

前两天同事让我在小组内分享一下 VIM，于是我花了一点时间写了个简短的教程。虽然准备有限，但分享过程中大家大多带着一种惊叹的表情，原来编辑器可以这样强大，这算是对我多年来使用 VIM 的最大鼓舞吧。所以分享结束之后，将这篇简短教程整理一下作为我 2014 年的第一篇 Blog。

目录

- [写在前面：Life Changing Editor](#)
- [什么是 VIM](#)
- [为什么选 VIM](#)
 - [为什么选其它](#)
 - [为什么犹豫选择它们](#)
 - [VIM >= SUM\(现代编辑器\)](#)
- [如何学习 VIM](#)
 - [一秒钟变记事本](#)
 - [VIM 的基本用法](#)
 - [VIM 进阶：插件](#)
 - [插件管理神器：Vundle](#)
 - [配色方案](#)
 - [导航与搜索](#)
 - [自动补全](#)
 - [语法](#)
 - [其它](#)
 - [终极配置: spf13](#)
 - [与其它软件集成](#)
 - [一些资源](#)
- [写在最后](#)

搭完网站之后的第一篇文章有些兴奋，先变身话痨简单回顾一下我是如何接触到 VIM 的，不感兴趣的同学可以直接跳过这一部分:-)

写在前面：Life Changing Editor

我是一个非常 **懒** 的人，对于效率有着近乎执拗的追求。比如我会花 2 个小时来写一个脚本，然后使用这个脚本瞬间完成一个任务，而不愿意花一个小时来手工完成这项任务，从绝对时间上来说，写脚本花的时间更长，但我依然乐此不疲。

工欲善其事，必先利其器，折腾各种各样的软件就成为了我的一大爱好，尤其是各种人称 **神器** 的工具类软件，而 [善用佳软](#) 是这类工具的聚集地，现在我使用的很多优秀的软件都得知于此，包括 VIM，所以，如果你和我一样，希望拥有众多“神器”，让工作事半功倍，可以关注此站。

第一次听说 VIM 已经是离开校园参加工作之后的事，那时部门内部大多使用 Source Insight 代替 Visual Studio 编写代码，大家都被它的代码管理，自动完成，代码跳转等功能所吸引，但一个领导说了句很多 Vimer 经常会说，至今仍让我记忆犹新的一句话：

世界上只有三种编辑器，EMACS、VIM 和其它

我很反对这种极端的言论，使用何种工具是一个人自由，只要能发挥一个工具最大的效率就行，不应该加以约束，更不应该鄙视。话虽如此，我却阻挡不住好奇心的驱使，琢磨着到底是什么样的编辑器会拥有这样高的评价。抱着这份好奇，我搜索到了 [善用佳软](#)，看到《[普通人的编辑利器——Vim](#)》，Dieken 的《[程序员的编辑器——VIM](#)》，以及王垠的《[Emacs 是一种信仰！世界最强编辑器介绍](#)》BANG.....想到不久前看到的 [一段话](#)：

南中国的雷雨天有怒卷的压城云、低飞的鸟和小虫，有隐隐的轰隆声呜呜咽咽……还有一片肃穆里的电光一闪。那闪电几乎是一棵倒着生长的树，发光发亮的枝丫刚刚舒展，立马结出一枚爆炸的果实，那一声炸响从半空中跌落到窗前，炸得人一个激灵，杯中一圈涟漪。

这种一个激灵的感觉不仅仅局限于雷雨天。在我读完上面几篇文章之后，简单的文字亦立刻击中僵中，炸的一个激灵。从此，我对编辑器的认识被完全颠覆。

很多孩子都有一个梦想：希望能够长大之后可以身着军装，腰插手枪，头戴警帽，遇到坏人之后潇洒拔出枪，瞬间解决战斗，除暴安良，匡扶正义。我这样的程序员们也有一个梦想：希望学成之后可以像电影里黑客们一样，对着满屏幕闪烁的各种符号，双手不离键盘噼里啪啦一阵乱敲，屏幕上的符号不断滚动，就攻破了几百公里之外的某某银行的服务器，向帐户里面增加一笔天文数字，然后潇洒的离去，神不知鬼不觉，留下不知所措的孩子们的梦想——警察叔叔们。这简直构成了程序员们的终极幻想:-P。VIM 的出现让我感觉离幻想更近了一步，呃，别想错了，我是指——双手不离键盘，噼里啪啦，黑客的范儿。不可否认，扮酷也是促使我学习 VIM 的一个重要原因:-P。

在一个激灵之后，接下来便是不可自拔的陷入 VIM 世界，于是网上搜索各种入门教程，_vimrc 的配置，折腾插件，研究奇巧淫技，将 VIM 打造成 IDE。那感觉就像世界从此就只有 VIM，写代码用 VIM，Visual Studio 用 VIM，Source Insight 用 VIM，甚至写 PDF，浏览网页都要用 VIM，够折腾吧。可是像 Vimer 们一样，我依然折腾着，并快乐着。如今，折腾一圈之后，随着对 Unix 的 KISS 设计哲学逐渐理解与认可：**把所有简单的事情做到极致**。所以在对待 VIM 的态度上也有了一定的转变，不再执著的将它打造成万能的 IDE，而仅仅让它将编辑功能发挥到极致，其它的事情交给其它更擅长的工具去做。Keep It Simple, Stupid.

在 VIM 的 [官方网站](#)上，对每个插件的评价是这样 [分类](#)的：

- Life Changing
- Helpful
- Unfulfilling

而我想将这个分类应用到使用的软件上，对于 VIM，它是毫无疑问的 Life Changing 。

什么是 VIM

以下两句对编辑器的最高评价足矣：

- VIM is the God of editors, EMACS is God's editor
- EMACS is actually an OS which pretends to be an editor

为什么选 VIM

我们所处的时代是非常幸运的，有越来越多的 [编辑器](#)，相对于 [古老的 VIM](#) 和 EMACS，它们被称为 **现代** 编辑器。我们来看看这两个古董有多大年纪了：

```
**EMACS** : 1975 ~ 2013 = 38 岁
**VI**    : 1976 ~ 2013 = 37 岁
**VIM**   : 1991 ~ 2013 = 22 岁
```

看到这篇文章的人有几个是比它们大的:-)

VIM 的学习曲线非常陡，[这里](#)有一个主流编辑器的学习曲线对比。既然学习 VIM 如此之难，而 **现代** 编辑器又已经拥有了如此多的特性，我们为什么要花大量的时间来学习这个老古董呢？

为什么选其它

先来看看为什么我们会选现在所使用的编辑器？(也许很多人直接用 IDE 自带的编辑器，我们暂且也把它们划到编辑器的范畴内。)这里我简单列举一些程序员期望使用的编辑拥有的功能：

- 轻量级，迅速启动（相对于 IDE）
- 特性
 - 语法高亮
 - 自动对齐
 - 代码折叠
 - 自动补全
 - 显示行号
 - 重定义 Tab
 - 十六进制编辑
 - 列编辑模式
 - 快速注释
 - 高级搜索，替代
 - 错误恢复
 - 迅速跳转
 - Mark
- 也许，美观也是一个诉求

但是…

为什么犹豫选择它们

总有一些理由让我们一再犹豫的选择它们，或者勉强使用它们：

- 太贵：虽然知道 VS 很贵，但看到价格时，还是被吓了一跳
 - Visual Studio Profession 2012 : 11645 元
 - UtralEdit : 420 元
 - Source Insight : 2500 元
 - \$\$
 - \$\$
 - \$\$
- 不能跨平台
 - VS, SI, UE, Notepad++这些只能在 Windows 上使用
 - Mac 上的 TextMate 只能运行于 Mac 上
- 不容易扩展

那么，还有别的选择么？

VIM >= SUM(现代编辑器)

首先，VIM 包含了上面列的所有现代编辑器的优点，并且远远多于此。

并且，VIM 拥有让你不再 **犹豫** 的其它特性：

- 无止尽的扩展：现在 VIM 的官方网站上已经有了 [4704](#) 个扩展，并且在不断增加…
- 完美的跨平台：
 - Windows : gVim
 - Linux : 内置默认 (e.g., man page)
 - Mac : MacVim
- 开源
- 用起来很酷
- 最关键的，\$

废话结束，开始进入正题。

如何学习 VIM

一秒钟变记事本

很多时候大家希望能够以最快的速度编辑文档，而不愿意花大量的时间在学习这一工具上，比如偶尔要去 Linux 改变一下配置。这时 VIM 有一种方法可以 **一秒钟变记事本**，打开 VIM 之后，只需要一个键 **i**，接下来所有的操作就和 Windows 上的记事本无异，你所喜爱与习惯的方向键也回来了。

这也并没有多神奇，它只是 VIM 提供了一种特殊的模式：Insert mode，在按过 i 之后，你可以在编辑器的左下角看到 INSERT 字样。但是因为 VIM 无法使用 CTRL-S 来保存，那么，在编辑完之后，如何保存退出呢？也很简单，先按 ESC，再输入 :wq，前面一步是告诉 VIM 退出 INSERT 模式，后面一个命令是保存退出。

