

# Minha configuração básica para o Neovim

fonte

Nos últimos artigos, comentei sobre o [porquê](#) de ter adotado o Neovim como meu editor principal e passei o [básico](#) para utilizar o programa. Porém, de fábrica ele não vem com extensões ou com uma aparência agradável. Abaixo, mostro a vocês como configurei o meu Neovim.

As linguagens de programação que uso no meu cotidiano são Javascript (JSX e TypeScript), CSS, Sass e ultimamente estou estudando Python, também. Portanto, a configuração que se seguirá dará suporte a estas tecnologias.

```

NVIM v0.5.1

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

```

*Neovim como vem de fábrica*

## Sobre minhas escolhas para a configuração

Como é de se imaginar, o Neovim é configurado por arquivos de texto.

Existem duas maneiras de configurá-lo: usando a linguagem Lua, ou o VimScript, criado para o Vim. Aqui utilizarei o último, pois este serve tanto para o Vim quanto para o Neovim. Assim eu mato dois coelhos com uma cajadada só.

Não tenho experiência com Lua, e no momento não tenho planos de aprender a linguagem apenas para configurar o programa. Tenho conhecimento de que existem maneiras mais modernas de fazer isto, mas prefiro manter as minhas configurações concisas.

No final você deste guia, você não terá um emaranhado de arquivos e diretórios, como você até pode ter visto em outros locais, mas sim um arquivo único, apenas com o necessário para um uso confortável.

## Editando opções

Como na maioria dos softwares, há uma maneira de editar opções e preferências. Estas podem ser testadas com o comando `:set <nome-da-opção>`. Experimente entrar em uma instância do Vim agora e digite `:set number`, você perceberá que os números de linha aparecerão na lateral. O problema disto é que estes comandos terão de ser repetidos cada vez que você reabrir o programa.

## Salvando configurações

Para manter estas configurações salvas, basta escrever estes comandos em um arquivo na sua pasta de usuário. O nome e local irá variar a depender se você usa o Vim ou o Neovim:

- **Vim:** `<usuário>/vimrc`
- **Neovim:** `<usuário>/config/nvim/init.vim`

Se esses arquivos/diretórios não existirem na sua pasta pessoal, basta criá-los manualmente. Lembrando que arquivos que iniciam com "." são ocultos por padrão em sistemas Unix (Linux, BSDs e MacOS). Portanto, certifique-se que esteja exibindo os arquivos escondidos em seu gerenciador de arquivos ou terminal.

## Opções básicas no arquivo de configuração

Antes de partirmos para coisas mais complexas, como extensões e suas configurações, vamos nos ater ao básico. Existem muitas opções para mexer, porém, as que coloquei abaixo são as que uso atualmente. Perceba que no arquivo ficam escritos os comandos que teriam de ser digitados todas as vezes que o editor abrisse.

```
" Options
set background=dark
set clipboard=unnamedplus
set completeopt=noinsert,menuone,noselect
set cursorline
set hidden
set inccommand=split
set mouse=a
set number
set relativenumber
set splitbelow splitright
set title
set timeoutlen=0
set wildmenu

" Tabs size
set expandtab
set shiftwidth=2
set tabstop=2
```

Eis a explicação de cada comando acima:

- `background=dark`: aplica o conjunto de cores para telas escuras. Não somente o fundo da tela, como pode parecer.

- `clipboard=unnamedplus`: habilita a área de transferência entre o Vim/Neovim e os demais programas do sistema.
- `cursorline`: destaca a linha atual no editor.
- `completeopt`: modifica o comportamento do menu de auto-completar para se comportar mais como uma IDE.
- `hidden`: esconde buffers<sup>1</sup> não usados.
- `inccommand=split`: mostra substituições em uma divisão da janela, antes de aplicar no arquivo.
- `mouse=a`: permite que o editor permita o uso do mouse.
- `number`: mostra o número das linhas na lateral.
- `relativenumber`: mostra as linhas a partir da atual. Útil para auxiliar em comandos que usam mais linhas.
- `splitbelow splitright`: configura o comportamento da divisão da tela com o comando `:split` (dividir a tela horizontalmente) e `:vsplit` (verticalmente). Neste caso, as telas sempre se dividirão abaixo da tela atual e à direita.
- `title`: mostra o título do arquivo
- `tttimeoutlen=0`: tempo em milissegundos para aceitar comandos.
- `wildmenu`: mostra um menu mais avançado para sugestões de auto-completar.
- `expandtab`: transforma tabulações em espaços.
- `shiftwidth=2`: quantidade de espaços ao indentar o texto.
- `tabstop=2`: número de espaços para as tabulações.

