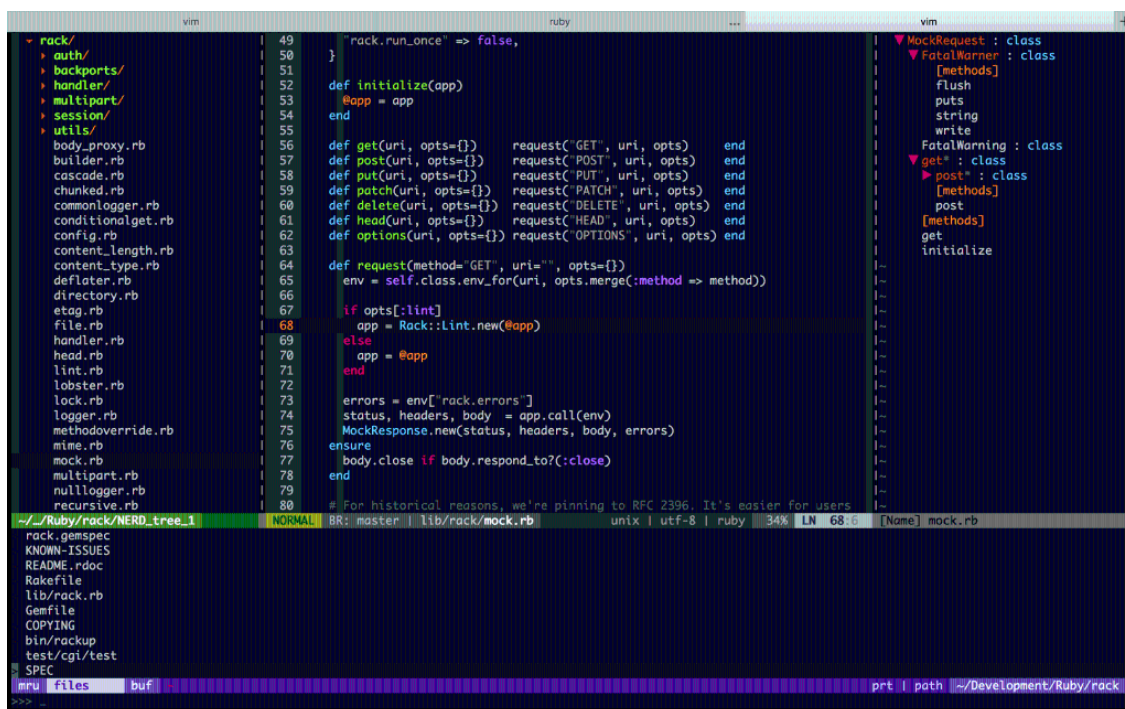


Vim 和 Emacs 一个称为神之编辑器一个被称为编辑器之神，固然很是夸张，但也足以说明这两款软件的优秀和在程序员界的地位。但是它们都已漫长的学习曲线让人望而生畏，阻止了大多数人进入。作为一名几乎完全使用 Vim 写各种代码、文档的人，我想把我自己平时使用的插件和配置整理下来，方便自己的总结和归纳，如果能有幸帮助到一些想学习 Vim 但是又不知道如何入门的人来说，那就再荣幸不过了。

在下面的内容中，我会介绍我使用的插件、Vim 的配置，最后如果你觉得这些配置手动太麻烦的话，我推荐你看我的另一篇文章（[从零搭建和配置 OSX 开发环境](#)），在那篇文章的末尾，我给出了一个自动化配置和管理 Vim 的方法。

先贴一张我的 Vim 的截图：



你看的到的插件

从上面那种截图中肉眼能看到的插件说起，把整个界面按照左窗口、主窗口、右窗口和下窗口命名，依次介绍出现在这个窗口中的主要插件。

主窗口

作为一款主要用于书写代码的文本编辑器，一个足够舒服、靓丽的配色当然是首要考虑的。我使用的配色主题是 molokai（[官方地址](#)），在你安装好了这个插件之后，你需要下面几行配置应用它：

```
" Switch syntax highlighting on, when the terminal has colors
if (t_Co > 2 || has("gui_running")) && !exists("syntax_on")
    syntax on
endif

" Javascript syntax highlight
syntax enable
```

```
" Set syntax highlighting for specific file types
autocmd BufRead,BufNewFile Appraisals set filetype=ruby
autocmd BufRead,BufNewFile *.md set filetype=markdown
autocmd Syntax javascript set syntax=jquery

" Color scheme
colorscheme molokai
highlight NonText guibg=#060606
highlight Folded guibg=#0A0A0A guifg=#9090D0
```