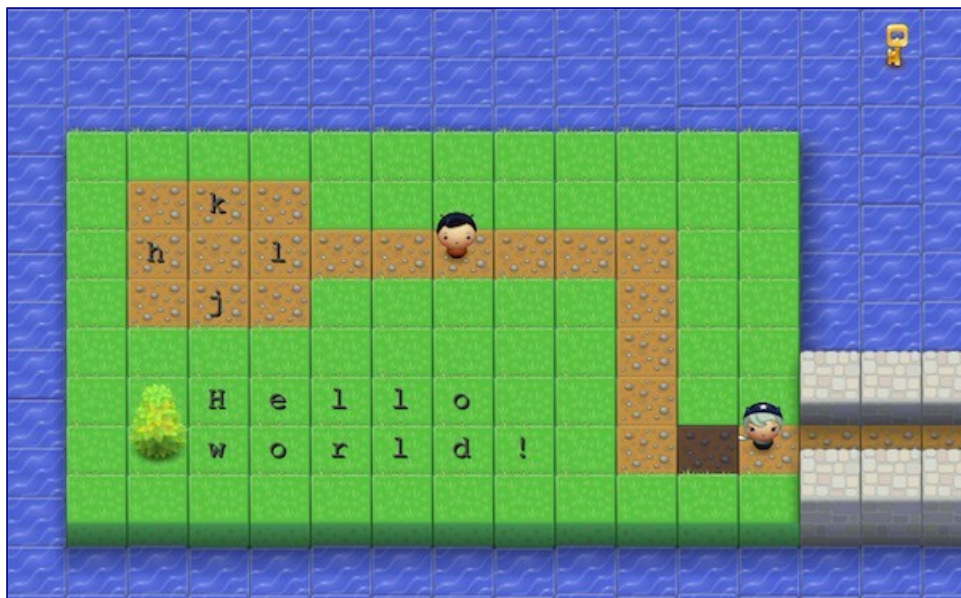
我见过很多人这样用，虽然说这很容易，但是有种暴殄天物的感觉，和给了你一把 AK47，你却把它当成棍子使一样。要发挥 AK47 的作用，还请向下看。

VIM 的基本用法

最好的入门教程非 VIM 自带的 [vimtutor](#) 莫属，它是 VIM 安装之后自带的简短教程，可以在安装目录下找到，只需半个小时左右的时间，就可以掌握 VIM 的绝大部分用法。这是迄今为止我见过的软件自带教程中最好的一个。

当然，网上的 VIM 教程也非常多，我之前看的是李果正的 [大家来学 VIM](#)，很适合入门。

另外推荐陈皓的 [简明 VIM 练级攻略](#)，或者创意十足的游戏 [VIM 大冒险](#)。

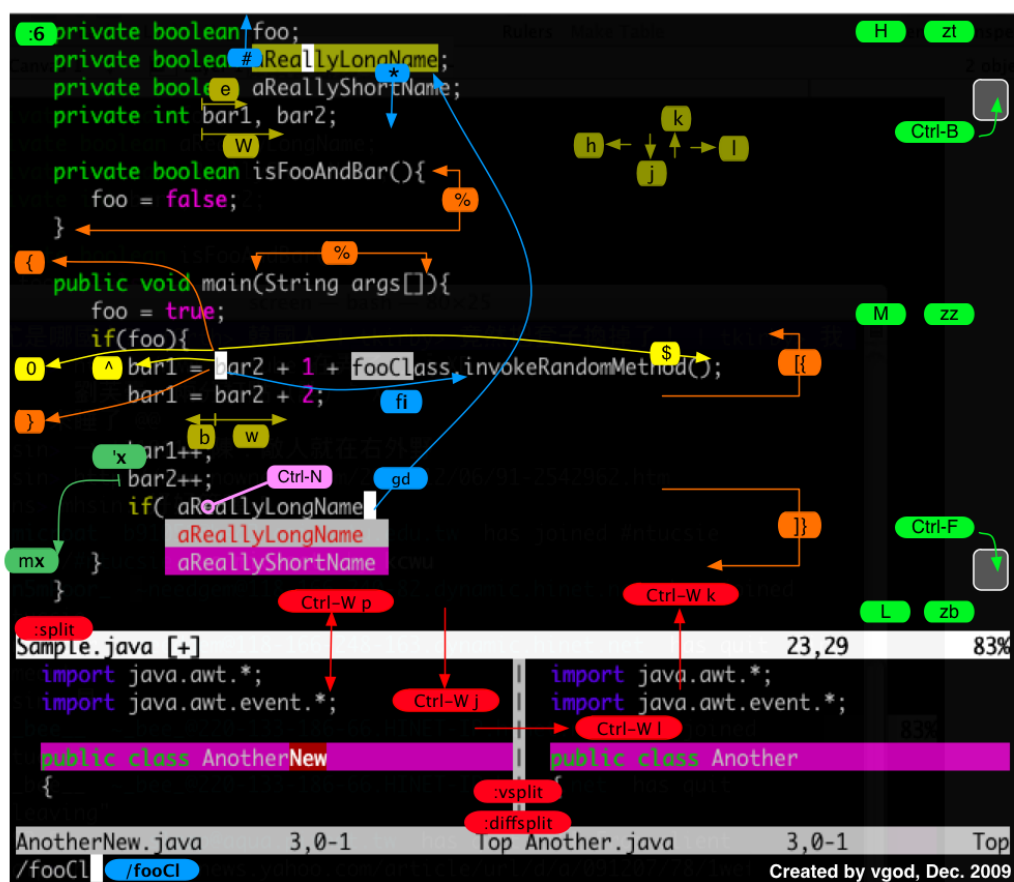


这游戏的创意实在是太赞了，打完游戏，你便掌握了 VIM，这才是真正的 **寓教于乐**，下面是摘自这个游戏的描述：

VIM Adventures is an online game based on VIM's keyboard shortcuts (commands, motions and operators). It's the "Zelda meets text editing" game. It's a puzzle game for practicing and memorizing VIM commands (good old VI is also covered, of course). It's an easy way to learn VIM without a steep learning curve.

最后在这里给大家分享一个 vgod 设计的 [VIM 命令图解](#)。这也是我看过的最好的命令图示，看完了前面的基本教程后，可以将它作为一个 cheat sheet 随时查看，相信用不了多久你也可以完全丢掉它。关于此图的详细解释可以参考 [这里](#)。

vim命令圖解



游標移動/範圍單位

字元(character)	
h	j
k	l
單字(word)	
w	b
W	B
e	
行(line)	
0	\$
^	
段落(paragraph)、區塊(block)	
{	}
[]
%	
螢幕(screen)、檔案(file)	
H	zt
M	zz
L	zb
C-B	C-F
gg	G
mx	'x
搜尋(search)	
*	#
fx	
gd	
/xxx	
n	N

ESC	
V	
C-v	
i	
R	
a	
A	
y	
d	
c	
x	
D	
C	
P	
J	
>	
<	
.	
u	
:w	
:q	
:e x	
:n	
:h	
:xx	
自動補	
C-N	
C-X C	
分割視	
:vsp	
:dif	
C-W	
C-W	

VIM 进阶：插件

在学完了上面任何一个教程之后，通过一段时间的练习，你已经可以非常熟练的使用 VIM。即使是“裸奔”，VIM 已经足够强大，能够完成日常的绝大部分工作。但 VIM 更加强大的是它的扩展机制，就像 Firefox 和 Chrome 的各种插件，它们将令我们的工具更加完美。网上有很多教程里写的插件已经过时，接下来我将介绍一些比较新的，非常有用的插件，看完之后，相信你一定会觉得蠢蠢欲动。

插件管理神器：Vundle

在这开始之前，先简单介绍 VIM 插件的管理方式。在我刚接触插件之时，安装一个插件需要：

1. 去官网下载
2. 解压
3. 拷贝到 VIM 的安装目录
4. 运行:help tags

这些步骤已经足够复杂，更加无法想象的是要 **更新** 或者 **删除** 一个插件时，因为它的文件分布在各个目录下，就比如 Windows 上的安装路径，Application data，用户数据，注册表等等，除非你对 VIM 的插件机制和要删的插件了如直掌，否则你能难将它删除干净。所以一段时间之后，VIM

的安装目录下简直就是一团乱麻，管理插件几乎成为了一项不可能完成的任务。想象一下，如果 Windows 上面没有软件管理工具，你如何安装，卸载一个软件吧。

但是这没有难倒聪明的 Vimer 们，他们利用 VIM 本身的特性，开发出了神器—— [Vundle](#)，配合上 [GitHub](#)，VIM 插件的管理变得前所未有的简单。来对比一下使用 Vundle 如何管理插件：

在按照官方的 [教程](#) 安装好 Vundle 之后，要安装一个插件时，你只需要：

1. 选好插件
2. 在 VIM 的配置文件中加一句 `Bundle 'your/script/path'`
3. 在 VIM 中运行 `:BundleInstall`

