1. 安装vim

在centos7环境下，源码编译安装vim需要安装编译环境：

$ yum install ncurses-devel

$ yum install gcc

在vim官网下载源码包并解压：

$ cd vim74/src

$ ./configure --with-features=huge --enable-rubyinterp --enable-pythoninterp --enable-luainterp --enable-multibyte --enable-cscope --enable-gui=gtk2

$ make

$ make install

在./configure时，可以配置参数，参数如下：

* --with-features=huge：支持最大特性
* --enable-rubyinterp：启用Vim对ruby编写的插件的支持
* --enable-pythoninterp：启用Vim对python编写的插件的支持
* --enable-luainterp：启用Vim对lua编写的插件的支持
* --enable-perlinterp：启用Vim对perl编写的插件的支持
* --enable-multibyte：多字节支持 可以在Vim中输入中文
* --enable-cscope：Vim对cscope支持
* --enable-gui=gtk2：gtk2支持,也可以使用gnome，表示生成gvim
* --with-python-config-dir=/usr/lib/python2.7/config-i386-linux-gnu/ 指定 python 路径
* --prefix=/usr：编译安装路径

如需要重新配置可 输入 make distclean #清理一下上一次编译生成的所有文件。

1. vim的一些基础设置：

前提：在源码编译安装vim时，可能会与原有的vi产生一些问题。

1、若遇到vim退格键(Backspace)不能用

先去掉讨厌的有关vi一致性模式，避免以前版本的一些bug和局限

set nocompatible

backspace有几种工作方式，默认是vi兼容的。对新手来说很不习惯。对老vi 不那么熟悉的人也都挺困扰的。可以用

set backspace=indent,eol,start

来解决。

indent: 如果用了:set indent,:set ai 等自动缩进，想用退格键将字段缩进的删掉，必须设置这个选项。否则不响应。  
eol:如果插入模式下在行开头，想通过退格键合并两行，需要设置eol。  
start：要想删除此次插入前的输入，需设置这个。

将以上两个命令加到vim的系统配置文件里就可以了。通过vim命令:ver可以看到系统配置文件的位置，一般在/etc/vimrc

2、正常的基础设置

set shiftwidth=4

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 tabstop=4

set nu

if has("syntax")

syntax on " 语法高亮

endif

colorscheme ron "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

"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 smartindent " 智能对齐方式

set softtabstop=4 " 设置软制表符的宽度

"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

下面是网上的参考设置：

" 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）

三、Vim的进阶设置（插件）：

**1、ctags**

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

yum 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。

$ cd 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

$ ctags -R \*

$ 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，

$ 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。

**2、taglist**

下载taglist包：

wget <http://jaist.dl.sourceforge.net/project/vim-taglist/vim-taglist/4.6/taglist_46.zip>

解压并将其分别放在~/.vim/plugin和~/.vim/doc目录中，

plugin/taglist.vim - taglist插件

doc/taglist.txt     - taglist帮助文件

使用下面的命令生成帮助标签（下面的操作在VIM中进行）：

:helptags ~/.vim/doc

生成帮助标签后，你就可以用下面的命令查看taglist的帮助了：

:help taglist.txt

vimrc的配置如下：

"-------------------------taglist------------------------

"-- 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

let Tlist\_Auto\_Open = 1

let Tlist\_Exit\_OnlyWindow=1 "如果taglist窗口是最后一个窗口，则退出vim

"let Tlist\_WinHeight = 100

"let Tlist\_WinWidth = 40

下面介绍常用的taglist配置选项，可以根据自己的习惯进行配置：

1、 Tlist\_Ctags\_Cmd选项用于指定你的Exuberant ctags程序的位置，如果它没在你PATH变量所定义的路径中，需要使用此选项设置一下；

2、如果你不想同时显示多个文件中的tag，设置Tlist\_Show\_One\_File为1。缺省为显示多个文件中的tag；

3、设置Tlist\_Sort\_Type为”name”可以使taglist以tag名字进行排序，缺省是按tag在文件中出现的顺序进行排序。按tag出现的范围（即所属的namespace或class）排序，已经加入taglist的TODO List，但尚未支持；