另外一个推荐的 vim 主题是 solarized([官方地址](#))。

在选定了一个适合自己的主题之后，就需要一些配置去解决排版的问题，比如字符编码和缩进等问题。

```
" Backspace deletes like most programs in insert mode
set backspace=2
" Show the cursor position all the time
set ruler
" Display incomplete commands
set showcmd
" Set fileencodings
set fileencodings=utf-8,bg18030,gbk,big5

filetype plugin indent on

" Softtabs, 2 spaces
set tabstop=2
set shiftwidth=2
set shiftround
set expandtab

" Display extra whitespace
set list listchars=tab:».,trail:.

" Make it obvious where 80 characters is
set textwidth=80
set colorcolumn=+1

" Numbers
set number
set numberwidth=5

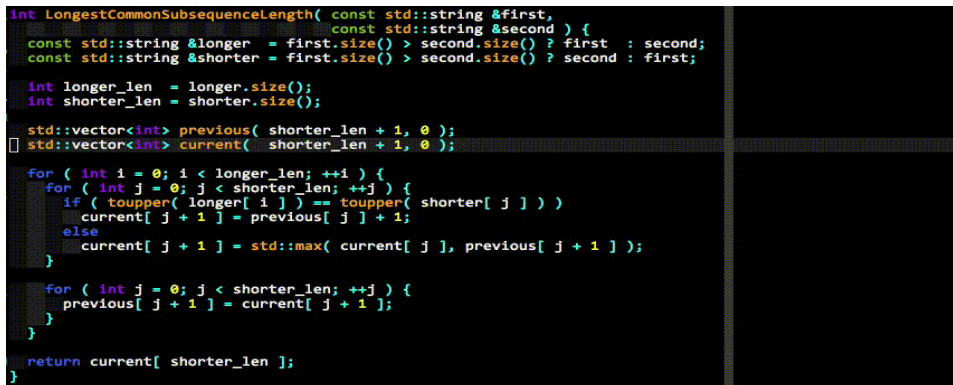
set matchpairs+=<:>
set hlsearch
```

在第 68 行，水平和垂直方向分别有一条高亮条，这是用来表示我当前光标所处于的行和列用的。实现它，只需要几行简单的配置就可以了：

```
" Highlight current line
au WinLeave * set nocursorline nocursorcolumn
au WinEnter * set cursorline cursorcolumn
set cursorline cursorcolumn
```

关于代码补全

有些人可能已经发现了，在我的主窗口中没有演示代码补全的功能，我需要对此做一个说明。我本人不喜欢过于强大的代码补全，所以默认的对于我来说已经完全足够了，如果你需要使用更强大的代码补全，我推荐你使用 YouCompleteMe ([官方地址](#))。



```
int LongestCommonSubsequenceLength( const std::string &first,
                                   const std::string &second ) {
    const std::string &longer = first.size() > second.size() ? first : second;
    const std::string &shorter = first.size() > second.size() ? second : first;

    int longer_len = longer.size();
    int shorter_len = shorter.size();

    std::vector<int> previous( shorter_len + 1, 0 );
    std::vector<int> current( shorter_len + 1, 0 );

    for ( int i = 0; i < longer_len; ++i ) {
        for ( int j = 0; j < shorter_len; ++j ) {
            if ( toupper( longer[ i ] ) == toupper( shorter[ j ] ) )
                current[ j + 1 ] = previous[ j ] + 1;
            else
                current[ j + 1 ] = std::max( current[ j ], previous[ j + 1 ] );
        }

        for ( int j = 0; j < shorter_len; ++j )
            previous[ j + 1 ] = current[ j + 1 ];
    }

    return current[ shorter_len ];
}
```

