Gvim 和 Vim 使用说明

茂松

将压缩包解压到/home/hostname/下，其中包括.vimrc、.gvimrc 文件和.vim 文件夹，覆盖已有的；接着安装 ctags 和 cscope，命令：

```
sudo apt-get install exuberant-ctags
sudo apt-get install cscope
```

1. （在.gvimrc 和.vimrc 中）定义自己的快捷命令方式；在 vim 普通模式下直接敲所需命令

```
"常用简写命令          !后面一定要有空格
nmap cd :cd
nmap ls :! ls
```

注意：在 vim 普通模式下键入“:ls”是查看同时打开所有文件的文件列表，不同于“:! ls”

```
nmap gcc :! gcc
nmap gl :! gcc -lGL -lGLU -lglut
nmap w :w
nmap wq :wq
nmap mk :! mkdir
nmap tch :! touch
nmap cp :! cp
nmap rm :! rm
nmap make :! make
```

2. 查找函数，变量定义：ctags（功能没有 cscope 强大）

要生成 tags 文件：在你要查看的源码“根目录”，执行

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

或者直接按“Ctrl+F12”快捷键，生成 tags 文件，文件大小和本项目的源代码总大小差不多；

使用：

案件“Ctrl+]”跳转到函数或者变量定义，按“Ctrl+t”，跳转回上一级，类似于栈操作

注意：如果找到的不是你想要的函数（有同名函数的原因），键入“:ts”查看找到的所有同名函数，选中自己想要的函数查看　　ts<==>tagslist

在.vim/sourceCode 中相应的文件夹中也要生成 tags，以便你要查找的函数不再你的项目中时使用，比如所标准的 C 库和 C++库或者 Java 库；

然后在.vimrc 中加入命令：

```
set tags=tags
set tags+=./tags,../tags,./*/tags
set tags+=/home/hostname/.vim/sourceCode/glibc-2.16.0/tags
set tags+=/home/hostname/.vim/sourceCode/stdcpp_for_ctags/tags
```

第一行是在打开的源文件的当前目录下查找 tags 文件；

第二行是在父目录或者更高级父母路中查找 tags 文件；

三四行是加载特定的目录下（你所需要的函数库中）的 tags 文件；

依照上面的方法，构造你需要的函数库

3. 按 F2 打开和关闭“文件浏览器”和“成员变量和(成员)函数浏览器”

4. 查找函数，变量定义：cscope

按 F5 键生成 cscope.files、cscope.in.out、cscope.out、cscope.po.out 同时也生成 tags

快捷键使用：（按 Ctrl+\组合键后，松开快速按另一个字母，可以在配置文件中更改快捷键，注意冲突情况）
Ctrl+\ s: 查找 C 语言符号，即查找函数名、宏、枚举值等出现的地方
Ctrl+\ g: 查找函数、宏、枚举等定义的位置，类似 ctags 所提供的功能
Ctrl+\ d: 查找本函数调用的函数
Ctrl+\ c: 查找调用本函数的函数
Ctrl+\ t: 查找指定的字符串
Ctrl+\ e: 查找 egrep 模式，相当于 egrep 功能，但查找速度快多了
Ctrl+\ f: 查找并打开文件，类似 vim 的 find 功能
Ctrl+\ i: 查找包含本文件的文

向回跳还是按“Ctrl+]”

搜索的结果将显示在 QuickFix 中，按 **F3** 键可以直接调出 **QuickFix** 窗口
配置文件中已经实现了，从子目录向父目录中搜索 cscope.out，实现自动加载 cscope.out 文件

5. 按 **F4** 或者 **F6** 实现多文件标签的切换，按 **F4** 是向左切换，按 **F6** 是向右切换

6. **F7** 实现 **grep** 功能，有 **ctags** 和 **cscope** 后，**grep** 基本用不上

7. 按 **Ctrl+Up**、**Ctrl+Down**、**Ctrl+Left**、**Ctrl+Right**（**Ctrl+**箭头键）切换光标所在窗口 **buffer** 的位置，相当于 **Ctrl+w+w**，不过,更好用

8. **F9** 键是一键编译，**Ctrl** 是编译并运行；只对简单项目有效，建议不使用，自己写 **Makefile**

9. 实现**{},[],'',””**的自动补全功能，不若不想使用，可以去掉**.gvimrc** 和**.vimrc** 中的**“{} [] '' ""**等自动补全”配置块

10. **OmniCppComplete**

（类成员或命名空间的补全功能：->、.、::号后的提示功能该功能是在 tags 文件基础上实现的）
注意：在编写完某一个类的头文件时，一定要重新生成 tags 文件（相当大的项目谨慎使用，浪费时间），以用来实现该功能

11. 多文件栏的管理

```
" minibufexpl.vim 的使用
:bn             打开当前 buffer 的下一个 buffer
:bp             打开当前 buffer 的前一个 buffer
:ls             当前打开的 buf
:e <filename>   打开文件
:b<tab>         自动补齐
:bd             删除 buf
 d              光标停在 buffer 栏上；删除光标所在的 buffer
:b num          打开指定的 buffer，num 指的是 buffer 开始的那个数字，比如，我
                想打开 buffer 值为 7 的文件，输入:b7 就 ok 了
设快捷键:
nmap vim :e     在 vim 下每次只能新打开一个文件，不能打开多个文件，gvim 下可以
nmap bd :bd
```

