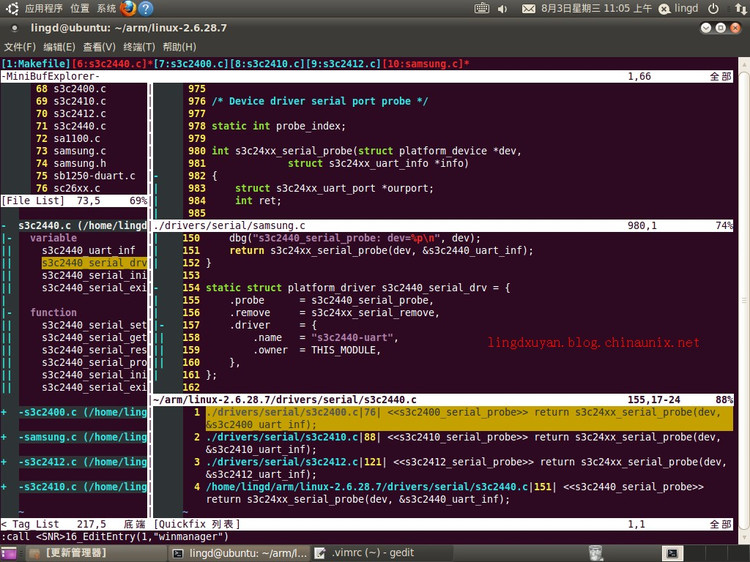
**一步步将vim改造成C/C++开发环境(IDE)**

我的vim IDE界面：



1、安装Vim和Vim基本插件  
首先安装好Vim和Vim的基本插件。这些使用apt-get安装即可：  
lingd@ubuntu:~/arm$sudo apt-get install vim vim-scripts vim-doc  
其中vim-scripts是vim的一些基本插件，包括语法高亮的支持、缩进等等。  
vim中文帮助文档tar包下载地址:  
http://sourceforge.net/projects/vimcdoc/files/vimcdoc/  
解压后其中有个doc文件夹, 将其中的内容全部复制到~/.vim/doc, 或者vim安装目录下的doc目录中, 此时vim中的help信息已经是中文的了.  
网页版中文帮助文档网址http://vimcdoc.sourceforge.net/doc/help.html  
首页就时vim帮助文档的目录，阅读起来更方便有效、更有针对性！  
   
2、Vim配置文件  
Vim强大的功能，其来源基本上就两个地方：Vim插件以及Vim配置文件。  
Vim 本身的系统配置文件夹是在/usr/share/vim/和/etc/vim/两个文件夹下。一般情况下，我们不会去改变这两个文件夹下的配置文件，而是 在用户文件夹/home/user（其中，user为用户名，我的用户名是lingd）下建立自己的配置文件。进入用户文件夹（/home/user/） 之后，用gedit新建一个名叫.vimrc的文件：  
lingd@ubuntu:~/arm$ cd ~  
lingd@ubuntu:~$ gedit .vimrc  
注：使用gedit主要是为了方便大段大段的文字粘贴！  
然后把下面的文字拷贝进这个文件之后保存：

" This line should not be removed as it ensures that various options are  
" properly set to work with the Vim-related packages available in Debian.  
debian.vim  
  
" Uncomment the next line to make Vim more Vi-compatible  
" NOTE: debian.vim sets 'nocompatible'. Setting 'compatible' changes numerous  
" options, so any other options should be set AFTER setting 'compatible'.  
set nocompatible  
  
" Vim5 and later versions support syntax highlighting. Uncommenting the  
" following enables syntax highlighting by default.  
if has("syntax")  
syntax on " 语法高亮  
endif  
colorscheme ron " elflord ron peachpuff default 设置配色方案，vim自带的配色方案保存在/usr/share/vim/vim72/colors目录下  
  
" detect file type  
filetype on  
filetype plugin on  
  
" If using a dark background within the editing area and syntax highlighting  
" turn on this option as well  
set background=dark  
  
" Uncomment the following to have Vim jump to the last position when  
" reopening a file  
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  
endif  
  
" The following are commented out as they cause vim to behave a lot  
" differently from regular Vi. They are highly recommended though.  
  
"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 autowrite " 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 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 tabstop=4 " 设置制表符(tab键)的宽度  
set softtabstop=4 " 设置软制表符的宽度   
set shiftwidth=4 " (自动) 缩进使用的4个空格  
set cindent " 使用 C/C++ 语言的自动缩进方式  
set cinoptions={0,1s,t0,n-2,p2s,(03s,=.5s,>1s,=1s,:1s "设置C/C++语言的具体缩进方式  
"set backspace=2 " 设置退格键可用  
set showmatch " 设置匹配模式，显示匹配的括号  
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  
"set hidden " Hide buffers when they are abandoned  
set mouse=a " Enable mouse usage (all modes) "使用鼠标  
set number " Enable line number "显示行号 "set previewwindow " 标识预览窗口 set history=50 " set command history to 50 "历史记录50条 "--状态行设置-- 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。 "--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 "--find setting-- set incsearch " 输入字符串就显示匹配点 set hlsearch

