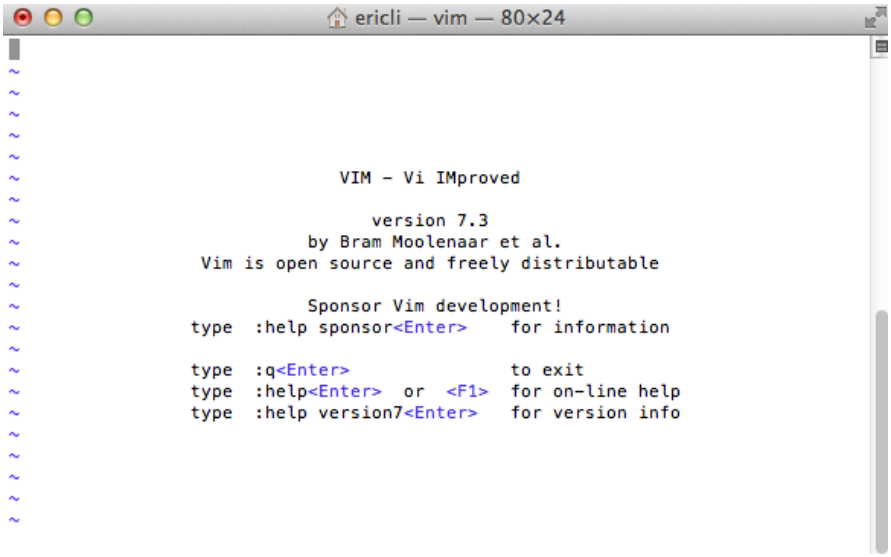
Mac开发利器之程序员编辑器MacVim学习总结

发布者：木一琦 时间：2013-06-18 09:36:08

Emacs和Vim都是程序员专用编辑器，Emacs被称为神的编辑器，Vim则是编辑器之神。至于两者到底哪个更好用，网络上两大派系至今还争论不休。不过，相比之下，Emacs更加复杂，已经不能算是一个编辑器了，有人这么说：Emacs是伪装成编辑器的操作系统。与之相反，Vim的定位很明确，就是要做一个强大的编辑器。由于笔者精力有限，决定还是选择自己认为相对简单点的Vim来学习。因此，笔者将会在本文跟大家介绍Mac下Vim的安装以及简单使用。

首先，Mac系统默认已经安装了Vim。打开终端，输入vim，回车。可看到如下界面，即表示Vim已经安装了。在vim里输入":q"

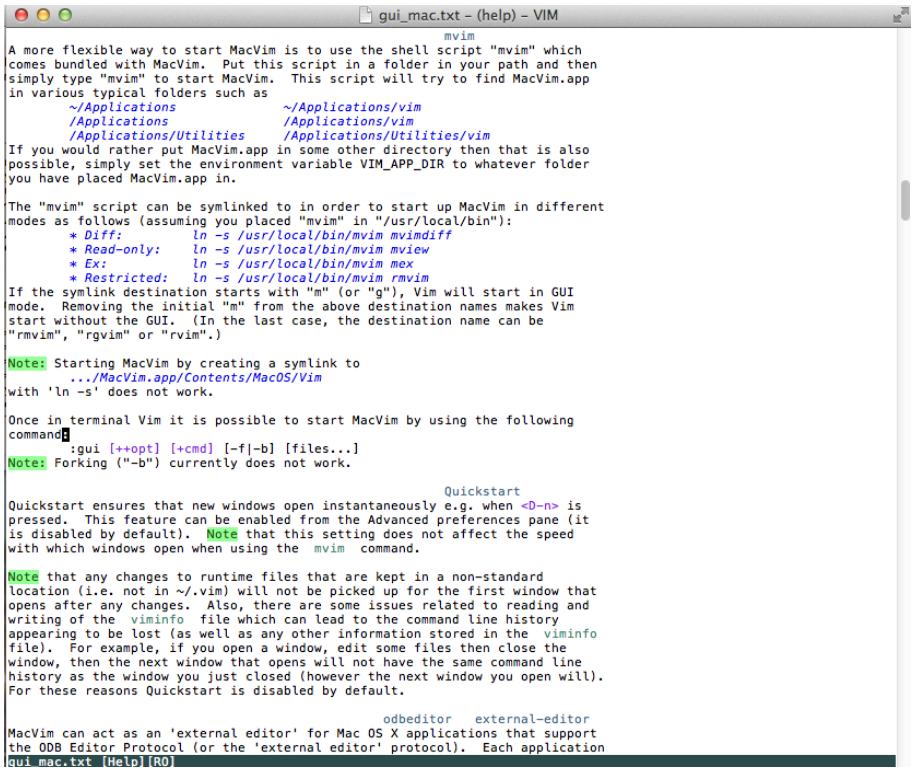


在终端上，界面非常之简洁，如果你需要更加复杂的界面以及使用Vim更多的特性，需要下载客户端安装。在Mac上目前存在两个版本的Vim客户端：

1、MacVim。使用Cocoa GUI，这是Mac上更新还很活跃的版本，也是Mac上最多人使用的版本。下载地址：http://code.google.com/p/macvim/

2、使用Carbon GUI的版本，但是这个版本目前基本上不再更新。下载地址：http://sourceforge.net/projects/macosxvim/files/

本文主要介绍MacVim，MacVim支持多窗口标签编辑以及全屏等新特性。下载了相应MacVim压缩包，解压以后将MacVim.app拖入/Applications目录下。启动MacVim，输入":h mvim", 按照提示，需要将mvim脚本文件拷贝到"/usr/local/bin"目录下。打开终端,输入命令 "cd /usr/local"以及"sudo mkdir bin",最后输入"sudo cp -f /Users/ericli/SoftWare/MacVim-snapshot-66/mvim /usr/local/bin/",这样就可以在终端输入命令"mvim"来快速启动MacVim 。如果你不想将MacVim.app放到/Applications目录下，只需要在终端设置VIM_APP_DIR变量为你想放置MacVim.app的目录并导入到$HOME/.bash_profile中即可，比如：export VIM_APP_DIR=$HOME/SoftWare/MacVim-snapshot-66/。在MacVim输入":h macvim", 可以查看MacVim的使用帮助介绍文档。



现在MacVime的界面还是很简陋，下面开始介绍如何简单配置MacVim：

1、创建~/.bashrc文件

在终端键入"cd $HOME" 和“touch .bashrc”，即创建成功。打开文件，输入“alias vim=mvim”，并保存别名变量。然后再在终端键入“source .bashrc”使.bashrc文件生效。这样，在终端输入vim和mvim都可以启动MacVim.app了。

在这里稍微解释下几个文件的作用，/etc/bashrc、/etc/profile是系统全局环境变量设置，给所有用户使用，~/.bashrc、~/.bash_profile、~/.bash_login、~/.profile是用户目录下的私有变量设置。当进入系统运行一个bash shell进程时，读取环境设定过程如下：1、读取全局环境变量设置/etc/profile,然后根据其内容读取/etc/profile.d、/etc/bashrc等设定，但是注意/etc/profile只在第一次运行bash shell时读取一次，而/etc/bashrc在每次运行bash shell都读取；2、然后读取当前用户目录下的~/.bash_profile、~/.bash_login或~/.profile三个中的其中一个文件的局部变量设置，而且只在第一次运行bash shell时读取，只有读取~/.bash_profile失败才会读取~/.bash_login，如果前边两文件读取失败才会读取~/.profile；3、最后根据~/.bash_profile的内容读取当前用户目录下的~/.bashrc文件中的局部环境变量设置，~/.bashrc文件在每次打开新的bash shell都读取一次。总结一下，/etc/bashrc、/etc/profile、~/.bashrc、~/.bash_profile、~/.bash_login、~/.profile都能设置环境变量，而/etc/profile、~/.bash_profile、~/.bash_login、~/.profile可以设定路径、环境变量等，只能登入的时候执行一次；/etc/bashrc、~/.bashrc保存路径、命令别名等，每次打开新的bash shell都会执行一次，通常前者/etc/profile、~/.bash_profile调用后者/etc/bashrc、~/.bashrc。更详细的资料可以参考该链接：http://blog.chinaunix.net/uid-24591881-id-2124780.html 。