卸载时只需：

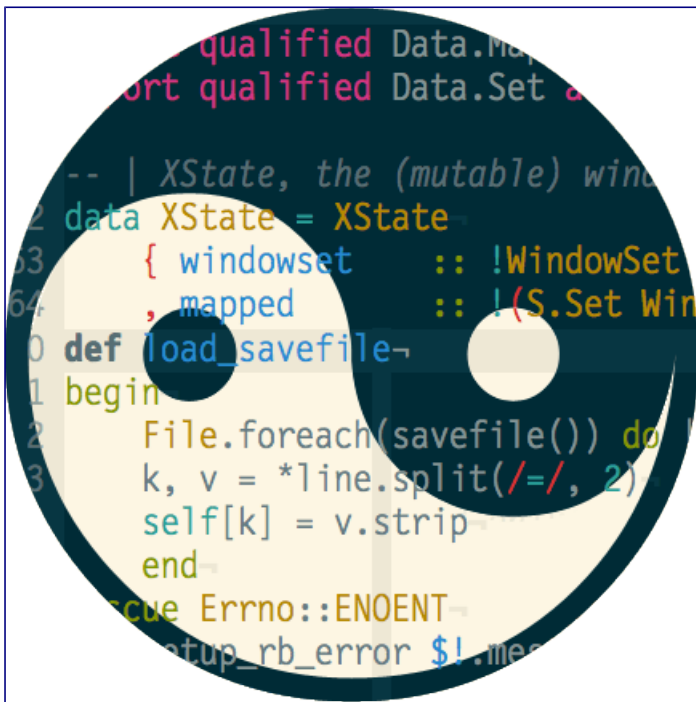
1. 去除配置文件中的 `Bundle 'your/script/name'`
2. 在 VIM 中运行 `:BundleClean`

更新插件就更加简单，只需一句 `:BundleUpdate`。现在你已经完全从粗活累活中解放了出来，从此注意力只需放在挑选自己喜欢的插件上，还有比这更美好的么？下面介绍的所有的插件都以它来管理。

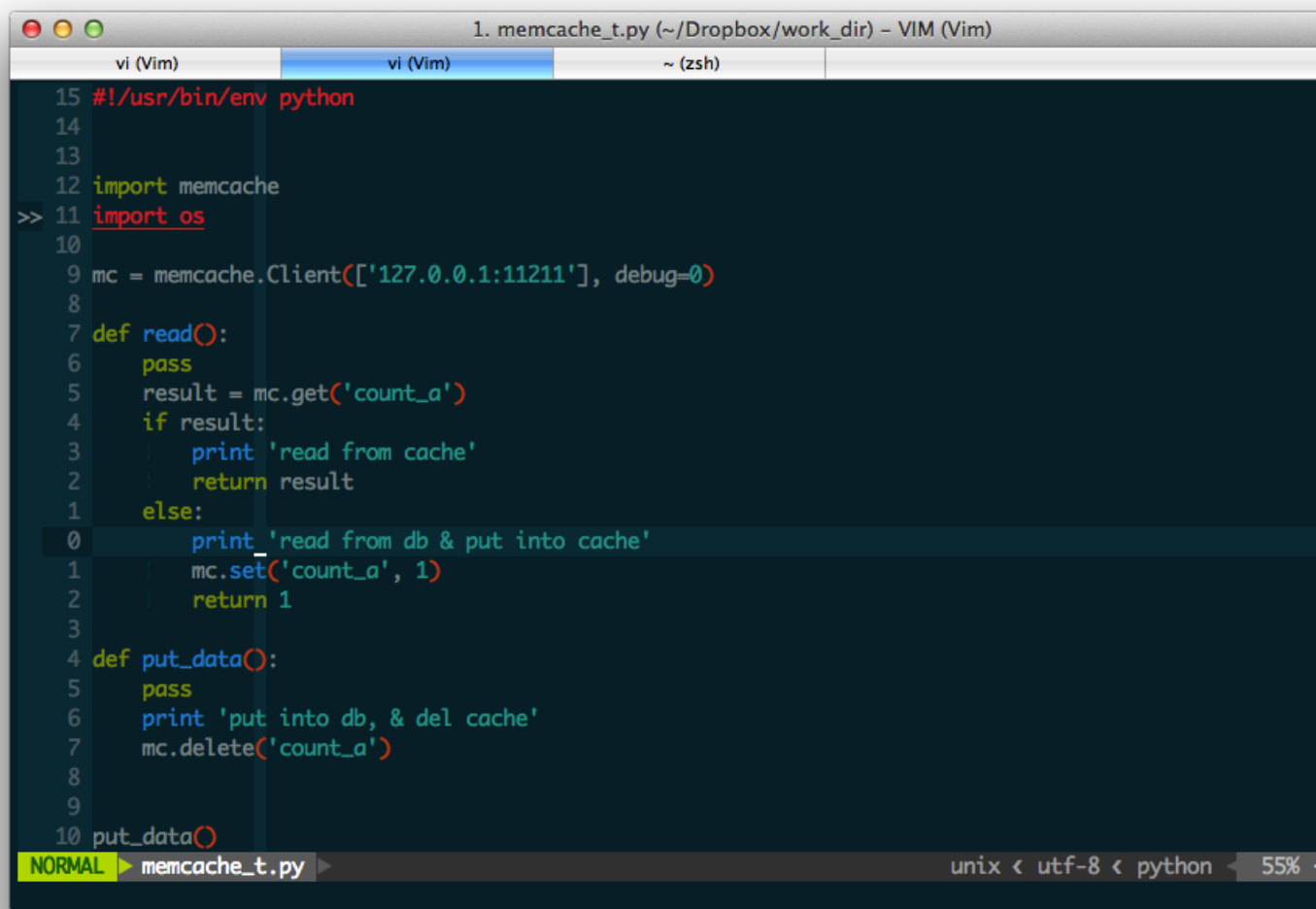
配色方案

你是否觉得用了许多年的白底黑字有些刺眼，又或者你是否厌倦了那单调枯燥？如果是，那好，VIM 提供了成百上千的 [配色方案](#)，终有一款适合你。

在所有的配色当中，最受欢迎的是这款 [Solarized](#)：



在 Github 上它有 [4,930](#) 个 Star，仅靠一个 配色方案 就得到如此多的 Star，可见它有多么的受欢迎。它有两种完全相反的颜色，一暗一亮，作者非常具有创意将它们设计成一个 阴阳八卦，赏心悦目。下面是采用这种配色的 VIM 截图：



```
15 #!/usr/bin/env python
14
13
12 import memcache
>> 11 import os
10
9 mc = memcache.Client(['127.0.0.1:11211'], debug=0)
8
7 def read():
6     pass
5     result = mc.get('count_a')
4     if result:
3         print 'read from cache'
2         return result
1     else:
0         print 'read from db & put into cache'
1         mc.set('count_a', 1)
2         return 1
3
4 def put_data():
5     pass
6     print 'put into db, & del cache'
7     mc.delete('count_a')
8
9
10 put_data()
```

NORMAL memcache_t.py unix < utf-8 < python 55%

Solarized 配色还有一个使它能够成为最受欢迎的配色方案的理由，除了 VIM 之外，它还提供了很多 [其它软件](#) 的配色方案，包括：Emacs , Visual Studio , Xcode , NetBeans , Putty , 各种终端等等，应该是除了默认的黑白配色之外用途最为广泛的一种了。目前我采用的就是这种配色方案的 dark background，它的对比度非常适合长期对着编辑器的程序员们。

还有一种很受欢迎的配色方案：[Molokai](#)，它是 Mac 上 TextMate 编辑器的一种经典配色，也非常适合程序员：


```
1. memcache_t.py (~/.Dropbox/work_dir) - VIM (Vim)
vi (Vim)      vi (Vim)      ~ (zsh)

15 #!/usr/bin/env python
14
13
12 import memcache
>> 11 import os
10
9 mc = memcache.Client(['127.0.0.1:11211'], debug=0)
8
7 def read():
6     pass
5     result = mc.get('count_a')
4     if result:
3         print 'read from cache'
2         return result
1     else:
0         print 'read from db & put into cache'
1         mc.set('count_a', 1)
2         return 1
3
4 def put_data():
5     pass
6     print 'put into db, & del cache'
7     mc.delete('count_a')
8
9
10 put_data()
NORMAL memcache_t.py unix < utf-8 < python 55%
```

导航与搜索

1. [NERDTree](#) - file navigation

```
1. vimrc (~/.linux_config/vim) - VIM (Vim)

12
11 "使用 Vundle来 管理 Vundle
10 Bundle 'gmarik/vundle'
9 " vim plugin bundle control, command model
8 " :BundleInstall      install
7 " :BundleInstall!    update
6 " :BundleClean       remove plugin not in list
5
4
3 ""file browser
2 Bundle 'vim-scripts/The-NERD-tree'
1 map <leader>n :NERDTreeToggle<CR>
0 let NERDTreeHighlightCursorline=1
1 let NERDTreeIgnore=[ '\.pyc$', '\.pyo$', '\.py\.$class$', '\.obj$', '\.o$', '\.so$', '\.egg$', '^\.git$' ]
2
3 ""D 代码片段
```

代码资源管理器现在已经成为了各种各样 IDE 的标配，这可以大大提高管理源代码的效率。这样的功能 VIM 自然不能少，NERD Tree 提供了非常丰富的功能，不仅可以以 VIM 的方式用键盘来操作目录树，同时也可以像 Windows 资源管理器一样用鼠标来操作。