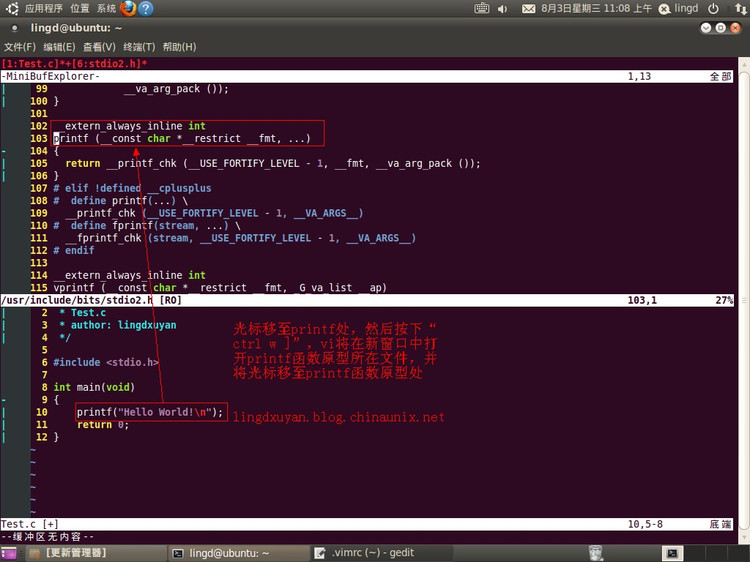
注：配置文件中，以单个双引号开头的文字为注释。  
保存文件之后，启动Vim。此时，Vim已经是这种效果了（语法高亮挺漂亮的–这个是由vim-scripts中的插件支持的）：  
  
3、ctags安装与配置  
ctags可以建立源码树的标签索引（标签就是一个标识符被定义的地方，如函数定义），使程序员在编程时能迅速定位函数、变量、宏定义等位置去查看原形  
以下是在ubuntu下ctags的下载安装和配置过程：  
下载并安装ctags，终端输入命令  
lingd@ubuntu:~/arm$ sudo apt-get install ctags  
建立源码索引，比如我经常需要查阅Linux的内核代码，而这些代码放在/home/lingd/arm/linux-2.6.24.7目录下  
那么在终端进入到该目录后，输入命令ctags -R \*，你会发现多了一个tags文件，这个就是索引文件  
lingd@ubuntu:~/arm$ cd linux-2.6.24.7  
lingd@ubuntu:~/arm/linux-2.6.24.7$ ls  
arch     crypto         include  kernel       mm              samples   usr  
block    Documentation  init     lib          net             scripts  
COPYING  drivers        ipc      MAINTAINERS  README          security  
CREDITS  fs             Kbuild   Makefile     REPORTING-BUGS  sound  
lingd@ubuntu:~/arm/linux-2.6.24.7$ ctags -R \*  
lingd@ubuntu:~/arm/linux-2.6.24.7$ ls  
arch     crypto         include  kernel       mm              samples   tags  
block    Documentation  init     lib          net             scripts   usr  
COPYING  drivers        ipc      MAINTAINERS  README          security  
CREDITS  fs             Kbuild   Makefile     REPORTING-BUGS  sound  
向vim注册索引文件tags的路径，  
lingd@ubuntu:~/arm/linux-2.6.24.7$ vi ~/.vimrc  
在打开文件的最后添加如下内容(当然，具体路径根据你自己的情况)

"--ctags setting--  
" 按下F5重新生成tag文件，并更新taglist  
map <F5> :!ctags -R --c++-kinds=+p --fields=+iaS --extra=+q .<CR><CR> :TlistUpdate<CR>  
imap <F5> <ESC>:!ctags -R --c++-kinds=+p --fields=+iaS --extra=+q .<CR><CR> :TlistUpdate<CR>  
set tags=tags  
set tags+=./tags "add current directory's generated tags file  
set tags+=~/arm/linux-2.6.24.7/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)

set tags+=./tags表示在当前工作目录下搜索tags文件  
set tags+=~/arm/linux-2.6.24.7/tags表示在搜寻tags文件的时候，也要搜寻~/arm/linux-2.6.24.7/文件夹下的tags文件。  
然后保存并退出vi。这样，你就可以用vim在任意地方查看有关Linux的函数原形  
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－  
tag命令用法：  
Ctrl＋］  跳到当前光标下单词的标签  
Ctrl＋O  返回上一个标签  
Ctrl＋T  返回上一个标签  
:tag TagName 跳到TagName标签  
以上命令是在当前窗口显示标签，当前窗口的文件替代为包标签的文件，当前窗口光标跳到标签位置。如果不希望在当前窗口显示标签，可以使用以下命令：  
:stag TagName 新窗口显示TagName标签，光标跳到标签处  
Ctrl＋W + ］  新窗口显示当前光标下单词的标签，光标跳到标签处  
当一个标签有多个匹配项时（函数 (或类中的方法) 被多次定义），":tags" 命令会跳转到第一处。如果在当前文件中存在匹配，那它将会被首先使用。  
可以用这些命令在各匹配的标签间移动：  
:tfirst    到第一个匹配  
:[count]tprevious 向前 [count] 个匹配  
:[count]tnext  向后 [count] 个匹配  
:tlast    到最后一个匹配  
或者使用以下命令选择要跳转到哪一个  
:tselect TagName  
输入以上命令后，vim会为你展示一个选择列表。然后你可以输入要跳转到的匹配代号 (在第一列)。其它列的信息可以让你知道标签在何处被定义过。  
以下命令将在预览窗口显示标签  
:ptag TagName 预览窗口显示TagName标签，光标跳到标签处  
Ctrl＋W + }  预览窗口显示当前光标下单词的标签，光标跳到标签处  
:pclose   关闭预览窗口  
:pedit file.h 在预览窗口中编辑文件file.h（在编辑头文件时很有用）  
:psearch atoi 查找当前文件和任何包含文件中的单词并在预览窗口中显示匹配，在使用没有标签文件的库函数时十分有用。   
   