2.配置~/.vimrc和~/.gvimrc

首先，在MacVim编辑器中输入“:version”,看到如下图示信息：

```
:version
VIM - Vi IMproved 7.3 (2010 Aug 15, compiled Dec 12 2012 16:22:25)
MacOS X (unix) version
Included patches: 1-754
Compiled by Bjorn Winckler <bjorn.winckler@gmail.com>
Huge version with MacVim GUI.  Features included (+) or not (-):
+arabic +autocmd +balloon_eval +browse ++builtin_terms +byte_offset +cindent +clientserver +clipboard
+cmdline_compl +cmdline_hist +cmdline_info +comments +conceal +cryptv +cscope +cursorbind +cursorshape
+dialog_con_gui +diff +digraphs +dnd -ebcdic +emacs_tags +eval +ex_extra +extra_search +farsi +file_in_path
+find_in_path +float +folding -footer +fork() +fullscreen -gettext -hangul_input +iconv +insert_expand +jumplist
+keymap +langmap +libcall +linebreak +lispindent +listcmds +localmap -lua +menu +mksession +modify_fname +mouse
+mouseshape +mouse_dec -mouse_gpm -mouse_jsbterm +mouse_netterm +mouse_sgr -mouse_sysmouse +mouse_urxvt
+mouse_xterm +multi_byte +multi_lang -mzscheme +netbeans_intg +odbeditor +path_extra +perl +persistent_undo
+postscript +printer +profile +python -python3 +quickfix +reltime +rightleft +ruby +scrollbind +signs +smartindent
-sniff +startuptime +statusline -sun_workshop +syntax +tag_binary +tag_old_static -tag_any_white -tcl +terminfo
+termresponse +textobjects +title +toolbar +transparency +user_commands +vertsplit +virtualedit +visual
+visualextra +viminfo +vreplace +wildignore +wildmenu +windows +writebackup -X11 -xfontset +xim -xsmp
-xterm_clipboard -xterm_save
   system vimrc file: "$VIM/vimrc"
     user vimrc file: "$HOME/.vimrc"
      user exrc file: "$HOME/.exrc"
  system gvimrc file: "$VIM/gvimrc"
    user gvimrc file: "$HOME/.gvimrc"
    system menu file: "$VIMRUNTIME/menu.vim"
  fall-back for $VIM: "/Applications/MacVim.app/Contents/Resources/vim"
Compilation: clang -c -I. -Iproto -DHAVE_CONFIG_H -DFEAT_GUI_MACVIM -Wall -Wno-unknown-pragmas -pipe  -DMACOS_X_UNIX
 -no-cpp-precomp  -g -O2 -U_FORTIFY_SOURCE -D_FORTIFY_SOURCE=1
Linking: clang   -L.    -L.         -L/usr/local/lib -o Vim -framework Cocoa -framework Carbon      -lncurses  -lico
nv -framework Cocoa    -fstack-protector -L/usr/local/lib  -L/System/Library/Perl/5.12/darwin-thread-multi-2level/CO
RE -lperl -lm -lutil -lc -framework Python   -framework Ruby
Press ENTER or type command to continue
```

从上图可以看出一些MacVim的配置信息：

$VIM系统变量的路径为："/Applications/MacVim.app/Contents/Resources/vim"

MacVim的系统配置文件vimrc的路径：“$VIM/vimrc”

用户配置文件vimrc的路径："$HOME/.vimrc"

MacVim编辑器的用户初始化配置文件的路径："$HOME/.exrc" (默认不存在，如果需要使用则在终端输入 “touch $HOME/.vimrc”创建)

MacVim的系统配置文件gvimrc的路径："$VIM/gvimrc"

用户配置文件gvimrc的路径："$HOME/.gvimrc"

MacVim的菜单文件的路径："$VIMRUNTIME/menu.vim"

另外解释下， $HOME为当前用户目录的路径，$VIMRUNTIME路径为"/Applications/MacVim.app/Contents/Resources/vim" . 可以在MacVim中输入”:echo $VIMRUNTIME“来查看这些系统变量的路径。

其次，从系统文件vimrc_example.vim和gvimrc_example.vim中copy标准的内容，保存到用户的配置文件~/.vimrc和~/.gvimrc中。在MacVim中输入如下命令：

:e $VIMRUNTIME/vimrc_example.vim
:saveas ~/.vimrc
:e $VIMRUNTIME/gvimrc_example.vim
:saveas ~/.gvimrc

3、添加插件

通常有如下两种方式添加插件：

其一：拷贝该 plugin_name.vim插件到你个人插件目录~/.vim/plugin/下，拷贝plugin_help.txt到你个人文档目录~/.vim/doc/下，若存在该插件syntax的vim支持文件则放到~/.vim/syntax/目录下，如果以上目录不存在到终端输入如下命令创建：

```
cd $HOME
mkdir .vim
mkdir .vim/plugin
mkdir .vim/doc
mkdir .vim/syntax
```

比如，拷贝matchit.txt到~/.vim/doc/下，拷贝matchit.vim到目录~/.vim/plugin/下，

其二，在你的vimrc配置文件最后加一行命令：

```
source $VIMRUNTIME/macros/matchit.vim0
```

最后，还有一种不常用的方式可以安装插件，将插件、文档、syntax支持文件分别放到$VIMRUNTIME/plugin/ 、$VIMRUNTIME/doc/、$VIMRUNTIME/syntax/三个目录下，或者/usr/share/vim/vim73/plugin/、/usr/share/vim/vim73/doc/、/usr/share/vim/vim73/syntax三个目录下都可以使MacVim自动加载插件。

以上安装插件的方法都比较麻烦，并且每个插件的文件分散到几个文件夹去，不方便管理。在这里给各位推荐一个很方便管理插件的Vim插件vundle：https://github.com/gmarik/vundle。vundle可以使得安装的插件的文件都放到同一个目录下，并且简单设置就可使用。安装方法如下：

首先，在终端输入如下命令：

```
git clone https://github.com/gmarik/vundle.git ~/.vim/bundle/vundle
```