## Sintaxe

Para adicionar suporte à sintaxe automática para os arquivos abertos:

```
filetype plugin indent on
syntax on
```

## Suporte a cores

Para habilitar 256 cores no terminal:

```
set t_Co=256
```

Abaixo segue uma lógica que fiz para que o Vim detecte se o sistema suporta maior número de cores. Note que fiz um condicional para detectar se o emulador de terminal é o da Apple. Por algum motivo as cores se comportam diferente nas versões que usei.

```
" True color if available
let term_program=$TERM_PROGRAM

" Check for conflicts with Apple Terminal app
if term_program !=? 'Apple_Terminal'
    set termguicolors
else
    if $TERM !=? 'xterm-256color'
        set termguicolors
    endif
endif
endif
```

Caso você não use o terminal da Apple, ignore o código acima e acrescente apenas o código dentro do `else`.

## Suporte ao itálico verdadeiro

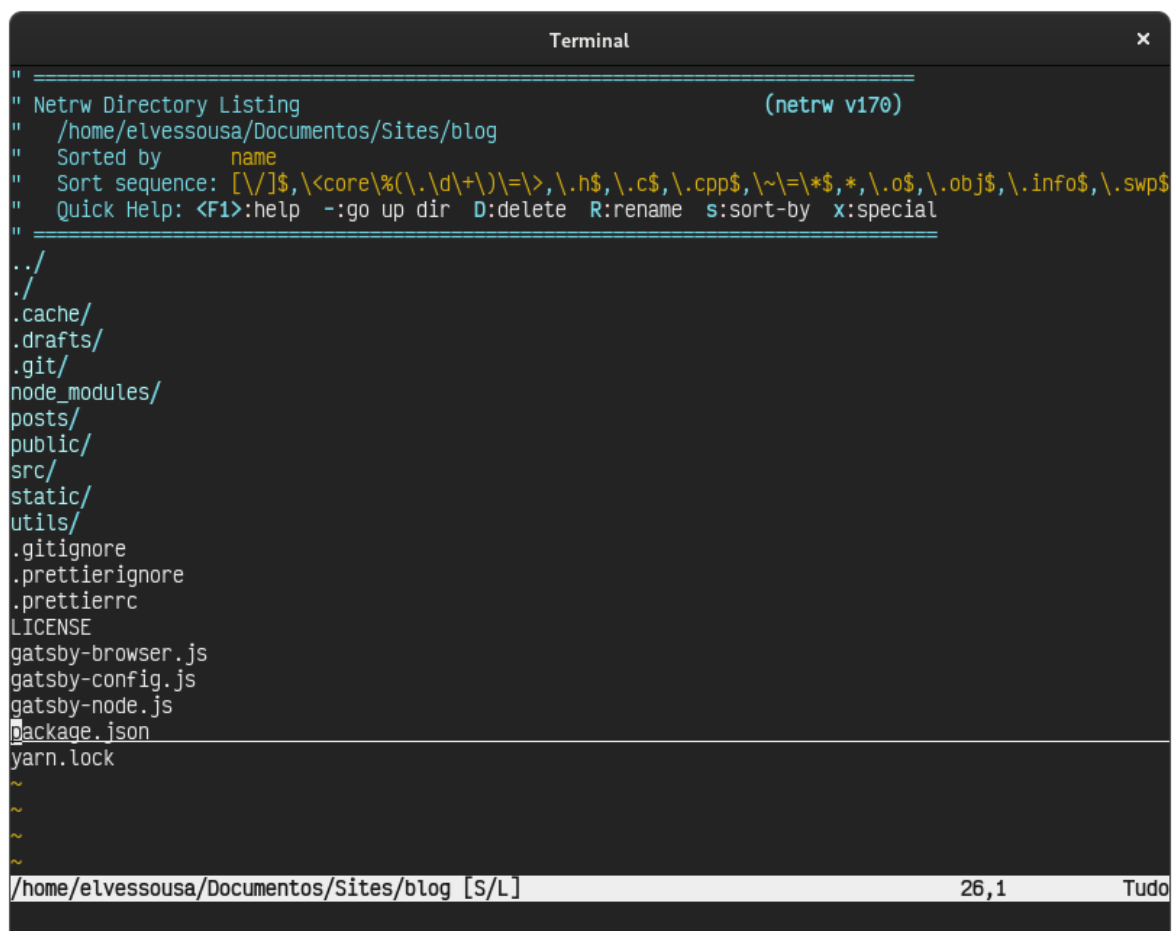
Se você é como eu e gosta de fontes com suporte ao itálico verdadeiro, acrescente estas linhas:

```
" Italics
let &t_ZH="\e[3m"
let &t_ZR="\e[23m"
```

Não me pergunte o que significam estes códigos aí. Só sei que um terminal com suporte a itálicos vai reconhecê-los como itálico.

