# vundle管理vim插件

<http://www.cnblogs.com/youxia/p/linux002.html>

<http://blog.csdn.net/sww_simpcity/article/details/18504071>

http://blog.csdn.net/namecyf/article/details/7787479

### 一、安装vundle

在用户的主目录中建立 .vim 目录，在 .vim 目录中建立 bundle 目录，以后Vundle自动下载的插件都保存在这里。然后，进入 Bundle 目录，使用 git clone 命令下载vundle.如下两种方式。

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

### 二、如何通过vundle安装插件

1. vim-scripts仓库中的插件，可以直接使用Bundle 'L9'这样的格式配置
2. github上其他用户的插件，使用Bundle 'hahaya/hahaya-vim.git'这样用户名加仓库名的方式配置(自己的插件全部使用这种方法配置)

### 三、vundle命令

* :BundleList -列举出列表中(.vimrc中)配置的所有插件
* :BundleInstall -安装列表中全部插件
* :BundleInstall! -更新列表中全部插件
* :BundleSearch foo -查找foo插件
* :BundleSearch! foo -刷新foo插件缓存
* :BundleClean -清除列表中没有的插件
* :BundleClean! -清除列表中没有的插件

### 四、配置.vimrc

上面已经解释了vundle的简单用法，下面配置.vimrc，然后介绍如何通过vundle管理插件，先在终端下执行vim ~/.vimrc命令，如果存在.vimrc则打开，不存在.vimrc则创建一个.vimrc并打开。在.vimrc中加入一些常用配置

### 五、使用vundle安装插件

前面已经介绍了.vimrc的基本配置，那么该怎么使用vundle来管理vim插件呢，在.vimrc中添加如下内容启用vundle管理vim插件的功能，并使用vundle来配置一个名为vim-powline的状态栏加强插件

*"开始使用Vundle的必须配置*

set *nocompatible*

filetype off

set *rtp*+=~/.vim/bundle/vundle/

call vundle#rc()

*"使用Vundle来管理Vundle*

Bundle 'gmarik/vundle'

*"PowerLine插件 状态栏增强展示*

Bundle 'Lokaltog/vim-powerline'

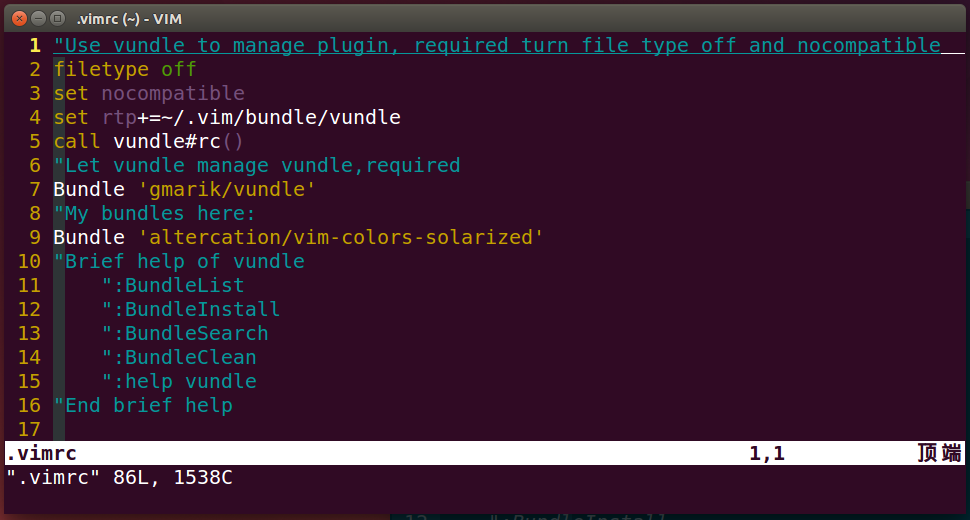
*"vim有一个状态栏 加上powline则有两个状态栏*

set *laststatus*=2

set *t\_Co*=256

let g:Powline\_symbols='fancy'

*"Vundle配置必须 开启插件*

filetype plugin indent on

（插图

)

保存.vimrc文件，然后重新打开一个vim，在NORMAL模式下输入命令:BundleInstall,首次执行会要求输入github帐号和密码，等待插件下载完成，然后重新启动vim就会看见漂亮的状态栏插件

### 六、使用vundle更新插件

使用vundle更新插件非常简单，只用打开一个vim，然后在NORMAL模式下输入命令:BundleInstall!