然后，在~/.vimrc里写入以下配置：

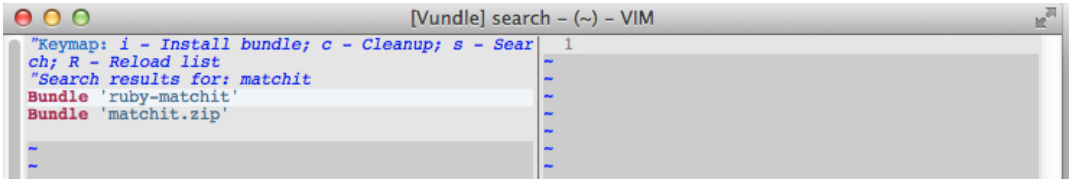
```
set nocompatible " be iMproved
filetype off " required!

set rtp+=~/.vim/bundle/vundle/
call vundle#rc()

" let Vundle manage Vundle
" required!
"这是vundle本身的设置
Bundle 'gmarik/vundle'

" My Bundles here:
"这里是设置你自己自定义的插件的设置vundle设置，注意：下载的插件git为：https://github.com/godlygeek/tabular.git，则设置为Bundle '
godlygeek/tabular';https://github.com/gmarik/vundle.git设置则为 Bundle 'gmarik/vundle'
" original repos on github
Bundle 'godlygeek/tabular'

" vim-scripts repos，vim-scripts的访问地址，格式则如下：
Bundle 'L9'
Bundle 'FuzzyFinder'
" non github repos ，非git的访问地址的，格式如下：
Bundle 'git://git.wincent.com/command-t.git'
" ...

filetype plugin indent on " required!
"
```

```
" Brief help
" :BundleList - list configured bundles
" :BundleInstall(!) - install(update) bundles
" :BundleSearch(!) foo - search(or refresh cache first) for foo
" :BundleClean(!) - confirm(or auto-approve) removal of unused bundles
"
" see :h vundle for more details or wiki for FAQ
" NOTE: comments after Bundle command are not allowed..
```

最后，登陆MacVim，运行`:BundleInstall`命令。这样，Vundle.vim管理插件就安装成功。接下来继续简单介绍下，Vundle插件的使用：

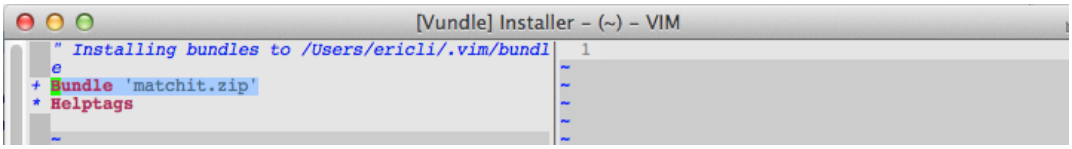
搜索新的插件：

:BundleSearch matchit



安装新的插件：

:BundleInstall matchit.zip



最后在~/.vimrc中的注释" vim-scripts repos下，加入如下命令并保存.vimrc文件，这样一个插件就安装成功了：

Bundle 'matchit.zip'

更新所有安装的插件(这个命令会根据~/.vimrc的Bundle 设置来更新插件)：

:BundleInstall
列出所有安装的插件列表：

:BundleList

清除插件命名(这个命令我测试有些插件删除不了，手动删除~/.vim/bundle的相关插件的目录，并将~/.vimrc中的相关配置去掉即可)：

:BundleClean

上述中的Tabular插件是用于编码中的字符对齐的，关于其具体用法见链接：https://github.com/godlygeek/tabular.用法是选定一段文本，然后输入相应的命令来根据相应的字符对齐。几个常用的命令如下：

:Tab /=
:Tab /:
:Tab /:\zs
:Tab /|
分别是选定文本按=、:、:\zs、|等符号对齐。

推荐一些插件：

NERD_tree ： 一个文件管理插件，一些常用命令： 打开一个目录树(:NERDTree <启动目录> | <bookmark>) 关闭目录树栏(:NERDTreeClose)；切换目录树栏（:NERDTreeToggle）；定义标签（:Bookmark <name>）；定义Root标签（:BookmarkToRoot <bookmark>)。。。。。。更多命令和用法见 :help NERD_tree。

word_complete ： 代码自动补全

SuperTab ： 省去Ctrl-n或Ctrl-p快捷键，通过按tab键快速显示补全代码.

xptemplate ： 快速自动完成一些if、switch、for、while结构模板代码，支持c、c++、Lua、Ruby、Php、html、css、javascript等多种语言。一般是输入结构体的关键字后，再按Ctrl-\组合键即可完成代码补全，然后按Tab键跳转到不同的位置替换模板内容。比如：输入for后按Ctrl-\组

合键即可快速完成for结构的模板代码。

ctags：一个扫描记录代码的语法元素，并记录为tag，方便代码定位跳转等操作，MacVim自带，但是据说有点问题，笔者用Vundle安装的貌似也有问题，推荐用MacPorts安装，然后在~/.gvimrc配置中加入： let Tlist_Ctags_Cmd="/opt/local/bin/ctags"。用法：在终端 cd 进入到你的项目根目录，输入语句即可将项目所有代码文件打上tag：

```
ctags -R --c++-kinds=+px --fields=+iaS --extra=+q .
```

taglist: 可以用Vundle安装，在编辑代码文件时，输入命令":TlistToggle"在右边就会出现当前类的函数或变量列表。输入命令“:tag <函数名或变量、类>”，如果只有一个文件定义了该函数或变量、类，vim打开该文件并将光标定位到对应的位置；如果多个文件有这个函数名或变量、类的tag，将给提示，并可输入“:tselect” ,显示可选的文件。快捷键跳转Ctrl+],Ctrl-o。

Cscope: 功能跟ctags差不多，不过更加强大，MacVim默认已经支持，输入“:version”命令查看.

OmniCppComplete:功能跟taglist差不多。

a.vim: 在.cpp文件和.h头文件间快速切换的插件。

grep.vim： 在工程中查找词汇的插件。

minibufexplorerpp:操作缓存buffer窗口。

quickfix：MacVim内置安装好了，不需要重新安装。显示一些命令查询结果以及编译错误等信息。

Command-t：用Commad-t命令快速查找切换文件。如果是用Vundle安装的话，还不能使用，在MacVim中输入“:CommandT”命令会报错。用Vundle安装好打开终端，输入如下命令，等待编译完毕后就可以使用了：

```
cd ~/.vim/bundle/Command-T/ruby/command-t
ruby extconf.rb
make
```

NERD_commenter.vim： 注释插件。

DoxygenToolkit.vim: 用于快速生成注释，并由注释生成文档。

winmanager： 可以用Vundle安装，管理窗口的插件，可以跟NERD_tree、Taglist插件结合，打造一个类似IDE的界面。只需要在NERD_tree.vim中加入如下代码：

```
let g:NERDTree_title = "NERDTree"
function! NERDTree_Start()
exec 'NERDTree'
endfunction
function! NERDTree_IsValid()
return 1
endfunction
```

