

VIM 常用配置及插件整理

(2013-11-20 18:47:03)

一 • VIM 的配置文件.vimrc 如下

```
"Filename: ~/.vimrc
"Author: gqyang
"Email: guoqing-yang@163.com
"
"=====
"
" VIM 配置
"
"=====
"
"在用户主目录下建立.vimrc 文件，复制本文内容到.vimrc 中
"插件目录
" ~/.vim
" ~/.vim/doc 帮助文件的目录
" ~/.vim/plugin 插件的目录
" ~/.vim/syntax 语法目录
"=====
"
" 常规设置
"
"=====
"
" 自动识别文件类型；用文件类型 plugin 脚本；使用缩进定义文件
"=====

filetype on
filetype plugin on
if has("autocmd")
    au BufReadPost * if line("'\"") > 1 && line("'\"") <= line("$") | exe "normal! g'\"" | endif
    "have Vim load indentation rules and plugins according to the detected filetype
    filetype plugin indent on
    filetype indent on
endif
"=====

" 自动加载和自动回写
"=====

" 设置当正在编辑的文件被外部的其它程序所修改后自动在 Vim 加载
if exists("&autoread")
    set autoread
endif
" 自动把内容写回文件：如果文件被修改过，在每个 :next、:rewind、:last、:first、:previous、:
stop、:suspend、:tag、:!、:make、CTRL-] 和 CTRL-^命令时进行；
" 用 :buffer、CTRL-O、CTRL-I、' {A-Z0-9} 或 ` {A-Z0-9} 命令转到别的文件时亦然。
set autowrite
"=====

" 可以为不同模式分别打开鼠标
```

```

"-----
" "n" 普通模式
" "v" 可视模式
" "i" 插入模式
" "c" 命令行模式
" "h" 编辑帮助文件时,所有前面的模式
" "a" 所有前面的模式
" "r" hit-enter 和 more-prompt 提示时
if exists("&mouse")
    set mouse=a " 使用鼠标这样设置后,不能用鼠标右键的"复制"了,解决方法,在复制之前,先按住
    Shift 键
endif
"-----

"文件和备份,不需要,所以复位
"-----

set nobackup
set nowritebackup

"-----

"文档格式
"-----

"set encoding=utf-8
set expandtab " 使用 space 代替 tab.
set shiftwidth=4 " (自动) 缩进使用的 4 个空格,用于 "cindent", ">>", "<<" 等
set softtabstop=4 " 设置软制表符的宽度
set tabstop=4 " 设置制表符(tab 键)的宽度
set smarttab
set autoindent " 设置自动对齐(缩进): 即每行的缩进值与上一行相等; 使用 noautoindent 取消设置
set smartindent " 智能对齐方式
set cindent " 使用 C/C++ 语言的自动缩进方式
set cinoptions={0,l,s,t0,n-2,p2s,(03s,=.5s,>1s,=1s,:1s " 设置 C/C++语言的具体缩进方式

"if has("syntax")
    syntax on " 语法高亮
"endif
"colorscheme zellner " 设置配色方案,它会在 'runtimepath' 里搜索"colors/{name}.vim", 载入
第一个找到的文件

set scrolloff=5 " 光标上下两侧最少保留的屏幕行数。这使你工作时总有一些可见的上下文。
set showmatch " 设置匹配模式,显示匹配的括号
set hidden " 没有保存的缓冲区可以自动被隐藏
set number " 显示行号
"set previewwindow " 标识预览窗口

"-----

"编辑操作
"-----

"set ignorecase " 搜索模式里忽略大小写
"set smartcase " 如果搜索模式包含大写字符,不使用 'ignorecase' 选项。只有在输入搜索模式并

```

且打开 'ignorecase' 选项时才会使用。

"set backspace=2 " 设置退格键可用

set linebreak " 整词换行

set whichwrap=b,s,<,>[,]" 光标从行首和行末时可以跳到另一行去

"命令行设置

set showcmd " 命令行显示输入的命令

set showmode " 命令行显示 vim 当前模式

"搜索设置

set incsearch

" 输入字符串就显示匹配点

set hlsearch

set history=50 " 历史记录 50 条

set viminfo='1000,<500

filetype plugin indent on " 加了这句才可以用智能补全

"set tags=~/.Vim_tags/tags " 设置 tags 文件的路径

"括号自动补全

:inoremap ((<ESC>i

:inoremap) <c-r>=ClosePair(')')<CR>

:inoremap { {<CR><ESC>O

:inoremap } <c-r>=ClosePair('}')<CR>

:inoremap [[<ESC>i

:inoremap] <c-r>=ClosePair(']')<CR>

:inoremap " "<ESC>i

:inoremap ' '<ESC>i

function! ClosePair(char)

if getline('.')[col('.') - 1] == a:char

return "\<Right>"

else

return a:char

endif

endfunction

"-----

"拼写检查

"-----

map <leader>sn]

map <leader>sp [

map <leader>sa zg

map <leader>s? z=

"-----

"状态栏相关的设置

"-----

set statusline=[%F]%y%r%m%*%=[Line:%l/%L,Column:%c][%p%]" 状态栏的显示格式

set laststatus=2 " 总显示最后一个窗口的状态行; 设为 1 则窗口数多于一个的时候显示最后一个窗口的状态行; 0 不显示最后一个窗口的状态行

set ruler " 标尺，用于显示光标位置的行号和列号，逗号分隔。每个窗口都有自己的标尺。如果窗口有状态行，标尺在那里显示。否则，它显示在屏幕的最后一行上。

"-----
"代码折叠
"-----

"set foldmarker={{,}} " 指定折叠标志
"set foldmarker={,} " 指定折叠标志
syntax region functionFold start="/(^({/|/S.*{}/\$/n)/@<=/_[^}]"
end="/(^{.*}/)@<!\$/n/(^})/@=" transparent fold
syntax region commentFold start="/" transparent fold keepend
syntax region commentFold2 start="~/(^//.*\$/n)/@<!//.*(/n///)/@=\$"
end="~/(^//.*\$/n)/@<=//.*(/n///)/@!\$" transparent fold keepend

"set foldmethod=marker " 标志用于指定折叠。
set foldmethod=syntax " 语法高亮项目指定折叠。
set foldlevel=100 " 不要自动折叠
"set foldopen=search " 不要打开折叠，当搜索进它里面时
"set foldopen=undo " 不要打开折叠，当做撤销进它里面时
"set foldcolumn=4 " 如果非零，指定宽度的列在窗口的一侧显示，指示折叠的打开和关闭。最大值为 12

"-----
"C, C++的调试(适用于在 Vim 中编译调试单个文件)
"-----

map <F7> :call Rungdb()<CR><CR><CR>
func! Rungdb()
 exec "w"
 exec "!gcc % -g -o %<"
 exec "!gdb %<"
endfunc
"*****
"

" 插件设置
"scope ctags omnicppcomplete Taglist QuickFix MiniBufferExplorer
"NERDTree WinManager A grep NERD Commenter SuperTag
"c-support
"-----

"快捷键设置
"F2 自动补全的代码
"F3 根据头文件补全代码
"F4 grep
"F5 make
"F6 make clean
"F7 gdb (单个文件编译及调试)
"F8 更新 ctags 数据库
"F9 编译
"Ctrl+F9 运行
"F10
"F12 切换.c 和.h 文件

```

"Tag C-X C-O 补全
"wm WinManager
"cscope 检索
"Ctrl+Shift+- 然后按 s 查找本 C 符号(可以跳过注释)
"Ctrl+Shift+- 然后按 g 查找本定义
"Ctrl+Shift+- 然后按 d 查找本函数调用的函数
"Ctrl+Shift+- 然后按 c 查找调用本函数的函数
"Ctrl+Shift+- 然后按 e 查找本 egrep 模式
"Ctrl+Shift+- 然后按 f 查找本文件
"Ctrl+Shift+- 然后按 i 查找包含本文件的文件
"*****
"-- ctags setting --
" 按下 F8 重新生成 tag 文件，并更新 taglist
map <F8> :!ctags -R --c++-kinds=+p --fields=+iaS --extra=+q.<CR><CR>:TlistUpdate<CR>
imap <F8><ESC>:!ctags -R --c++-kinds=+p --fields=+iaS --extra=+q.<CR><CR>:TlistUpdate<CR>
set tags=tags;
set autochdir
"set tags+=./tags "add current directory's generated tags file
"set tags+=~/Vim_Tags/tags "add new tags file(刚刚生成 tags 的路径，在 ctags -R 生成 tags 文件
后，不要将 tags 移动到别的目录，否则 ctrl+] 时，会提示找不到源码文件)

"-- scope setting --
"用法：
"<1>、为源码建立一个 cscope 数据库
"lingd@ubuntu:~/arm/linux-2.6.28.7$ cscope -Rbq
"lingd@ubuntu:~/arm/linux-2.6.28.7$ ls cscope.*
"cscope.in.out cscope.out cscope.po.out
"<2>、用 vim 打开某个源码文件，末行模式下，输入 ":cs add /home/./cscope.out home/..."，添加
cscope 数据库到 vim。已将 vim 配置为启动时，自动添加当前目录下的 cscope 数据库
"<3>、完成前两步后，现在就可以用 "cs find c" 等 Cscope 查找命令查找关键字了。我们已在.vimrc
中将 "cs find c" 等 Cscope 查找命令映射为<C-_{c 等快捷键（按法是先按 Ctrl+Shift+-，然后很快按
下 c）
"-- Cscope setting --
if has("cscope")
    set csprg=/usr/bin/cscope " 指定用来执行 cscope 的命令
    set cst=0 " 设置 cstag 命令查找次序：0 先找 cscope 数据库再找标签文件；1 先找标
签文件再找 cscope 数据库
    set cst " 同时搜索 cscope 数据库和标签文件
    set cscopequickfix=s-,c-,d-,i-,t-,e- " 使用 QuickFix 窗口来显示 cscope 查找结果
    set noscverb
    if filereadable("cscope.out") " 若当前目录下存在 cscope 数据库，添加该数据库到 vim
        cs add cscope.out
    elseif $CSCOPE_DB != "" " 否则只要环境变量 CSCOPE_DB 不为空，则添加其指定的数据库到
vim
        cs add $CSCOPE_DB
    endif
    set csverb
endif

" 将:cs find c 等 Cscope 查找命令映射为<C-_{c 等快捷键（按法是先按 Ctrl+Shift+-，然后很快再