### 七、使用vundle卸载插件

使用vundle卸载插件也很简单，只需在.vimrc去掉绑定插件的命令及插件的配置(注释掉即可，以免以后会使用)，假如需要卸载vim-powerline这个插件，首先在.vimrc中注释掉以下内容：

Bundle 'Lokaltog/vim-powerline'

*"vim有一个状态栏 加上powline则有两个状态栏*

set *laststatus*=2

set *t\_Co*=256

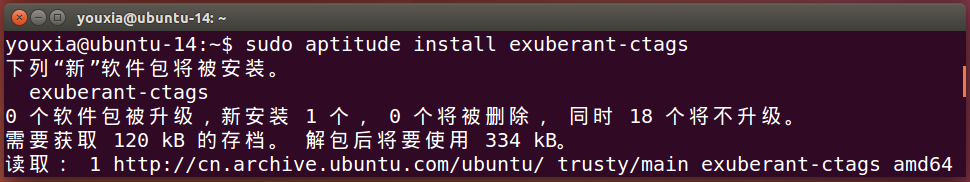
let g:Powline\_symbols='fancy'

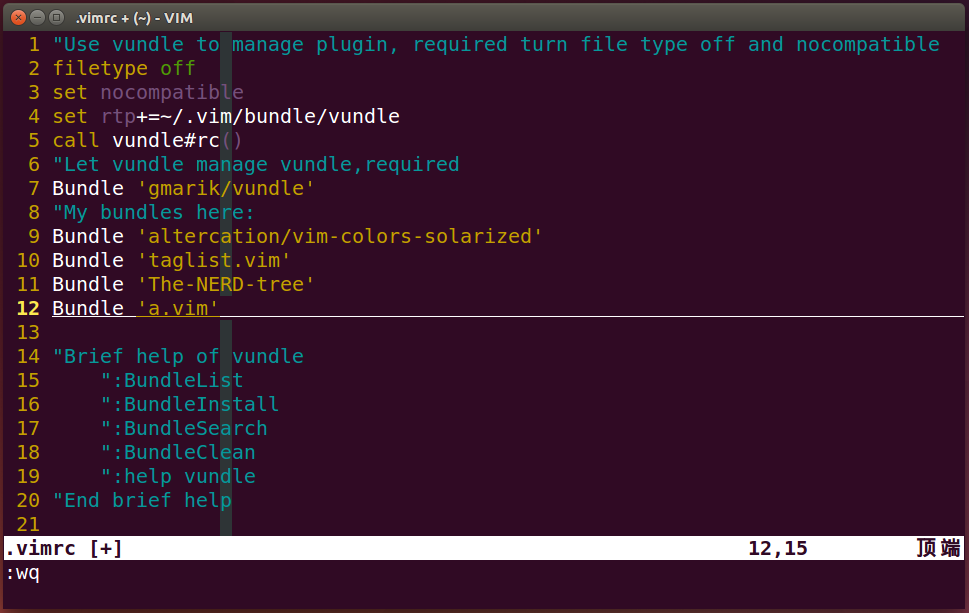
保存.vimrc文件，重新打开一个vim，在NORMAL模式下输入命令:BundleClean,打开~/.vim/bundle已经看不到插件vim-powerline的相关文件