左窗口

左窗口是一个用于浏览目录结构的插件 nerdtree ([官方地址](#))。同样一些简单的配置之后，它便能为你提供提供一个方便够用的功能。

```
" NERD tree
let NERDChristmasTree=0
let NERDTreeWinSize=35
let NERDTreeChDirMode=2
let NERDTreeIgnore=['\~$', '\.pyc$', '\.swp$']
let NERDTreeShowBookmarks=1
let NERDTreeWinPos="left"
" Automatically open a NERDTree if no files where specified
autocmd vimenter * if !argc() | NERDTree | endif
" Close vim if the only window left open is a NERDTree
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTreeType") &&
b:NERDTreeType == "primary") | q | endif
" Open a NERDTree
nmap <F5> :NERDTreeToggle<cr>
```

右窗口

在我的截图中，右窗口陈列出了我当前打开的 rb 文件中声明的类、变量及方法等等。这是一款名叫 tagbar 的插件，它为我们提供了一个简单的方式去浏览当前文件的结构，并且支持在各个标签之间快捷的跳转。同理，安装之后，需要一些配置：

```
" Tagbar
let g:tagbar_width=35
let g:tagbar_autofocus=1
nmap <F6> :TagbarToggle<CR>
```

如果你发现默认的 Tagbar 不能支持你的语言，比如 Css, Clojure, Markdown 等等，你可以参照[这篇文章](#)为它提供额外的支持。

下窗口

下窗口包含了两个部分：一个是用于全局搜索的窗口和一个状态条。

全局搜索是一个基于文件名的搜索功能，可以快速定位一个文件。这是 ctrlp 这个插件提供的功能。下面是 ctrlp 的一些配置：

```
" ctrp
set wildignore+=*/tmp/*,*.so,*.swp,*.zip,*.png,*.jpg,*.jpeg,*.gif " MacOSX/Linux
let g:ctrlp_custom_ignore = '\v[\/]\. (git|hg|svn)$'
```

ctrlp 默认会使用 grep 进行搜索，效率低且慢。所以，我使用了 ag 去替换默认的搜索功能。ag 是一款轻量级的搜索工具，速度非常快。为了集成 ag，需要添加下列配置：

```
if executable('ag')
  " Use Ag over Grep
  set grepprg=ag\ --nogroup\ --nocolor
  " Use ag in CtrlP for listing files.
  let g:ctrlp_user_command = 'ag %s -l --nocolor -g "'
  " Ag is fast enough that CtrlP doesn't need to cache
  let g:ctrlp_use_caching = 0
endif
```

下面状态条中会依次显示：当前模式、Git 分支、文件路径、文件是否保存以及当前所载行和列的信息。这是通过 vim-powerline 来实现的。其中显示 Git 信息需要配合 vim-fugitive 插件一些使用。

```
set laststatus=2 " Always display the status line
set statusline+=%{fugitive#statusline()} " Git Hotness
```

小结

通过以上的配置，你就可以拥有一些如第一张图所示的那样，看起来还不错的编辑器。当然，Vim 之所以如此倍受推崇，只是依靠这些还是远远不够的。接下来，我要介绍一些看不见的插件来实实在在的提升 Vim 体验。

看不见的实用插件

现代化的插件管理

在我的另一篇文章中（[从零搭建和配置 OSX 开发环境](#)），我已经详细介绍过 Vundle 这个管理 Vim 插件的一个软件，这里不做过多介绍。

在 Vim 中执行你想要运行的命令

vim-run-interactive 让你可以在 Vim 中执行几乎任何你想要在命令行中执行的命令。举例来说，假设你有条 git update 的自定义命令，你可以通过:RunInInteractiveShell git update 来执行它，而不需要退出 Vim。添加一条配置，可以简化这个步骤：