--help: 可以将打开目录树的功能绑定到你喜欢的快捷键上，比如: map <leader>e :NERDTreeToggle<CR>

2. [CtrlP](#) - fast file finder

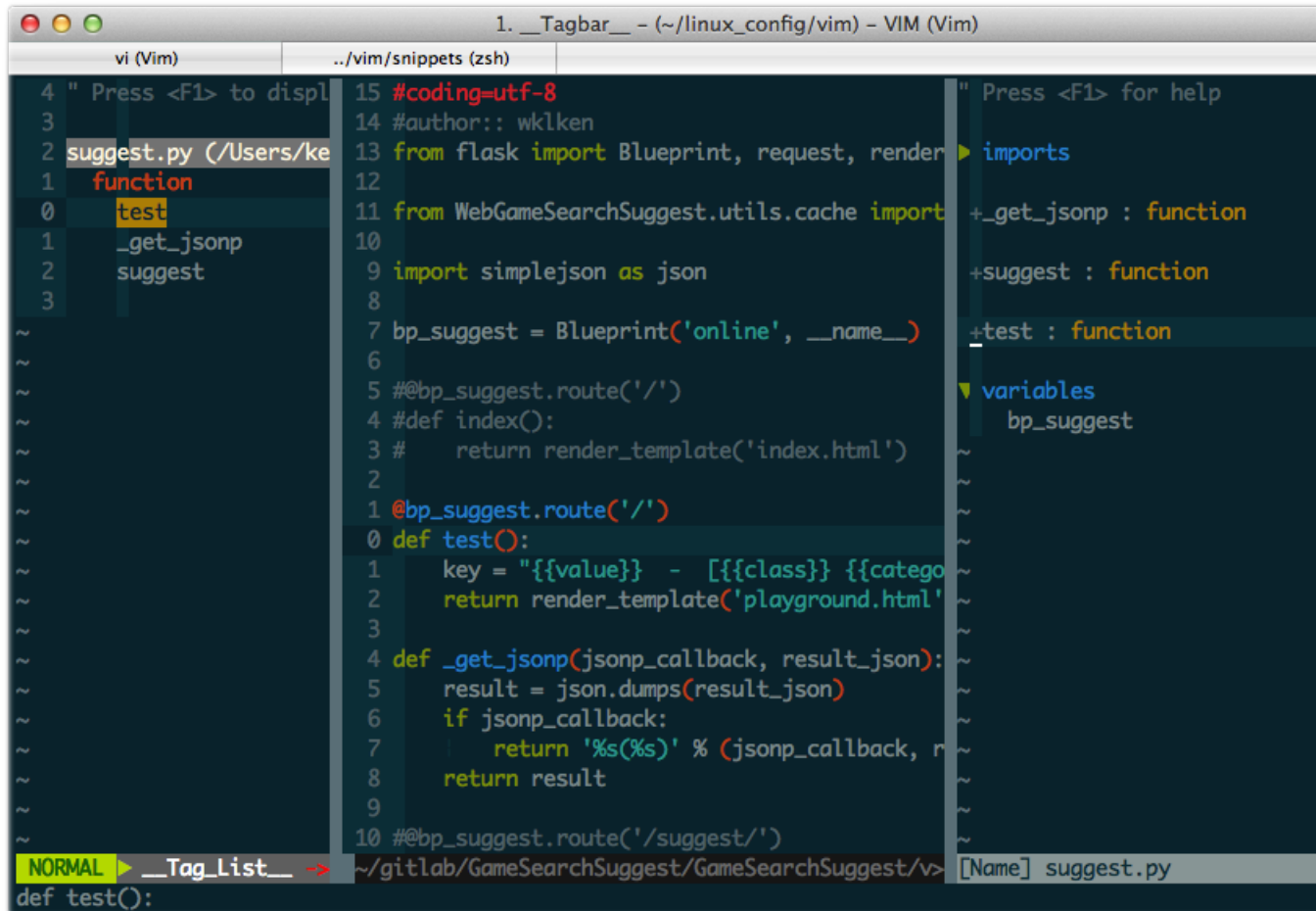
```
4 let python_highlight_all = 1
3
2 "for file search ctrlp
1 Bundle 'kien/ctrlp.vim'
0 let g:ctrlp_map = '<leader>p'
1 let g:ctrlp_cmd = 'CtrlP'
2 "set wildignore+=*/tmp/*,*.so,*.swp,*.zip      " MacOSX/Linux
3 let g:ctrlp_custom_ignore = '\.git$|\.hg$|\.svn$|\.rvm$'
4 let g:ctrlp_working_path_mode=0
5 let g:ctrlp_match_window_bottom=1
NORMAL > vimrc
oldvimrc
README.md
python.vim
gui_vim.png
snippets/go.snippets
doc/tags
doc/taglist.txt
plugin/taglist.vim
snippets/sh.snippets
snippets/all.snippets
snippets/css.snippets
snippets/python.snippets
snippets/html.snippets
snippets/java.snippets
snippets/snippets.snippets
mru > files > buf > -
prt < path < ~/linux_c
>>> _
```

如果说上面介绍的 NERD Tree 极大的方便了源代码的管理方式，那 CtrlP 可以称的上是革命性的，杀手级的 VIM 查找文件插件。它以简单符合直觉的输入方式，极快的响应速度，精确的准备度，带你项目中自由穿越。它可以模糊查询定位，包括工程下的所有文件，已经打开的 buffer，buffer 中的 tag 以及最近访问的文件。在这之前，我用的是 [lookupfiles](#)，因为依赖了其它的插件和应用程序，这个上古时代的插件逐渐被抛弃了。自从有了它，NERD Tree 也常常被我束之高阁。

据说它模仿了 Sublime 的名字和功能，我没用过 Sublime，但是听说 CtrlP 这个功能是 Sublime 最性感的功能之一。可以去它的 [官网](#) 看看。

--help: 这个插件另一个令人称赞的一点在于无比简单直观的使用方式，正如其名: Ctrl+P，然后享受它带来的快感吧。

3. [Taglist](#) - source code browser



想必使用过 Visual Studio 和 Source Insight 的人都非常喜爱这样一个功能：左边有一个 Symbol 窗口，它列出了当前文件中的宏、全局变量、函数、类等信息，鼠标点击时就会跳到相应的源代码所在的位置，非常便捷。Taglist 就是实现这个功能的插件。可以说 symbol 窗口是程序员不可缺少的功能，当年有很多人热衷于借助 taglist、ctags 和 cscope，将 VIM 打造成一个非常强大的 Linux 下的 IDE，所以一直以来，taglist 在 VIM 官方网站的 scripts 排列榜中一直高居 [榜首](#)，成为 VIM 使用者的必备插件。

--help: 最常见的做法也是将它绑定到一个快捷键上，比如：map <silent> <F9> :TlistToggle<CR>

4. [Tagbar](#) - tag generation and navigation

```
14 #
13 #     word = request.args.get('key')
12 #     if word:
11 #         word = word.lower()
10 #
9 #     result = get_keyword_suggest(word)
8 #     if result:
7 #         result_json.update({'ok': True})
6 #         result_json.update({'data': result})
5 #     else:
4 #         result_json.update({'ok': False})
3 #     return _get_jsonp(jsonp_callback, result_json)
2
1 @bp_suggest.route('/suggest/')
0 def suggest():
1     #推 荐
2     jsonp_callback = request.args.get('callback', '')
3
4
5     word = request.args.get('key')
6     if word:
7         word = word.lower()
8
9     result = get_keyword_suggest(word)
```

看起来 Tagbar 和上面介绍的 Taglist 很相似，它们都是展示当前文件 Symbol 的插件，但是两者有一定的区别，大家可以从上图的对比中得知，两者的关注点不同。总的来说 Tagbar 对面向对象的支持更好，它会自动根据文件修改的时间来重新排列 Symbol 的列表。它们以不同的纬度展示了当前文件的 Symbol。