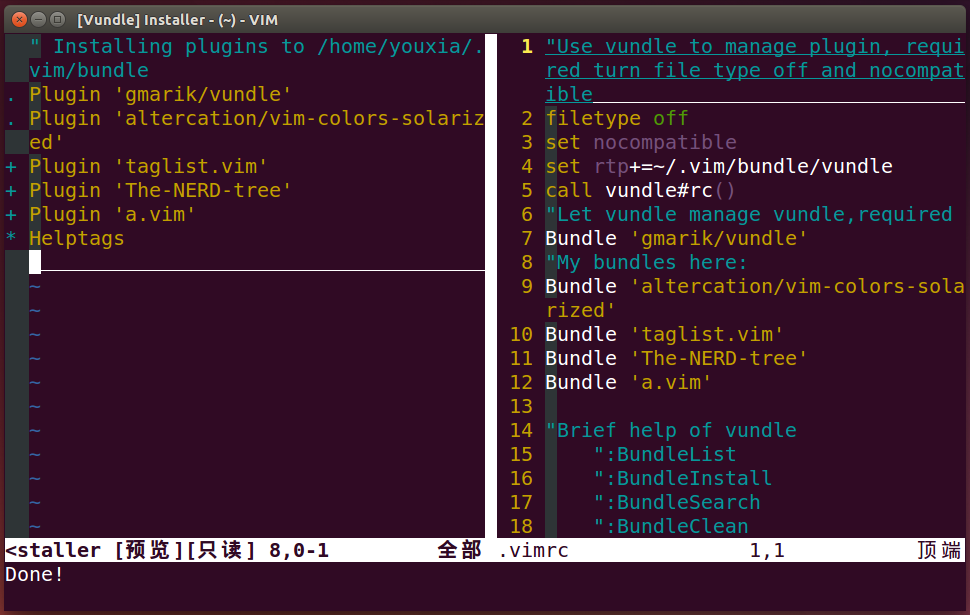
八、插件使用排名

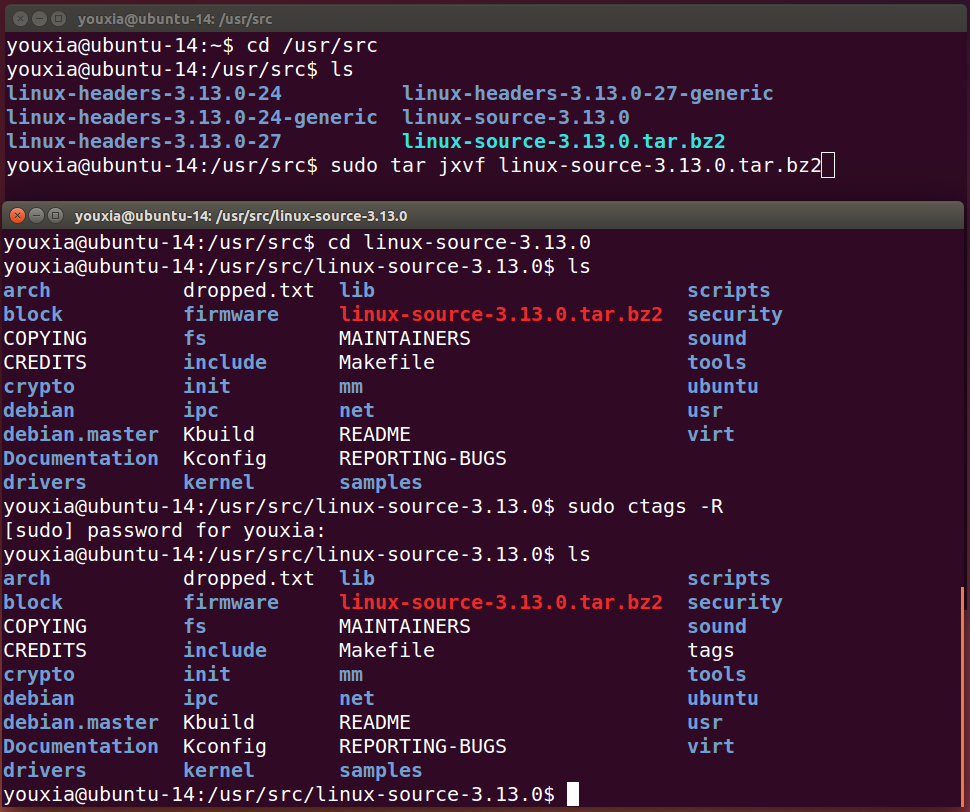
看哪些插件的下载率比较靠前。地址这里：<http://www.vim.org/scripts/script_search_results.php?order_by=downloads>