4、如果你在想taglist窗口是最后一个窗口时退出VIM，设置Tlist\_Exit\_OnlyWindow为１；

5、如果你想taglist窗口出现在右侧，设置Tlist\_Use\_Right\_Window为１。缺省显示在左侧。

6、在gvim中，如果你想显示taglist菜单，设置Tlist\_Show\_Menu为１。你可以使用Tlist\_Max\_Submenu\_Items和Tlist\_Max\_Tag\_Length来控制菜单条目数和所显示tag名字的长度；

7、缺省情况下，在双击一个tag时，才会跳到该tag定义的位置，如果你想单击tag就跳转，设置Tlist\_Use\_SingleClick为１；

8、如果你想在启动VIM后，自动打开taglist窗口，设置Tlist\_Auto\_Open为1；

9、如果你希望在选择了tag后自动关闭taglist窗口，设置Tlist\_Close\_On\_Select为1；

10、当同时显示多个文件中的tag时，设置Tlist\_File\_Fold\_Auto\_Close为１，可使taglist只显示当前文件tag，其它文件的tag都被折叠起来。

11、在使用:TlistToggle打开taglist窗口时，如果希望输入焦点在taglist窗口中，设置Tlist\_GainFocus\_On\_ToggleOpen为1；

12、如果希望taglist始终解析文件中的tag，不管taglist窗口有没有打开，设置Tlist\_Process\_File\_Always为1；

13、Tlist\_WinHeight和Tlist\_WinWidth可以设置taglist窗口的高度和宽度。Tlist\_Use\_Horiz\_Window为１设置taglist窗口横向显示；

在taglist窗口中，可以使用下面的快捷键：

<CR>    跳到光标下tag所定义的位置，用鼠标双击此tag功能也一样  
o             在一个新打开的窗口中显示光标下tag  
<Space>       显示光标下tag的原型定义  
u             更新taglist窗口中的tag  
s             更改排序方式，在按名字排序和按出现顺序排序间切换  
x             taglist窗口放大和缩小，方便查看较长的tag  
+             打开一个折叠，同zo  
-             将tag折叠起来，同zc  
\*             打开所有的折叠，同zR  
=             将所有tag折叠起来，同zM  
[[            跳到前一个文件  
]]            跳到后一个文件  
q             关闭taglist窗口  
<F1>          显示帮助

可以用“:TlistOpen”打开taglist窗口，用“:TlistClose”关闭taglist窗口。或者使用“:TlistToggle”在打开和关闭间切换。在我的vimrc中定义了下面的映射，使用“,tl”键就可以打开/关闭taglist窗口：

map <silent> <leader>tl :TlistToogle<cr>

Taglist插件还提供了很多Ex命令，你甚至可以用这些命令创建一个taglist的会话，然后在下次进入VIM时加载此会话。

**3、vundle插件管理器**

Vundle是基于Git仓库的插件管理软件。Vundle将插件的安装简化为类似yum软件安装的过程，只要:BundleInstall插件就安装完了，:BundleClean之后插件就卸载了。

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

更新.vimrc配置文件：

set nocompatible " be iMproved

filetype off " required!

" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

call vundle#rc()

" let Vundle manage Vundle

" required!

Bundle 'gmarik/vundle'

" 可以通过以下四种方式指定插件的来源

" a) 指定Github中vim-scripts仓库中的插件，直接指定插件名称即可，插件明中的空格使用“-”代替。

Bundle 'L9'

“ b) 指定Github中其他用户仓库的插件，使用“用户名/插件名称”的方式指定

Bundle 'tpope/vim-fugitive'

Bundle 'Lokaltog/vim-easymotion'

Bundle 'rstacruz/sparkup', {'rtp': 'vim/'}

Bundle 'tpope/vim-rails.git'

" c) 指定非Github的Git仓库的插件，需要使用git地址

Bundle 'git://git.wincent.com/command-t.git'

" d) 指定本地Git仓库中的插件

Bundle 'file:///Users/gmarik/path/to/plugin'

" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

filetype plugin indent on " required!

