

# VIM 配置推荐方案

---

本人水平有限，出于帮助师弟师妹的心理而编写了这个简单的文档，如有错漏，还请指出，本文仅供参考。

作者：吴俊宇

联系方式：shibuyanorailgun@sina.com

日期： Aug 31 2013

## 前言：

这个文档的目的在于，快速设置插件和配置，提高 **VIM** 下面的工作效率，基本上都是短平快而且最基本实用的插件配置。

本文基于读者已经了解 **vim** 基本操作的假设，否则请回去阅读黄东师兄的文档。

首先我们要说一下怎么在 **linux** 上配置软件，不同与 **Windows** 和 **mac** 下点按钮的风格，

**linux** 大部分情况下都是手动写配置文档或者是修改环境变量，例如 **git**

有 **.gitconfig**, **vim** 也是可以依靠 **.vimrc** 和一些其他脚本来实现配置，你更加知道你在干什么。**~/.vimrc** 的脚本是看起来有点像是命令行，一些多语言通用的配置，一般写在这里，**~/.vim** 文件夹存放一些插件，或者是一些语法文件（这里的说法可能有些粗糙，不太准确）。所以我们要做的事情就是，修改这两个东西。

本文大概的条目是：

- 1 多文件管理，编辑与快速切换
- 2 定义的浏览，查看
- 3 快捷键定义
- 4 自动补全

通常来说，配置只需按照本文顺序从头到尾进行即可。

正文：

## 一、 多文件管理，编辑与快速切换

### 1. Ex

简单粗暴的第一个内置功能，直接用 `vim` 打开编辑文件夹或者 `:Explorer`，你会得到一个文件浏览的页面，你可以在上面选择文件，其实我最开始就是这么干的，你可以根据上方的 `Quickhelp` 做一些事情，排序，打开编辑之类的，不过个人感觉并不够方便。

```
" Press ? for help

.. (up a dir)
/home/cennix/workspace/cpp/
| -a.out*
| -Curry.cpp
| -Curry.h
| -Curry.o
| -in
| -main.cpp
| -main.o
| -makefile
| -ss1.png
| -ss2.png
| -vim-note.rd
|
|
|
|
```

## 2. NerdTree

接下来我们要看到第一个插件：

NerdTree, <https://github.com/scrooloose/nerdtree>。

你可以下载之后，解压（`man unzip`，或者自己 `google`）并将文件放置在 `.vim` 中，并

在 `.vimrc` 文件中加入这句话：

```
let g:NERDTreeDirArrows=0
```

这是为了防止编码问题，详细你可以 `google`：）。NerdTree 的详细用法可以参考 `doc`，

但是我一般比较常用的就两个功能：NerdTreeToggle 和 menu。在命令模式输入

NerdTreeToggle 打开 NerdTree，wow！这就类似于 IDE 的文件导航栏。按 `j`，`k` 或者

方向键移动光标选取文件，

```
" Press ? for help
.. (up a dir)
/home/cennix/workspace/cpp/
-a.out*
-Curry.cpp
-Curry.h
-Curry.o
-in
-main.cpp
-main.o
-makefile
-vim-note.rd

1
2 #include <iostream>
3
4 using namespace std;
5
6 #include "../Curry.h"
7
8 int main(int argc, char *argv[])
9 {
10     Curry c("Curry");
11
12     return 0;
13 } /* end function main */
14

~
~
~
~
~
~
~
~
~
~
```

按 **m** 打开操作菜单，提供新建，删除，复制，移动等功能。

```

.. (up a dir)
/home/cennix/workspace/cpp/
-a.out*
-Curry.cpp
-Curry.h
-Curry.o
-in
-main.cpp
-main.o
-makefile
-vim-note.rd

```

```

3
4 using namespace std;
5
6 #include "../Curry.h"
7
8 int main(int argc, char** argv)
9 {
10     Curry c("Curry");
11
12     return 0;
13 } /* end function main
14
~
~
~
~
~
~
~
~
~
~

```

```

/home/cennix/workspace/cpp      main.cpp [+]
NERDTree Menu. Use j/k/enter and the shortcuts indicated
=====
> (a)dd a childnode
(m)ove the current node
(d)elete the current node
(c)opy the current node

```

### 3. WinManager

我没怎么用过这个插件，但是反响不错的样子，因为我个人偏爱隐藏 **NerdTree** 和

TagList 的分屏，所以没怎么用。

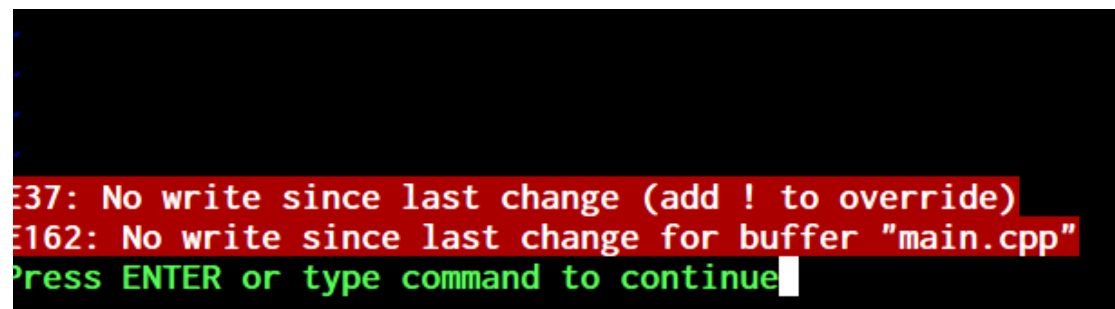
部分人可能会有兴趣：WinManager:

<https://github.com/vim-scripts/winmanager>.