九、几款插件配置

（1）taglist.vim需要tags文件的支持，而tags文件需要使用tags命令生成，所以需要安装exuberant-ctags，如下图：  


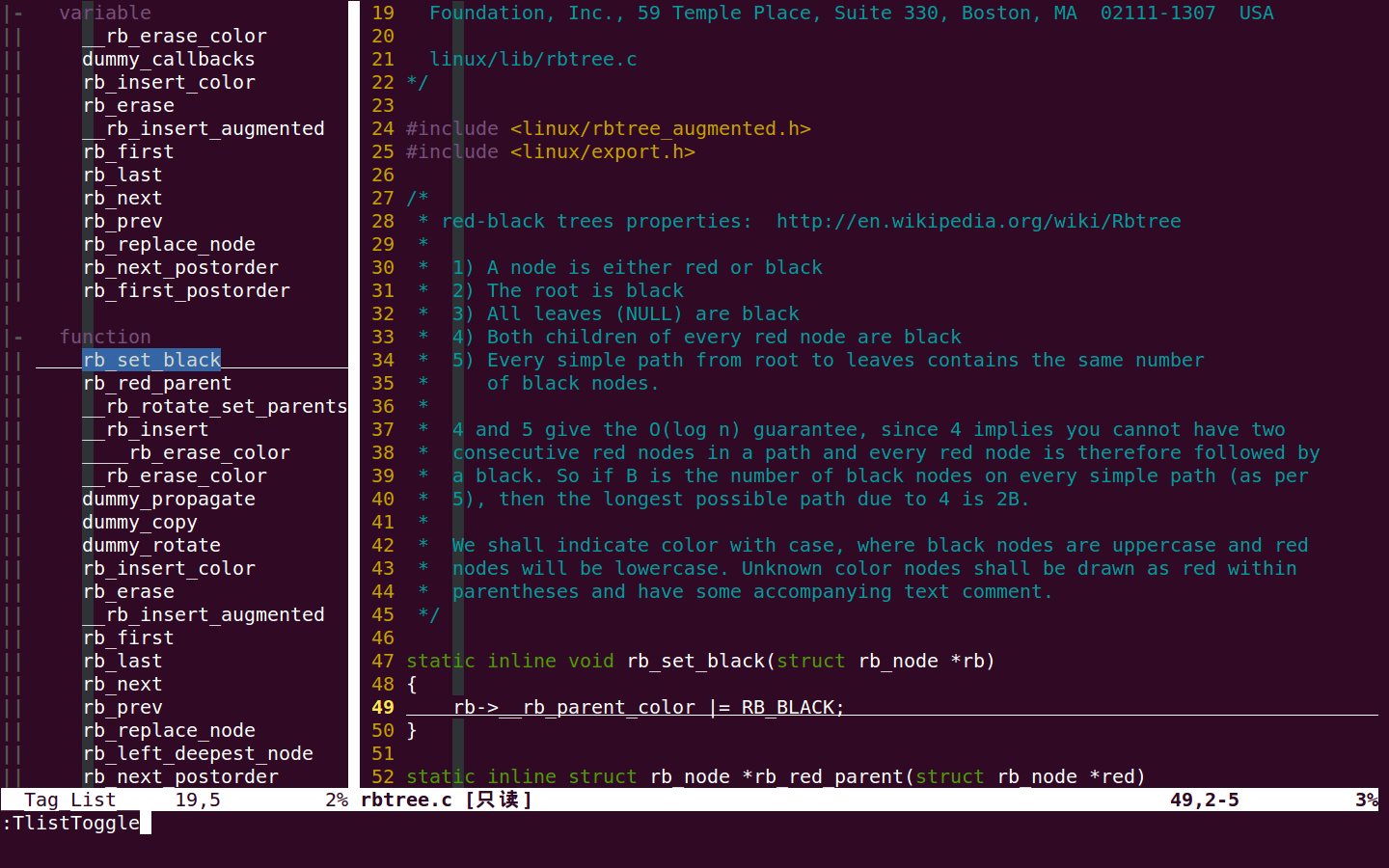
　　再然后，为Vim安装插件。从下面的图中可以看出，除了 taglist.vim和The-NERD-tree，我还安装了a.vim，该插件可以方便地实现在.c文件和.h文件之间跳转。当插件名称中含有空格 的时候，以“-”代替，如The NERD tree插件，在配置文件中需写成“The-NERD-tree”。配置插件如下图：  


　　安装插件如下图：  


　　再然后，进入Linux内核源代码包的安装目录，解压Linux内核源代码，并生成tags文件，如下图：（这两个步骤都需要有点耐心，因为解压和生成tags这两个操作都很花时间。）  


　　阅读代码前，需要告诉Vim使用哪一个tags文件，在这里，就是刚刚使用 ctags 生成的文件，可以通过 :set tags=/usr/src/linux-source-3.13.0/tags 来设置。为了方便起见，我把该设置放到了 .vimrc 配置文件中，如下图：

　　最后，启动Vim，找到相应的代码进行浏览。如果编辑的是一个目录的话（如下面的 :e . 命令，表示编辑当前目录），会自动跳出The-NERD-tree的界面，在目录之间导航非常方便，如下图：

　　这里以Linux内核中红黑树的实现为例。使用The-NERD-tree定位到 lib 目录下的 rbtree.c 文件，按Enter键即可打开代码。输入 :TlistToggle 命令可以打开Taglist窗口，如下图：  