最简单的使用方法举例  
用vi在任意目录写一个Test.c文件，内容如下:

int main(void)  
{  
printf("Hello World!\n");  
return 0;  
}

写好后末行模式输入w保存好（不要退出vi），按Esc回到指令模式，把光标停留在printf上  
然后按 Ctrl＋W + ］，vi会自动跳到Linux系统函数printf()处，这时我们能查看printf()的原形  
查看完了，按Ctrl＋o(回到上一个标签) 就回到原来的地方

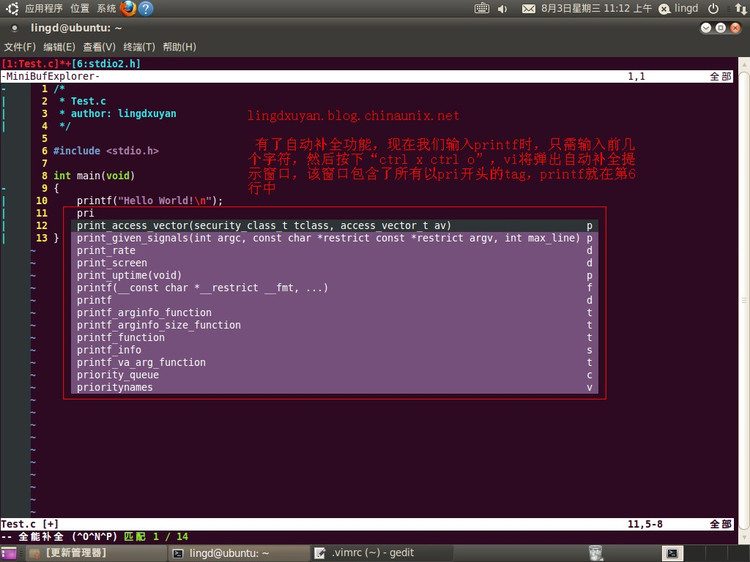
  
4、管理vim插件——vim-addons  
通 过vim-addons，我们可以管理vim插件。我们在sudo apt-get install vim vim-scripts vim-doc时，一般会自动安装上vim-addons。若未安装可通过sudo apt-get install vim-addon-manager手动安装。安装完成后，就可以用vim-addons管理vim插件了。  
# 系统中已有的vim-scripts中包含的插件及其状态：  
lingd@ubuntu:~$ vim-addons status  
# Name                     User Status  System Status  
align                       removed       removed        
alternate                   removed       removed        
bufexplorer                 removed       removed        
calendar                    removed       removed        
closetag                    removed       removed        
colors sampler pack         removed       removed        
cvsmenu                     removed       removed        
debPlugin                   removed       removed        
detectindent                removed       removed        
doxygen-toolkit             removed       removed        
editexisting                removed       removed        
enhanced-commentify         removed       removed        
gnupg                       removed       removed        
info                        removed       removed        
justify                     removed       removed        
lbdbq                       removed       removed        
markdown-syntax             removed       removed        
matchit                     removed       removed        
minibufexplorer             installed     removed        
nerd-commenter              removed       removed        
omnicppcomplete             installed     removed        
po                          removed       removed        
project                     installed     removed        
python-indent               removed       removed        
secure-modelines            removed       removed        
snippetsEmu                 removed       removed        
sokoban                     removed       removed        
supertab                    removed       removed        
surround                    removed       removed        
taglist                     installed     removed        
tetris                      removed       removed        
utl                         removed       removed        
vcscommand                  removed       removed        
vimplate                    removed       removed        
whatdomain                  removed       removed        
winmanager                  removed       removed        
xmledit                     removed       removed      
上 面我们介绍了如何独立于系统配置文件之外，建立自己的Vim配置文件。当我们自己下载安装Vim插件的时候，也可以另外建立目录，放置我们自己的插件。这 个目录一般为/home/user/.vim，另外还需要建立一个插件子目录，一个插件文档子目录，以上的可以进入/home/user目录下通过下面的 命令执行：  
lingd@ubuntu:~$ mkdir .vim  
lingd@ubuntu:~$ cd .vim  
lingd@ubuntu:~/.vim$ mkdir plugin  
lingd@ubuntu:~/.vim$ mkdir doc   
# vim官方插件的安装，xxxx是要安装的插件名，以status中显示的名称为准。安装插件xxxx时使用以下命令（前提是在目录/home/user/.vim/下建立好了plugin和doc两个文件夹）  
vim-addons install xxxx  
 关于vim-addons命令的详细用法，可以通过“man vim-addons”查看其帮助文档  
  
