = '|'
let g:indentguides_tabchar = '|'

" Fim das configurações do lexima.vim
```

## Maps Importantes

---

Também é interessante que você tenha algumas maps (atalhos) que facilitem a edição de arquivos. Todas esses maps podem ser facilmente mudados para uma outra tecla.

Alguns exemplos são:

### Salvar arquivos com CTRL + S

---

```
nnoremap <C-s> :w!<CR>
```

### Sair do Vim com CTRL + q

---

```
nnoremap <C-q> :qa<CR>
```

### Alternar entre abas (buffers) com F1 e F2

---

```
nnoremap <F1> :bprevious<CR>
nnoremap <F2> :bnext<CR>
```

### Alternar a posição de uma linha com SHIFT + seta para cima e SHIFT + seta para baixo

---

```
nnoremap <silent> <s-Down> :m +1<CR>
nnoremap <silent> <s-Up> :m -2<CR>
```

### Copiar um texto e enviar esse texto para a área de transferência

---

```
vnoremap <C-c> "+y<CR>
```

# E Agora?

---

Agora você já está com um Neovim bem configurado. Agora você deve aprender mais sobre o Vim, atalhos, sobre os modos, etc. Um **ÓTIMO** vídeo em português para aprender é esse do Calango Hackerclub

E também buscar mais plugins, configurações, adaptações, etc. Específicos para o que você faz no editor. Uma boa dica é trocar o Dalton por um tema que você já utiliza ou outro que você achar agradável.

Muito obrigada por ler ❤️👉 e me segue nas redes, é tudo @lissatransborda 👁👁