　　在源文件中浏览时，使用Ctrl+]快捷键，可以非常方便地在符号的使用和定义之间跳转。

使用文件浏览器和窗口管理器 -- 插件: WinManager

通过WinManager插件来将TagList窗口和netrw窗口整合起来, 就像图1中的图标2和3组成的那个效果  
现在在你的~/.vimrc中增加下面两句

|  |
| --- |
| let g:winManagerWindowLayout='FileExplorer|TagList' nmap wm :WMToggle<cr> |

十.CScope使用

安装Cscope:  
如果你的系统中有cscope命令, 则可以跳过这一小段, 如果没有, 就先跟着我一起安装一个吧.  
在Cscope的主页: http://cscope.sourceforge.net 下载一个源码包, 解压后编译安装:

|  |
| --- |
| # ./configure # make # make install       // 需要root权限 |

先在~/vimrc中增加一句:

|  |
| --- |
| :set cscopequickfix=s-,c-,d-,i-,t-,e- |

这个是设定是否使用 quickfix 窗口来显示 cscope 结果, 用法在后面会说到。  
  
跟Ctags一样, 要使用其功能必须先为你的代码生成一个cscope的数据库, 在项目的根目录运行下面的命令:

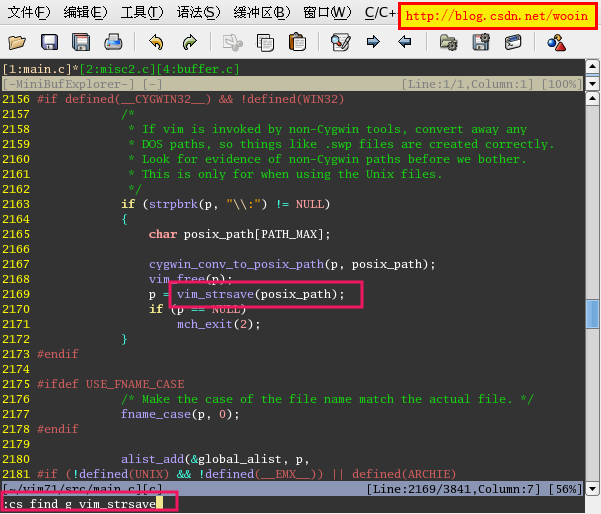
|  |
| --- |
| $ cd /home/wooin/vim71/ $ cscope -Rbq # 此后会生成三个文件 $ ll cscope.\* -rw-rw-r-- 1 wooin wooin 1.1M 2007-09-30 10:56 cscope.in.out -rw-rw-r-- 1 wooin wooin 6.7M 2007-09-30 10:56 cscope.out -rw-rw-r-- 1 wooin wooin 5.1M 2007-09-30 10:56 cscope.po.out # 打开文件, 开始Coding $ cd src $ vi main.c |

进入vim后第一件事是要把刚才生成的cscope文件导入到vim中来, 用下面的命令:

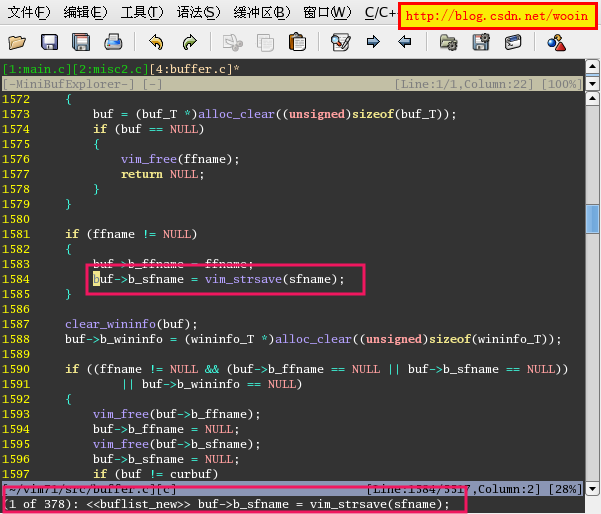
|  |
| --- |
| :cs add /home/wooin/vim71/cscope.out /home/wooin/vim71 |

上面这条命令很重要, 必须写全, 不能只写前半句:  
:cs add /home/wooin/vim71/cscope.out  
因为源码是多级目录的, 如果这样写, cscope是无法在子目录中的源码中工作的, 当然, 如果你的源码都在同一级目录中就无所谓了. 如果你要经常用cscope的话, 可以把上面那句加到~/.vimrc中去.  
  