并且在winmanager.vim的找到下面代码，增加一句代码：

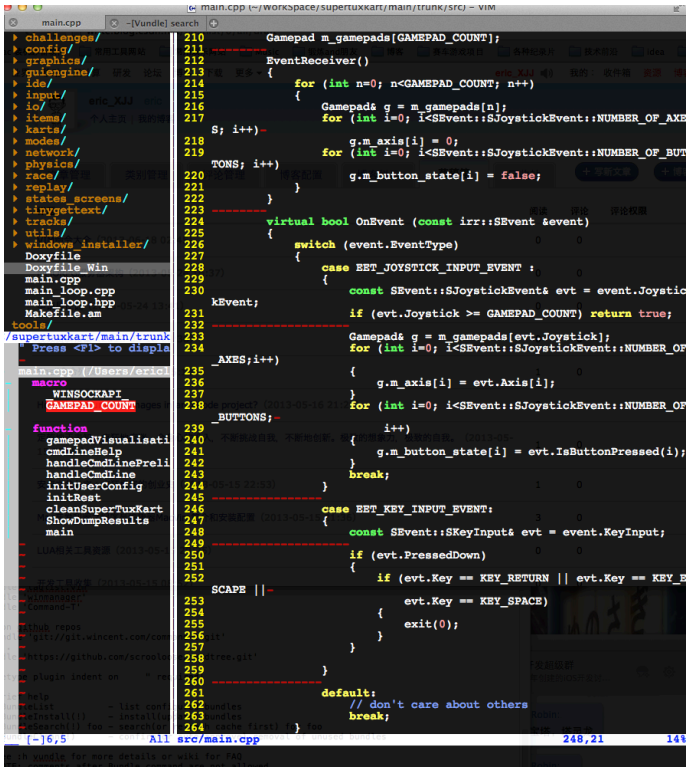
```
" toggle showing the explorer plugins.
function! <SID>ToggleWindowsManager()
if IsWinManagerVisible()
call s:CloseWindowsManager()
else
call s:StartWindowsManager()
"NERDTree 打开的时候有一个空白窗口，需要关闭
exec 'q'
end
endfunction
```

最后在~/.gvimrc中加入如下配置：

```
"在vim左侧显示窗口Taglist和文件列表窗口
"let g:winManagerWindowLayout='FileExplorer,BufExplorer|TagList'
```

```
let g:winManagerWindowLayout='NERDTree, BufExplorer|TagList'
"打开vim时自动打开winmanager
let g:AutoOpenWinManager = 1
"定义打开关闭winmanager的快捷键为 wt组合键命令
nmap wt :WMToggle<cr>
```

重启，即可看到一个类似IDE的Vim界面。



4、Vim的模式与一些常用命令

i、Vim的模式

Vim有三种模式，Normal 、Insert 、 Visual。在Normal模式下，可以输入运行命令；在Insert模式下，可以输入编辑文本；在Visual模式下，可以对选定的文本运行命令操作并该命令仅仅作用于选定文本。启动Vim的默认模式是Normal模式，输入 i、I、a、A、o、O、s、S、r、R可以进入Insert模式，其中区别在于：

进入Insert模式的命令区别

命令	区别
i	在光标之前插入字符
I	在光标所在行的所有字符开头之前插入字符
a	在光标之后插入字符
A	在光标所在行的所有字符之后插入字符
o	在当前光标的下面打开新的一行
O	在当前光标的上面打开新的一

	行
s	删除当前光标下的字符
S	删除当前光标所在行的所有字符
r	用新输入的字符替换当前光标所在字符，然后立即返回Normal模式
R	在当前光标位置所在字符开始往后一直用新输入的字符替换所有原有字符

在Nomal模式中输入v进入Visual模式。 在Insert模式或Visual模式中按Esc键即返回Normal模式。

ii、Vim的常用命令（不做特殊注释的话，下面的命令一般在Normal模式生效,一般对文本操作的命令在Visual模式下也同样生效）

(1)、光标的移动：

h(左)、j（下）、k（上）、l（右）移动光标。还可以在命令之前输入数字，指定要移动多少行。比如：7h表示左移7个光标，6j表示光标下移6行。

光标向前跳多个词到达词首，比如5个词，输入5w;光标向后跳7个词到达词首，输入7b。跳到3个单词后的词末，输入3e;跳到3个单词前的词末，输入3ge。

移动到行首第一个非空字符，输入"^"或按Home键；移动到光标所在行的第一个字符（包括空格)；移动到行尾，输入"$"或按End键。注意，这个只有"$"或按End键之前加数字会生效。

向前移动到某个指定的字符，比如向前找第三个“h”，输入”3fh“；向前把光标移动到第四个"b"字符的前一个字符上，输入”4tb“。向后移动到某个指定的字符，比如向后查找第三个”h“，输入”3Fh“;向后把光标移动到第四个”b“的后一个字符上，输入”4Tb“。这四个命令都可以使用”;“命令重复，","命令反向重复。光标不会移出当前行。

"%"命令可以将当前光标所在的括弧移动到匹配的括弧的位置，比如：从”)“移动到”(“，反之亦然。

gd命令到达光标所在处函数或变量定义之处。

”*“和”#“命令匹配当前光标所在单词, ”*“移动光标的下一个匹配单词，而”#“命令是移动光标到上一个匹配单词。

移动到指定行，比如想移动到第30行，输入30G或输入30gg。没有数字限制的话，G命令将光标移动到文件的尾部，gg命令移动光标到文件开始。

移动都某部分，比如到文件大约10%的行位置，输入”10%“。

H命令移动光标到视野内的第一行，M移动到视野内的中部，L移动光标到视野内的最后一行。

确定光标的位置，输入Ctrl-G命令。

Ctrl-f光标向前移动一整屏，Ctrl-b光标向后移动一整屏。

”zt“命令将光标所在行移动到屏幕的顶部，”zz“命令将光标所在行移动到屏幕的中部，”zb“将光标移动到屏幕的底部。

(2)、删除字符：

删除某个字符，移动光标到所在字符，然后输入x或dl（光标下的字符）、X或dh(光标签的字符)；删除多个字符，可以在x之前加数字；

删除从当前光标到所在行上指定的某个字符之间的所有字符（包括该指定字符）

删除某行，移动光标到所在行，然后输入dd；

删除换行符将两行连起来，光标移动到要合并的两行的第一行，然后输入J。

利用命令组合模式，操作符-动作。d命令后跟一个光标移动动作，将删除从当前位置到光标移动到的目标位置的全部内容。例如：删除从当前光标向前的5个单词，命令为d5w；删除从当前光标位置到行尾的字符，命令为d$。

db表示从当前位置删除到前一个单词的开头。diw表示删除光标上的单词(不包括空格)。daw表示删除光标上的单词（包括空格）。

从当前光标位置上的字符一直删除到指定的字符（包括指定字符），用df命令，比如，删除到r，命令为dfr。从当前光标位置删除到指定字符的前一个字符，用dt命令，比如：dtr。

dG 表示从当前光标位置一直删除到文件的末尾。dgg表示从当前光标位置一直删除到文件的开始。

(3)、修改字符：

修改字符的操作命令是c。作用是删除字符后自动切换到Insert模式，比删除操作命令多了一个切换到Insert模式的功能。

修改光标下的 字符，cl.

修改光标前的字符，ch.

修改当前光标后的3个词，组合命令是c3w。

修改当前光标之前的单词，cb.

修改光标上的单词，不包括空格，ciw。

修改光标上的单词，包括空格，caw。

修改到指定字符（包括指定字符），比如修改到e，命令为cfe。

修改到指定字符之前一个字符，比如，修改到e之前的字符，cte.