5、vim自动补全——OmniCppComplete  
vim的自动补全功能可通过其插件OmniCppComplete实现。  
安装OmniCppComplete  
lingd@ubuntu:~$ vim-addons install omnicppcomplete  
配置OmniCppComplete  
在vim配置文件/home/user/.vimrc中加入如下的配置：

"-- omnicppcomplete setting --  
" 按下F3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 <F3> <C-X><C-O>  
" 按下F2根据头文件内关键字补全  
imap <F2> <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

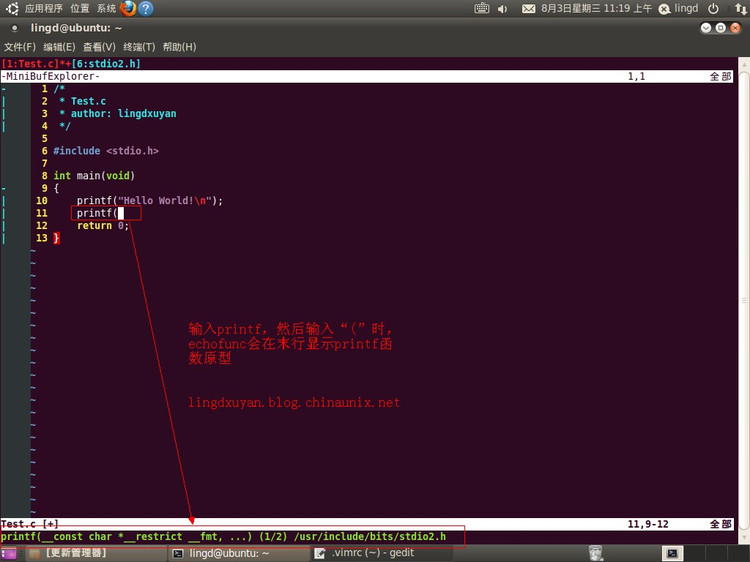
（前几行就是提供了C++中的./->/::等操作符的提示和自动完成）。  
OmniCppComplete 是基于ctags数据库即tags文件实现的(基于ctags生成的索引信息来实现自动补全的)，所以在ctags -R生成tags时还需要一些额外的选项，这样生成的tags文件才能与OmniCppComplete配合运作。使用下列命令生成tags文件，就可以 与OmniCppComplete配合运作：  
ctags -R --c++-kinds=+p --fields=+iaS --extra=+q .  
--c++-kinds=+p  : 为C++文件增加函数原型的标签  
--fields=+iaS   : 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  
4.4  4.4.3  
# 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  
4.4  4.4.3  tags  
# 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

# 用vi打开前面的Test.c文件，在printf("Hello World!\n")下一行中，输入pri，然后按下Ctrl+X Ctrl+O，此时vi会弹出一个窗口，所有以pri开头的tag都会出现在这个窗口中，printf就出现在第6行中  
lingd@ubuntu:~$ cd ~  
lingd@ubuntu:~$ vim Test.c



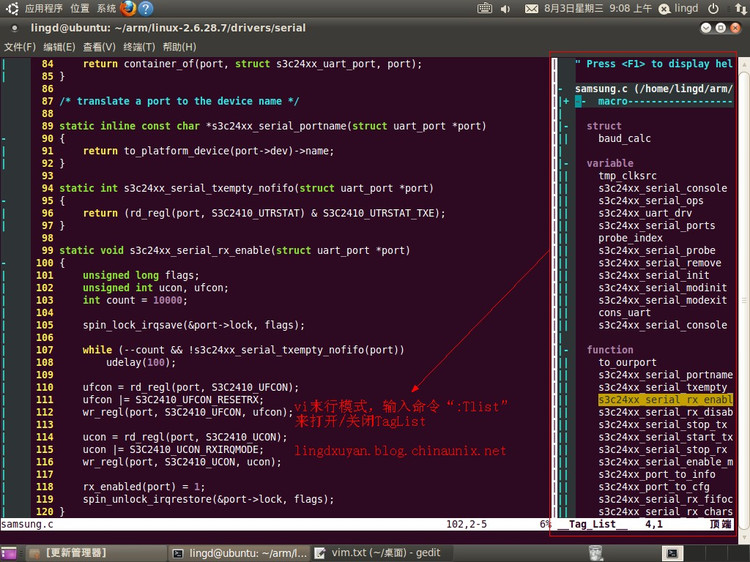
注 意：在自动补全的点，Vim必须知道可能补全的定义。比如说，在namespace std命名空间下的变量和函数，必须要用using namespace std;暴露出来，否则是不能补全的。在.cpp文件中还可以，在.h文件中这样就不是好的做法了。暂时不知道这个问题是由于我自己配置错误还是程序没有 实现。  
当自动补全下拉窗口弹出后，一些可用的快捷键:  
Ctrl+P  向前切换成员  
Ctrl+N  向后切换成员  
Ctrl+E  表示退出下拉窗口, 并退回到原来录入的文字  
Ctrl+Y  表示退出下拉窗口, 并接受当前选项  
其他补全方式:  
Ctrl+X Ctrl+L 整行补全  
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  
帮助文档  
:help omnicppcomplete  
   