如果想要安装新的插件，在“call vundle#rc()" 和 ”filetype plugin indent on"之间添加新的Bundle ‘插件名称’即可。

安装插件：打开vim，在命令行里输入BundleInstall即可，如果要卸载插件，同样输入BundleClean即可。

Vundle常用命令：

:BundleList -列举列表(也就是.vimrc)中配置的所有插件

:BundleInstall -安装列表中的全部插件

:BundleInstall! -更新列表中的全部插件

: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

**4、vim智能补全插件YouCompleteMe**

YCM是vim的一款基于语义的智能补全插件。该插件的功能与以下插件相冲突：  
- clang\_complete  
- AutoComplPop  
- Supertab  
- neocomplcache

虽然可以补全很多种编程语言，但YCM主要还是针对c++的，此外它对python的支持也不错。要想尽可能地发挥YCM的功能，推荐安装以下软件：

- vim (尽量新的版本，至少 >= 4.3.7，截至 2013-10-27)  
- git (作为vundle插件的支持)  
- vim 插件 Vundle （参考：[https://github.com/gmarik/vundle](http://jump.bdimg.com/safecheck/index?url=rN3wPs8te/r8jfr8YhogjfUWFoMgIRa80hsyRxjCI9JuYUN99HYivEhdiWJwZ15tmLSLhrX9ajJ0LU4OngLi73fqoP32cyrIQbcgB/YT9hVQ2eIlQjC3YPA92sJ3O3yIiBm084QAE+J2Pah4egTNMg==)）  
- vim 插件 syntastic 与 ListToggle  
- CMake (>=2.8) (为c++项目生成 compile database)  
- pylint (高级python语法检查支持)

安装pylint，首先安装Mercurial（HG）：

wget <https://www.mercurial-scm.org/release/mercurial-3.6.tar.gz>

$ tar –zxvf mercurial-3.6.tar.gz

$ vim Makefile

找到PREFIX=/usr/local，将其中的/usr/local改成实际的安装目录。然后编译安装

$ make install

注：需先安装python-docutils，否则Mercurial的文档会生成失败，报如下错误：

abort: couldn't generate documentation: docutils module is missing  
please install python-docutils or see http://docutils.sourceforge.net/  
make[1]: \*\*\* [hg.1] 错误 255  
make[1]: Leaving directory `/home/nieyong/download/build/mercurial-2.6.1/doc'  
make: \*\*\* [doc] 错误 2