修改光标所在行一整行，cc。

从当前光标位置修改到行尾，c$。

cG，修改到文件的末尾。

cgg，修改到文件的开头。

(4)、替换字符

r命令是进入Insert模式的命令之一，进入Insert模式等待你输入一个字符替换当前光标上的字符后自动返回Normal模式。

在r命令之前加入数字前缀，可以用一个字符替换当前光标后的多个字符。比如：3rp。但是例外的是，如果替换的字符是回车键，则只会用一个换行符替换多个字符。

使用R命令启动Replace模式, 这个模式下在当前光标位置所在字符开始往后一直可以持续用新输入的字符替换所有原有字符,直到你退出Replace模式。在这个模式下，使用退格键Backspace，原来被替换的字符会恢复。

(5)、查询字符：

从当前光标位置向前查找当前行上的某个指定字符的第一个字符，命令是f。比如： 查找w，fw。F也和f命令一样，但是查找方向相反。

从当前光标位置向前查找当前行上的某个指定字符的前一个字符，命令是t。比如： 查找o之前的字符，to。T也和t命令一样，但是查找方向相反。

向前查找字符单词命令, ”/“后加上想查找的字符单词。比如：”/a“ 就是查找字符”a“。想要向前查找当前光标后第三个匹配的a，还需要输入”3n“命令。向后第三个则输入”3N“。

向后查找字符，”?“后加上想查找的字符。想要向前查找光标后第三个匹配的字符，还需输入”3N“。向后第三个则输入”3n“。

查找字符通常区分大小写的，如果不想区分大小写，可在~/.vimrc文件中加入命令"set ignorecase"命令。

还可以将光标移动要查找的单词上，然后输入”*“命令,这样将取得当前光标上的单词并向前查找该单词。想要向前查找光标后第2个单词，可以输入"2*"。

向后查找当前光标上的单词，则可以输入”#“。想要向后查找当前光标的第2个单词，可以输入”2#“。

输入”/di“,也可能查找到单词”media“。如果想限定di只匹配单词开头，可以输入”/<di“。如果只想查找以di结尾的单词，则可以输入”/di>“。如果只想查找完整的单词"di"，输入”/<di>“。注意，”*“和”#“命令也是匹配整个单词查找的，如果想要部分匹配，输入”g*“和”g#“。

"/^di"，只匹配行首。"/di$"只匹配行未。”/^di$“仅仅匹配包括”di“的行并不带空格。

”/d.i“只匹配第一个字符是”d“,第二个字符是任意字符，第三个字符是”i“的字符串。

”/di/.“匹配带特殊字符”.“的字符串”di.“。

(6)、复制字符

使用d、x或其它命令删除文本的时候，这些被删除的文字将会被Vim缓存起来。可以使用p命令将该被删除的文本在当前光标的前面(比如前一行)重新粘贴出来。

使用y（yank：抽出）操作符命令可直接把文字拷贝到寄存器中，然后用p命令粘贴到其他地方。比如：拷贝4个单词，4yw（包括了单词后面的空格，用ye则不包括单词后的空格）。

yl拷贝当前光标下的字符，yh拷贝当前光标之前的字符。

yw拷贝当前光标到下一个单词的开头，yb拷贝当前光标到上一个单词的开头。yiw，拷贝光标上的单词（不包括空格）。yaw,拷贝光标上的单词（包括空格）。

y$拷贝到行尾。

yy命令拷贝一整行，快捷方式为Y。

yG，拷贝到文件的末尾。

ygg，拷贝到文件的开头。

（7）、文本对象

当处于某个单词或句子的中间，而又想删除整个单词或句子时，可以使用文本对象。比如：diw可以删除一整个单词（iw表示Inner word，不包括单词后的空格），如果想要包含单词后的空格也删除，可用命令daw。同样地，cis表示修改一整个句子（is表示Inner Sentence，不包括句子后的空格），如果想要包含句子后的空格也修改，则用命令cas。

同样地，在Visual模式下，也可以用文本对象命令来选中更多的文本。比如使用aw选中当前的整个单词，再次使用则选中多一个单词。

文本对象的表达式：<操作命令>a<文本对象>或<操作命令>i<文本对象>,操作命令可以是任何命令，如d，y，c；文本对象可以是一个单词w，一个句子s，一个段落p，也可以是一个特殊的字符："、'、)、}、]。

使用 :help text-objects 可以在帮助里查看更多的文本对象的更多命令内容。

使用:help operator 可以在帮助里查看更多的操作符命令内容。

(8)、撤销与重做：

撤销上一个编辑操作，输入u；撤销所有在前一个编辑行上的操作,输入U;重做，输入Ctrl+r。

"."命令表示重复最后一次除u命令、Ctrl-r命令和冒号命令之外的任何操作命令。

(9)、查找命令

输入"/"、”?“、”:"后，按方向键上箭头或下箭头，可以查找同样的历史命令记录。

(10)、命令次数：

可以在很多命令之前加上数字，表示执行该命令的次数。

在组合命令的中，数字可以放在不同的地方。比如：修改5个单词，5cw和c5w都一样；删除10个单词，可以写成10cw、c10w，或者2c5w、5c2w。其中的差别在于数字修饰的命令不一样，但是作用是一样的。

(11)、退出：

保存文件并退出，输入ZZ或":wq"或“:x”；放弃修改并退出，输入“:q!”；强制退出所有打开的文件,输入“:qa!”

(12)、放弃执行命令：

Esc按键可以终止大部分命令。

(13)、帮助：

输入“:help”进入帮助窗口；输入ZZ，退出帮助窗口。

要获得具体的帮助，比如关于u命令的帮助，输入类似“:help u”。

默认显示Normal模式的帮助，要显示其他模式的命令，可以使用“i_”前缀。

获取命令参数的帮助，比如：“:help -e”.

使用“:helptags ~/.vim/doc” 命令产生插件的本地tags文件，从而可以在帮助中查询插件的文档。

(14)、标签跳转：

被 [ ]包起来的文字是一个标签链接，光标移动到 [ ]之间，按Ctrl+]组合键跳转到标签链接的地方。Ctrl+t或 Ctrl + O跳回前一个标签。

使用G命令或者”n“命令或者查找”/“命令跳到其他行后，Vim会记录下你跳过来的地方，使用两个单引号('')即可跳回原来的地方；或者使用Ctrl-o命令跳到较旧的地方；Ctrl-i命令或Tab键跳到较新的地方。输入":jumps"命令可以看到光标跳转的位置列表信息。

使用”m“命令标记当前光标的位置，位置的名称只能为a~z的单个字母(区分大小写)。比如：定义当前光标的位置为”E“，命令为"mE"。设置后可以通过单引号'+光标名称，或者上句号`+光标名称可以跳到指定的光标位置，区别是单引号 ’E命令跳转到标记光标位置所在行的首个非空字符（列的位置跟标记的时候不同），而上句号 `E跳转到标记的光标原来位置（包括行和列的位置都跟标记的时候一样）。输入”:marks“命令可以查看所有带名称的标记信息列表，其中有几个特殊的标记。比如：单引号 ' ，表示跳转前的位置。双引号 "表示Normal模式最后编辑的位置。中括号 [表示文件第一行的第一列。 中括号]表示文件最后一行的第一列。 ^ 表示Insert模式下最后修改的位置。 . 表示无论是Insert模式或Normal模式最后修改的位置的开头。

(15)、显示Vim相关的文件路径

:scriptnames命令可以查找~/.vimrc或~/.gvimrc或插件等文件的路径。

(16)、在vim中打开编辑文件命令：

```
:e! ~/.vimrc
```

保存文件：

```
:w
```

另存为：

```
:saveas pathFile
```

当同时打开多个文件，在多个文件间切换命令：

```
:bn //下一个文件
:bp //前一个文件
```

(17)、Visual模式

之前介绍过，在Normal模式下，输入v，进入Visual模式。这样，你对光标作任何移动操作，从当前光标位置到移动到的位置之间的文本都会高亮，此时输入操作命令，比如删除命令d，则高亮部分文本会被删除。