6、提示函数原型echofunc  
echofunc可以在命令行中提示当前输入函数的原型。  
echofunc下载地址：http://www.vim.org/scripts/script.php?script\_id=1735  
下载完成后，把echofunc.vim文件放到 ~/.vim/plugin文件夹中  
当 你在vim插入(insert)模式下紧接着函数名后输入一个"("的时候, 这个函数的声明就会自动显示在命令行中。如果这个函数有多个声明, 则可以通过按键"Alt+-"和"Alt+="向前和向后翻页, 这个两个键可以通过设置g:EchoFuncKeyNext和g:EchoFuncKeyPrev参数来修改。这个插件需要tags文件的支持, 并且在创建tags文件的时候要加选项"--fields=+lS"（OmniCppComplete创建的tag文件也能用）, 整个创建tags文件的命令如下:  
$ ctags -R --fields=+lS  
其他插件说明详见echofunc.vim



 如果你在编译vim时加上了"+balloon\_eval"特性，那么当你把鼠标放在函数名上的时候会有一个tip窗口弹出, 该窗口中也会有函数的声明  
  
7、标签浏览器Taglist  
Taglist用于列出了当前文件中的所有标签（宏, 全局变量, 函数名等）  
安装Taglist  
lingd@ubuntu:~$ vim-addons install taglist  
配置Taglist  
在vim配置文件/home/user/.vimrc中加入如下的配置：

"-- 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=1 "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

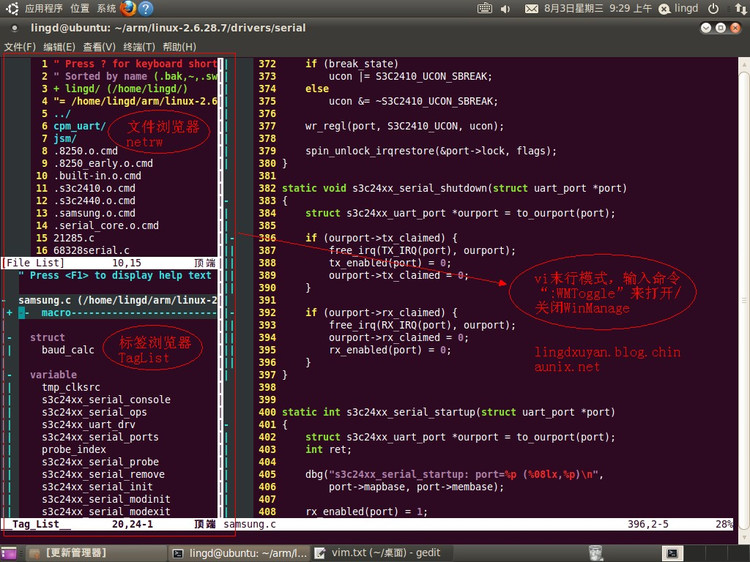
进入vim后用命令":Tlist"打开/关闭taglist窗口  
帮助文档  
:help taglist.txt



 8、文件浏览器和缓冲区管理器WinManager  
WinManager用于管理文件浏览器和缓冲区（buffer）。2.0以上版本的WinManager还可以管理其他IDE类型插件，不过要用户在插件中增加一些辅助变量和hook来支持WinManager（帮助文档有相关说明）。  
Taglist 插件本身就提供了对WinManager的支持，不需要我们去修改它。这里，我们就用WinManager来管理文件浏览器netrw和标签浏览器 Taglist。netrw是标准的vim插件, 已经随vim一起安装进系统里了, 不需要我们自行下载安装。  
安装WinManager  
lingd@ubuntu:~$ vim-addons install winmanager  
配置WinManager  
在vim配置文件/home/user/.vimrc中加入如下的配置：

"-- 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>

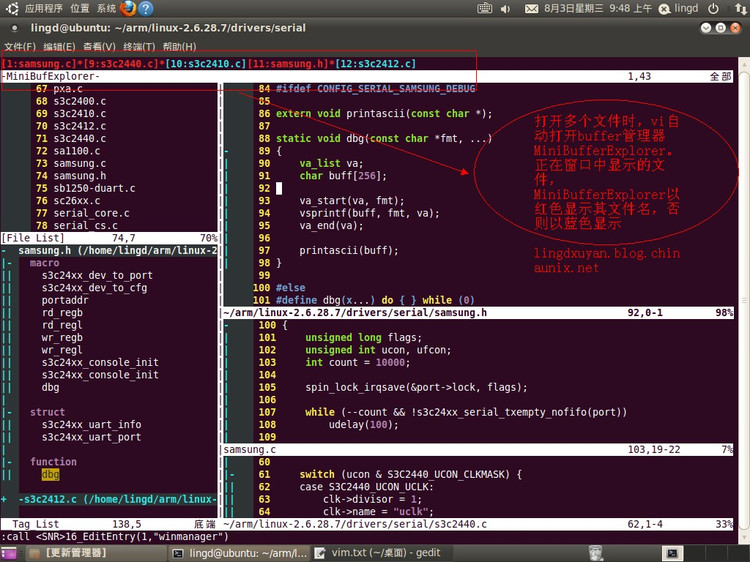
常用命令  
:WMToggle 打开/关闭WinManage，不过我们在配置文件.vimrc中做了快捷键映射，所以直接按wm就可以打开/关闭WinManage  
文件浏览器命令（在文件浏览器窗口中使用）  
<enter>或双击  如果光标下是目录, 则进入该目录; 如果光标下文件, 则打开该文件  
<tab>   如果光标下是目录, 则进入该目录; 如果光标下文件, 则在新窗口打开该文件  
<F5> 刷新列表  
-  返回上一层目录  
c  使浏览目录成为vim当前工作目录  
d  创建目录  
D  删除当前光标下的目录或文件  
i  切换显示方式  
R  文件或目录重命名  
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  
winmanager帮助文档  
:help winmanager  
netrw帮助文档  
:help netrw