12. **F10** 键是注释光标所在行，**F11** 是取消注释（光标所在行）

13. “空格”，用来实现某个块的折叠和打开，包括**{}**、**/* */**等

14. **snippets** 插件的使用（快速插入常用结构）

具体语言在~/.vim/snippets/目录下，打开相应语言的文件，查看常用的结构，以便使用：

结构特点：注意使用方法

```
# For Loop
snippet for
	for (${1:i} = 0; $1 < ${2:count}; $1${3:++}) {
		${4:/* code */}
	}

# If Condition
snippet if
	if (${1:/* condition */}) {
		${2:/* code */}
	}
snippet ef
	else if (${1:/* condition */}) {
		${2:/* code */}
	}
snippet el
	else {
		${1}
	}
```

以上为例：

for 循环：键入：for 再接着按 Tab 键，程序就会插入 for 代码块

```
	for (i = 0; i < count; i++) {
		/* code */
	}
```

对应原始结构：按 tab 键跳转{}对应的块，顺序是数字顺序

```
	for (${1:i} = 0; $1 < ${2:count}; $1${3:++}) {
		${4:/* code */}
	}
```

if 结构：键入：if 再接着按 Tab 键，程序就会插入 if 代码块

```
	if (/* condition */) {
		/* code */
	}
```

键入：ef 再接着按 Tab 键，程序就会插入 else if 代码块

```
	else if (/* condition */) {
		/* code */
	}
```

键入：el 再接着按 Tab 键，程序就会插入 else 代码块

```
	else {

	}
```

其他结构要查看~/.vim/snippets/目录下的文件内容

15.函数参数列表的提示和补全功能的实现；**code_complete.vim** 的使用

使用也是根据 tags 文件实现的，

比如：ITutorial 类中含有成员函数声明：

```
bool frameRenderingQueued(const Ogre::FrameEvent &evt);
```

用途 1：函数定义时

写完头文件时要重新生成 tags 文件，然后在实现文件中

```
bool frameRenderingQueued(
```

停留在“(”括号处，

（在 vim 插入模式下）按 “Ctrl+]”，显示该函数的补全参数列表的列表，如果有多个，选择你想要的那个；

```
bool ITutorial3::frameRenderingQueued(const Ogre::FrameEvent& evt)
bool ITutorial3::frameRenderingQueued const Ogre::FrameEvent& evt)  BaseApplication.cpp
{                                     const Ogre::FrameEvent& arg)  ITutorial3.cpp
    //we want to run everything in th const Ogre::FrameEvent&)      /home/song/.vim/sourceC
    //but we also want to do somethin const FrameEvent& evt)        /home/song/.vim/sourceC
```

用途 2：函数调用时

（在 vim 插入模式下）按 “Ctrl+j”，显示该函数的提示参数列表的列表，如果有多个，选择你想要的那个；注意区别

```
bool ITutorial3::frameRenderingQueued(`<const Ogre::FrameEvent& evt>`)
bool ITutorial3::frameRenderingQueued `<const Ogre::FrameEvent& evt>`)  BaseApplication.cpp
{                                     `<const Ogre::FrameEvent& arg>`)  ITutorial3.cpp
    //we want to run everything in th `<const Ogre::FrameEvent&>`)      /home/song/.vim/sou
    //but we also want to do somethin `<const FrameEvent& evt>`)        /home/song/.vim/sou
```

再按一次“Ctrl+j”，光标会跳转到：

```
bool ITutorial3::frameRenderingQueued(`<const Ogre::FrameEvent& evt>`)
```

根据提示键入参数，如果有多个参数，则填入一个参数后，再按“Ctrl+j”，跳到另一个地方，以此类推；知道填完参数为止；

16.增加 python 功能：

包含的文件有：.vim/syntax/python.vim、.vim/python_dict/complete-dict、.vim/plugin/python_fold.vim

.vimrc 和.gvimrc 新增内容：

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"python 配置
let g:pydiction_location = '~/.vim/python_dict/complete-dict'
"通过 Ctrl+n 来进行补全了。
"inoremap <silent> <buffer> <C-n>
```

17.增加搜索高亮显示配置：

类似 firefox 的搜索：在搜索时，输入的词句的逐字符高亮，按回车键确定搜索的字符后，全部高亮显示匹配的字符串

F8： 取消所有的高亮显示；

Ctrl+x： 向下查找光标所在的字符串

Ctrl+a： 向上查找光标所在的字符串

.vimrc 和.gvimrc 新增内容:

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
" 搜索和匹配
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
"设置高亮搜索
:set hlsearch
"取消显示所有高亮内容
nnoremap <F8> :noh<return><esc>
"设置查找光标所在位置的字符串
nmap <C-x> <S-*> "向下查找光标所在的字符串
nmap <C-a> <S-#> "向上查找光标所在的字符串

set showmatch " 高亮显示匹配的括号
"set ignorecase " 在搜索的时候忽略大小写
set incsearch " 在搜索时，输入的词句的逐字符高亮（类似 firefox 的搜索）
set laststatus=1        " 总是显示状态行
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

未完待续，可以根据自己的理解更改。

该.vim 文件中有我自己割更改的文件:

```
              包括   .vim/snippets/c.snippets
                     .vim/snippets/cpp.snippets
                     .vim/plugin/code_complete.vim    “极其重要”
```

加入 vim 插件管理工具 Vunble 插件

首先安装 git: sudo apt-get install git;
清空.vim 下所有插件文件夹，除了 sourceCode 文件夹;
使用 git 命令: git clone https://github.com/gmarik/vundle.git ~/.vim/bundle/vundle
在文件夹.vim 中生成 bundle/vundle 目录，这样就可以使用 Bundle 命令了，常用的命令有 ：

BundleList	打印插件清单
BundleInstall	安装.vimrc 中声明的所有插件
BundleInstall!	更新.vimrc 中声明的所有插件
BundleClean	删除除了.vimrc 中声明的无用的插件
BundleSearch XXX	搜索想要的插件，在 vim-scripts 用户中搜索