--help: 同 Taglist 一样，可以这样绑定它的快捷键，`nmap <silent> <F4> :TagbarToggle<CR>`

5. [Tasklist](#) - eclipse task list

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #TODO: test
4
5
6 cs'"
7 "hello world"
8
9 ds'
10 hello world
11
12 ysiw]
13 [ hello ]
NORMAL ▶ ~/tmp/common.py ▶ unix < utf-8 < python < 1% <
5 Bundle 'hdira/python-syntax'
4 let python_highlight_all = 1
3
2
1 ""D task list
0 Bundle 'vim-scripts/TaskList.vim'
1 map <leader>td <Plug>TaskList
2
3 ""D for go lang
4 Bundle 'jnrwhiteh/vim-golang'
5 ""D edit history 同类 Undotree
6 Bundle 'sjl/gundo.vim'
vimrc 81% <
```

这是一个非常有用的插件，它能够标记文件中的 `FIXME` 、 `TODO` 等信息，并将它们存放到一个任务列表当中，后面随时可以通过 `Tasklist` 跳转到这些标记的地方再来修改这些代码，是一个十分方便实用的 `Todo list` 工具。

--help: 通常只需添加一个映射: `map <leader>td <Plug>TaskList`

自动补全



1. [YouCompleteMe](#) - visual assist for vim

这是迄今为止，我认为 `VIM` 历史上最好的插件，没有之一。为什么这么说？因为作为一个程序员，这个功能必不可少，而它是迄今为止完成的最好的。从名字可以推断出，它的作用是代码补全。不管是在 `Source Insight`，还是安装了 `Visual Assist` 的 `Visual Studio` 中，代码补全功能可以极大的提高生产力，增加编码的乐趣。大学第一次遇到 `Visual Assist` 时带给我的震撼至今记忆犹新，那感觉就似百兽之王有了翅膀，如虎添翼，从此只要安装有 `Visual Studio` 的地方我第一时间就会安装 `Visual Assist`。

而作为编辑器的 `VIM`，一直以来都没有一个能够达到 `Visual Assist` 哪怕一成功力的插件，不管是自带的补全，`omnicppcomplete`，`neocompletercache`，完全和 `Visual Assist` 不在一个数量级上。`Visual Assist` 借助于 `Visual Studio`，它的补全是语义层面的，它完全能够理

解程序语言，而 VIM 的这些插件仅仅是基于文本匹配，虽然最近的 `neocompletestache` 已经好了很多，但准确率非常低。所以在写代码时，即使 VIM 用得再顺手，绝大部分情况下我还是倾向于 Visual Studio + Visual Assist 。

但是 YouCompleteMe 的出现彻底的改变了这一现状，它对代码的补全完全终于也达到了编译器级别，绝不弱于 Visual Assist，遇到它是我使用 VIM 之后最兴奋的一件事。为什么一个编辑器的插件可以做到如此的神奇，原因就在于它基于 [LLVM/clang](#)，一个 Apple 公司为了代替 GNU/GCC 而支持的编译器，正因为 YouCompleteMe 有了编译器的支持，而不再像以往的插件一样基于文本来进行匹配，所以准确率才如此之高。其次，由于它是 C/S 架构，会在本机创建一个服务器端，利用 clang 来解析代码，然后将结果返回给客户端，所以也就解决了 VIM 是单线程而造成的各种补全插件速度奇慢的诟病，在使用时，几乎感觉不到任何的延时，体验达到了 Visual Assist 的级别。

YouCompleteMe 也是所有的插件当中安装最为复杂的一个，这是因为需要用 clang 来编译相应的库。因为 clang 在 Linux 和 Mac 平台上支持的非常好，所以在这两个平台上安装相对简单。但是 clang 并没有官方支持 Windows，所以 YouCompleteMe 插件也没有官方支持 Windows。可这么好的东西，活跃在 Windows 上聪明的 Vimer 们怎么可能容忍这种事情呢，有人就提供了 [Windows Installation Guide](#)，已经编译好了各种版本的 YouCompleteMe 插件，可以参考这个 Guide 来安装。我并没有采用它，而是参考了 [这里](#)，自己编译了 YouCompleteMe，其实也不难，一步一步按照介绍的步骤，相信你也可以。

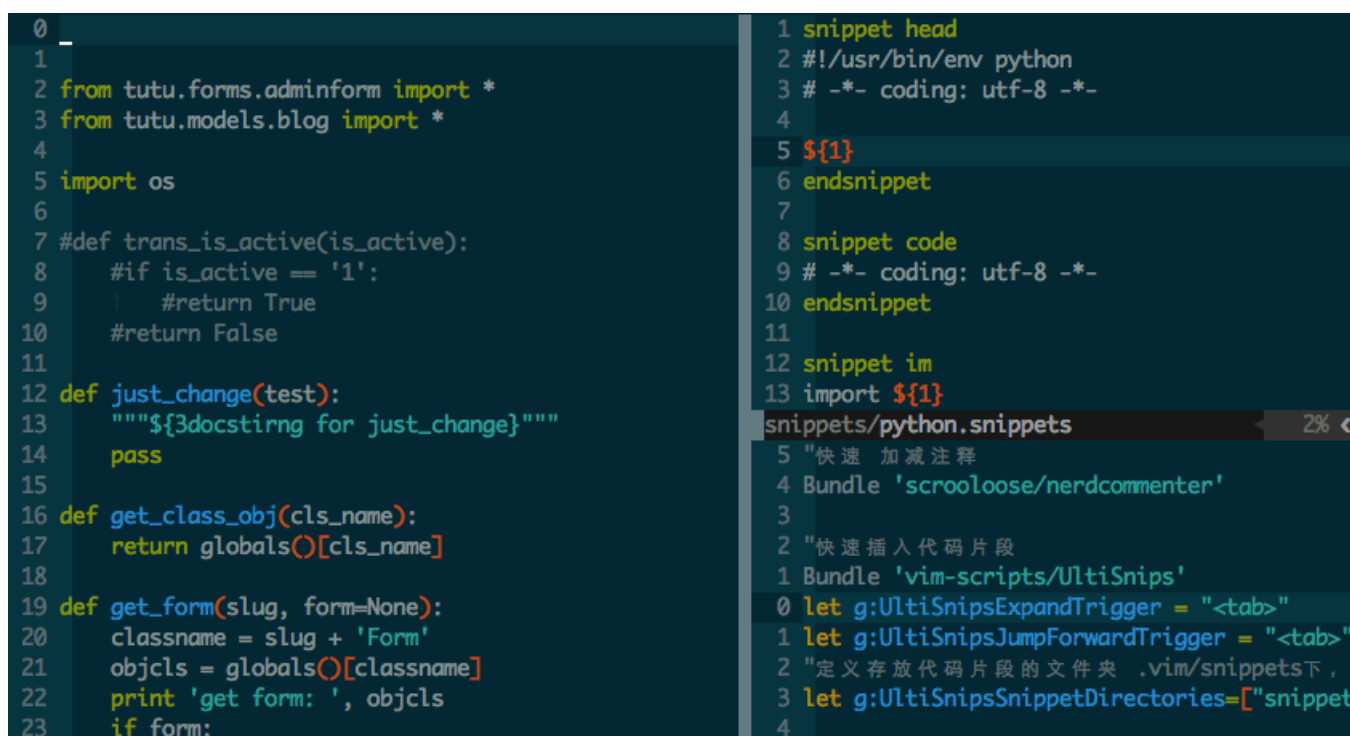
YouCompleteMe 除了补全以外，还有一个非常重要的作用：代码跳转，同样可以达到编译器级别的准确度，媲美 Visual Assist 与 Source Insight。

有了 YouCompleteMe 之后，是时候抛弃昂贵的 Visual Assist 与 Source Insight 了。赶快安装尝试吧:-)

--help: 只要设置好项目的 `.ycm_extra_conf.py`，自动补全功能就可以完美的使用了。通常一个全局的 `.ycm_extra_conf.py` 足矣。代码跳转可以绑定一个快捷键：
`nnoremap <leader>jd :YcmCompleter`

`GoToDefinitionElseDeclaration<CR>`，很好理解，先跳到定义，如果没找到，则跳到声明处。

2. [UltiSnips](#) - ultimate snippets



这是什么？相信大家经常在写代码时需要在文件开头加一个版权声明之类的注释，又或者在头文件中需要：`#ifndef... #def... #endif` 这样的宏，亦或者写一个 `for` 、`switch` 等很固定的代码片段，这是一个非常机械的重复过程，但又十分频繁。我十分厌倦这种重复，为什么不能有一种快速输入这种代码片段的方法呢？于是，各种 snippets 插件出现了，而它们之中，UltiSnips 是最好的一个。比如上面的一长串 `#ifndef... #def... #endif`，你只需要输入 `ifn<TAB>`，怎么样，方便吧。更为重要的一点是它支持扩展，你可以随心所欲的编辑你自己的 snippets。

现在它可以和上面介绍的 YouCompleteMe 插件一块使用，比如在敲完 `ifn` 时，YouCompleteMe 会将这个 snippet 也放在下拉框中让你选择，这样你就不用去记何时按 `<TAB>` 来展开 snippets，YouCompleteMe 已经帮你完成。

去它的 [网站](#) 看看，有几个视频，绝对亮瞎你的双眼(需要翻墙)。

--help: 它和 YouCompleteMe 一块使用时会有一定的冲突，因为两者都默认绑定了 `<TAB>` 键，可以参考各自的 help 文档，将其中一个绑定到其它的快捷键，或者借助 [其它的插件](#) 让它们兼容。

3. Zen Coding - hi-speed coding for html/css

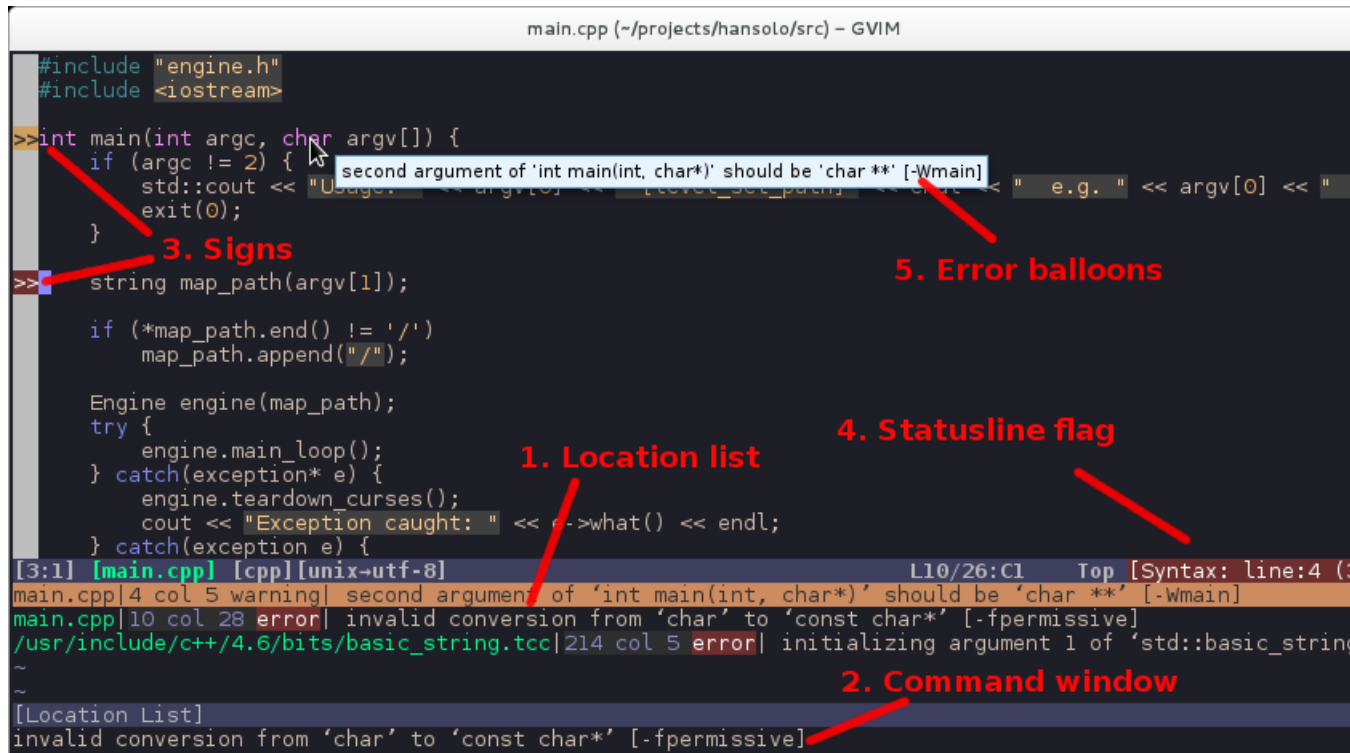
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ru">
<head>
  <title>Title</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
</head>
<body>
</body>
</html>
```