 9、buffer管理器MiniBufferExplorer  
MiniBufferExplorer 用于浏览和管理buffer，如果只打开一个文件，是不会显示在屏幕上的，而打开多个文件之后，会自动出现在屏幕上。vim也有自带的buffer管理工 具，不过只有:ls, :bnext, :bdelete 等的命令, 既不好用, 又不直观.  
关于vim缓冲区（buffer）和窗口的概念（详见:help windows）  
"缓冲区" 是一块内存区域，里面存储着正在编辑的文件。如果没有把缓冲区里的文件存盘，那么原始文件不会被更改。  
"窗口" 被用来查看缓冲区里的内容。你可以用多个窗口观察同一个缓冲区，也可以用多个窗口观察不同的缓冲区。  
"屏幕" Vim 所用的整个工作区域，可以是一个终端模拟窗口，也被叫做 "Vim 窗口"。一个屏幕包含一个或多个窗口，被状态行和屏幕底部的命令行分割。  
        +-------------------------------+  
屏幕    | 窗口 1        | 窗口 2        |  
        |               |               |  
        |               |               |  
        |=== 状态行  ===|==== 状态行 ===|  
        | 窗口 3                        |  
        |                               |  
        |                               |  
        |========== 状态行 =============|  
        |命令行                         |  
        +-------------------------------+  
安装MiniBufferExplorer  
lingd@ubuntu:~$ vim-addons install minibufexplorer  
配置MiniBufferExplorer  
在vim配置文件/home/user/.vimrc中加入如下的配置：

" -- MiniBufferExplorer --  
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

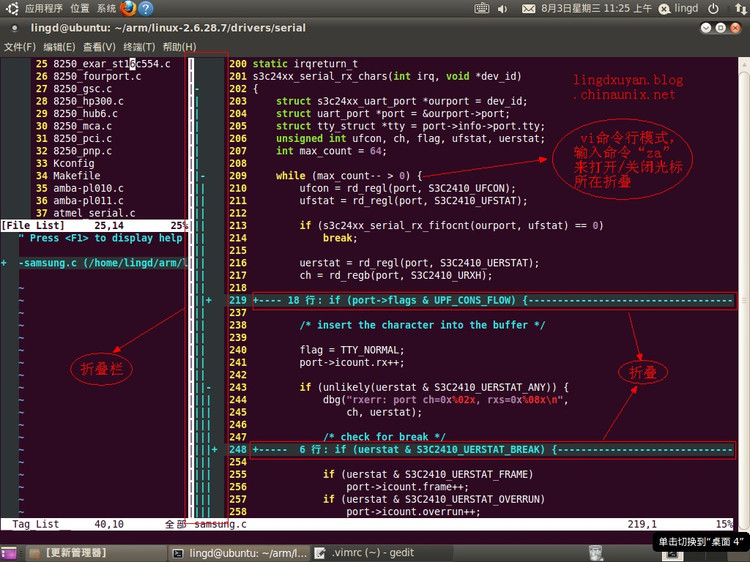
常用命令  
<Tab>  移到上一个buffer  
<Shift-Tab> 移到下一个buffer  
<Enter>  打开光标所在的buffer  
d   删除光标所在的buffer



 10、代码折叠fold  
折叠用于把缓冲区内某一范围内的文本行显示为屏幕上的一行。就像一张纸，要它缩短  
些，可以把它折叠起来:  
 +------------------------+  
 | 行 1                   |  
 | 行 2                   |  
 | 行 3                   |  
 |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  
 \                        \  
  \\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\  
  / 被折叠的行             /  
 /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/  
 | 行 12                  |  
 | 行 13                  |  
 | 行 14                  |  
 +------------------------+  
那些文本仍然在缓冲区内而没有改变。受到折叠影响的只是文本行显示的方式。  
折叠的好处是，通过把多行的一节折叠成带有折叠提示的一行，会使你更好地了解对文本  
的宏观结构。  
折叠方式foldmethod  
vim提供以下6种方法来选定折叠方式：  
manual 手工定义折叠  
indent 更多的缩进表示更高级别的折叠  
expr 用表达式来定义折叠  
syntax 用语法高亮来定义折叠  
diff 对没有更改的文本进行折叠  
marker 对文中的标志折叠  
折叠级别foldlevel  
'foldlevel' 是个数值选项：数字越大则打开的折叠更多。  
当 'foldlevel' 为 0 时，所有的折叠关闭。  
当 'foldlevel' 为正数时，一些折叠关闭。  
当 'foldlevel' 很大时，所有的折叠打开。  
折叠栏foldcolumn  
'foldcolumn' 是个数字，它设定了在窗口的边上表示折叠的栏的宽度。当为0时，没有折叠栏。最大是12。  
一个打开的折叠由一栏来表示，顶端是 '-'，其下方是 '|'。这栏在折叠结束的地方结束。当折叠嵌套时，嵌套的折叠出现在被包含的折叠右方一个字符位置。  
一个关闭的折叠由 '+' 表示。  
当折叠栏太窄而不能显示所有折叠时，显示一数字来表示嵌套的级别。  
在折叠栏点击鼠标，可以打开和关闭折叠：  
- 点击 '+' 打开在这行的关闭折叠  
- 在任何其他非空字符上点击，关闭这行上的打开折叠  
在vim配置文件/home/user/.vimrc中加入如下的配置：