```

按下 c)

```
nmap <C-_>s :cs find s <C-R>=expand("<cword>")<CR><CR>:copen<CR><CR>
nmap <C-_>g :cs find g <C-R>=expand("<cword>")<CR><CR>
nmap <C-_>d :cs find d <C-R>=expand("<cword>")<CR><CR>:copen<CR><CR>
nmap <C-_>c :cs find c <C-R>=expand("<cword>")<CR><CR>:copen<CR><CR>
nmap <C-_>t :cs find t <C-R>=expand("<cword>")<CR><CR>:copen<CR><CR>
nmap <C-_>e :cs find e <C-R>=expand("<cword>")<CR><CR>:copen<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>:copen<CR><CR>
```

--- omnicppcomplete setting ---

set nocp
" 按下 F2 自动补全代码，注意该映射语句后不能有其他字符，包括 tab；否则按下 F3 会自动补全一些乱码

```
imap <F2><C-X><C-O>
" 按下 F3 根据头文件内关键字补全
imap <F3><C-X><C-I>
set completeopt=menu,menuone " 关掉智能补全时的预览窗口
let OmniCpp_MayCompleteDot = 1 " autocomplete with .
let OmniCpp_MayCompleteArrow = 1 " autocomplete with ->
let OmniCpp_MayCompleteScope = 1 " autocomplete with ::
let OmniCpp_SelectFirstItem = 2 " select first item (but don't insert)
let OmniCpp_NamespaceSearch = 2 " search namespaces in this and included files
let OmniCpp_ShowPrototypeInAbbr = 1 " show function prototype in popup window
let OmniCpp_GlobalScopeSearch=1 " enable the global scope search
let OmniCpp_DisplayMode=1 " Class scope completion mode: always show all members
"let OmniCpp_DefaultNamespaces=["std"]
let OmniCpp_ShowScopeInAbbr=1 " show scope in abbreviation and remove the last column
let OmniCpp_ShowAccess=1
```

--- Taglist setting ---

```
let Tlist_Ctags_Cmd='ctags' "因为我们放在环境变量里，所以可以直接执行
let Tlist_Use_Right_Window=0 "让窗口显示在右边，0 的话就是显示在左边
let Tlist_Show_One_File=0 "让 taglist 可以同时展示多个文件的函数列表
let Tlist_File_Fold_Auto_Close=1 "非当前文件，函数列表折叠隐藏
let Tlist_Exit_OnlyWindow=1 "当 taglist 是最后一个分割窗口时，自动推出 vim
"是否一直处理 tags. 1:处理;0:不处理
let Tlist_Process_File_Always=1 "实时更新 tags
let Tlist_Inc_Winwidth=0
```

--- QuickFix setting ---

" 按下 F6，执行 make clean
map <F6> :make clean<CR><CR><CR>
" 按下 F5，执行 make 编译程序，并打开 quickfix 窗口，显示编译信息
map <F5>:make<CR><CR><CR>:copen<CR><CR>
" 以上的映射是使上面的快捷键在插入模式下也能用
imap <F6><ESC>:make clean<CR><CR><CR>
imap <F5><ESC>:make<CR><CR><CR>:copen<CR><CR>

```

"-- MiniBufferExplorer setting--
let g:miniBufExplMapWindowNavVim=1 " 按下 Ctrl+h/j/k/l, 可以切换到当前窗口的上下左右窗口
let g:miniBufExplMapWindowNavArrows=1 " 按下 Ctrl+箭头, 可以切换到当前窗口的上下左右窗口
let g:miniBufExplMapCTabSwitchBufs=1 " 启用以下两个功能: Ctrl+tab 移到下一个 buffer 并在当前窗口打开; Ctrl+Shift+tab 移到上一个 buffer 并在当前窗口打开; ubuntu 好像不支持
"let g:miniBufExplMapCTabSwitchWindows=1 " 启用以下两个功能: Ctrl+tab 移到下一个窗口; Ctrl+Shift+tab 移到上一个窗口; ubuntu 好像不支持
let g:miniBufExplModSelTarget=1 " 不要在不可编辑内容的窗口(如 TagList 窗口)中打开选中的 buffer

"--NERDTree setting--
let g:NERDTree_title="[NERDTree]"
"let g:winManagerWindowLayout="NERDTree|TagList"

function! NERDTree_Start()
    exec 'NERDTree'
endfunction

function! NERDTree_IsValid()
    return 1
endfunction

"-- WinManager setting --
let g:winManagerWindowLayout='FileExplorer|TagList' " 设置我们要管理的插件
"let g:persistentBehaviour=0 " 如果所有编辑文件都关闭了, 退出 vim
nmap wm :WMToggle<cr>

"-- A setting --
nnoremap <silent><F12>:A<CR>

"-- grep setting --
nnoremap <silent><F4>:Grep<CR>"

"-- NERD Commenter setting --
",ca, 在可选的注释方式之间切换, 比如 C/C++ 的块注释和行注释//
",cc, 注释当前行
",c, 切换注释/非注释状态
",cs, 以"性感"的方式注释
",cA, 在当前行尾添加注释符, 并进入 Insert 模式
",cu, 取消注释
"Normal 模式下, 几乎所有命令前面都可以指定行数
"Visual 模式下执行命令, 会对选中的特定区块进行注释/反注释
let mapleader=","

"-- SuperTag setting --
let g:SuperTabDefaultCompletionType="context"

"-- c-support setting --