比一般的 C/C++/Java 等更多重复劳动的语言估计要算 HTML/CSS 这类前端语言了吧，为此前端大牛发明了 Zen Coding，去 [这里](#) (需翻墙) 看看演示视频，相当令人震撼。如果是写前端的话，强烈推荐此插件。

`--help`: 可以去[这里](#)参考前端工程师们写的中文教程 [1](#), [2](#)

语法

1. [Syntastic](#) - integrated syntax checking



这是一个非常实用的插件，它能够实时的进行语法和编码风格的检查，利用它几乎可以做到编码完成后无编译错误。并且它还集成了静态检查工具：lint，可以让你的代码更加完美。更强大的它支持近百种编程语言，像是一个集大成的实时编译器。出现错误之后，可以非常方便的跳转到出错处。强烈推荐。

--help: 这是一个后台运行的插件，不需要手动的任何命令来激活它。

2. [Python-mode](#) - Python in VIM

如果你需要写 Python，那么 Python-mode 是你一定不能错过的插件，靠它就可以把你的 VIM 打造成一个强大的 Python IDE，因为它可以做到一个现代 IDE 能做的一切：

- 查询 Python 文档
- 语法及代码风格检查
- 运行调试
- 代码重构
-

所以，有了它，你就等于有了一个现代的 Python IDE，各位 Pythoner 们，还等什么呢？

--help: 默认情况下该插件已经绑定了几个快捷键：

K -> 跳到 Python doc 处
<leader>r -> 运行当前代码
<leader>b -> 增加/删除断点

其它

1. [Tabularize](#) - align everything

```
2
3 vim-easy-align
4 =====
5
6 Where's Lennon?
7 -----
8
9 Paul McCartney 1942
10 George Harrison 1943
11 Ringo Starr 1940
12 Pete Best 1941
13
14
15 Formatting table
16 -----
17 ```
18
19 | Option | Type | Default | Description |
20 |--|--|--|--|
21 | threads | Fixnum | 1 | number of threads in the thread pool |
22 | queues | Fixnum | 1 | number of concurrent queues |
23 | queue_size | Fixnum | 1000 | size of each queue |
24 | interval | Numeric | 0 | dispatcher interval for processing |
25 | batch | Boolean | false | enables batch processing mode |
[1] DEMO.md [markdown] [Git(master)] 9,1 1%
```

这个插件的作用是用于按等号、冒号、表格等来对齐文本，参考下面这个初始化变量的例子：

```
int var1 = 10;
float var2 = 10.0;
char *var_ptr = "hello";
```

运行 Tabularize /= 可得：

```
int var1      = 10;
float var2    = 10.0;
char *var_ptr = "hello";
```

另一个常见的用法是格式化文件头：

```
file: main.cpp
author: feihu
date: 2013-12-17
description: this is the introduction to vim
license:
TODO:
```

运行 Tabularize `/:/r0` 可得:

```
file      : main.cpp
author    : feihu
date      : 2013-12-17
description : this is the introduction to vim
license   :
TODO      :
```

另一种对齐方式, 运行 Tabularize `/:/r1c1l0` :

```
      file : main.cpp
    author : feihu
      date : 2013-12-17
description : this is the introduction to vim
    license :
      TODO :
```

对于写代码的人来说, 还是非常有用的。因为没有找到对应的图, 所以这里就用 [另外一个插件](#) 的动画来代替了, Tabular 的功能比它更为强大。

--help: 通常会绑定这样一些快捷键:

```
nmap <Leader>a& :Tabularize /&<CR>
vmap <Leader>a& :Tabularize /&<CR>
nmap <Leader>a= :Tabularize /=<CR>
vmap <Leader>a= :Tabularize /=<CR>
nmap <Leader>a: :Tabularize /:<CR>
vmap <Leader>a: :Tabularize /:<CR>
nmap <Leader>a:: :Tabularize /:\zs<CR>
vmap <Leader>a:: :Tabularize /:\zs<CR>
nmap <Leader>a, :Tabularize /,<CR>
vmap <Leader>a, :Tabularize /,<CR>
nmap <Leader>a,, :Tabularize /,\zs<CR>
vmap <Leader>a,, :Tabularize /,\zs<CR>
nmap <Leader>a<Bar> :Tabularize /<Bar><CR>
vmap <Leader>a<Bar> :Tabularize /<Bar><CR>
```

2. [Easymotion](#) - jump anywhere

```
11 let Tlist_WinWidth = 25
10
9
8 "状态栏增强展示
7 Bundle 'Lokaltog/vim-powerline'
6 "if want to use fancy,need to add font patch -> git clone git://gist.github.com/1630581.git ~/.fonts/
5 "let g:Powerline_symbols = 'fancy'
4 let g:Powerline_symbols = 'unicode'
3
2
1 "括号显示增强
0 Bundle 'kien/rainbow_parentheses.vim'
1 let g:rbpt_colorpairs = [
2     \ ['brown',      'RoyalBlue3'],
3     \ ['Darkblue',   'SeaGreen3'],
4     \ ['darkgray',   'DarkOrchid3'],
5     \ ['darkgreen',  'firebrick3'],
6     \ ['darkcyan',   'RoyalBlue3'],
7     \ ['darkred',    'SeaGreen3'],
8     \ ['darkmagenta', 'DarkOrchid3'],
9     \ ['brown',      'firebrick3'],
10    \ ['gray',        'RoyalBlue3'],
11    \ ['black',       'SeaGreen3'],
12    \ ['darkmagenta', 'DarkOrchid3'],
13    \ ['Darkblue',    'firebrick3'],
NORMAL vimrc unix < utf-8 < vim 71%
```

VIM 本身的移动方式已经是极其高效快速，它在编辑器的世界中独树一帜，算是一个极大的创新。而如果说它的移动方式是一个创新的话，那么 Easy Motion 的移动方式就是一个划时代的革命。利用 VIM 的 #w 、 #b 、 :# 等操作，移动到一个位置就像是大炮瞄准一个目标，它可以精确到一个大致的范围内。而 Easy Motion 可以比作是精确制导，它可以准备无误的定位到一个字母上。

这种移动方式我曾在 Firefox 和 Chrome 的 VIM 插件中看到过，跳转到一个超链时就采用了同样的方式，但是由于浏览网页的特殊性与随意性，当时我没有适应。在编辑的时候就不一样了，编辑更加专注，更带有目的性，所以它能够极大的提高移动速度。享受这种光标指间跳跃，指随意动，移动如飞的感觉:-P

--help: 插件默认的快捷键是: <leader><leader>w ，效果如上图所示。

3. [NERDCommenter](#) - comment++

```
3 from tutu.forms.adminform import *
4 from tutu.models.blog import *
5
6 import os
7
8 def trans_is_active(is_active):
9     if is_active == '1':
10         return True
11     return False
12
13 def just_change(test):
14     """${3docstring for just_change}"""
15
NORMAL ▶ ~/tmp/common.py + ▶ unix < utf-8 < python 6%
5
4 "自动补全单引号，双引号等 Bundle 'underlog/ClosePairs'
3 Bundle 'Raimondi/delimitMate'
2
1 "快速 加减注释
0 Bundle 'scrooloose/nerdcommenter'
1
2 "D 代码片段
3 "Bundle 'vim-scripts/UltiSnips'
4 ""D for golang
5 "Bundle 'liambitoh/vim-golang'
```