"--fold setting--  
set foldmethod=syntax " 用语法高亮来定义折叠  
set foldlevel=100 " 启动vim时不要自动折叠代码  
set foldcolumn=5 " 设置折叠栏宽度

常用命令  
za  打开/关闭在光标下的折叠  
zA  循环地打开/关闭光标下的折叠  
zo  打开 (open) 在光标下的折叠  
zO  循环打开 (Open) 光标下的折叠  
zc  关闭 (close) 在光标下的折叠  
zC  循环关闭 (Close) 在光标下的所有折叠  
zM  关闭所有折叠  
zR  打开所有的折叠  
帮助文档  
:help usr\_28.txt  
:help fold.txt

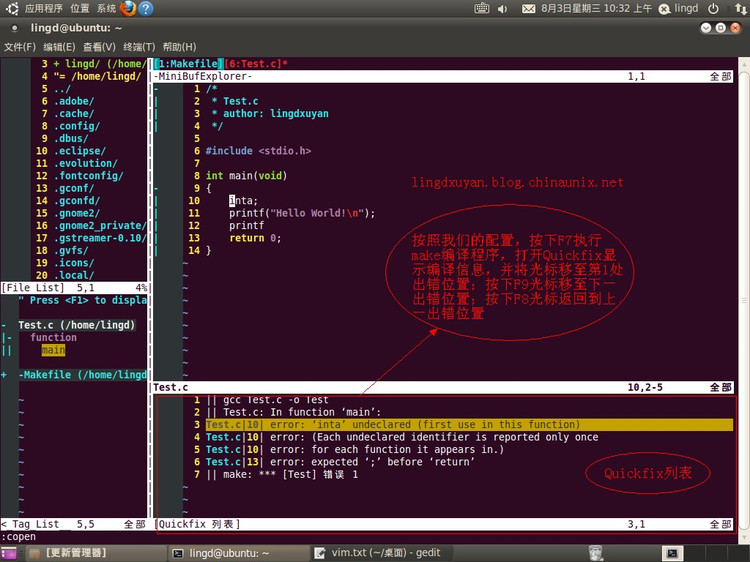


11、项目目录数管理器Project  
Project插件是用来显示项目的目录树的，这个目录树是默认保存在~/.vimprojects文件中。  
安装Project  
[lingd@ubuntu:~$](mailto:lingd@ubuntu:%7E$) vim-addons install project   
Project目录树可以通过下面的步骤生成：  
1） 打开vim在命令模式下输入 :Project，在屏幕的最左边就会出现一个project框。不过因为没有初始化暂时是空的  
2）在命令模式下（不是插入模式）输入\C （大写的C），会出现下面这些信息：  
Enter the Name of the Entry: xxxx （输入项目名称）  
Enter the Absolute Directory to Load: /xxx/xxx/xxx （输入项目根目录的绝对路径）  
Enter the CD parameter: . （“.”为当前目录）或者和项目根目录一致  
Enter the File Filter: \*.\* （符合条件的源文件，可以是\*.cpp/\*.h等）  
PS：项目目录可以嵌套。而且更改之后在~/.vimprojects文件中就能看到内容，你可以手动进行更改。

12、quickfix命令集  
通过quickfix命令集，你可在 Vim 内编译程序并直接跳转到出错位置进行修正。你可以接着重新编译并做修正，直到不再出错为止。  
在vim配置文件/home/user/.vimrc中加入如下的配置：

"-- QuickFix setting --  
" 按下F6，执行make clean  
map <F6> :make clean<CR><CR><CR>  
" 按下F7，执行make编译程序，并打开quickfix窗口，显示编译信息  
map <F7> :make<CR><CR><CR> :copen<CR><CR>  
" 按下F8，光标移到上一个错误所在的行  
map <F8> :cp<CR>  
" 按下F9，光标移到下一个错误所在的行  
map <F9> :cn<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 <F7> <ESC>:make<CR><CR><CR> :copen<CR><CR>  
imap <F8> <ESC>:cp<CR>  
imap <F9> <ESC>:cn<CR>