## NetRW: O gerenciador de arquivos padrão do Vim

*Aviso: Caso já conheça um pouco do Vim e não tenha interesse no NetRW, pule esta parte.*



```
" =====
" Netrw Directory Listing                                (netrw v170)
" /home/elvessousa/Documentos/Sites/blog
"   Sorted by      name
"   Sort sequence:  [\/]$, \<core%\(\.d\+\)\>=\>, \.h$, \.c$, \.cpp$, \~\=\*$, *, \.o$, \.obj$, \.info$, \.swp$
"   Quick Help: <F1>:help  -:go up dir  D:delete  R:rename  s:sort-by  x:special
" =====
.. /
. /
.cache/
.drafts/
.git/
node_modules/
posts/
public/
src/
static/
utils/
.gitignore
.prettierrc
.prettierrc
LICENSE
gatsby-browser.js
gatsby-config.js
gatsby-node.js
package.json
yarn.lock
~
~
~
~
/home/elvessousa/Documentos/Sites/blog [S/L] 26,1  Tudo
```

*NetRW como vem de fábrica*

Se você já usou alguma IDE alguma vez na vida, deve ter percebido que há sempre uma árvore na lateral com os arquivos do projeto. O Vim/Neovim usam por padrão o NetRW, que traz esta mesma função.

Ao abrir o Vim ou o Neovim usando o comando `[n]vim .` para abrir o diretório do projeto, a listagem de arquivos irá aparecer dentro dele. Ele também pode ser aberto usando os comandos `:Ex`, `:Vex` ou `:Sex`.

Você pode usá-lo com as configurações padrão, mas pessoalmente, algumas coisas me incomodam, portanto, usei as configurações a seguir:

```
" File browser
let g:netrw_banner=0
let g:netrw_liststyle=0
let g:netrw_browse_split=4
let g:netrw_altv=1
let g:netrw_winsize=25
let g:netrw_keepdir=0
let g:netrw_localcopydircmd='cp -r'
```

Eis a explicação:

- `g:netrw_banner=0`: esconde a mensagem que aparece no topo por padrão.

- `g:netrw_liststyle=0`

: muda a exibição dos arquivos.

- `0` mostra somente um diretório por vez.
- `1` mostra os dados dos arquivos.
- `2` mostra os arquivos em colunas.
- `3` mostra como uma árvore onde os diretórios abertos ficam expandidos.

- `g:netrw_browse_split=4`

: muda como os arquivos são abertos.

- `1` abre arquivos em uma divisão horizontal.
- `2` abre em uma divisão vertical.
- `3` abre em uma nova aba.
- `4` abre em uma janela anterior, evitando a criação de mais divisões.

- `g:netrw_altv=1`: alterna a exibição do NetRW para a esquerda.
- `g:netrw_winsize=25`: limita o tamanho da janela para 25% do espaço disponível em tela.
- `g:netrw_keepdir=0`: mantém o diretório que acessado anteriormente.
- `g:netrw_localcopydircmd`: modifica o comando usado para copiar arquivos. Por padrão, o NetRW não copia apenas pastas vazias. Para mudar isto, mudei o comando padrão para `cp -r` para que a cópia ocorra recursivamente.

Para facilitar a criação de arquivos, adicionei mais estas configurações:

```
" Create file without opening buffer
function! CreateInPreview()
  let l:filename = input('please enter filename: ')
  execute 'silent !touch ' . b:netrw_curdir.'/'.l:filename
  redraw!
endfunction

" Netrw: create file using touch instead of opening a buffer
function! Netrw_mappings()
  noremap <buffer>% :call CreateInPreview(<cr>
endfunction

augroup auto_commands
  autocmd filetype netrw call Netrw_mappings()
augroup END
```

Isto evita que o NetRW abra uma tela vazia apenas para criar o arquivo.

Se você deseja usar o NetRW, acredito que isto seja o mínimo necessário para usá-lo confortavelmente.