安装python-docutils使用命令：

$ pip install docutils

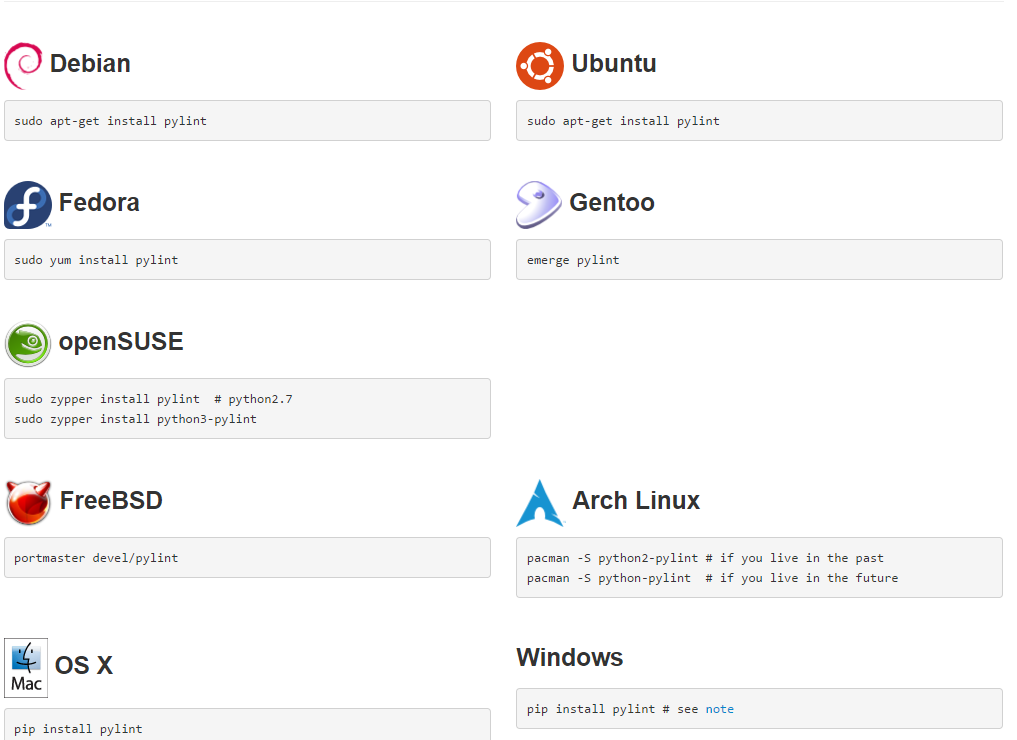
安装结束之后，安装pylint，从源码安装：

$ hg clone https://bitbucket.org/logilab/pylint/

$ hg clone https://bitbucket.org/logilab/astroid

$ mkdir logilab && touch logilab/\_\_init\_\_.py

$ hg clone http://hg.logilab.org/logilab/common logilab/common



YCM语法补全模块的编译：

为了让YCM实现语法补全，还需要编译语法补全模块和编辑一个配置文件。在终端下执行以下命令：  
$HOME/.vim/bundle/YouCompleteMe  
install.sh --clang-completer  
即可安装语法补全模块。

**[重要] 配置文件 .ycm\_extra\_conf.py**这个文件决定了YCM在进行c系语言(c,c++,etc.) 语法补全时的行为。默认的样板配置文件在  
$HOME/.vim/bundle/YouCompleteMe/cpp/ycm/.ycm\_extra\_conf.py  
 对于YCM来说，当打开一个代码文件时，插件将顺着文件所在的路径一直向上查找，如果搜索到第一个配置文件将立刻读入。如果一直搜索到根目录依旧无法找到配置文件，语法补全将不被启用。由此可知，文件所在目录的配置文件优先级最高，根目录的优先级最低。  
 一种比较好的使用方法是在每个项目中创建一个配置文件，或者将项目根据语言进行分类，在每个语言文件夹下建立一个配置文件。  
 默认配置文件是支持c++的，但是需要修改一处地方。可以将该文件拷贝出来并编辑：  
cp $HOME/.vim/bundle/YouCompleteMe/cpp/ycm/.ycm\_extra\_conf.py ~  
vim ~/.ycm\_extra\_conf.py  
 找到以下内容：  
# NOTE: This is just for YouCompleteMe; it's highly likely that your project  
# does NOT need to remove the stdlib flag. DO NOT USE THIS IN YOUR  
# ycm\_extra\_conf IF YOU'RE NOT 100% YOU NEED IT.  
try:  
final\_flags.remove( '-stdlib=libc++' )  
except ValueError:  
pass  
 将之删除后YCM才会补全c++标准库的内容。这样，一个最小能用的配置文件就出炉了。

自定义ycm\_extra\_conf：

虽然已经有了标准的c++配置文件。但有时还需要在某些地方让YCM支持c语言，或者让它支持第三方库比如Qt4或者OpenCV等。为此，需要修改配置文件的细节参数。有此需要的童鞋可以自行根据样板文件的注释进行修改。另外，本人写了一个自动生成.ycm\_extra\_conf.py的脚本，目前支持c和Qt common libs的自定义。见这里：[https://github.com/robturtle/newycm\_extra\_conf.py](http://jump.bdimg.com/safecheck/index?url=rN3wPs8te/r8jfr8YhogjfUWFoMgIRa8d+lJlSz2Ey30H2IZv2pxoDjXdSftCvDowvKOM8Mr1xPOK0zx5HoJFeY6ziLC4MyGmLSLhrX9ajJ0LU4OngLi73fqoP32cyrIQbcgB/YT9hVQ2eIlQjC3YPA92sJ3O3yIiBm084QAE+J2Pah4egTNMg==)