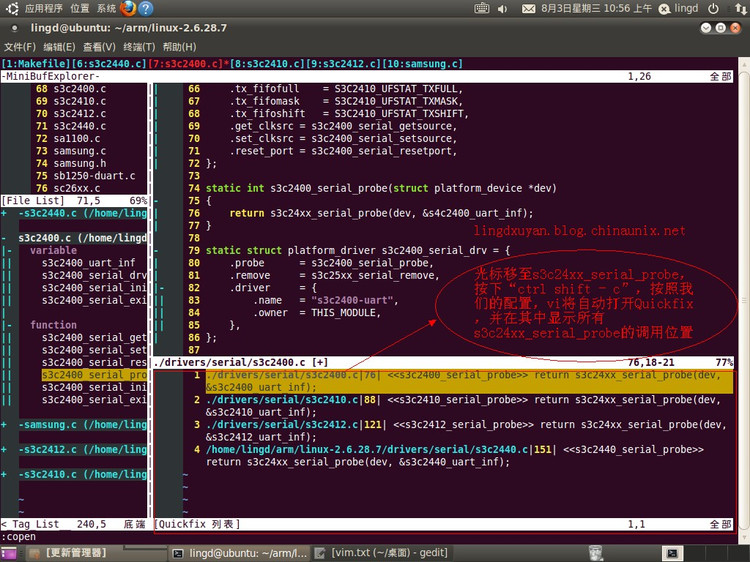
帮助文档  
:help usr\_30  
:help quickfix  
下面的命令运行 "make" (包括你所给出的参数) 程序并捕捉其运行结果: >  
 :make {arguments}  
如果编译时出现错误，按 <Enter>，回到vim界面，看不到出错信息了！这时，可以运行以下命令  
:cw[indow]  
打开quickfix窗口来查看出错信息，它会自动跳到第一处出错的地方。然后，你可以双击出错某一条出错信息，vim就会自动跳转到相应的出错位置  
:cn[ext]  光标移到下一个错误所在的行  
:cp[revious] 光标移到上一个错误所在的行  
:cfirst  到第一处错误  
:clast  到最后一处错误  
:cc   空间不够时，Vim 会缩短出错信息。如果你想查看详细信息，可以使用此命令  
:cl[ist] 列出所有出错信息的概览（只有那些含有文件名或行数的错误信息会被显示，需要查看那些并不含文件名或行数的信息可用“:cl[ist]!”命令）



 13、Cscope  
Cscope是一个类似于ctags的工具，不过其功能比ctags强大很多。  
安装Cscope  
lingd@ubuntu:~$ sudo apt-get install cscope  
在vim配置文件/home/user/.vimrc中加入如下的配置：

"-- Cscope setting --  
if has("cscope")  
set csprg=/usr/bin/cscope " 指定用来执行cscope的命令  
set csto=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  
map <F4> :cs add ./cscope.out .<CR><CR><CR> :cs reset<CR>  
imap <F4> <ESC>:cs add ./cscope.out .<CR><CR><CR> :cs reset<CR>  
" 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>

cscope的主要功能是通过其"find"子命令来实现的  
"cscope find"的用法:  
cs find c|d|e|f|g|i|s|t name  
0 或 s  查找这个 C 符号(可以跳过注释)  
1 或 g  查找这个定义  
2 或 d  查找这个函数调用的函数  
3 或 c  查找调用过这个函数的函数  
4 或 t  查找这个字符串  
6 或 e  查找这个 egrep 模式  
7 或 f  查找这个文件  
8 或 i  查找包含这个文件的文件  
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 cscope.out"（该命令已被我们映射为快捷键F4），添加cscope数据库到vim。因为我们已将vim配置为启动时，自动添加当前目录下的 cscope数据库，所以你再添加该cscope数据库时，vim会提示“重复cscope数据库 未被加入“  
<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）  
   
帮助文档  
:help if\_cscop  
注意：帮助文档if\_cscop中，快捷键映射nmap <C-\_>i :cs find i ^<C-R>=expand("<cfile>")<CR>$<CR>是有误的！  
比 如光标“header.h”上按下<C-\_>i，也就是查找包含“header.h“的文件。但是按照这个映射规则，则是将<C- \_>i映射为cs find i ^header.h$，也就是查找包含“^header.h$“的文件，这显然不是我们想要的结果。该映射规则应该改成nmap <C-\_>i :cs find i <C-R>=expand("<cfile>")<CR><CR>



 14、vim常用快捷键  
%  跳转到配对的括号去  
[[  跳转到当前或者上一代码块(函数定义、类定义等)的开头去(但要求代码块中'{'必须单独占一行)  
][  跳转到当前代码块(函数定义、类定义等)的结尾去(但要求代码块中'{'必须单独占一行)  
]]  跳转到下一代码块(函数定义、类定义等)的开头去(但要求代码块中'{'必须单独占一行)  
[/  跳到注释开头（只对/\* \*/注释有效）  
]/ 跳到注释结尾（只对/\* \*/注释有效）  
gD  跳转到当前文件内标识符首次出现的位置，可用于跳转到全部变量的定义处；查找局部变量时，要是多个函数中定义了该局部变量，使用gD进行查找，找到的变量定义可能不是你需要的  
gd 跳转到当前函数内标识符首次出现的位置，可用于跳转到局部变量的定义处  
''  跳转到光标上次停靠的地方, 是两个', 而不是一个"  
mx  设置书签,x只能是a-z的26个字母  
`x  跳转到书签处("`"是1左边的键)  
>  增加缩进,"x>"表示增加以下x行的缩进  
<  减少缩进,"x<"表示减少以下x行的缩进