又是一个写代码必备的插件，用于快速，批量注释与反注释。它适用于任何你能想到的语言，会根据不同的语言选择不同的注释方式，方便快捷。

--help: 十分简单的用法，默认配置情况下选择好要注释的行后，运行 <leader>cc 注释，<leader>cu 反注释，也可以都调用 <leader>c<SPACE>，它会根据是否有注释而选择来注释还是取消注释。

4. [Surround](#) - managing all the “{}[]()''<>” etc

```
3
2
1 cs"
0 'hello world'
1
2 ds'
3 'hello world'
4
5 ysiw]
6 hello
7
NORMAL ▶ ~/tmp/common.py ▶ unix < utf-8 < python 3%
5 "定义存放代码片段的文件夹 .vim/snippets下，使用自定义和默认的，将会的到全局，有冲突的会提示
4 let g:UltiSnipsSnippetDirectories=["snippets", "bundle/UltiSnips/UltiSnips"]
3
2 " 快速加入修改环绕字符
1 Bundle 'tpope/vim-surround'
0 "for repeat -> enhance surround.vim, . to repeat command
1 Bundle 'tpope/vim-repeat'
2
3
4
5
6 ""D task list
```

在写代码时经常会遇到配对的符号，比如 {}[]()''<> 等，尤其是标记类语言，比如 html, xml，它们完全依赖这种语法。现代的各种编辑器一般都可以在输入一半符号的时候帮你自动补全另外一半。可有的时候你想修改、删除或者是增加一个块的配对符号时，它们就无能为力了。

Surround 就是一个专门用来处理这种配对符号的插件，它可以非常高效快速的修改、删除及增加一个配对符号。如果你经常和这些配对符号打交道，比如你是一个前端工程师，那么请一定不要错过这样一个神级插件。

--help: : 部分常用快捷键如下:

```
Normal mode
ds - delete a surrounding
cs - change a surrounding
ys - add a surrounding
yS - add a surrounding and place the surrounded text on a new line +
indent it
yss - add a surrounding to the whole line
ySs - add a surrounding to the whole line, place it on a new line +
indent it
ySS - same as ySs

Visual mode
s - in visual mode, add a surrounding
S - in visual mode, add a surrounding but place text on new line +
indent it
```

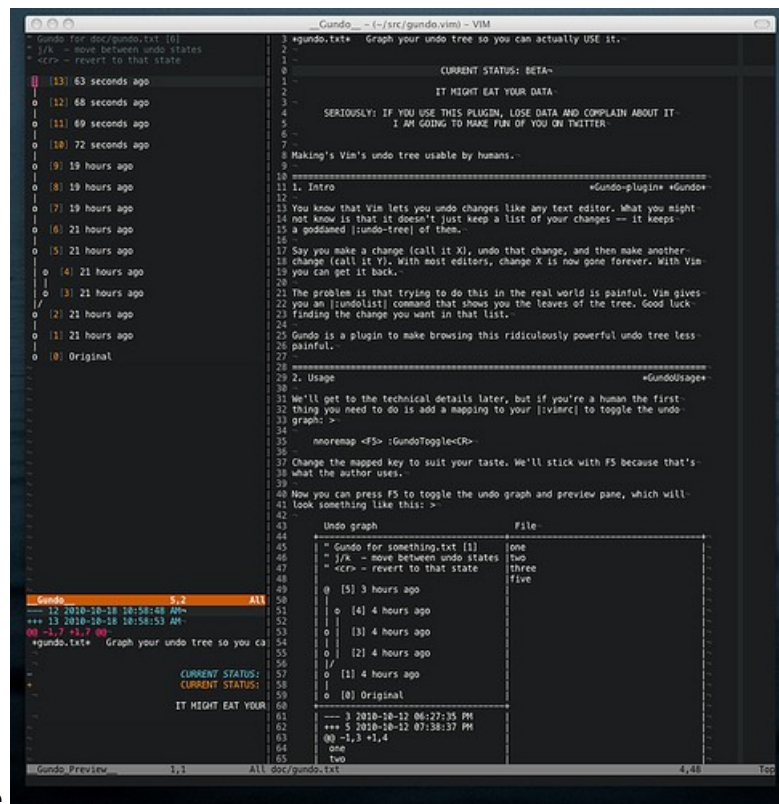
Insert mode

<CTRL-s> - in insert mode, add a surrounding

<CTRL-s><CTRL-s> - in insert mode, add a new line + surrounding + indent

<CTRL-g>s - same as <CTRL-s>

<CTRL-g>S - same as <CTRL-s><CTRL-s>



5. [Gundo](#) - time machine

现代编辑器都提供了多次的撤消和重做功能，这样你就可以很放心的修改文档或者恢复文档。可是假如你操作了 5 次，然后撤消 2 次，再重新编辑后，你肯定是无法回到最开始的 3 次编辑了，因为在你复杂的操作后，编辑器维护的 Undo Tree 实际上出现了分支，而一般的 CTRL+Z 和 CTRL+R 无法实现这么复杂的操作。

这时 VIM 的优势又体现了出来，它不仅提供无限撤消，VIM 7.3 之后还有永久撤消功能，即使文件关闭后再次打开，之前的修改仍然可以撤消。而 Gundo 提供了一个树状图形的撤消列表，下方还有每次修改的差异对比，分支一目了然，相当于一个面向撤消与编辑操作的版本控制工具。有了它，你的文件编辑就像是有了一台时光机，可以随心所欲的回到任何时间，乘着你的时光机，放心大胆的去穿梭时空吧:-P

--help: 通常会将这句加入 `_vimrc` : `nnoremap <Leader>u :GundoToggle<CR>`

6. [Sessionman](#) - session manager

这是 VIM 的 Session Manager，作用很简单，管理 VIM 的会话，可以让你在重新打开 VIM 之后立刻进行之前的编辑状态，就像 Windows 的休眠一样，相信它一定是你工作的好伴侣。

--help: 我的配置如下：

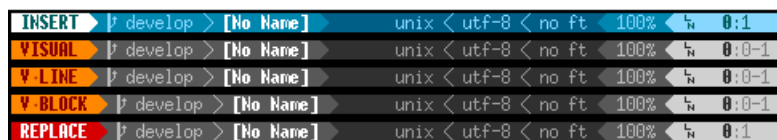
```
set sessionoptions=blank,buffers,curdir,folds,tabpages,winstate
```

```
nmap <leader>sl :SessionList<CR>
nmap <leader>ss :SessionSave<CR>
nmap <leader>sc :SessionClose<CR>
```

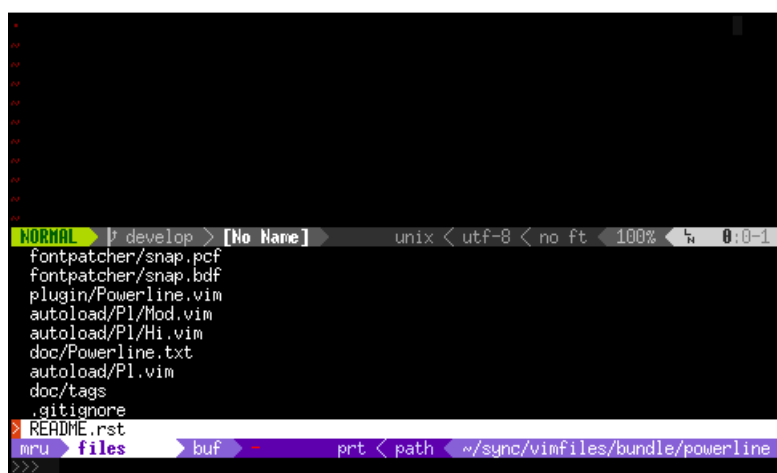
7. [Powerline](#) - ultimate statusline utility



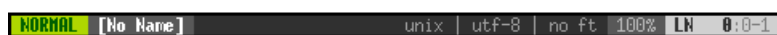
Normal mode



Mode-specific highlighting for all vim modes



Custom statuslines for many plugins and filetypes



Default appearance with unpatched font

增强型的状态栏插件，可以以各种漂亮的颜色展示状态栏，显示文件编码，类型，光标位置，甚至可以显示版本控制信息。不仅功能强大，写着代码时看着下面赏心悦目的状态状，心情也因此大好。像我一样的外观控一定无法抗拒它:-)