在Visual模式下光标的移动操作跟Normal模式差不多。注意，在Visual模式下，"o"命令表示返回到选中文字的另一端，这跟Normal模式不同。

（18）、快捷键与一些常用的命令：

一些经常使用的组合命令，存在等价的快捷单字符命令：

dl 的等价快捷键为 x，作用删除当前光标下的字符。

dh的等价快捷键为 X，作用删除光标左边的字符。

d$的等价快捷键为D，作用删除到行尾。

cl的等价快捷键为s，作用修改当前光标的字符。

cc的等价快捷键为S，作用修改一整行。

c$的等价快捷键为C, 作用修改到行尾。

yy的等价快捷键为Y，作用拷贝一整行。

一些常用的命令的如下：

c 修改操作命令

d 删除操作命令

y 复制到寄存器的操作命令

~ 修改选中的字符的大小写，原来大写的转换成小写，原来小写的转成大写。在Visual模式和Normal模式下都能生效。 注意，只有tiledeop设置后，这个才能成为一个操作命令，才可用该命令与其他光标移动的命令组合使用。

g~ 修改选中的字符的大小写操作命令，原来大写的转换成小写，原来小写的转成大写。

gu 修改选中的字符为小写操作命令。

gU 修改选中的字符为大写操作命令。

! 过滤警告操作命令

gq 选中文本格式化，文本拼接组合成句子或段落命令。在Normal模式和Visual模式下都能生效。

> 选中文本向右Tab缩进。在Normal模式和Visual模式下都能生效。

< 选中文本向左Tab缩进。在Normal模式和Visual模式下都能生效。

gd 到达光标所在处函数或变量定义之处。

块操作命令过程： 移动到想要进行操作的位置，比如行首或行尾；按组合键**Ctrl-v**，进入块操作模式；移动光标到任何地方，选定块操作的范围，比如：**hjkl**命令；最后输入**I**（在块的每一行首插入字符）或**A**（块的每一行尾部插入）进入**Insert**模式，然后输入想要插入的字符，按**Esc**键使块的每一行同样的位置生效。

自动补全功能：在**Insert**模式下，输入一个词的开头，然后按组合键**Ctrl-n**或**Ctrl-p**，就出现自动补全的提示。

宏录制：**q + <宏> +** 操作队列 **+ q** ，**@<宏> , @@**。**<宏>** 可以是任意单个字母（区分大小写）或任意单个数字， **q<宏>** 进入**recording**模式，该模式下你的操作记录会保存到寄存器**<宏>** 中；然后输入**@<宏>**操作命令将重新执行被记录到寄存器**<宏>**中的命令；**@@**命令跟**@<宏>**的作用一样。

可视化选择：**v,V,Ctrl-v. v,V**都可以进入**Visual**模式，而**Ctrl-v**可以进入**Visual Block**模式（即块操作模式），在这两个模式下，都可以移动光标选定编辑文本。对选定文本执行**c**，**y**，**d**等操作，下面有几个常用的对选定文本的操作：

J：把所有行连起来变成一行。
>或**<**操作，文本向左右**Tab**缩进。
=操作：选中文本自动缩进。

分屏：**[N]-Ctrl-w-s**、**[N]-ctrl-w-S**或**:[N]sp[lit],**组合命令将**Vim**的屏幕高度分出一个新的具有**N**行的屏幕**,**其中**[N]**和**[lit]**选项可以不输入。即**:sp**或**:5split**都是合法的分屏命令。

[N]-Ctrl-w-v或**:[N]vs[plit]**组合命令将**Vim**的屏幕宽度分出一个新的具有**N**列的屏幕，其中**[N]**和**[plit]**选项可以不输入，即**:vs**或**:5vsplit**都是合法的分屏命令。

[N]-Ctrl-w-n或**:[N]new** 组合命令将**Vim**的屏幕高度分出一个新的具有**N**行的屏幕，并打开一个新的空文件开始编辑。

:[N]vne[w] 组合命令将**Vim**的屏幕宽度分出一个新的具有**N**列的屏幕，并打开一个新的空文件开始编辑。

:q或**:q!**、**:close**或**:close!**（最后一个窗口不关闭）、**:hide**（最后一个窗口不关闭）可以关闭当前屏幕窗口。**:on[ly][!]** 除了当前窗口所有其他窗口都关闭。

Ctrl-w-w、**Ctrl-w-**方向键（包括**hjkl**和箭头方向键），可以在多个窗口中切换。**Ctrl-t**（**top-left**窗口）、**Ctrl-b**（**bottom-right**窗口）、**Ctrl-p**(上一个窗口)也是常用的切换窗口命令。

Ctrl-w-r、 **Ctrl-w-R**、 **Ctrl-w-x**、 **Ctrl-w-J\H\K\L\T**等可以改变窗口的布局。

Ctrl-w-+(增加尺寸)、 **Ctrl-w--**（减少尺寸）增加或减少当前窗口屏幕尺寸。**Ctrl-w-_**(竖屏最大化)、**Ctrl-w-|**（横屏最大化）用来最大化当前分窗口屏幕。

自己配置不同的语言函数自动补全的步骤：

创建~/.vim/dict 目录, 然后创建包含各种语言函数列表的txt文件；最后在~/.gvimrc文件加入如下命令：

```
au FileType cpp setlocal dict+=~/.vim/dict/cpp_function_list.txt
```