## NetRW e seus problemas

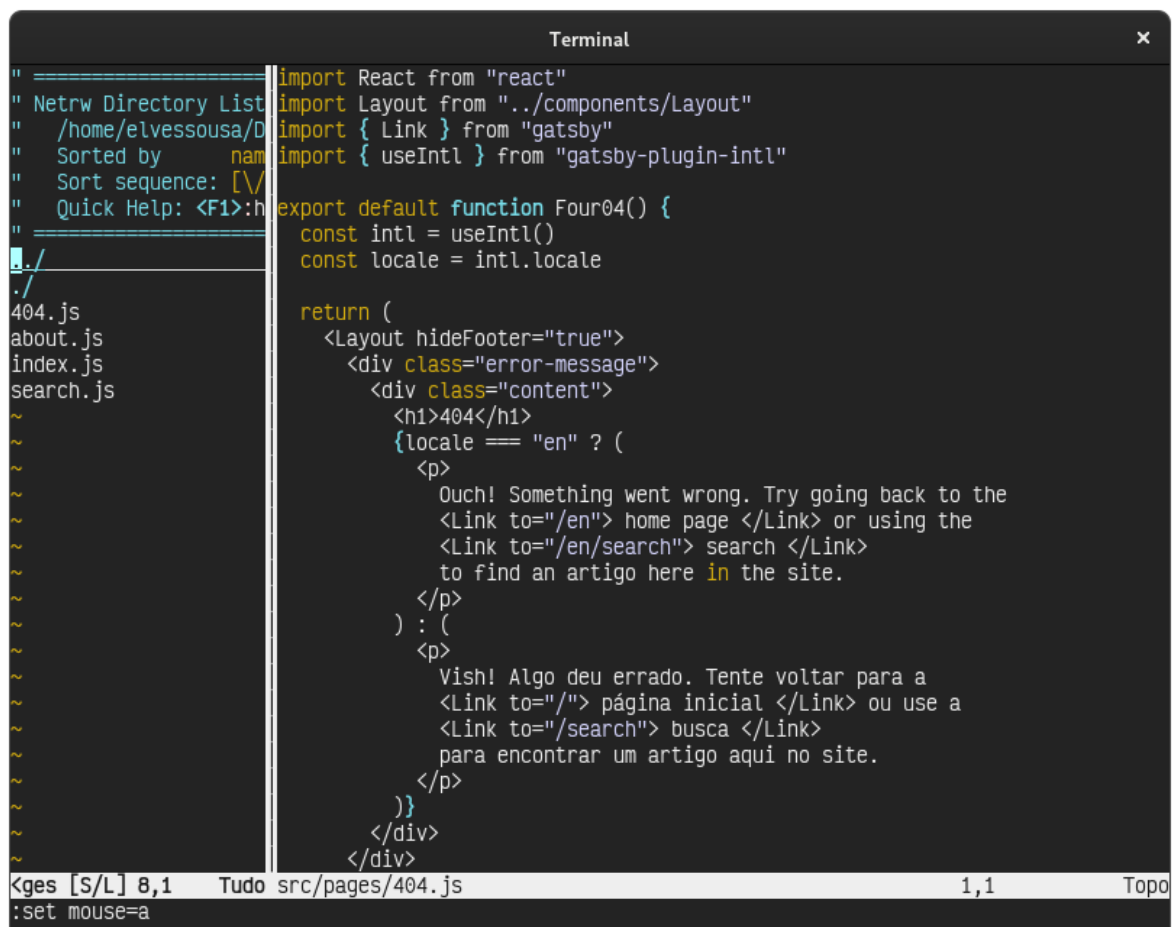
Existem várias críticas ao NetRW, pela forma como causa uma bagunça nos arquivos abertos. Pelo que pesquisei, parece que usar o modo árvore `g:netrw_liststyle=3` costuma piorar este comportamento. Os erros são aleatórios, dificultando a correção e, além disso, o processo para atualizar somente ele é mais complexo do que deveria ser.

Por isso existem várias alternativas ao NetRW, e a mais famosa delas é o NERD Tree, que uso em minha configuração principal.

---

## Extensões

---



```
" Netrw Directory List
/home/elvessousa/D
Sorted by name
Sort sequence: [\\
Quick Help: <F1>:h

import React from "react"
import Layout from "../components/Layout"
import { Link } from "gatsby"
import { useIntl } from "gatsby-plugin-intl"

export default function Four04() {
  const intl = useIntl()
  const locale = intl.locale

  return (
    <Layout hideFooter="true">
      <div class="error-message">
        <div class="content">
          <h1>404</h1>
          {locale === "en" ? (
            <p>
              Ouch! Something went wrong. Try going back to the
              <Link to="/en"> home page </Link> or using the
              <Link to="/en/search"> search </Link>
              to find an artigo here in the site.
            </p>
          ) : (
            <p>
              Vish! Algo deu errado. Tente voltar para a
              <Link to="/"> página inicial </Link> ou use a
              <Link to="/search"> busca </Link>
              para encontrar um artigo aqui no site.
            </p>
          )}
        </div>
      </div>
    </Layout>
  )
}
```

*Estado do Neovim usando as configurações feitas até agora*

## Configuração do instalador

Finalmente, chegou a hora de configurar os plugins. Ao contrário do VS Code ou outros editores, existem diversas opções para instaladores de extensões: Neobundle, Vundle, Vim-Plug, etc. Aqui usarei este último, que parece ser o mais usado hoje em dia. Para instalar, entre no terminal e rode o seguinte comando:

Para o Vim:

```
$ curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

Para o Neovim:

```
$ 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'
```

Esses comandos acima dão conta do necessário para ter o Vim-Plug disponível no seu editor.

## Instalação de extensões

O Vim-Plug lê uma parte do seu arquivo de configuração para localizar a extensão no Github e instalá-la no editor. Essa parte é delimitada pela seguinte estrutura:

```
call plug#begin()
    Plug '<usuário-do-github>/<nome-do-repositório>'
call plug#end()
```

Dá para acrescentar diversas extensões de uma vez, basta acrescentar vários `Plug` conforme o desejado. Existe um site que cataloga estes plugins para facilitar a descoberta, caso queira visitar. Chama-se Vim Awesome. O link está no final do artigo.

## As extensões que uso

Eis o que uso no meu Vim/Neovim:

```
call plug#begin()
    " Appearance
    Plug 'vim-airline/vim-airline'
    Plug 'ryanoasis/vim-devicons'

    " Utilities
    Plug 'sheerun/vim-polyglot'
    Plug 'jiangmiao/auto-pairs'
    Plug 'ap/vim-css-color'
    Plug 'preservim/nerdtree'
    Plug 'kien/ctrlp.vim'

    " Completion / linters / formatters
    Plug 'neoclide/coc.nvim', {'branch': 'master', 'do': 'yarn install'}
    Plug 'plasticboy/vim-markdown'

    " Git
    Plug 'airblade/vim-gitgutter'
call plug#end()
```

- **Vim Airline:** modifica a barra de estado para deixá-la mais agradável que a versão padrão.
- **Vim Devicons:** mostra ícones na interface, baseado em fontes de ícone<sup>2</sup>.
- **Vim Polyglot:** pacote de sintaxes para diversas linguagens de programação.
- **Auto Pairs:** fecha automaticamente os parênteses, colchetes e chaves ao digitar.
- **Vim CSS Color:** mostra as cores (hexadecimal, RGB, HSL) diretamente no código.

- **NERDTree**: barra lateral para acesso aos arquivos do projeto. Substitui o NetRW que comentei anteriormente.
- **CTRLP**: busca rápida de arquivos dentro do projeto.
- **CoC (Conquer of Completion)**: fornece auto-completar de modo bastante semelhante ao disponível no VS Code. Entro em detalhes dele, mais à frente.
- **Vim Markdown**: melhor suporte para sintaxe de arquivos Markdown.
- **Vim GitGutter**: mostra alterações do Git em arquivos abertos.

Escrever estas linhas com os endereços não é o suficiente para instalar as extensões. Primeiro, você deve salvar o arquivo de configuração e reiniciar o Vim, para que o editor obedeça às configurações recém-adicionadas. Após isto, execute o comando `:PlugInstall`.

```
Terminal
2 Updated. Elapsed time: 5.695911 sec.
1 [=]
3
1 - Finishing ... Done!
2 - sobrio: Resolving deltas: 100% (7/7), done.

2 call plug#begin()
1 " Appearance
3 Plug 'elvessousa/sobrio'
1 call plug#end()
2
3 " Window stuff
4 filetype plugin indent on
5 syntax on
6
7 " Options
8 set background=dark
9 set clipboard=unnamedplus
10 set cursorline
11 set hidden
12 "set incommand=split
13 set ttymouse=xterm2
14 set mouse=a
15 set number
16 set path+=**
17 set relativenumber
18 set splitbelow splitright
19 set title
20 set timeoutlen=0
21 set ttyfast
22 set wildmenu
23
24 " Cursors
25 let &t_SI = "\<Esc>[6 q"
26 let &t_SR = "\<Esc>[4 q"
27 let &t_EI = "\<Esc>[2 q"
28
```

*Vim Plug em funcionamento*

Uma janela irá se abrir e mostrar o processo de instalação de cada extensão. Após a instalação, você terá de reiniciar o Vim mais uma vez para usar as novas extensões.

Para desinstalar, remova a linha do plugin que deseja se livrar do arquivo de configuração e após reiniciar o editor, rode o comando `:PlugClean`.

## Esquemas de cor

Se tem algo que é fundamental para uma boa experiência de desenvolvimento, é um bom esquema de cores. Estes são instalados da mesma forma que as extensões.

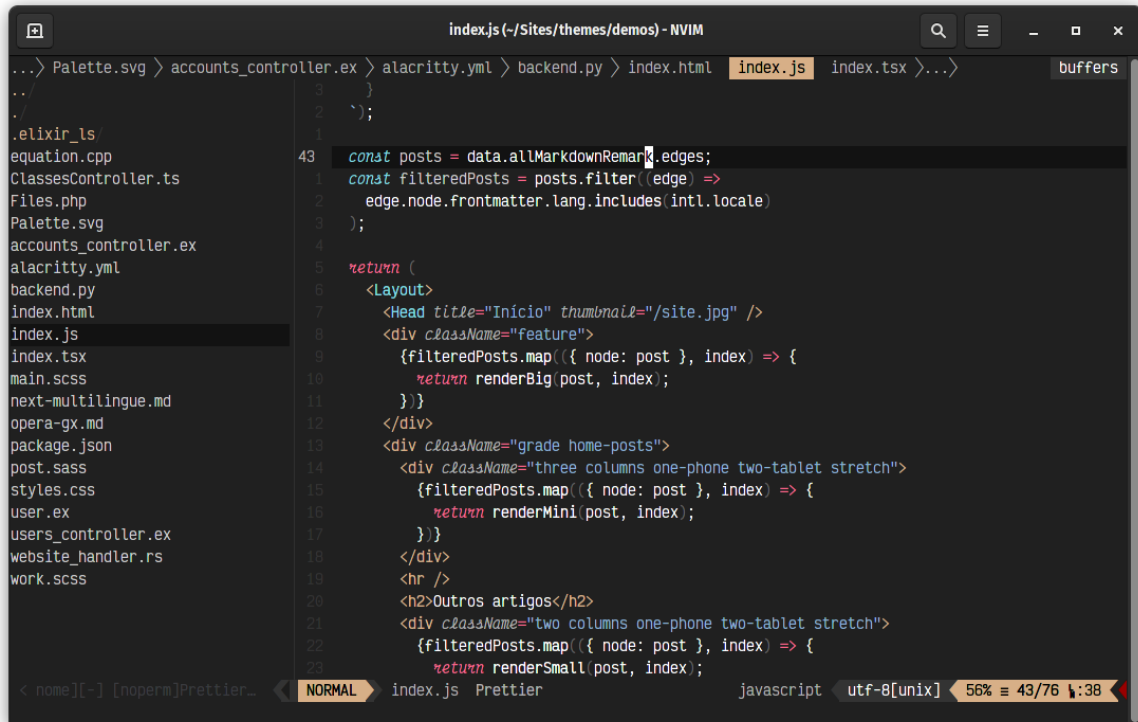
Primeiro, acrescenta-se a linha com o caminho para o repositório Github do tema:



```
call plug#begin()
    Plug 'morhetz/gruvbox'
call plug#end()
```

Para usar o tema, adicione o comando `colorscheme gruvbox` no arquivo de configuração. Isto já será o suficiente para sair daquela tela preta e cores gritantes que vêm por padrão.

## Tema que uso



*Sobrio: o tema que uso no meu dia-a-dia*

Apesar de dar o exemplo com o tema GruvBox, não é este o tema que uso. Esquemas de cor são coisas muito subjetivas e conforme usei este tema, acabei por enjoar. O mesmo aconteceu com outros. Visto que não encontrava um que me agradasse, fiz o meu próprio. Se chama Sobrio. Se quiser testá-lo basta seguir a mesma receita acima, mudando apenas o repositório:

```
call plug#begin()
    Plug 'elvessousa/sobrio'
call plug#end()
```

Depois é só acrescentar a linha `colorscheme sobrio` e reiniciar o editor. Fique à vontade para deixar seu comentário e sugestão sobre ele. Gostaria de saber sua opinião. Se quiser saber mais sobre ele, deixe o link para o site no final do artigo.

---

## Configuração dos plugins

### Devicons

Devicons é uma extensão que adiciona ícones à interface do Vim para efeitos estéticos. Para que ele funcione, é necessário ter fontes com suporte a ícones instaladas no sistema.

Procure por "NERD Fonts" na internet e você encontrará várias delas. A forma de instalar fontes no sistema variam bastante conforme o sistema operacional e não vou entrar em detalhes aqui. Portanto, baixe, instale e aplique a fonte para o seu emulador de terminal. Fora isso, nenhuma outra configuração é necessária.

## Airline



Pouca coisa para configurar aqui. Basicamente é dizer qual tema usar e se você deseja usar o modo "Powerline", que são essas linhas com formas de setas.

```
let g:airline_theme='sobrio'
let g:airline_powerline_fonts = 1
let g:airline#extensions#tabline#enabled = 1
```

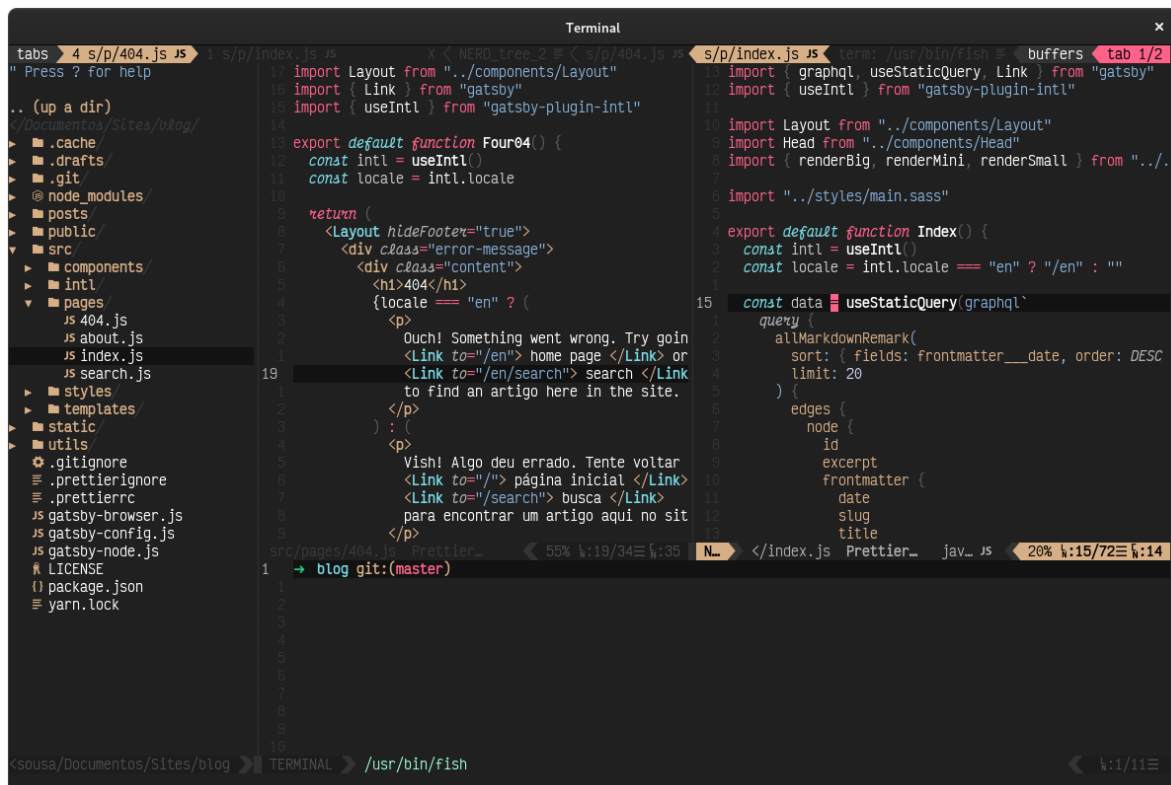
A última opção acima mostra habilita o Airline para a barra de abas.

## NERDTree

NERDTree é o substituto do NetRW. Por padrão já funciona muito bem, apenas adicionei uma linha para mostrar os arquivos ocultos por padrão.