--help：简单实用，无需多余的配置。

终极配置: spf13

至此，我经常用到的所有插件都介绍完了，如果你也都安装尝试一下的话，相信很容易就配置出来符合你个人习惯的强大的 IDE。也许有人会想，这么多的主题、个性化设置、插件，配置太麻烦，有没有已经配置好的，可以直接拿来使用呢？其实我当时也有一样的想法，在折腾了很久之后，发现 `_vimrc` 已经非常庞大且混乱，亟需整理。再后来就发现了它， `spf13`：



```
4 "
5 "
6 "
7 "
8 "
9 "
10 "
11 " This is the personal .vimrc file of Steve Francia.
12 " While much of it is beneficial for general use, I would
13 " recommend picking out the parts you want and understand.
14 "
15 " You can find me at http://spf13.com
16 " }
17
18 +-- 21 lines: Environment -----
19
20 +-- 66 lines: Bundles -----
21
22 " General {
23   set background=dark           " Assume a dark background
24   if !has('gui')
25     set term=$TERM              " Make arrow and other keys work
26   endif
27   filetype plugin indent on     " Automatically detect file types.
28   syntax on                     " syntax highlighting
29   set mouse=a                   " automatically enable mouse usage
30   scriptencoding utf-8
31   autocmd BufEnter * if bufname("") !~ "^\[A-Za-z0-9\]*://" | lcd %:~ | #
32   " always switch to the current file directory.
33
34   " set autowrite                " automatically write a file when leav#
35   set shortmess+=filnrxo0tT     " abbrev. of messages (avoids 'hit ente#
36   set viewoptions=folds,options,cursor,unix,slash " better unix / windows#
37   set virtualedit=onemore       " allow for cursor beyond last character
38   set history=1000              " Store a ton of history (default is 20)
39   set spell                     " spell checking on
40   set hidden                    " allow buffer switching without saving
41
42 +-- 6 lines: Setting up the directories -----
43 " }
44
45 +-- 50 lines: Vim UI -----
46
47 +-- 13 lines: Formatting -----
48
49 +-- 80 lines: Key (re)Mappings -----
50
51 +-- 202 lines: Plugins -----
52
53 +-- 13 lines: GUI Settings -----
54
55 +-- 41 lines: Functions -----
56
57 N BR: 3.0 | .vimrc | unix | utf-8 | vim | 2% | LN 15 C 4
```

它是 [Steve Francia's Vim Distribution](#)，但是组织的非常整洁，容易扩展，并且跨平台，易于安装维护。在看到的所有 `_vimrc` 中，这是写的最漂亮的一个。只需要一个简单的脚本就可以 [安装](#)，这里面利用了方便的 Vundle 集成了绝大部分前面介绍的插件，并且还有大量其它的插件，具体可以看它的 `.vimrc.bundles`。

因为它完美的结构组织，你完全可以在不修改它任何文件的基础上，对应增加几个自己的 `~/.vimrc.local`，`~/.vimrc.bundles.local`，`~/.vimrc.before.local` 文件来增加自己的个性化配置，或者增加删除插件，可扩展性极强。在我的 `_vimrc` 乱成一团的情况，果

断 fork 并安装了这个 Distribution，增加了自己的一些配置，最终形成了现在的 VIM。如果你也不愿折腾配置，那么完全可以直接安装它，省事方便的同时还可以学习一下它的组织结构，一举两得。

与其它软件集成

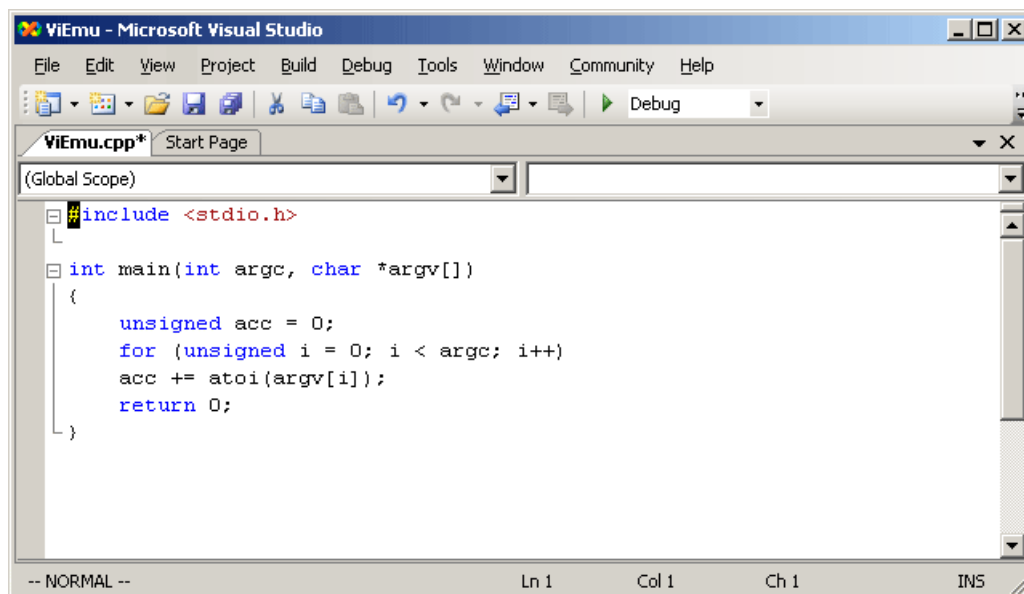
因为 VIM 的操作方式广泛为人们所逐渐接受，尤其是经常工作在 Linux 下的人们，所以它越来越多的被集成到其它一些常用的工具上，我用过的就包括：

- Visual Studio

本身 Windows 下的 gVim 安装包在安装时会提供一个集成到 Visual Studio 中的插件 VsVim，可以选择安装，但它是另开一个 VIM 的窗口来编辑当前的文件，我并不习惯这种方式，所以又找到了 [ViEmu](#)，它完美的将 VIM 的操作方式集成到了 Visual Studio 中，让你根本感觉不到这是在使用 Visual Studio。更加强大的是，它可以完美的和 Visual Assist 集成：

Build 1854 contains a workaround for case=58034. Create a binary registry value named TrackCaretVisibility under HKCU\Software\Whole Tomato\Visual Assist X\VANet10 and set its value to 00 for compatibility with ViEmu. (The value defaults to 01 and is created for you upon exiting VS the first time you run 1854 or higher.) Note you need to close all IDEs before editing this registry key, to avoid Visual Assist X overwriting your change when it exits.

在遇到 YouCompleteMe 之前，这就是我所采用的编程环境。但这是一个商业版的插件，只有 30 天的试用期，如果你真的喜欢它的，完全可以买下它，绝对物超所值。更为强大的是它还支持 Xcode、Word、Outlook、SQL Server，这一定是一个极端的 Vimer 的项目:-)，来看看它的动画：



- Source Insight

VIM 也可以集成到 Source Insight 中，不过我没有去找相应的插件，只找一种和前面介绍的 VsVim 一样的方法：

- 在 Source Insight 菜单中，Options-Custom Commands
- Run: “C:\Program Files\Vim\vim74\gvim.exe” -remote-silent +%l %f
- Dir: %d
- Add 之后再 Options-Key Assignments，将它绑定到一个快捷键中，比如 F11

这样编辑一个文件时，如果你想打开 VIM 时，直接按 F11，它就会跳到当前行，编辑完之后关闭 VIM，又回到 Source Insight 中。这种方法我用过一段时间，后来由于很少用 Source Insight 写代码，也逐渐淡忘了。

- Firefox/Chrome

在狂热于 VIM 的年代，我曾想把一切操作都变成 VIM 的方式，包括上网。所以就找到了 [Vimperator](#)，但终究由于上网是一种更加随性、无目的的行为，拿着鼠标随便点点完全可以了，所以也就放弃它，回归到正常的操作方式下，有兴趣的可以把玩一下，很有意思，之前谈到的 Easy Motion 我就在这里见识过。Chrome 下也有相应的 [插件](#)。

一些资源

最后附上一些有趣有用的资源：

- 一篇非常好的为什么使用 VIM 的文章，请看 [这里](#)
- 为什么 VIM 使用 HJKL 作为方向键？请看 [这里](#)
- 为什么 VIM 和 EMACS 被称为最好的编辑器？这看 [这里](#)
- VIM 作者的演讲：《[高效编辑的 7 个习惯](#)》，视频请点 [这里](#)

写在最后

网上可能有很多人像我之前一样，过于关注工具本身，而忽略了一个非常重要的问题：工具之所以称为工具，仅仅在于它是被人们拿来使用，只要顺手就好，用它来做的事情才是关键。对于我们开发人员来说，专业知识永远比工具更为重要。自打 VIM 出生以来，就有几个亘古不变的话题：

- VIM vs Emasc
- VIM vs 其它编辑器
- VIM vs IDE

争论从来没有平息过，从远古时期的大牛们，到刚刚踏入 VIM 阵营的我们，也从来没有一个结论。也许很多人争吵已经不再是单单的编辑器之争，而是出于维护心目中最好的工作方式，甚至哲学之争。但对于大部分人来说，只要你的工具足够称手，那么多写几行代码，多看些书，远比参与这些无休止的争吵强得多。但如果你更深一步，开发出更好的编辑器，或者插件，那又另当别论了。

这篇教程至此也将告一段落，说是教程，本文却并没有详细的介绍如何入门，反而回忆了一大段个人学习 VIM 的经历，然后介绍了常用的优秀插件。也许看完本文，你并不一定能够学会 VIM，但是它提供

了很多比本文更有价值去学习的资源，给了你一个整体的认识，让你看到 VIM 可以强大到什么程度，避免走很多弯路。看完本文之后，你能够知道如何入门，如何去选插件，我想，对于本文来说，这就够了。

(全文完)