YCM变量配置：

在日常使用中，有一些建议修改的变量。本人的.vimrc中对YCM的配置如下：  
" YouCompleteMe  
nnoremap <leader>jd :YcmCompleter GoToDefinitionElseDeclaration<CR>  
nnoremap <F5> :YcmForceCompileAndDiagnostics<CR>  
"let g:ycm\_global\_ycm\_extra\_conf = '~/.ycm\_extra\_conf.py'  
"Do not ask when starting vim  
let g:ycm\_confirm\_extra\_conf = 0  
let g:syntastic\_always\_populate\_loc\_list = 1  
 其中，g:ycm\_global\_ycm\_extra\_conf 定义了全局配置文件，对于懒人来说是个好用的功能。当YCM在向上搜索的路径中找不到配置文件后，YCM将导入全局配置文件。g:ycm\_confirm\_extra\_conf 决定了在导入配置时是否需要手动确认。设置为0后YCM就不会老是在启动vim的时候来烦你了。

本示例中安装了syntastic插件进行语法检查，因此g:syntastic\_always\_populate\_loc\_list设置为1将更方便地使用其功能。又因为本示例安装了ListToggle，打开错误/警告提示面板将变成简单的一个快捷键。在vim中，<leader>键可以自由设置，一般来说它是逗号","  
进行至此，可用的快捷键如下：  
<leader>jd 跳转到当前函数的定义或声明。  
F5 刷新错误/警告。（其实没什么用，每次保存时候都会自动刷新的）  
<leader>l 打开错误/警告提示面板  
<leader>q 打开quickfix窗口

Python 高级语法检查：  
只要安装了pylint就可以自动启用了

[大杀器]compile commands database

对于比较复杂的应用很多第三方库而且想最小化链接库的c++项目，比手动自定义.ycm\_extra\_conf.py更方便的方法就是，生成Clang编译命令数据库。  
 一种非常方便的生成方法就是使用CMake，well well在linux下搞c++，CMake应该是最好的自动工具之一了吧。至于其他自动构建工具，请自行google。  
 在使用cmake命令的时候加上 -DCMAKE\_EXPORT\_COMPILE\_COMMANDS=ON 参数即可导出数据库。对于使用cmake的惯例来说，命令大致如下：  
在项目根目录中：  
mkdir build  
cd build  
cmake -DCMAKE\_EXPORT\_COMPILE\_COMMANDS=ON ..  
然后在根目录中的.ycm\_extra\_conf.py中，找到这一行：  
compilation\_database\_folder = ''  
将包含数据库的文件夹填入，如果是上例，就改成：  
compilation\_database\_folder = 'build'  
即可。如此，配置文件将根据数据库的内容工作，而原先配置文件里的所有flags都将被忽略。  
 很显然，上述过程也可以通过一个脚本自动化，甚至可以在vim保存文件时让上述命令自动运行以刷新数据库。这个功能留给我们的想象空间还是很大的。  
 不过目前来说，只有用了许多第三方库的项目用这个方法才省心，所以自动化就留给有心人去做吧。

**5、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

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

**6、树形目录插件NERDTree**

安装NERDTree插件

先下载，官网：<http://www.vim.org/scripts/script.php?script_id=1658>

解压缩之后，把 plugin/NERD\_tree.vim 和doc/NERD\_tree.txt分别拷贝到~/.vim/plugin 和 ~/.vim/doc 目录。