```

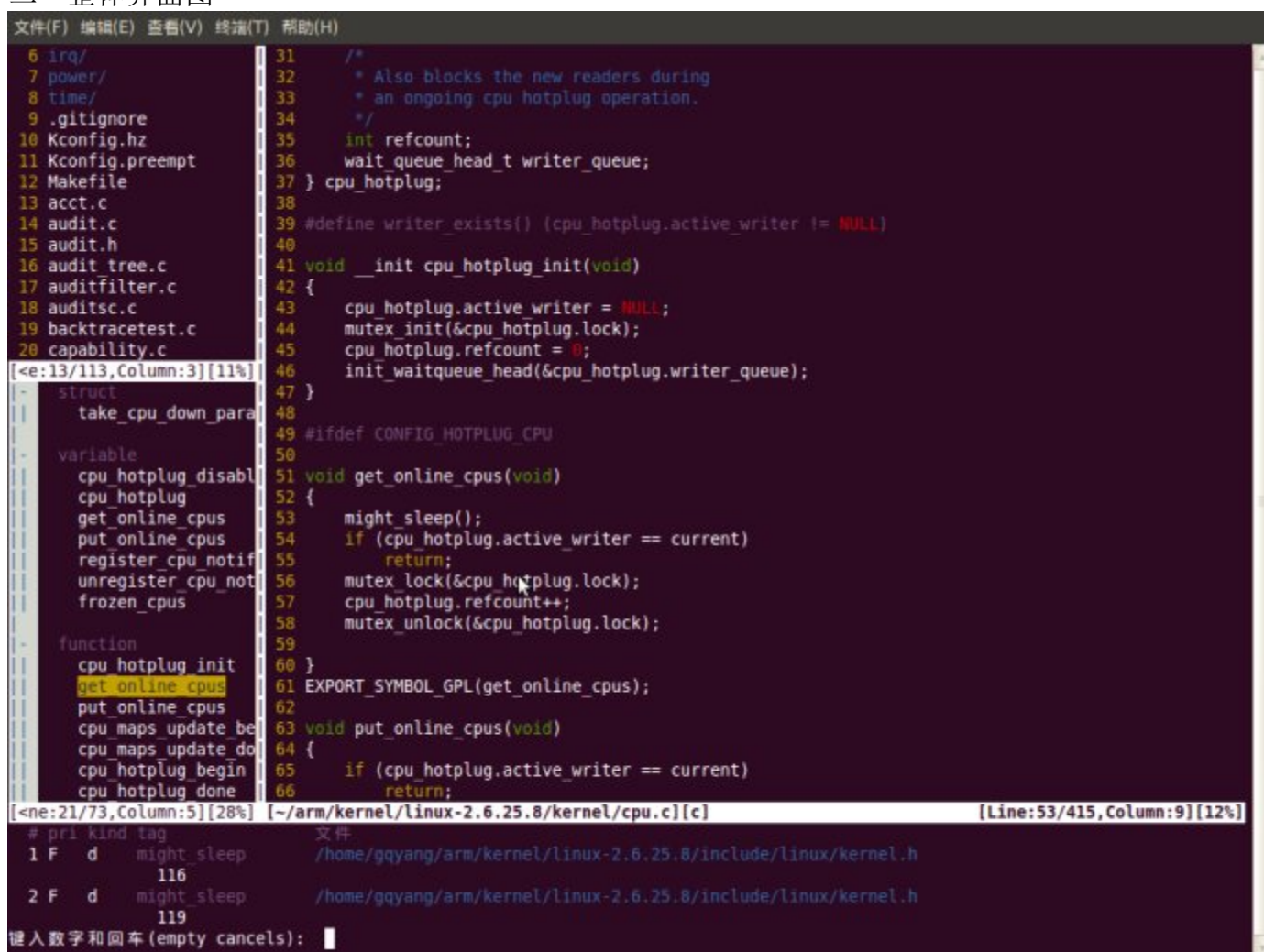
”解压，有 c-support、ftplugin、doc、plugin 四个文件，拷贝到\$HOME/.vim/目录下。并在.vimrc 中用命令 filetype plugin on 激活插件功能。

”功能一：自动为*.c 文件添加文件头说明

”\$ vim ~/.vim/c-support/templates/Templates 可改变头说明

”快捷键帮助信息详见 doc 的 pdf 文档

二 • 整体界面图



VIM 整体界面图

三 • 对涉及到的插件信息整理如下

1. ctags 安装与配置

Ctags 工具是用来遍历源代码文件生成 tags 文件，这些 tags 文件能被编辑器或其它工具用来快速查找定位源代码中的符号（tag/symbol），如变量名，函数名等。比如，tags 文件就是 Taglist 和 OmniCppComplete 工作的基础。

安装步骤跟大多数软件的从源代码安装步骤一样。

- 1) 从 <http://ctags.sourceforge.net/> 下载源代码包后，解压缩生成源代码目录，
- 2) 然后进入源代码根目录执行 ./configure，
- 3) 然后执行 make，
- 4) 编译成功后执行 make install。

Ubuntu 中直接 `sudo apt-get install ctags` 就可以了。

在 ~/.vimrc 中的配置如下：

```
"-- ctags setting --
" 按下 F8 重新生成 tag 文件，并更新 taglist
map <F8> :!ctags -R --c++-kinds=+p --fields=+iaS --extra=+q.<CR><CR>:TlistUpdate<CR>
imap <F8><ESC>:!ctags -R --c++-kinds=+p --fields=+iaS --extra=+q.<CR><CR>:TlistUpdate<CR>
set tags=tags;
set autochdir
"set tags+=./tags "add current directory's generated tags file
"set tags+=~/Vim_Tags/tags "add new tags file(刚刚生成 tags 的路径，在 ctags -R 生成 tags 文件
后，不要将 tags 移动到别的目录，否则 ctrl+] 时，会提示找不到源码文件)
```

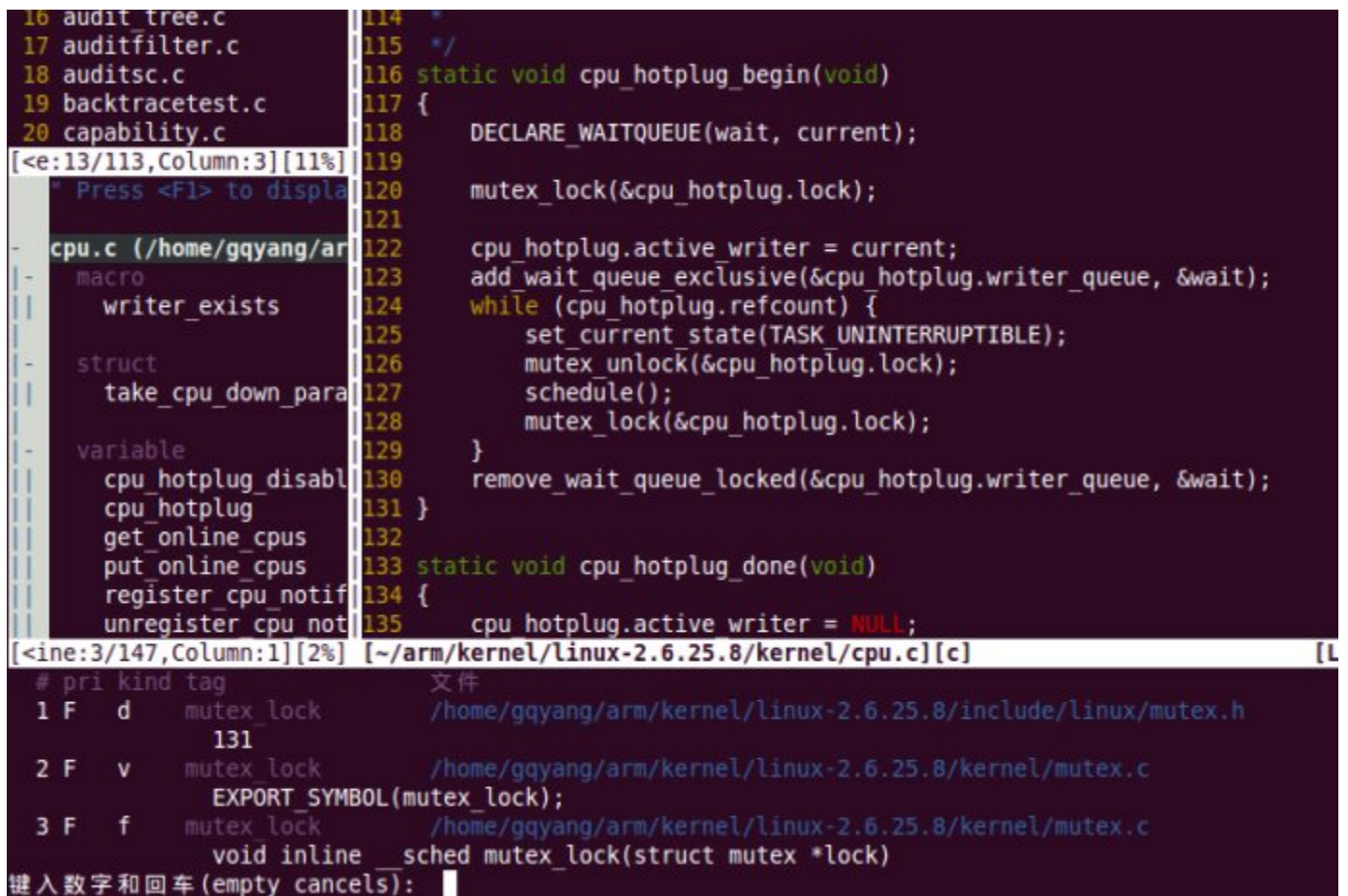
ctags 安装完成。

使用 Ctags 的也很简单。进入我们的项目代码根目录，执行以下命令：ctags -R，会在源代码目录生成 tags 文件。配置文件中已经配置了 F8 快捷键，所以可以进入代码根目录后，打开 Vim，按下 F8 快捷键自动生成 tags 文件。此时，我们已经具有定义跳转的功能了。有两组快捷键是最常用的。