```
" File browser
let NERDTreeShowHidden=1
```

Sabe como é, gosto de ver os arquivos `.env` ou `.gitignore` sem ter de digitar `I` antes.



Neovim com as configurações até agora

## CTRLP

Este plugin dá acesso a uma pesquisa de arquivos do projeto, e ativado pelo atalho de teclado que dá nome à extensão. Se você já usou o VS Code, vai se habituar bem rápido. A única configuração que fiz foi esta:

```
let g:ctrlp_user_command = ['.git/', 'git --git-dir=%s/.git ls-files -oc --exclude-standard']
```

Traduzindo a configuração em português: o CTRLP irá ignorar todos os arquivos e diretórios mencionados no arquivo `.gitignore` do seu projeto.

## Vim Markdown

Se você não usa Markdown, ignore esta parte.

Como utilizo bastante o formato Markdown para escrever em meu blog, instalei esta extensão para melhorar a exibição deste tipo de arquivo no meu Neovim.

```
" Disable math tex conceal feature
let g:tex_conceal = ''
let g:vim_markdown_math = 1

" Markdown
let g:vim_markdown_folding_disabled = 1
let g:vim_markdown_frontmatter = 1
let g:vim_markdown_conceal = 0
let g:vim_markdown_fenced_languages = ['tsx=typescriptreact']
```

Uso cabeçalhos (Frontmatter) em YAML no topo para que o Gatsby faça a mágica dele, e sem esse plugin a sintaxe sai quebrada, assim como os blocos de código, que saíam sem a formatação desejada.

## Coc: Conquer of Completion

Este é o plugin que exige mais configuração. O Conquer of Completion trabalha como um servidor que disponibiliza ferramentas de auto-completar e correção de código, nos mesmos moldes do VS Code. Este suporta extensões também, e estas são responsáveis para adicionar o suporte às linguagens de programação desejadas.

O ponto fraco do CoC é que é baseado em NodeJS, portanto, as dependências dos pacotes instalados ocuparão um certo espaço em disco, como todos os projetos em Node. Dos que testei até agora, foi o que mais atendeu às minhas expectativas, com menos configuração.

Para instalar extensões é simples: basta rodar o comando `:CocInstall <nome-da-extensão>` e as funcionalidades estarão disponíveis para usar.

Eis as extensões que uso:

- **coc-tsserver**: suporte para JavaScript, TypeScript e React.
- **coc-css**: suporte para CSS, Sass e SCSS.
- **coc-eslint**: analisa os arquivos JavaScript e TypeScript.
- **coc-emmet**: adiciona atalhos para escrever menos em arquivos HTML, CSS, Sass, SCSS e JavaScript estendido (JSX, TSX)
- **coc-pyright**: suporte a Python.

- **coc-prettier**: formata o código de acordo com padrões bem aceitos na comunidade.

Para instalar todos de uma vez, basta executar o comando:

```
:CocInstall coc-tsserver coc-css coc-eslint coc-emmet coc-pyright coc-prettier
```

## Prettier com coc-prettier

```
" Language server stuff
command! -nargs=0 Prettier :call CocAction('runCommand', 'prettier.formatFile')
```

O comando acima irá rodar o Prettier e formatar o arquivo sempre que se salva o arquivo. Isso ajuda bastante, pois perco menos tempo dando tabulações para ajustar o código.

## Configuração do Coc em JSON

Fora o `init.vim` ou `.vimrc` que usamos até aqui, eu uso também um arquivo a parte chamado `coc-settings.json`. Com ele é possível modificar o comportamento de suas extensões.