下面我们来操练一下, 查找函数vim\_strsave()的定义, 用命令:

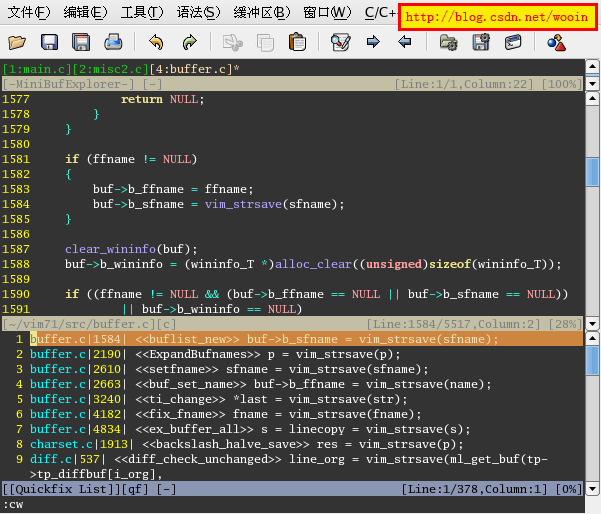
|  |
| --- |
| :cs find g vim\_strsave |

如下图:  
  
(--- 图8 ---)  
按下回车后会自动跳转到vim\_strsave()的定义处. 此时你肯定会说Ctags也可以做到这个呀, 那么下面说的这个Ctags就无法做到了, 我想查找vim\_strsave()到底在那些地方被调用过了, 用命令:

|  |
| --- |
| :cs find c vim\_strsave |

按下回车后vim会自动跳转到第一个符合要求的地方, 并且在命令栏显示有多少符合要求的结果, 如图:  
  
(--- 图9 ---)  
如果自动跳转的位置你不满意, 想看其他的结果, 可以用下面的命令打开QuickFix窗口:

|  |
| --- |
| :cw |

如图:  
  
(--- 图10 ---)  
这时你就可以慢慢挑选了^\_^  
cscope的主要功能是通过同的子命令"find"来实现的  
"cscope find"的用法:  
cs find c|d|e|f|g|i|s|t name

|  |  |
| --- | --- |
| 0 或 s | 查找本 C 符号(可以跳过注释) |
| 1 或 g | 查找本定义 |
| 2 或 d | 查找本函数调用的函数 |
| 3 或 c | 查找调用本函数的函数 |
| 4 或 t | 查找本字符串 |
| 6 或 e | 查找本 egrep 模式 |
| 7 或 f | 查找本文件 |
| 8 或 i | 查找包含本文件的文件 |

如果每次查找都要输入一长串命令的话还真是件讨人厌的事情, Cscope的帮助手册中推荐了一些快捷键的用法, 下面是其中一组, 也是我用的, 将下面的内容添加到~/.vimrc中, 并重启vim:

|  |
| --- |
| nmap <C-\_>s :cs find s <C-R>=expand("<cword>")<CR><CR> nmap <C-\_>g :cs find g <C-R>=expand("<cword>")<CR><CR> nmap <C-\_>c :cs find c <C-R>=expand("<cword>")<CR><CR> nmap <C-\_>t :cs find t <C-R>=expand("<cword>")<CR><CR> nmap <C-\_>e :cs find e <C-R>=expand("<cword>")<CR><CR> nmap <C-\_>f :cs find f <C-R>=expand("<cfile>")<CR><CR> nmap <C-\_>i :cs find i ^<C-R>=expand("<cfile>")<CR><CR> nmap <C-\_>d :cs find d <C-R>=expand("<cword>")<CR><CR> |

当光标停在某个你要查找的词上时, 按下<C-\_>g, 就是查找该对象的定义, 其他的同理.  
按这种组合键有一点技巧,按了<C-\_>后要马上按下一个键,否则屏幕一闪就回到nomal状态了  
<C-\_>g的按法是先按"Ctrl+Shift+-", 然后很快再按"g"  
  
很奇怪, 其中的这句:  
nmap <C-\_>i :cs find i ^<C-R>=expand("<cfile>")<CR>$<CR>  
在我的vim中无法工作, 但是我改成:  
nmap <C-\_>i :cs find i <C-R>=expand("<cfile>")<CR><CR>  
就可以正常工作了, 不知道是什么原因? 有哪位朋友知道请告诉我.  
  