Ctrl-] 跳转到光标所在符号的定义。

Ctrl-t 回到上次跳转前的位置。

更多功能通过命令 man ctags 或在 Vim 命令行下运行 help ctags 查询。



The screenshot shows the Vim editor with a taglist window on the left. The taglist window displays a list of tags from the file /home/gqyang/arm/kernel/linux-2.6.25.8/kernel/mutex.c. The tags include mutex_lock, EXPORT_SYMBOL(mutex_lock), and void inline __sched mutex_lock(struct mutex *lock). The main window shows the source code of the mutex_lock function, which is a static inline function that calls mutex_lock_slowpath. The cursor is positioned at the start of the function definition.

```
16 audit_tree.c
17 auditfilter.c
18 auditsc.c
19 backtracetest.c
20 capability.c
[<e:13/113,Column:3][11%]
" Press <F1> to displa
cpu.c (/home/gqyang/arm
macro
writer_exists
struct
take_cpu_down_param
variable
cpu_hotplug_disable
cpu_hotplug
get_online_cpus
put_online_cpus
register_cpu_notifier
unregister_cpu_notifier
114 *
115 */
116 static void cpu_hotplug_begin(void)
117 {
118     DECLARE_WAITQUEUE(wait, current);
119
120     mutex_lock(&cpu_hotplug.lock);
121
122     cpu_hotplug.active_writer = current;
123     add_wait_queue_exclusive(&cpu_hotplug.writer_queue, &wait);
124     while (cpu_hotplug.refcount) {
125         set_current_state(TASK_UNINTERRUPTIBLE);
126         mutex_unlock(&cpu_hotplug.lock);
127         schedule();
128         mutex_lock(&cpu_hotplug.lock);
129     }
130     remove_wait_queue_locked(&cpu_hotplug.writer_queue, &wait);
131 }
132
133 static void cpu_hotplug_done(void)
134 {
135     cpu_hotplug.active_writer = NULL;
136 }
[<ine:3/147,Column:1][2%] [~/arm/kernel/linux-2.6.25.8/kernel/cpu.c][c] [L
# pri kind tag 文件
1 F d mutex_lock /home/gqyang/arm/kernel/linux-2.6.25.8/include/linux/mutex.h
131
2 F v mutex_lock /home/gqyang/arm/kernel/linux-2.6.25.8/kernel/mutex.c
EXPORT_SYMBOL(mutex_lock);
3 F f mutex_lock /home/gqyang/arm/kernel/linux-2.6.25.8/kernel/mutex.c
void inline __sched mutex_lock(struct mutex *lock)
键入数字和回车(empty cancels):
```

查找 mutex_lock 函数

2. Taglist 安装与配置

Taglist 提供源代码符号的结构化视图，可以列出当前文件中的所有标签（宏，全局变量，函数名等）。

安装步骤:

1) 从 http://www.vim.org/scripts/script.php?script_id=273 下载安装包。

2) 将 Taglist 安装包解压, 把解压后的文件夹 (doc 和 plugin) 复制到在 ~/.vim 目录, 提示重名时选择合并, 其他插件安装过程类似。

3) 进入 ~/.vim/doc 目录, 在 Vim 下运行 "helptags." 命令。此步骤是将 doc 下的帮助文档加入到 Vim 的帮助主题中, 这样我们就可以通过在 Vim 中运行 "help taglist.txt" 查看 taglist 帮助。

4) 在 .vimrc 文件中进行配置

-- Taglist setting --

```
let Tlist_Ctags_Cmd='ctags' "因为我们放在环境变量里, 所以可以直接执行
let Tlist_Use_Right_Window=0 "让窗口显示在右边, 0 的话就是显示在左边
let Tlist_Show_One_File=0 "让 taglist 可以同时展示多个文件的函数列表
let Tlist_File_Fold_Auto_Close=1 "非当前文件, 函数列表折叠隐藏
let Tlist_Exit_OnlyWindow=1 "当 taglist 是最后一个分割窗口时, 自动退出 vim
"是否一直处理 tags. 1:处理;0:不处理
let Tlist_Process_File_Always=1 "实时更新 tags
let Tlist_Inc_Winwidth=0
```

taglist 安装完成。

在 Vim 命令行下运行 TlistToggle 命令就可以打开 Taglist 窗口, 再次运行 TlistToggle 则关闭。

```

14 audit.c
15 audit.h
<e:13/113,Column:3][11%]
cpu.c (/home/gqyang/ar
- macro
|   writer_exists
- struct
|   take_cpu_down_para
- variable
|   cpu_hotplug_disabl
|   cpu_hotplug
|   get_online_cpus
|   put_online_cpus
|   register_cpu_notif
|   unregister_cpu_not
|   frozen_cpus
- function
|   cpu_hotplug_init
|   get_online_cpus
|   put_online_cpus
|   cpu_maps update be
|   cpu_maps update do
|   cpu_hotplug_begin
|   cpu_hotplug_done
|   register_cpu_notif
|   unregister_cpu_not
|   check_for_tasks
|   take cpu down
107 * - Last rea
108 * - A new re
109 * - The writ
110 *   non zero
111 *
112 * However, t
113 * get_online
114 *
115 */
116 static void c
117 {
118     DECLARE_W
119
120     mutex_loc
121
122     cpu_hotpl
123     add wait
124     while (cp
125         set_d
126         mutex
127         sched
128         mutex
129     }
130     remove_wa
131 }
132
133 static void c
134 {
135     cpu_hotpl
136     mutex_unl
137 }

```

3. Cscope 安装与配置

Cscope 是一个类似于 ctags 的工具，不过其功能比 ctags 强大很多，提供交互式查询语言符号功能，如查询哪些地方使用某个变量或调用某个函数。

安装步骤：ubuntu 直接 `sudo apt-get install cscope` 即可。

在 .vimrc 文件中进行配置：

```

"-- scope setting --
"-- Cscope setting --
if has("cscope")
    set csprg=/usr/bin/cscope    " 指定用来执行 cscope 的命令
    set cst=0                    " 设置 cstag 命令查找次序：0 先找 cscope 数据库再找标签文件；1 先找标
    签文件再找 cscope 数据库
    set cst                      " 同时搜索 cscope 数据库和标签文件
    set cscopequickfix=s-,c-,d-,i-,t-,e- " 使用 QuickFix 窗口来显示 cscope 查找结果
    set nocsverb
    if filereadable("cscope.out") " 若当前目录下存在 cscope 数据库，添加该数据库到 vim
        cs add cscope.out
    elseif $CSCOPE_DB != ""      " 否则只要环境变量 CSCOPE_DB 不为空，则添加其指定的数据库到
vim
        cs add $CSCOPE_DB
    endif

```

```
set csverb
endif
```

” 将:cs find c 等 Cscope 查找命令映射为<C->c 等快捷键（按法是先按 Ctrl+Shift+~, 然后很快再按下 c）

```
nmap <C->s :cs find s <C-R>=expand("<word>")<CR><CR>:copen<CR><CR>
nmap <C->g :cs find g <C-R>=expand("<word>")<CR><CR>
nmap <C->d :cs find d <C-R>=expand("<word>")<CR><CR>:copen<CR><CR>
nmap <C->c :cs find c <C-R>=expand("<word>")<CR><CR>:copen<CR><CR>
nmap <C->t :cs find t <C-R>=expand("<word>")<CR><CR>:copen<CR><CR>
nmap <C->e :cs find e <C-R>=expand("<word>")<CR><CR>:copen<CR><CR>
nmap <C->f :cs find f <C-R>=expand("<file>")<CR><CR>
nmap <C->i :cs find i <C-R>=expand("<file>")<CR><CR>:copen<CR><CR>
```

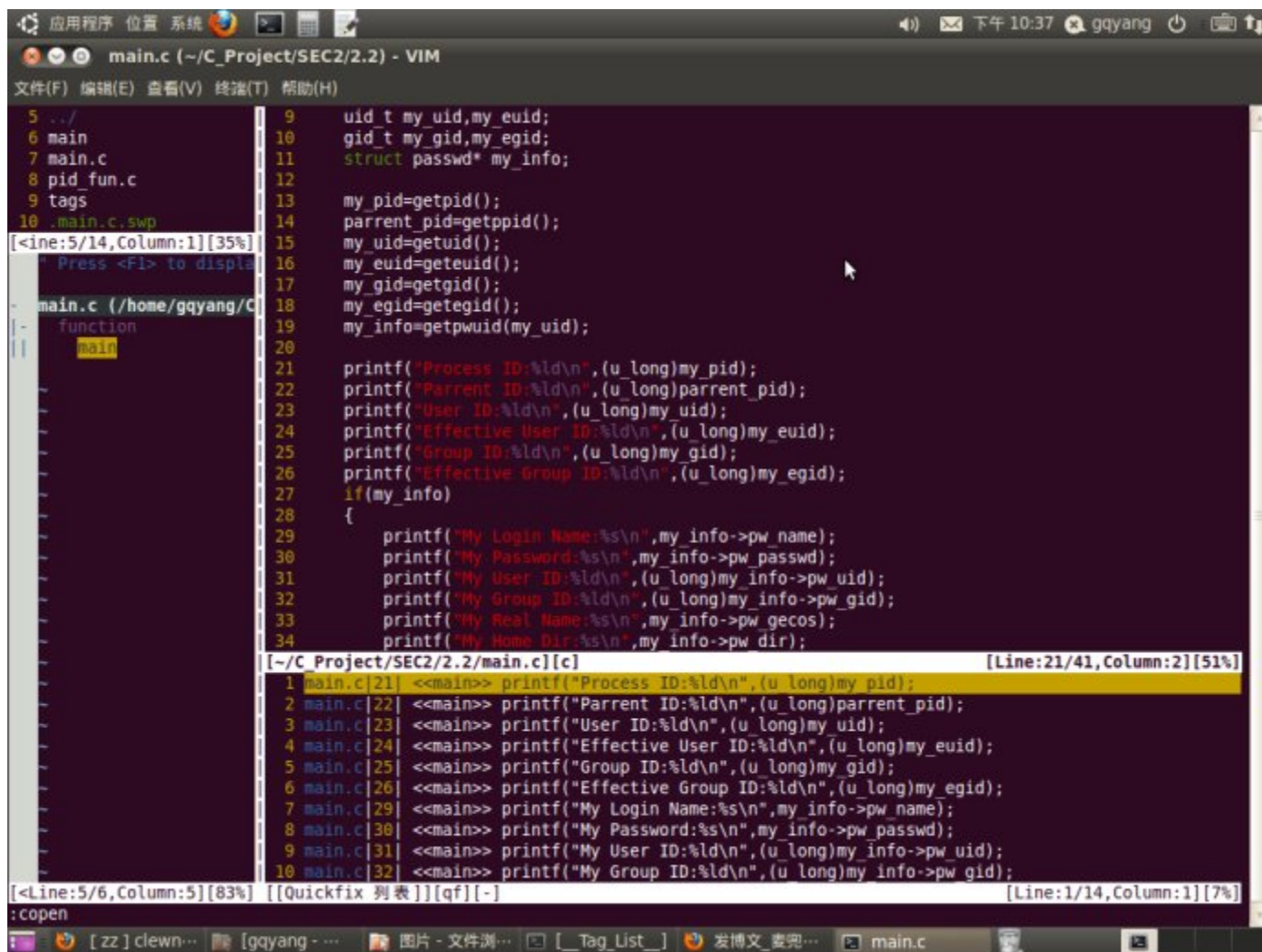
使用方法:

1) 为源码建立一个 cscope 数据库

```
@ubuntu:~/arm/linux-2.6.28.7$ cscope -Rbq
@ubuntu:~/arm/linux-2.6.28.7$ ls cscope.*
"cscope.in.out cscope.out cscope.po.out
```

2) 用 vim 打开某个源码文件, 末行模式下, 输入 “:cs add /home/././cscope.out home/...”, 添加 cscope 数据库到 vim。已将 vim 配置为启动时, 自动添加当前目录下的 cscope 数据库

3) 完成前两步后, 现在就可以用 “cs find c” 等 Cscope 查找命令查找关键字了。我们已在.vimrc 中将 “cs find c” 等 Cscope 查找命令映射为<C->c 等快捷键（按法是先按 Ctrl+Shift+~, 然后很快按下 c）



```
5 ./
6 main
7 main.c
8 pid fun.c
9 tags
10 .main.c.swp
[Line:5/14,Column:1][35%]
Press <F1> to display help
main.c (/home/gqyang/C
function
main
9 uid_t my_uid,my_euid;
10 gid_t my_gid,my_egid;
11 struct passwd* my_info;
12
13 my_pid=getpid();
14 parent_pid=getppid();
15 my_uid=getuid();
16 my_euid=geteuid();
17 my_gid=getgid();
18 my_egid=getegid();
19 my_info=getpwuid(my_uid);
20
21 printf("Process ID:%ld\n",(u_long)my_pid);
22 printf("Parrent ID:%ld\n",(u_long)parent_pid);
23 printf("User ID:%ld\n",(u_long)my_uid);
24 printf("Effective User ID:%ld\n",(u_long)my_euid);
25 printf("Group ID:%ld\n",(u_long)my_gid);
26 printf("Effective Group ID:%ld\n",(u_long)my_egid);
27 if(my_info)
28 {
29     printf("My Login Name:%s\n",my_info->pw_name);
30     printf("My Password:%s\n",my_info->pw_passwd);
31     printf("My User ID:%ld\n",(u_long)my_info->pw_uid);
32     printf("My Group ID:%ld\n",(u_long)my_info->pw_gid);
33     printf("My Real Name:%s\n",my_info->pw_gecos);
34     printf("My Home Dir:%s\n",my_info->pw_dir);
[Line:21/41,Column:2][51%]
1 main.c[21] <<main>> printf("Process ID:%ld\n",(u_long)my_pid);
2 main.c[22] <<main>> printf("Parrent ID:%ld\n",(u_long)parent_pid);
3 main.c[23] <<main>> printf("User ID:%ld\n",(u_long)my_uid);
4 main.c[24] <<main>> printf("Effective User ID:%ld\n",(u_long)my_euid);
5 main.c[25] <<main>> printf("Group ID:%ld\n",(u_long)my_gid);
6 main.c[26] <<main>> printf("Effective Group ID:%ld\n",(u_long)my_egid);
7 main.c[29] <<main>> printf("My Login Name:%s\n",my_info->pw_name);
8 main.c[30] <<main>> printf("My Password:%s\n",my_info->pw_passwd);
9 main.c[31] <<main>> printf("My User ID:%ld\n",(u_long)my_info->pw_uid);
10 main.c[32] <<main>> printf("My Group ID:%ld\n",(u_long)my_info->pw_gid);
[Line:1/14,Column:1][7%]
:copen
[ zz ] clewn... [gqyang - ...] 图片 - 文件浏... [_ Tag_List_] 发博文_麦兜... main.c
```

4. OmniCppComplete 安装与配置

OmniCppComplete 主要提供输入时实时提供类或结构体的属性或方法的提示和补全。该功能要 tags 文件的支持。

安装步骤:

1) 从 http://www.vim.org/scripts/script.php?script_id=1520 下载安装包。

2) 将 omnicppcomplete-0.41 安装包解压,把解压后的文件夹复制到在 ~/.vim 目录,提示重名时选择合并。

3) 进入 ~/.vim/doc 目录,在 Vim 下运行 "helptags." 命令。

4) 在 .vimrc 文件中进行配置

-- omnicppcomplete setting --

set nocp

" 按下 F2 自动补全代码,注意该映射语句后不能有其他字符,包括 tab; 否则按下 F3 会自动补全一些乱码

imap <F2><C-X><C-O>

" 按下 F3 根据头文件内关键字补全

imap <F3><C-X><C-I>

set completeopt=menu,menuone " 关掉智能补全时的预览窗口

let OmniCpp_MayCompleteDot = 1 " autocomplete with .

let OmniCpp_MayCompleteArrow = 1 " autocomplete with ->

```

let OmniCpp_MayCompleteScope = 1 " autocomplete with ::
let OmniCpp_SelectFirstItem = 2 " select first item (but don't insert)
let OmniCpp_NamespaceSearch = 2 " search namespaces in this and included files
let OmniCpp_ShowPrototypeInAbbr = 1 " show function prototype in popup window
let OmniCpp_GlobalScopeSearch = 1 " enable the global scope search
let OmniCpp_DisplayMode = 1 " Class scope completion mode: always show all members
"let OmniCpp_DefaultNamespaces=["std"]
let OmniCpp_ShowScopeInAbbr = 1 " show scope in abbreviation and remove the last column
let OmniCpp_ShowAccess = 1

```

omnicppcomplete 安装完成。

OmniCppComplete 是基于 ctags 数据库即 tags 文件实现的(基于 ctags 生成的索引信息来实现自动补全的)，所以在 ctags -R 生成 tags 时还需要一些额外的选项，这样生成的 tags 文件才能与 OmniCppComplete 配合运作。使用下列命令生成 tags 文件，就可以与 OmniCppComplete 配合运作：

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

--c++-kinds=+p : 为 C++ 文件增加函数原型的标签

--fields=+iaS : 在标签文件中加入继承信息(i)、类成员的访问控制信息(a)、以及函数的指纹(S)

--extra=+q : 为标签增加类修饰符。注意，如果没有此选项，将不能对类成员补全

vim 自动补全功能的测试

为了测试自动补全功能，我们先下载 C++ 一份 C++ 标准库的源代码。

```
lingd@ubuntu:~$ sudo apt-get install build-essential
```

然后在 /usr/include/c++ 下就可以找到标准库的头文件了。

```
lingd@ubuntu:~$ cd /usr/include/c++
```

```
lingd@ubuntu:/usr/include/c++$ ls
```

在此文件夹下生成能与 OmniCppComplete 配合运作的 tags 文件

```
lingd@ubuntu:/usr/include/c++$ ctags -R --c++-kinds=+p --fields=+iaS --extra=+q.
```