（19）菜单快捷键

编辑菜单：

Undo 快捷键 command+z

Redo 快捷键 shift+command+z

Cut 快捷键 command+x （剪切的文本可以粘贴于其他程序或vim）

Copy 快捷键 command+c （复制的文本可以粘贴于其他程序或vim）

Paste 快捷键 command+v

Select All 快捷键 command+a

文件菜单：

New window 快捷键 Command+n

New Tab 快捷键 Command+t

Open New File 快捷键爱你 Command+o

Save 快捷键 Command+s

Save as 快捷键 shifit+Command+s

Close window 快捷键 Shift+Command+w

Close 快捷键 Command+w

5、最后贴出本人的~/.vimrc和~/.gvimrc配置：

~/.vimrc配置如下：

```
" An example for a vimrc file.一般设置cli相关设置，即命令行相关设置，插件相关设置。原因加载顺序是先读取vimrc配置，然后读取plugin插件，然后加载GUI，最后再读取gvimrc配置文件。所以，GUI以及快捷键一般放到gvimrc里设置，有时候在vimrc设置跟界面显示相关的没作用，要在gvimrc里设置才有用。vimrc配置是vim，gvimrc配置文件是gvim，如果想要vim也有配色等，可以将界面相关的设置放在vimrc文件里重新设置一下。

"------------------------------------------------------------------------
" 基本设置
"------------------------------------------------------------------------
" When started as "evim", evim.vim will already have done these settings.
if v:progname =~? "evim"
finish
endif

"启用Vim的特性，而不是Vi的（必须放到配置的最前边）
set nocompatible

" 设置编码
set encoding=utf-8
set fenc=utf-8
set fileencodings=ucs-bom,utf-8,cp936,gb2312,gb18030,big5
```

```
"显示行号
set number

"设置默认打开窗口大小
set lines=70 columns=100

"设置窗口透明度
set transparency=10

"设置背景色
"set bg=dark

"用 koehler 调色板
colorscheme koehler

"隐藏工具栏和滑动条
set guioptions=aAce

"设置标签栏
"最多30个标签
set tabpagemax=30
"显示标签栏
set showtabline=2


"设定文件浏览器目录为当前目录
"set bsdir=buffer
"set autochdir

"保存100条命令历史记录
set history=100

"总是在窗口右下角显示光标的位置
set ruler

"在Vim窗口右下角显示未完成的命令
set showcmd

" 启用鼠标
if has('mouse')
set mouse=a
endif

"设置语法高亮
if &t_Co > 2 || has("gui_running")
syntax on
endif


"--------------------------------------------------------------------------
" 文本操作设置
"--------------------------------------------------------------------------
```

```
"设置字体
set gfn=Courier:h15

"设置自动缩进
"设置智能缩进
set tabstop=4
set shiftwidth=4
set softtabstop=4
set expandtab
set smarttab


"设置Tab键跟行尾符显示出来
set list lcs=tab:>-,trail:-

"设置自动换行
set wrap

"设置Insert模式和Normal模式下Left和Right箭头键左右移动可以超出当前行
set whichwrap=b,s,<,>,[,]

"设置光标移动到屏幕之外时，自动向右滚动10个字符
set sidescroll=10




"设置使~命令成为操作符命令，从而使~命令可以跟光标移动命令组合使用
set tildeop

"在Insert模式下，设置Backspace键如何删除光标前边的字符。这里三个值分别表示空白字符，分行符和插入模式之前的字符。
set backspace=indent,eol,start

"定义键映射，不使用Q命令启动Ex模式，令Q命令完成gq命令的功能---即文本格式化。
map Q gq

" CTRL-U 在Insert模式下可以删除当前光标所在行所在列之前的所有字符. Insert模式下，在Enter换行之后，可以立即使用CTRL-U命令删除换行符
。
inoremap <C-U> <C-G>u<C-U>

"使 "p" 命令在Visual模式下用拷贝的字符覆盖被选中的字符。
vnoremap p <Esc>:let current_reg = @"<CR>gvs<C-R>=current_reg<CR><Esc>


"----------------------------------------------------------------------------
" 搜索设置
"----------------------------------------------------------------------------
"打开搜索高亮
set hlsearch

"忽略大小写
```

```
set ignorecase

"在查找时输入字符过程中就高亮显示匹配点。然后回车跳到该匹配点。
set incsearch

"设置查找到文件尾部后折返开头或查找到开头后折返尾部。
set wrapscan



"------------------------------------------------------------------------------
" 文件设置
"------------------------------------------------------------------------------

"设置当Vim覆盖一个文件的时候保持一个备份文件，但vms除外（vms会自动备份。备份文件的名称是在原来的文件名上加上 "~" 字符
if has("vms")
set nobackup
else
set backup
endif


if has("autocmd")
"启用文件类型检测并启用文件类型相关插件，不同类型的文件需要不同的插件支持，同时加载缩进设置文件, 用于自动根据语言特点自动缩进.
filetype plugin indent on

"将下面脚本命令放到自动命令分组里，这样可以很方便地删除这些命令.
augroup vimrcEx
au! "删除原来组的自动命令

"对于所有文件类型，设置textwidth为78个字符.
autocmd FileType text setlocal textwidth=78

//vim启动后自动打开NerdTree
autocmd vimenter * NERDTree
autocmd vimenter * if !argc() | NERDTree | endif
"设置关闭vim NerdTree窗口
autocmd bufenter * if (winnr("$") == 1 && exists("b:NERDTreeType") && b:NERDTreeType == "primary") | q | endif

"当打开编辑文件时总是自动执行该脚本，跳转到最后一个有效的光标位置Mark标记。当一个事件正在处理时，不执行该脚本命令。
"行首的反斜杠用于把所有语句连接成一行，避免一行写得太长.
autocmd BufReadPost *
\ if line("'\"") > 1 && line("'\"") <= line("$") |
\ exe "normal! g`\"" |
\ endif

augroup END
else
"Enter换行时总是使用与前一行的缩进等自动缩进。
set autoindent
"设置智能缩进
set smartindent
endif
```

```
"编辑一个文件时，你所编辑的内容没保存的情况下，可以把现在的文件内容与编辑之前的文件内容进行对比，不同的内容将高亮显示
if !exists(":DiffOrig")
command DiffOrig vert new | set bt=nofile | r ++edit # | 0d_ | diffthis
\ | wincmd p | diffthis
endif


"------------------------------------------------------------------------------
" 插件设置
"------------------------------------------------------------------------------

"插件相关的设置
"matchit 的字符匹配自定义设置
let b:match_words = '\<if\>:\<endif\>,'
\ . '\<while\>:\<continue\>:\<break\>:\<endwhile\>'



"Vundle 的配置

filetype off " required!

set rtp+=~/.vim/bundle/vundle/
call vundle#rc()

" let Vundle manage Vundle
" required!
"管理Vim插件
Bundle 'gmarik/vundle'


" My Bundles here:
" original repos on github
"文本按字符对齐
Bundle 'godlygeek/tabular'

" vim-scripts repos
"实现“begin”/“end”类似地匹配,~/.vimrc文件中添加自定义的设置：let b:match_words = '\<if\>:\<endif\>,'
\ . '\<while\>:\<continue\>:\<break\>:\<endwhile\>'
Bundle 'matchit.zip'
Bundle 'moria'
Bundle 'word_complete.vim'
Bundle 'SuperTab'
Bundle 'xptemplate'
Bundle 'ctags.vim'
Bundle 'taglist.vim'
Bundle 'winmanager'
Bundle 'Command-T'

" non github repos
"Bundle 'git://git.wincent.com/command-t.git'
```

```
" ...
Bundle 'https://github.com/scrooloose/nerdtree.git'

filetype plugin indent on " required!
"
" Brief help
" :BundleList - list configured bundles
" :BundleInstall(!) - install(update) bundles
" :BundleSearch(!) foo - search(or refresh cache first) for foo
" :BundleClean(!) - confirm(or auto-approve) removal of unused bundles
"
" see :h vundle for more details or wiki for FAQ
" NOTE: comments after Bundle command are not allowed..
```