在linux命令行界面，输入vim

输入  :NERDTree ，回车进入当前目录的树形界面，通过小键盘上下键，能移动选中的目录或文件

目录前面有+号，摁Enter会展开目录，文件前面是-号，摁Enter会在右侧窗口展现该文件的内容，并光标的焦点focus右侧。

ctr+w+h  光标focus左侧树形目录，ctrl+w+l 光标focus右侧文件显示窗口。多次摁 ctrl+w，光标自动在左右侧窗口切换

光标focus左侧树形窗口，摁? 弹出NERDTree的帮助，再次摁？关闭帮助显示

输入:q回车，关闭光标所在窗口

配置：

"-------------------------nerdtree--------------------

" 设置NerdTree

map <F3> :NERDTreeMirror<CR>

map <F3> :NERDTreeToggle<CR>

"autocmd VimEnter \* NERDTree

"wincmd w

"autocmd VimEnter \* wincmd w

   "NERDTree配置{{{  
                map <F1> :NERDTreeToggle<CR>  
                map <C-F1> :NERDTreeFind<CR>  
                let NERDTreeChDirMode=2 "选中root即设置为当前目录  
                let NERDTreeQuitOnOpen=1 "打开文件时关闭树  
                let NERDTreeShowBookmarks=1 "显示书签  
                let NERDTreeMinimalUI=1 "不显示帮助面板  
                let NERDTreeDirArrows=1 "目录箭头 1 显示箭头 0传统+-|号  
        "}}}

常用命令：

NERDTreeToggle “打开 关闭树

NERDTree [<start-directory> | <bookmark>] “这个配合标签用起来很爽  
NERDTreeFind “在树中找到当前文件  
Bookmark <name> “mark个书签  
BookmarkToRoot <bookmark> “打开书签并设置为root，用pwd查看当前工作目录  
ClearBookmarks [<bookmarks>] “清除书签

常用快捷键：

o…….Open files, directories and bookmarks………………..|NERDTree-o|  
go……Open selected file, but leave cursor in the NERDTree…..|NERDTree-go|  
t…….Open selected node/bookmark in a new tab……………..|NERDTree-t|  
T…….Same as ‘t’ but keep the focus on the current tab……..|NERDTree-T|  
i…….Open selected file in a split window…………………|NERDTree-i|  
gi……Same as i, but leave the cursor on the NERDTree……….|NERDTree-gi|  
s…….Open selected file in a new vsplit…………………..|NERDTree-s|  
gs……Same as s, but leave the cursor on the NERDTree……….|NERDTree-gs|  
O…….Recursively open the selected directory………………|NERDTree-O|  
x…….Close the current nodes parent………………………|NERDTree-x|  
X…….Recursively close all children of the current node…….|NERDTree-X|

e…….Edit the current dif……………………………….|NERDTree-e|  
D…….Delete the current bookmark ………………………..|NERDTree-D|  
P…….Jump to the root node………………………………|NERDTree-P|  
p…….Jump to current nodes parent………………………..|NERDTree-p|  
K…….Jump up inside directories at the current tree depth…..|NERDTree-K|  
J…….Jump down inside directories at the current tree depth…|NERDTree-J|  
<C-J>…Jump down to the next sibling of the current directory…|NERDTree-C-J|  
<C-K>…Jump up to the previous sibling of the current directory.|NERDTree-C-K|  
C…….Change the tree root to the selected dir……………..|NERDTree-C|  
u…….Move the tree root up one directory………………….|NERDTree-u|  
U…….Same as ‘u’ except the old root node is left open……..|NERDTree-U|

r…….Recursively refresh the current directory…………….|NERDTree-r|

R…….Recursively refresh the current root…………………|NERDTree-R|  
m…….Display the NERD tree menu………………………….|NERDTree-m|  
cd……Change the CWD to the dir of the selected node………..|NERDTree-cd|  
I…….Toggle whether hidden files displayed………………..|NERDTree-I|  
f…….Toggle whether the file filters are used……………..|NERDTree-f|  
F…….Toggle whether files are displayed…………………..|NERDTree-F|  
B…….Toggle whether the bookmark table is displayed………..|NERDTree-B|  
q…….Close the NERDTree window…………………………..|NERDTree-q|  
A…….Zoom (maximize/minimize) the NERDTree window………….|NERDTree-A|