cscope的其他功能你可以通过帮助手册自己慢慢学习  
  
reset : 重新初始化所有连接。  
用法  : cs reset

十一：QuickFix 窗口

|  |  |
| --- | --- |
|  |  |
|  | 在上一节的图10中大家可以看到在窗口下面有一个显示查询结果的窗口, 这个窗口中列出了查询命令的查询结果, 用户可以从这个窗口中选择每个结果进行查看, 这个窗口叫"QuickFix"窗口, 以前也是一个vim的插件来的, 只不过现在成了vim的标准插件, 不用你在去安装了, QuickFix窗口的主要作用就是上面看到的那个功能: 输出一些供选择的结果, 可以被很多命令调用, 更详细的介绍和使用方法请用下面的命令打开QuickFix的手册来学习吧:   |  | | --- | | :help quickfix |   这里我一个常用的例子来再介绍一种QuickFix窗口的使用方法. 这个例子是要模仿平时我们编程时, 当编译出错时, QuickFix会把出错的信息列出来, 供我们一条条地查看和修改. 首先还是用vim打开~/vim71/src/main.c, 事先最好先编译过vim71, 否则一会儿编译的时候有点慢, 或者你也可以自己写一个小的有错误的程序来跟着我做下面的步骤, 见下图:  (--- 图11 ---) 我们修改一下main.c, 人为地造成几处错误, 在第1019行增加了一个baobao\_wu的没有任何定义的字符串, 删除了第1020行最后的一个括号")", 然后用下面的命令进行编译:   |  | | --- | | :make |   显然编译会报很多错误, 当编译结束并退出到源码界面时, 刚才编译器报的错误都已经看不到了, 但是我们可以用QuickFix窗口再将错误信息找出来, 用下面的命令调出QuickFix窗口:   |  | | --- | | :cw |   此时你就可以看如下图所示的QuickFix窗口了:  (--- 图12 ---) 在下面的QuickFix窗口中我们可以找到每一个编译错误, 同样你可以用鼠标点击每一条记录, 代码会马上自动跳转到错误处, 你还可以用下面的命令来跳转:   |  | | --- | | :cn        // 切换到下一个结果 :cp        // 切换到上一个结果 |   如果你经常使用这两个命令, 你还可以给他们设定快捷键, 比如在~/.vimrc中增加:   |  | | --- | | nmap <F6> :cn<cr> nmap <F7> :cp<cr> |   其还有其他的命令/插件也会用到QuickFix窗口, 但是用法基本上的都是类似的, 本文后面还会用到QuickFix窗口, 接着往下看吧. |

十二 快速浏览和操作Buffer -- 插件: MiniBufExplorer

|  |  |
| --- | --- |
|  |  |
|  | |  |  | | --- | --- | | 下载地址 | http://www.vim.org/scripts/script.php?script\_id=159 | | 版本 | 6.3.2 | | 安装 | 将下载的 minibufexpl.vim文件丢到 ~/.vim/plugin 文件夹中即可 | | 手册 | 在minibufexpl.vim 文件的头部 |   在编程的时候不可能永远只编辑一个文件, 你肯定会打开很多源文件进行编辑, 如果每个文件都打开一个vim进行编辑的话那操作起来将是多麻烦啊, 所以vim有buffer(缓冲区)的概念, 可以看vim的帮助: :help buffer vim自带的buffer管理工具只有:ls, :bnext, :bdelete 等的命令, 既不好用, 又不直观. 现在隆重向你推荐一款vim插件(plugin): MiniBufExplorer  使用方法: 重新启动vim, 当你只编辑一个buffer的时候MiniBufExplorer派不上用场, 当你打开第二个buffer的时候, MiniBufExplorer窗口就自动弹出来了, 见下图:  (--- 图13 ---) 上面那个狭长的窗口就是MiniBufExplorer窗口, 其中列出了当前所有已经打开的buffer, 当你把光标置于这个窗口时, 有下面几个快捷键可以用:   |  |  | | --- | --- | | <Tab> | 向前循环切换到每个buffer名上 | | <S-Tab> | 向后循环切换到每个buffer名上 | | <Enter> | 在打开光标所在的buffer | | d | 删除光标所在的buffer |   以下的两个功能需要在~/.vimrc中增加:   |  | | --- | | let g:miniBufExplMapCTabSwitchBufs = 1 |  |  |  | | --- | --- | | <C-Tab> | 向前循环切换到每个buffer上,并在但前窗口打开 | | <C-S-Tab> | 向后循环切换到每个buffer上,并在但前窗口打开 |   如果在~/.vimrc中设置了下面这句:   |  | | --- | | let g:miniBufExplMapWindowNavVim = 1 |   则可以用<C-h,j,k,l>切换到上下左右的窗口中去,就像: C-w,h j k l    向"左,下,上,右"切换窗口. 在~/.vimrc中设置:   |  | | --- | | let g:miniBufExplMapWindowNavArrows = 1 |   是用<C-箭头键>切换到上下左右窗口中去 |