~/.gvimrc配置如下：

```
" An example for a gvimrc file.一般设置GUI和快捷键等
"-------------------------------------------------------------------------------
" gvimrc导入的默认设置
"-------------------------------------------------------------------------------
" 设置窗口底部命令有两行
set ch=2

" 输入文本时隐藏鼠标
set mousehide

"键位映射，使shift-insert快捷键像在 Xterm程序中一样工作
map <S-Insert> <MiddleMouse> "鼠标中键代表快捷粘贴
map! <S-Insert> <MiddleMouse>

" Vim版本5.0或以上下面脚本工作.
if version >= 500

"高亮C注释字符串
let c_comment_strings=1

"如果没启动语法高亮，启动语法高亮
if !exists("syntax_on")
syntax on
endif

"启动搜索高亮.
set hlsearch

" 设置颜色
" 文本背景为浅灰色
" 文本的最后一行为深灰色
" 光标设置成绿色, C当 ":lmap"映射激活时变成青色
" 常量设置为没有底线，但常量的背景颜色浅一些
```

```
highlight Normal guibg=grey90
highlight Cursor guibg=Green guifg=NONE
highlight lCursor guibg=Cyan guifg=NONE
highlight NonText guibg=grey80
highlight Constant gui=NONE guibg=grey95
highlight Special gui=NONE guibg=grey95

endif




"------------------------------------------------------------------------------
" 基本设置
"------------------------------------------------------------------------------
" When started as "evim", evim.vim will already have done these settings.
" 设置编码
set encoding=utf-8
set fenc=utf-8
set fileencodings=ucs-bom,utf-8,cp936,gb2312,gb18030,big5

"显示行号
set number

"设置默认打开窗口大小
set lines=70 columns=100

"设置窗口透明度
set transparency=10

"设置背景色
"set bg=dark

"用 koehler 调色板
colorscheme koehler

"隐藏工具栏和滑动条
set guioptions=aAce

"开启自带的tab栏
set showtabline=2

"总是在窗口右下角显示光标的位置
set ruler

"在Vim窗口右下角显示未完成的命令
set showcmd

"设置语法高亮
if &t_Co > 2 || has("gui_running")
syntax on
endif
```

```
"------------------------------------------------------------------------------
" 文本操作设置
"------------------------------------------------------------------------------
"设置字体
set gfn=Courier:h15

"设置自动缩进
set tabstop=4
set shiftwidth=4
set softtabstop=4
set noexpandtab

"设置Tab键跟行尾符显示出来
set list lcs=tab:>-,trail:-


"设置自动换行
set wrap

"设置光标移动到屏幕之外时，自动向右滚动10个字符
set sidescroll=10


"------------------------------------------------------------------------------
" 搜索设置
"------------------------------------------------------------------------------
"打开搜索高亮
set hlsearch

"忽略大小写
set ignorecase

"在查找时输入字符过程中就高亮显示匹配点。然后回车跳到该匹配点。
set incsearch

"设置查找到文件尾部后折返开头或查找到开头后折返尾部。
set wrapscan



"------------------------------------------------------------------------------
" 文件设置
"------------------------------------------------------------------------------



"------------------------------------------------------------------------------
" 插件界面设置
"------------------------------------------------------------------------------

"ctags的设置，这里的ctags是用MacPorts安装的，用Vundle安装的'ctags.vim'有问题，系统自带的ctags也有点问题
let Tlist_Ctags_Cmd="/opt/local/bin/ctags"
```

```
let Tlist_Exist_OnlyWindow=1 " 如果taglist窗口是最后一个窗口，则退出vim
let Tlist_Auto_Update " Update the tag list automatically

"在vim左侧显示窗口Taglist和文件列表窗口
"let g:winManagerWindowLayout='FileExplorer,BufExplorer|TagList'
let g:winManagerWindowLayout='NERDTree, BufExplorer|TagList'
"打开vim时自动打开winmanager
let g:AutoOpenWinManager = 1
"定义打开关闭winmanager的快捷键
nmap wt :WMToggle<cr>
```

本文走马观花地带领大家快速浏览了MacVIm的一些基本功能，也相当于笔者的一个学习总结。实际上，上边谈到的每一个内容都可以独立写成一篇博客来介绍，对于Vim的更多具体用法还需要去进行更深入的学习。写这篇文章，笔者参考了很多资料，在这里罗列一些出来供读者参考。

参考资料：

Mac OS X使用之——新年第一天弘法寺许愿，MacVim小试

vim基本命令

vim 命令(全)

手把手教你把Vim改装成一个IDE编程环境(图文)

把VIM打造成一个真正的IDE(3)

将Vim改造为强大的IDE—Vim集成Ctags/Taglist/Cscope/Winmanager/NERDTree/OmniCppComplete（有图有真相）

简明 Vim 练级攻略

标签： Mac 开发 利器 程序员 编辑器 MacVim学习 总结

相关信息

更多

- 我们在囧途之程序员转型记
- 我们在囧途之程序员新人篇
- 【开发工具】vim编辑器实用技巧总结
- 优秀程序员的良好的学习方式，特征，生活和学
- 写给自己和程序员的一些话(12年末-13年初总结)
- 三年程序员生涯的感悟、总结和憧憬
- 黑马程序员_多态的学习(上)
- 我的程序员之路----2012在匍匐中前进
- 我们在囧途之程序员做私活小记
- 写在冬日的第一天--一个女程序员第七年工作总
- 漫谈程序员系列：一张图道尽程序员的出路
- ASP.NET常用在线编辑器使用方法总结
- ASP.NET常用在线编辑器使用方法总结（二）
- DLP Coder:C#写的一个用于编辑DLP投影机程序
- 程序员成功之路
- Java程序员从笨鸟到菜鸟之（一百零二）sql注入

您的回应...

提交

相关话题

查看全部

- 求解惑--关于我的程序员之路... (1回应)
- 程序员的迷茫之要不要啃基础 (1回应)
- 怎样开发一个andriod 的pdf阅读器同时也可以编辑文 (0回应)
- Wecenter 问答程序编辑器上怎样加代码高亮功能？ (0回应)
- 软件程序员关于兴趣那点事 (3回应)
- 为什么我们不要 .NET 程序员? (3回应)
- 推荐一款bootstrap的好用福编辑器 (2回应)
- 新手php程序员的烦恼~· (2回应)
- 为什么很多程序员英语很差？ (2回应)
- 求推荐 网页内容编辑器 (2回应)
- 程序员转数据挖掘相关的工作的问题 (2回应)
- 有没有适用于移动端的，后台文本编辑器？或者解决 (1回应)
- 用vs2008开发windows窗口程序，语言c# (1回应)
- 我想知道一名程序员需要都了解什么知识？ (1回应)
- 编辑器与UI结合，dialog里面编辑器失效 (1回应)
- 为什么eclipse for j2ee （juno）突然不能设置jsp文件 (1回应)
- Java程序员成长疑问,求指点，求指教 (1回应)
- PHP程序员如何选择第二门语言 (1回应)
- 压迫程序员的storyboard， (1回应)
- 作为一名应届毕业生没有工作经验的程序员，应该如 (1回应)
- C语言开发的应用程序可以和Tomcat服务器连接通信 (1回应)
- 编辑器的字体样式无法控制的解决方法 (1回应)
- 用java做一个文本编辑器并提供eclipse的编译和运行 (1回应)
- 豆瓣的编辑器 (1回应)

热点话题

回应

1. **java采用什么IDE编程比较** 10
2. .NET面试当中碰到：设计模 10
3. B/S网站如何做实时消息提 9
4. JavaScript的 onclick事件是 9
5. php 多维数组合并的问题 9
6. php如何生成十进制00到20 9
7. 求一枚前端大神指点指点 8
8. 那种高端大气公司开发PHP 7

也许你感兴趣

换一批


本书的角落，靠窗的座位book nooks,window
来自：图片 关注度：4


道阻且长，勿忘归途
来自：图片 关注度：4


首页-洁婷乐萱专卖店-天猫Tmall.com
来自：图片 关注度：4


乐视控股董事长贾跃亭：风口论者都是机会主义
来自：文章 关注度：4


来自相册
来自：图片 关注度：4


蛋
来自：图片 关注度：4


乘风归来在12月16日19时42分分享的素材图片
来自：图片 关注度：4


PHP 7.0.0 正式版终于发布了，速度是 PHP 5.6
来自：文章 关注度：4


信息过剩的时代如何打动用户?
来自：文章 关注度：4


中国概念股周四多数下跌 唯品会跌 6.6 %
来自：文章 关注度：4


小宋佳亮相电影《万物生长》北京首映式
来自：图片 关注度：4


Inktober 2015 Week 3,Xavier Collette:All the
来自：图片 关注度：4

热门标签

更多

Android Java Linux 智能 开发 互联网 手机 实现 美元 移动 发布 问题
windows 方法 设计 中国 公司 如何 网络 解决 微软 推出 服务器 苹果
文件 算法 技术 产品 服务 系统