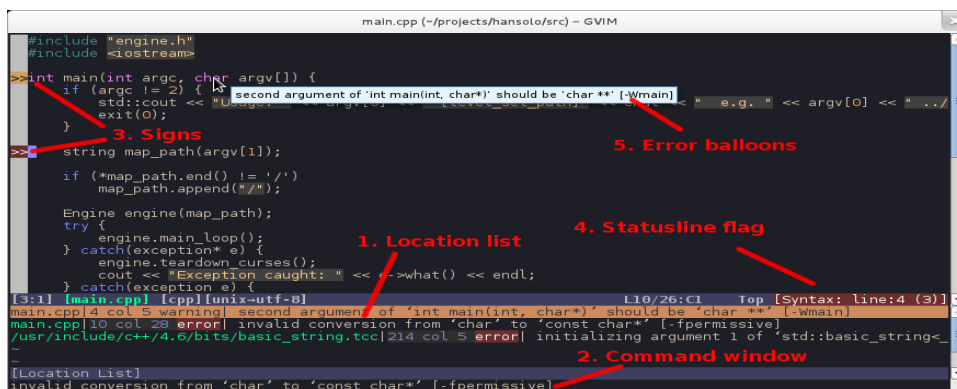
```
" Run commands that require an interactive shell
nnoremap <Leader>r :RunInInteractiveShell<space>
```

如此一来，你可以通过<Leader> + r + 命令键来激活执行命令。如果你不知道什么是 Leader 键，你可以去百度或者 Google 一下。

Vim 的语法检查

Vim 中有个很强大的语法检查插件，它支持几乎所有常用的语言的语法检测

[syntastic(<https://github.com/scrooloose/syntastic>)]。附上一张来自官方的截图：



为了让它更好的工作，同样需要一些配置：

```
" configure syntastic syntax checking to check on open as well as save
let g:syntastic_check_on_open=1
let g:syntastic_html_tidy_ignore_errors=[" proprietary attribute \"ng-"]
let g:syntastic_always_populate_loc_list = 1
let g:syntastic_auto_loc_list = 1
let g:syntastic_check_on_wq = 0
set statusline+=%#warningmsg#
set statusline+=%{SyntasticStatuslineFlag()}
set statusline+=%*
```

Rails 集成开发套件

我是一名 Ruby 的爱好者，所以 Vim 中少不了针对 Ruby 的一系列插件。我作为一名 Web 开发者，Rails 这个大名鼎鼎的框架自然也是有所涉猎。所以在我的 Vim 中有着针对它们开发的一套插件。

插件列表：

- [ruby-vim](#)：在快速的在 module, class, method 中跳跃。
- [vim-bundler](#)：在 Vim 中集成 Bundler。
- [vim-endwise](#)：自动补全 end 关键字。
- [vim-rails](#)：它的功能很多，可以说是用 Vim 开发 Rails 不可缺少的一个插件。更详细的信息，可以前往它的官方网站获取。
- [vim-rspec](#)：在 Vim 中执行 Rspec 测试。

```
" Cucumber navigation commands
autocmd User Rails Rnavcommand step features/step_definitions -glob=**/*
-suffix=_steps.rb
autocmd User Rails Rnavcommand config config -glob=**/* -suffix=.rb
-default=routes

" RSpec.vim mappings
map <Leader>t :call RunCurrentSpecFile()<CR>
map <Leader>s :call RunNearestSpec()<CR>
map <Leader>l :call RunLastSpec()<CR>
map <Leader>a :call RunAllSpecs()<CR>
```

更多好用的工具

还有很多好用的插件，如果每个都一一说明，那么篇幅再长一倍怕也是不够。所以，我这里就把一些好用的插件列出来，有兴趣的可以自己看看。

- [rename.vim](#)：在 Vim 中为文件重命名。
- [vim-coffee-script](#)：在 Vim 中舒心的编写、编译 Coffeescript。
- [vim-mkdir](#)：当你在 Vim 中新建文件的时候，自动帮你创建不存在的目录。
- [vim-surround](#)：快速的删除、修改和添加 括号、引号、XML 标签等等。
- [matchit](#)：用 % 去在两个对应的字符间跳转。
- [tComment](#)：快速注释、反注释代码。
- [emmet-vim](#)：Emmet 的 Vim 版。
- [tabular](#)：快速对齐。
- [snipmate.vim](#)：快速的代码片段。
- [vim-easymotion](#)：在文件中快速定位。
- [vim-instant-markdown](#)：Vim 中对 Markdown 文档的实时预览。

备注

有更多的一些插件我没有都列出来，它们一般用于特定语法的开发，不一定适合所有人。你可以参考[从零搭建和配置 OSX 开发环境](#)这篇文章，自动管理、配置你的 Vim 环境。

来自：<http://yuez.me/jiang-ni-de-vim-da-zao-cheng-qing-qiao-qiang-da-de-ide/>