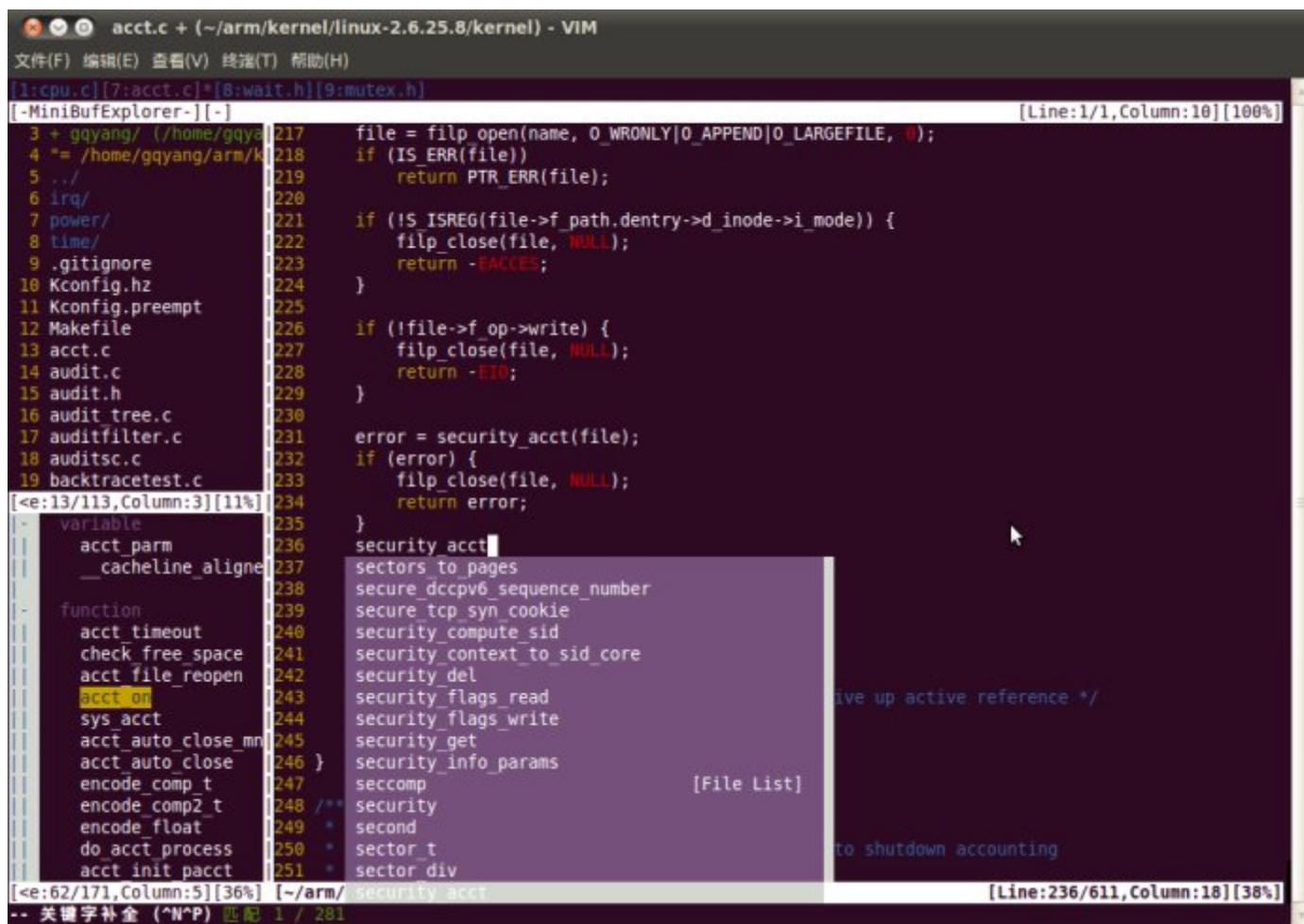
```
lingd@ubuntu:/usr/include/c++$ ls
```

在 vim 的配置文件中如下内容，然后在编程的时候就可以使用自动补全功能了。

```
lingd@ubuntu:/usr/include/c++$ vi ~/.vimrc
```

```
set tags+=/usr/include/c++/tags
```

更多功能通过在 Vim 命令行下运行 "help omnicppcomplete" 查询。



5. SuperTab 安装与配置

SuperTab 使 Tab 快捷键具有更快捷的上下文提示功能。跟 OmniCppComplete 一样，SuperTab 也是一个 Vim 插件。

安装步骤：

1) 从 http://www.vim.org/scripts/script.php?script_id=1643 下载安装包。

2) 这个安装包跟之前的几个 Vim 插件不同，它是一个 vmb 文件。用 VIM 打开此文件（\$ vim supertab.vmb），在命令行输入:so %即可。

3) 在.vimrc 文件中进行配置

-- SuperTag setting --

```
let g:SuperTabDefaultCompletionType="context"
```

SuperTab 安装完成。

SuperTab 使用很简单，只要在输入变量名或路径名等符号中途按 Tab 键，就能得到以前输入过的符号列表，并通过 Tab 键循环选择。

使用 SuperTab 后，<Tab>键缩进不能用了，如果前面有字符按下<Tab>键后就会进行补全，而不是缩进，如果哪位朋友知道，请一定记得告诉我 guoqing-yang@163.com 。

6. Winmanager, NERDTree 和 MiniBufExplorert 安装与配置

这三个插件，主要优化布置 Vim 的界面。NERDTree 提供树形浏览文件系统的界面，MiniBufExplorer 提供多文件同时编辑功能，而 Winmanager 将这 NERDTree 界面和 Taglist 界面整合起来，使 Vim 更像 VS！

安装步骤:

- 1) 从 http://www.vim.org/scripts/script.php?script_id=1658
http://www.vim.org/scripts/script.php?script_id=159
http://www.vim.org/scripts/script.php?script_id=95 下载安装包。
- 2) 将 NERDTree 安装包解压, 把解压后的文件夹复制到在 ~/.vim 目录, 提示重名时选择合并。
- 3) 进入 ~/.vim/doc 目录, 在 Vim 下运行 "helptags." 命令。
- 4) MiniBufExplorer 只有一个 .vim 文件, 将其拷贝到 ~/.vim/plugin 目录。
- 5) Winmanager 安装与 NERDTree 步骤相同。
- 6) 在 .vimrc 文件中进行配置。

```
-- WinManager setting --
let g:winManagerWindowLayout='FileExplorer|TagList' " 设置我们要管理的插件
"let g:persistentBehaviour=0 " 如果所有编辑文件都关闭了, 退出 vim
nmap wm :WMToggle<cr>

--NERDTree setting--
let g:NERDTree_title="[NERDTree]"
"let g:winManagerWindowLayout="NERDTree|TagList"

function! NERDTree_Start()
    exec 'NERDTree'
endfunction

function! NERDTree_IsValid()
    return 1
endfunction

-- MiniBufferExplorer setting--
let g:miniBufExplMapWindowNavVim = 1 " 按下 Ctrl+h/j/k/l, 可以切换到当前窗口的上下左右窗口
let g:miniBufExplMapWindowNavArrows = 1 " 按下 Ctrl+箭头, 可以切换到当前窗口的上下左右窗口
let g:miniBufExplMapCTabSwitchBufs = 1 " 启用以下两个功能: Ctrl+tab 移到下一个 buffer 并在当前窗口打开; Ctrl+Shift+tab 移到上一个 buffer 并在当前窗口打开; ubuntu 好像不支持
"let g:miniBufExplMapCTabSwitchWindows = 1 " 启用以下两个功能: Ctrl+tab 移到下一个窗口; Ctrl+Shift+tab 移到上一个窗口;
let g:miniBufExplModSelTarget = 1 " 不要在不可编辑内容的窗口 (如 TagList 窗口) 中打开选中的 buffer
```