?…….Toggle the display of the quick help…………………|NERDTree-?|

**7、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。

winmanager下载地址：<http://www.vim.org/scripts/script.php?script_id=1440>

参考：<http://blog.itpub.net/25897606/viewspace-1244602/>

备注：.vmb插件使用方法为,使用vim打开.vmb文件，在命令行输入命令：so %回车后即可。

1. 参考方案

 http://geek.csdn.net/news/detail/34295

1. 我自己的vimrc文件



具体如下：

set shiftwidth=4

set tabstop=4

set nu

set autoindent

if has("syntax")

syntax on

endif

set nocompatible

set backspace=indent,eol,start

colorscheme ron

set autowrite

set mouse=a

set history=50

set ruler

set showcmd

set showmode

"-------------------------coding-------------------------

set fileencoding=utf-8

set fileencodings=utf-8,gbk,cp936

set termencoding=utf-8,gb18030,gb2312,gbk

set encoding=utf-8

"-------------------------ctags--------------------------

"map <C-F12> :!ctags -R --c++-kinds=+p --fields=+iaS --extra=+q .<CR>

" 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=/home/yxf/.vim/tags

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)

"-------------------------taglist------------------------

"-- 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

let Tlist\_Auto\_Open = 0

let Tlist\_Exit\_OnlyWindow=1 "如果taglist窗口是最后一个窗口，则退出vim

"let Tlist\_WinHeight = 100

"let Tlist\_WinWidth = 40

"------------------------- WinManager setting ---------------------------

" 设置winmanager

" 设置界面分割,设置我们要管理的插件

let g:winManagerWindowLayout = "NERDTree|TagList"

"设置winmanager的宽度，默认为25

let g:winManagerWidth = 30

"定义打开关闭winmanager快捷键为F8

nmap <silent> <F8> :WMToggle<cr>

"在进入vim时自动打开winmanager

let g:persistentBehaviour=0 " 如果所有编辑文件都关闭了，退出vim

let g:AutoOpenWinManager = 1 "在进入vim时自动打开winmanager

let g:winManagerOnRightSide = 1

"-------------------------vundle--------------------------

filetype off " required! /\*\* 从这行开始，vimrc配置 \*\*/

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

call vundle#rc()

" let Vundle manage Vundle

" required!

Bundle 'gmarik/vundle'

" My Bundles here: /\* 插件配置格式 \*/

"

" original repos on github （github网站上非vim-scripts仓库的插件，按下面格式填写）

"bundle 'tpope/vim-fugitive'

"bundle 'lokaltog/vim-easymotion'

"bundle 'rstacruz/sparkup', {'rtp': 'vim/'}

"bundle 'tpope/vim-rails.git'

" vim-scripts repos （vim-scripts仓库里的，按下面格式填写）

"bundle 'l9'

"bundle 'FuzzyFinder'

" non github repos (非上面两种情况的，按下面格式填写)

"bundle 'git://git.wincent.com/command-t.git'

" ...

"filetype plugin indent on " required

"------------------------------------------------------> YouCompleteMe

Bundle 'Valloric/YouCompleteMe'

Bundle 'Valloric/ListToggle'

Bundle 'scrooloose/syntastic'

let g:ycm\_global\_ycm\_extra\_conf = '~/.vim/bundle/YouCompleteMe/third\_party/ycmd/cpp/ycm/.ycm\_extra\_conf.py '

nnoremap <leader>jd :YcmCompleter GoToDefinitionElseDeclaration<CR>

nnoremap <F5> :YcmForceCompileAndDiagnostics<CR>

let g:ycm\_confirm\_extra\_conf=0 "打开vim时不再询问是否加载ycm\_extra\_conf.py配置

let g:ycm\_collect\_identifiers\_from\_tag\_files = 1 "使用ctags生成的tags文件

"------------------------------------------------------> NERDTree

Bundle 'scrooloose/nerdtree'

" 设置NerdTree

map <F3> :NERDTreeMirror<CR>

map <F3> :NERDTreeToggle<CR>

let g:NERDTree\_title = "[NERDTree]"

function! NERDTree\_Start()

exe 'NERDTree'

endfunction

function! NERDTree\_IsValid()

return 1

endfunction

"autocmd VimEnter \* NERDTree

"wincmd w

"autocmd VimEnter \* wincmd w

filetype plugin indent on " required! /\*\* vimrc文件配置结束 \*\*/

" /\*\* vundle命令 \*\*/

" 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..