```
{
  "coc.preferences.formatOnSaveFiletypes": [
    "css",
    "sass",
    "scss",
    "markdown",
    "html",
    "json",
    "javascript",
    "javascriptreact",
    "typescript",
    "typescriptreact",
    "python"
  ],
  "python.formatting.provider": "black",
  "python.formatting.blackPath": "~/.local/bin/black",
  "prettier.singleQuote": true,
  "javascript.autoClosingTags": true,
  "typescript.autoClosingTags": true,
  "emmet.includeLanguages": [
    "javascript",
    "javascriptreact",
    "typescript",
    "typescriptreact",
    "html"
  ],
  "codeLens.enable": false
}
```

Eis a explicação das configurações acima:

1. Configurei o CoC para formatar alguns tipos de arquivos ao salvar.
2. Configurei o Prettier para preferir aspas simples na formatação.
3. Informei qual o formatador de código para python, "black" neste caso, e passei o caminho para o black no meu sistema.
4. Habilitei o fechamento automático para tags de JavaScript estendido (JSX)

5. Habilitei o Emmet para mais formatos de arquivo.
6. Desabilitei o Code Lens, pois este dava umas travadas ocasionalmente.

Note que caso você não use Python, basta ignorar estas configurações. Para o formatador de Python funcionar, ele deve estar instalado no sistema:

```
$ pip install black
```

## Atalhos de teclado

Assim como em outros programas, existe uma maneira de configurar atalhos de teclado. Isto pode ser feito para cada modo do editor. As possibilidades são muitas. Para adicionar um atalho de teclado no Vim, basta adicionar a expressão:

```
[letra-inicial-do-modo-alvo]noremap <tecla-ou-letra> :Comando
```

A estranha palavra `noremap` vem de "***Non Recursive Mapping***", "Mapeamento Não-Recurso". Isto significa que o atalho criado não irá sobrescrever outro atalho igual, evitando conflitos.

Como sou um pouco minimalista, não adicionei muitos atalhos.

```
" Normal mode remappings
nnoremap <C-q> :q!<CR>
nnoremap <F4> :bd<CR>
nnoremap <F5> :NERDTreeToggle<CR>
nnoremap <F6> :sp<CR>:terminal<CR>

" Tabs
nnoremap <S-Tab> gT
nnoremap <Tab> gt
nnoremap <silent> <S-t> :tabnew<CR>
```

Eis a explicação dos mapeamentos acima:

- `Ctrl+q`: fecha a tela aberta.
- `F4`: fecha o arquivo aberto.
- `F5`: mostra ou esconde a árvore do NERDTree.
- `F6`: abre uma divisão inferior e abre um terminal nela.
- `Shift+Tab`: alterna para a aba anterior.
- `Shift+t`: cria uma aba.
- `Tab`: alterna para a próxima aba aberta.

Note que há vários `<CR>`. Estes representam todas as vezes que você teria de apertar `Enter` no modo de comando. **CR** é a abreviação de "*Carriage Return*", ou "Retorno do Carro": termo da época em que se usavam máquinas de escrever e hoje entendemos como a tecla `Enter` ou `Return`.

Todos que usei são para o modo normal, por isso `noremap` esta em todas as linhas. Caso algum fosse para o modo de inserção seria `inoremap`, se fosse para o modo visual seria `vnoremap` e assim por diante.

## Comandos automáticos

Para automatizar algumas coisas, o Vim possibilita definir o que ele chama de `autocmd`: são comandos automáticos realizados em determinados eventos do programa.

```
" Auto Commands
augroup auto_commands
    autocmd BufWrite *.py call CocAction('format')
    autocmd FileType scss setlocal iskeyword+=@-@
augroup END
```

Acima defini duas ações: uma para formatar o código Python ao salvar o arquivo, e outra para resolver uma inconsistência que tinha com arquivos SCSS.

---

## Conclusão

Sem dúvidas, este foi o artigo mais longo que fiz até agora, mas se você seguiu as instruções, você já tem um editor bem funcional. É claro que há outros plugins, mas conforme você se acostuma e pega confiança, você testa outros que se adequam melhor às suas necessidades e linguagens de programação.

O ideal é que este arquivo não cresça demais, pois quanto mais extensões, mais isto pode afetar o desempenho. Existem outras configurações que não detalhei aqui no artigo, pois não são importantes para a maioria dos casos. De qualquer forma, eu disponibilizo o link para o repositório com os meus **dotfiles** para você conferir no final do artigo.

Até a próxima!

## Links

- [Meus dotfiles](#)
- [Sobrio](#)
- [Vim Awesome](#)