十三：c/h文件间相互切换 -- 插件: A

|  |  |
| --- | --- |
|  |  |
|  | |  |  | | --- | --- | | 下载地址 | http://www.vim.org/scripts/script.php?script\_id=31 | | 版本 |  | | 安装 | 将a.vim 放到 ~/.vim/plugin 文件夹中 | | 手册 | 无 |   下面介绍它的用法: 作为一个C程序员, 日常Coding时在源文件与头文件间进行切换是再平常不过的事了, 直接用vim打开其源/头文件其实也不是什么麻烦事, 但是只用一个按键就切换过来了, 这是多么贴心的功能啊.... 安装好a.vim后有下面的几个命令可以用了:   |  |  | | --- | --- | | :A | 在新Buffer中切换到c/h文件 | | :AS | 横向分割窗口并打开c/h文件 | | :AV | 纵向分割窗口并打开c/h文件 | | :AT | 新建一个标签页并打开c/h文件 |   其他还有一些命令, 你可以在它的网页上看看, 我都没用过, 其实也都是大同小异, 找到自己最顺手的就行了.  我在~/.vimrc中增加了一句:   |  | | --- | | nnoremap <silent> <F12> :A<CR> |   意思是按F12时在一个新的buffer中打开c/h文件, 这样在写程序的时候就可以不假思索地在c/h文件间进行切换, 减少了按键的次数, 思路也就更流畅了 |

十四：自动补全

|  |  |
| --- | --- |
|  |  |
|  | 用过Microsoft Visual Studio的朋友一定知道代码补全功能, 输入一个对象名后再输入"."或者"->", 则其成员名都可以列出来, 使Coding流畅了许多, 实现很多懒人的梦想, 现在我要告诉你, 这不再是Microsoft Visual Studio的专利了, vim也可以做到! 下面由我来教你, 该功能要tags文件的支持, 并且是ctags 5.6版本, 可以看看前文介绍tags文件的章节.  我这里要介绍的功能叫"new-omni-completion(全能补全)", 你可以用下面的命令看看介绍:   |  | | --- | | :help new-omni-completion |   你还需要在~/.vimrc文件中增加下面两句:   |  | | --- | | filetype plugin indent on |   打开文件类型检测, 加了这句才可以用智能补全   |  | | --- | | set completeopt=longest,menu |   关掉智能补全时的预览窗口  请确定你的Ctags 5.6已经安装好, 并且生成的tags文件已经可以用了, 那么我们就要抄家伙开搞了. 用vim打开源文件   |  | | --- | | $ vi /home/wooin/vim71/src/main.c |   设置tags文件   |  | | --- | | :set tags=/home/wooin/vim71/tags |   随便找一个有成员变量的对象, 比如"parmp", 进入Insert模式, 将光标放在"->"后面,  然后按下"Ctrl+X Ctrl+O", 此时会弹出一个下列菜单, 显示所有匹配的标签, 如下图:  (--- 图18 ---) 此时有一些快捷键可以用:   |  |  | | --- | --- | | Ctrl+P | 向前切换成员 | | Ctrl+N | 向后切换成员 | | Ctrl+E | 表示退出下拉窗口, 并退回到原来录入的文字 | | Ctrl+Y | 表示退出下拉窗口, 并接受当前选项 |   如果你增加了一些成员变量, 全能补全还不能马上将新成员补全, 需要你重新生成一下tags文件, 但是你不用重启vim, 只是重新生成一下tags文件就行了, 这时全能补全已经可以自动补全了, 还真够"全能"吧.  vim中的其他补全方式还有:   |  |  | | --- | --- | | Ctrl+X Ctrl+L | 整行补全 | | Ctrl+X Ctrl+N | 根据当前文件里关键字补全 | | Ctrl+X Ctrl+K | 根据字典补全 | | Ctrl+X Ctrl+T | 根据同义词字典补全 | | Ctrl+X Ctrl+I | 根据头文件内关键字补全 | | Ctrl+X Ctrl+] | 根据标签补全 | | Ctrl+X Ctrl+F | 补全文件名 | | Ctrl+X Ctrl+D | 补全宏定义 | | Ctrl+X Ctrl+V | 补全vim命令 | | Ctrl+X Ctrl+U | 用户自定义补全方式 | | Ctrl+X Ctrl+S | 拼写建议 | |