三个插件安装完成。

现在进入我们的项目目录, 打开 Vim, 按下组合快捷键 w-m 就可以我们的崭新的 Vim 了! 再次按下 w-m 就可关闭界面。

MiniBufferExplorer 常用命令

<Tab> 移到上一个 buffer

<Shift-Tab> 移到下一个 buffer

<Enter> 打开光标所在的 buffer

d 删除光标所在的 buffer

WinManager 常用命令

wm 打开/关闭 WinManage

文件浏览器命令（在文件浏览器窗口中使用）

<enter>或双击 如果光标下是目录，则进入该目录；如果光标下文件，则打开该文件

<tab> 如果光标下是目录，则进入该目录；如果光标下文件，则在新窗口打开该文件

<F5> 刷新列表

- 返回上一层目录

c 使浏览目录成为 vim 当前工作目录

d 创建目录

D 删除当前光标下的目录或文件

i 切换显示方式

R 文件或目录重命名

s 选择排序方式

r 反向排序列表

x 定制浏览方式，使用你指定的程序打开该文件

NERDTree 常用命令

回车，打开文档；r，刷新工程目录文件列表；I，显示或隐藏隐藏文件

7. QuickFix 安装与配置

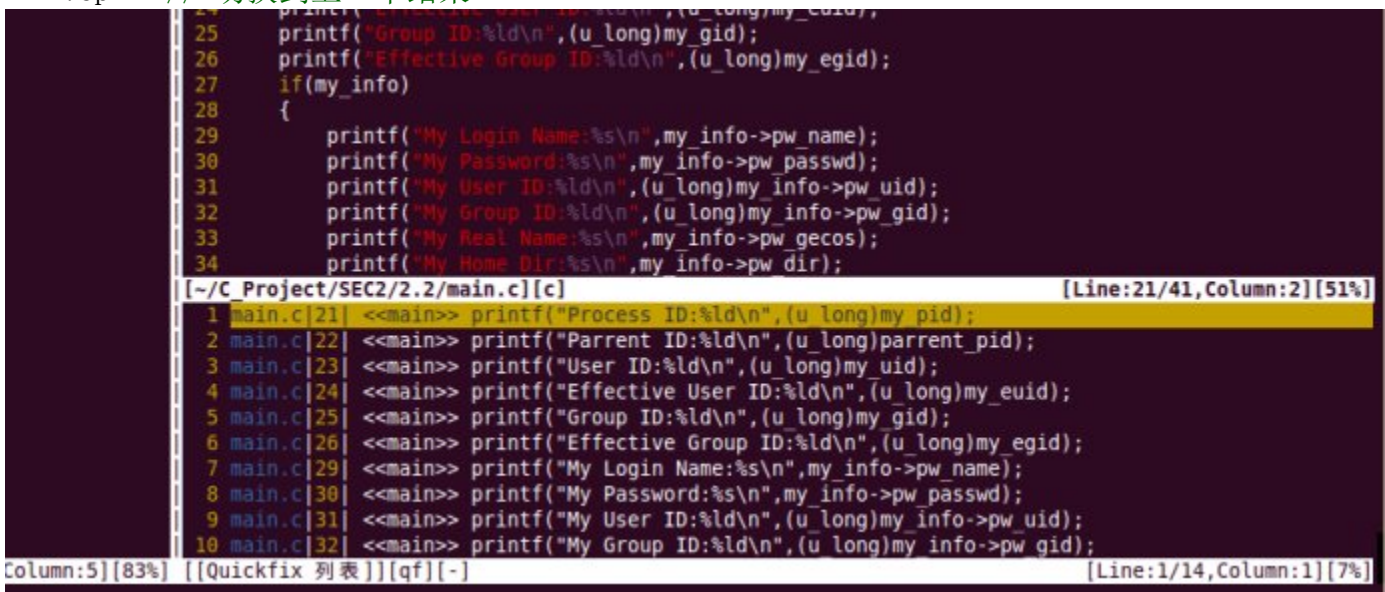
Cscope 查找有一个显示查询结果的窗口，这个窗口中列出了查询命令的查询结果，用户可以从这个窗口中选择每个结果进行查看，这个窗口叫“QuickFix”窗口，以前也是一个 vim 的插件来的，只不过现在成了 vim 的标准插件，不用去安装了，QuickFix 窗口的主要作用就是上面看到的那个功能：输出一些供选择的结果，可以被很多命令调用，更详细的介绍和使用方法见 help quickfix 件。

QuickFix 常用命令

:cw 调出 QuickFix 窗口

:cn // 切换到下一个结果

:cp // 切换到上一个结果



```
24 printf("Effective User ID:%ld\n", (u_long)my_euid);
25 printf("Group ID:%ld\n", (u_long)my_gid);
26 printf("Effective Group ID:%ld\n", (u_long)my_egid);
27 if(my_info)
28 {
29     printf("My Login Name:%s\n", my_info->pw_name);
30     printf("My Password:%s\n", my_info->pw_passwd);
31     printf("My User ID:%ld\n", (u_long)my_info->pw_uid);
32     printf("My Group ID:%ld\n", (u_long)my_info->pw_gid);
33     printf("My Real Name:%s\n", my_info->pw_gecos);
34     printf("My Home Dir:%s\n", my_info->pw_dir);
}

[~/C Project/SEC2/2.2/main.c][c] [Line:21/41, Column:2] [51%]
1 main.c[21] <<main>> printf("Process ID:%ld\n", (u_long)my_pid);
2 main.c[22] <<main>> printf("Parrent ID:%ld\n", (u_long)parrent_pid);
3 main.c[23] <<main>> printf("User ID:%ld\n", (u_long)my_uid);
4 main.c[24] <<main>> printf("Effective User ID:%ld\n", (u_long)my_euid);
5 main.c[25] <<main>> printf("Group ID:%ld\n", (u_long)my_gid);
6 main.c[26] <<main>> printf("Effective Group ID:%ld\n", (u_long)my_egid);
7 main.c[29] <<main>> printf("My Login Name:%s\n", my_info->pw_name);
8 main.c[30] <<main>> printf("My Password:%s\n", my_info->pw_passwd);
9 main.c[31] <<main>> printf("My User ID:%ld\n", (u_long)my_info->pw_uid);
10 main.c[32] <<main>> printf("My Group ID:%ld\n", (u_long)my_info->pw_gid);

Column:5] [83%] [[Quickfix 列表]] [qf] [-] [Line:1/14, Column:1] [7%]
```

8. Grep 安装与配置

进行工程内全局查找，可以借助此插件实现。

安装步骤：

- 1) 从 http://www.vim.org/scripts/script.php?script_id=311 下载安装包。
- 2) 把 grep.vim 文件复制到 ~/.vim/plugin 文件夹。
- 3) 在.vimrc 文件中进行配置。

-- grep setting --

nnoremap <silent><F4>:Grep<CR>

taglist 安装完成。

将光标移到要 grep 的地方上，然后按下 F4 键。

```
50
51 void get_online_cpus(void)
52 {
53     might_sleep();
54     if (cpu_hotplug.active_writer == current)
55         return;
56     mutex_lock(&cpu_hotplug.lock);
57     cpu_hotplug.refcount++;
58     mutex_unlock(&cpu_hotplug.lock);
59
60 }
```

```
cpu hotplug done | 71
[<e:21/171,Column:5][12%] [~/arm/kerr
Search for pattern: might_sleep
```

在最下面的命令行会显示：

Search in files: *

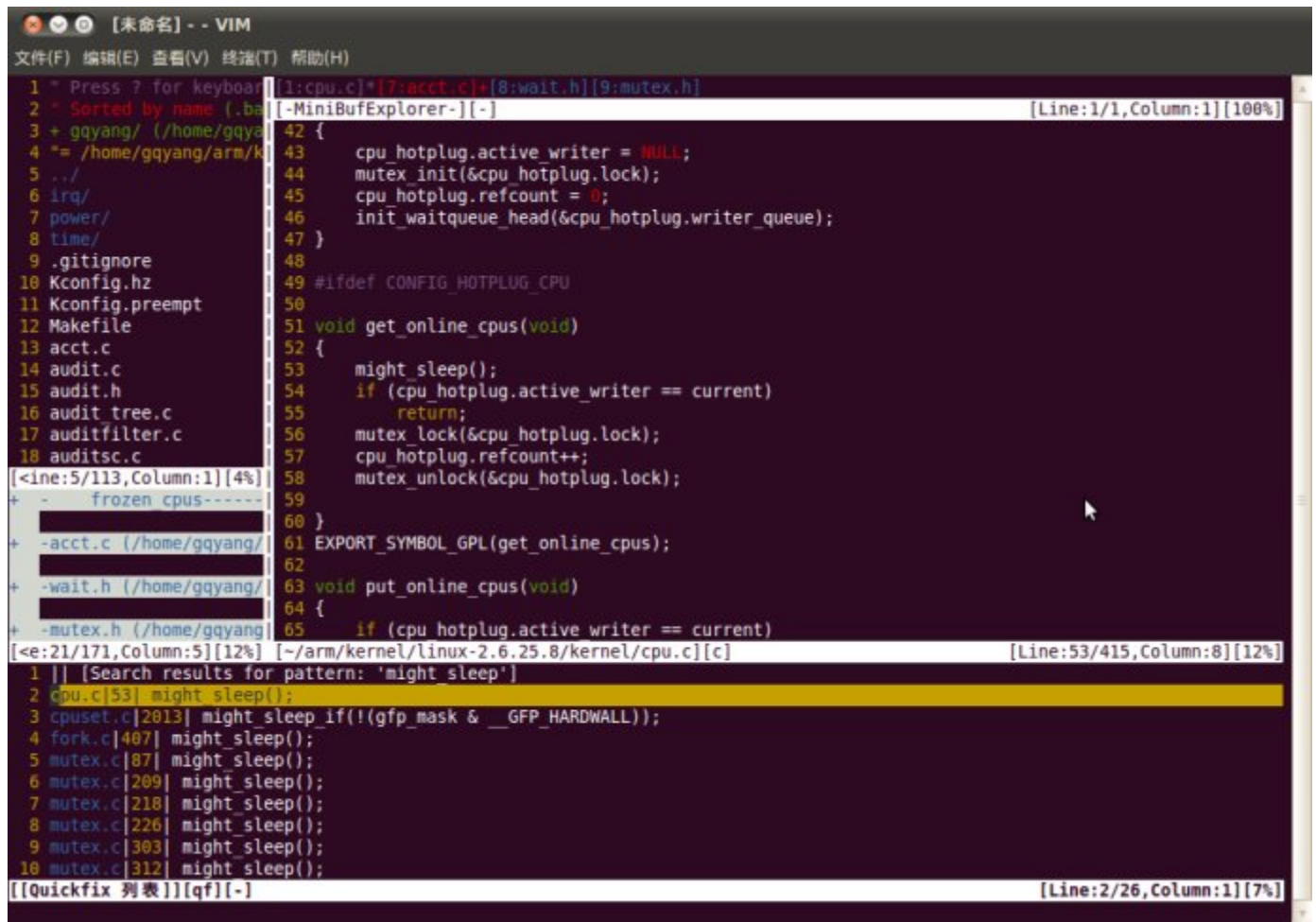
```
[<e:21/171,Column:5][12%] [~/arm
Search in files: *
```

是问你搜索范围，默认是该目录下的所有文件，此时你还可以编辑该行，比如你只想搜索源码文件：

Search in files: *.c *.h

```
cpu_hotplug_begin | 70     if (unlikely(writer_exists()) &&
cpu_hotplug done   | 71     wake up(&cpu_hotplug.writer qu
[<e:21/171,Column:5][12%] [~/arm/kernel/linux-2.6.25.8/kernel/cpu.c]
Search in files: *.c
```

然后在按下回车，会在弹出的 QuickFix 窗口中列出所有符合条件的搜索结果，你可以在其中查找你想要的结果，



9. A 插件安装与配置

c/h 文件间相互切换。

安装步骤：

- 1) 从 http://www.vim.org/scripts/script.php?script_id=31 下载安装包。
- 2) 将 a.vim 复制到 ~/.vim/plugin 文件夹中。
- 3) 在 .vimrc 文件中进行配置。

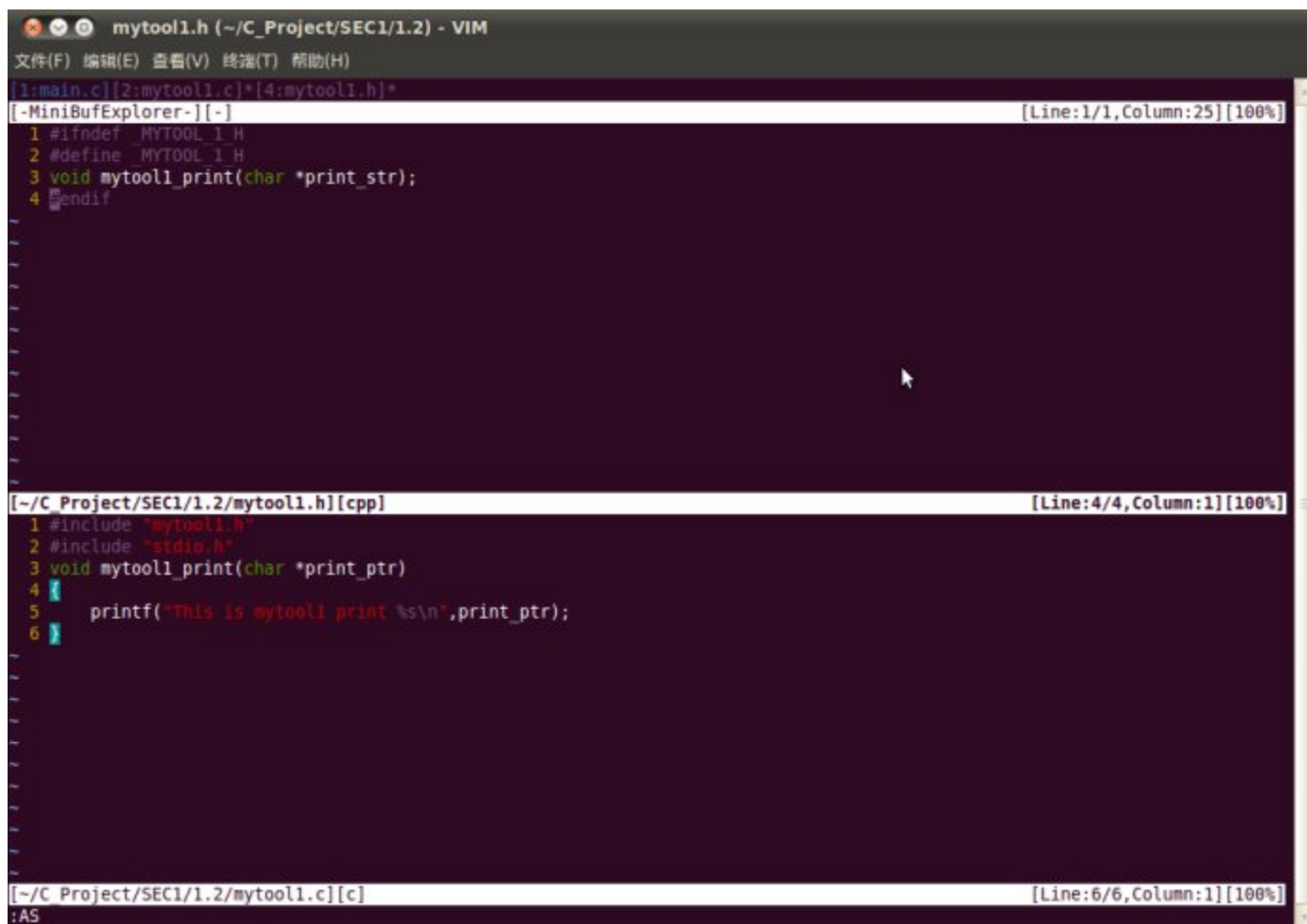
“-- A setting --

nnoremap <silent><F12>:A<CR>

A 安装完成。

常用命令

: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



```
mytool1.h (~/.C_Project/SEC1/1.2) - VIM
文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)
[1:main.c][2:mytool1.c][4:mytool1.h]*
[-MiniBufExplorer-][-] [Line:1/1,Column:25][100%]
1 #ifndef MYTOOL1_H
2 #define MYTOOL1_H
3 void mytool1_print(char *print_str);
4 #endif

[~/C_Project/SEC1/1.2/mytool1.h][cpp] [Line:4/4,Column:1][100%]
1 #include "mytool1.h"
2 #include "stdio.h"
3 void mytool1_print(char *print_ptr)
4 {
5     printf("This is mytool1 print %s\n",print_ptr);
6 }

[~/C_Project/SEC1/1.2/mytool1.c][c] [Line:6/6,Column:1][100%]
:AS
```

10. c-support 安装与配置

C.vim 即 C-Support vim，能够帮助 C 程序员很好的完成一些重复性的工作，节约时间，并保护你的键盘。插件作者 Fritz Mehner，编写 c.vim 的宗旨是“Write and run programs. Insert statements, idioms, comments”。

安装步骤：

- 1) 从 http://www.vim.org/scripts/download_script.php?src_id=9679 下载安装包。
- 2) 将 cvim 安装包解压，把解压后的文件夹复制到在 ~/.vim 目录，提示重名时选择合并。
- 3) 在 .vimrc 文件中进行配置。

-- c-support setting --

在 .vimrc 中用命令 filetype plugin on 激活插件。

\$ vim ~/.vim/c-support/templates/Templates 可改变头说明

快捷键帮助信息详见 doc 的 pdf 文档

C-Support 安装完成。

gqyang@gqyang-ubuntu: ~/C_Project/SEC1/1.2

文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

```
1 /*
2 * =====
3 *
4 *      Filename:  test.c
5 *
6 *      Description:
7 *
8 *      Version:   1.0
9 *      Created:   2013年 11月 20日  23时 01分 27秒
10 *      Revision:  none
11 *      Compiler:  gcc
12 *
13 *      Author:    gqyang (PE), guoqing-yang@163.com
14 *      Company:   NanShan University
15 *
16 * =====
17 */
18
19
```

test.c + (~/C_Project/SEC1/1.2) - VIM

文件(F) 编辑(E) 查看(V) 终端(T) 帮助(H)

```
10 *      Revision:  none
11 *      Compiler:  gcc
12 *
13 *      Author:    gqyang (PE), guoqing-yang@163.com
14 *      Company:   NanShan University
15 *
16 * =====
17 */
18
19 #include  **
20
21 #if  PI
22
23 #else      /* -----  not PI  ----- */
24
25 #endif      /* -----  not PI  ----- */
26 int
27 main ( int argc, char *argv[] )
28 {
29
30     while ( ) {
31     }
32     return EXIT_SUCCESS;
33 }          /* -----  end of function main  ----- */
34 void
35 abc ( )
36 {
37     return ;
38 }          /* -----  end of function abc  ----- */
39
```

[~/C_Project/SEC1/1.2/test.c][c][+]

[Line:22/39,Column:1][56%]

-- 插入 --