## 4. Hidden

你会发现，编辑一个文件的时候想要打开修改另一个文件，却暂时不想写入文件，你希望留着这个样子，编辑另一个文件，稍后再回来，可是一旦 `:edit` 另一个文件的时候，他却会提醒你要 `:write`，等你编辑完回来，你再 `:edit` 原先的文件，发现光标在文件顶端，非常不方便；特别是利用 `ctags` 跳转查看定义的时候（稍后提及），`Ctrl + t` 回来，发现光标跑到顶上去，你不得不重新移动光标的时候就特别郁闷。

解决方案特别简单，添加 `set hidden` 即可，这时候你打开新的文件并不会关闭旧文件，随你跳来跳去，你会发现，这比 IDE 还 HAPPY（部分 IDE 并不会记住之前编辑的光标位置），稍后要退出的时候，`vim` 会提醒你有部分文件还未保存，用 `wall` 命令全部写入或者 `qall!` 直接退出即可。



```
E37: No write since last change (add ! to override)
E162: No write since last change for buffer "main.cpp"
Press ENTER or type command to continue
```

## 二、 定义的浏览，查看

### 1. Ctags

这个东西很重要，`vim` 下面的定义跳转，自动补全都要依靠他。

首先我们要安装 `ctags`，这个一个简单的分析工具，用来分析文件中的，额。。。东西。。。

你可以看一下 `man` 文档里面有讲他可以分析什么东西。

如果没有预先安装的话，可以用 `sudo apt-get install exuberant-ctags` 安装或者是通过源代码安装，事实上通过源代码安装有些好处，比如，你可以控制编译选项，你也更加清楚软件依赖。

安装完之后，在源文件文件夹下 `ctags -R *`（其实 `fields` 也是个很有用的选项，建议你 `man` 一下），出现一个 `tags` 文件，这就是分析的结果，有兴趣可以看一下他在讲什么东西。

```
1 !_TAG_FILE_FORMAT 2 /extended format; --format=1 will not append ;"  
2 !_TAG_FILE_SORTED 1 /0=unsorted, 1=sorted, 2=foldcase/  
3 !_TAG_PROGRAM_AUTHOR Darren Hiebert /dhiebert@users.sourceforge.net/  
4 !_TAG_PROGRAM_NAME Exuberant Ctags //  
5 !_TAG_PROGRAM_URL http://ctags.sourceforge.net /official site/  
6 !_TAG_PROGRAM_VERSION 5.9~svn20110310 //  
7 CC makefile /^CC=g++$/;" m  
8 CFLAG makefile /^CFLAG=-c -g -Wall $/;" m  
9 Curry Curry.cpp /^Curry::Curry(const string &s)$/;" f class:Curry  
10 Curry Curry.h /^class Curry {$/;" c  
11 EXE makefile /^EXE=a.out$/;" m  
12 LDFLAG makefile /^LDFLAG=$/;" m  
13 OBJ makefile /^OBJ=$(SRC:.cpp=.o)$/;" m  
14 SRC makefile /^SRC=main.cpp Curry.cpp$/;" m  
15 m_s Curry.h /^ std::string m_s$/;" m class:Curry  
16 main main.cpp /^int main(int argc, char *argv[])$/;" f  
17 operator () Curry.cpp /^bool Curry::operator() () const {$/;" f clas
```

## 2. `Ctrl + ]` 和 `Ctrl + T`

这个东西真是非常有用！特别是你看开源项目的源代码的时候，可以直接查看定义，非常方便；在你自己的项目中，你也可以用它来查看变量的定义，函数的定义注释之类的。

将光标移动到标记（例如变量，类型名），按下 `Ctrl + ]` 可以跳转到定义处，按下 `Ctrl + T` 跳转回来（配合 `hidden`，不然你会发现频繁写入和光标移动令人厌烦）。

```
1
2 #include <iostream>
3
4 using namespace std;
5
6 #include "../Curry.h"
7
8 int main(int argc, char *argv[])
9 {
10     Curry c("Curry");
11
12     return 0;
13 } /* end function main */
14
```

```
1
2 #include <string>
3
4 class Curry {
5
6     public:
7
8         explicit Curry(const std::string &s);
9
10        bool operator() () const;
11
12        private:
13
14            std::string m_s;
15 };

```

如图，跳转到 Curry 定义。

### 3. TagList

又是一个插件，这个插件用得不如 NerdTree 多，但是也非常有用，特别是你们的项目里面有那么多方法，导航是非常重要的，这个插件可以导航你的类型和方法，宏定义。

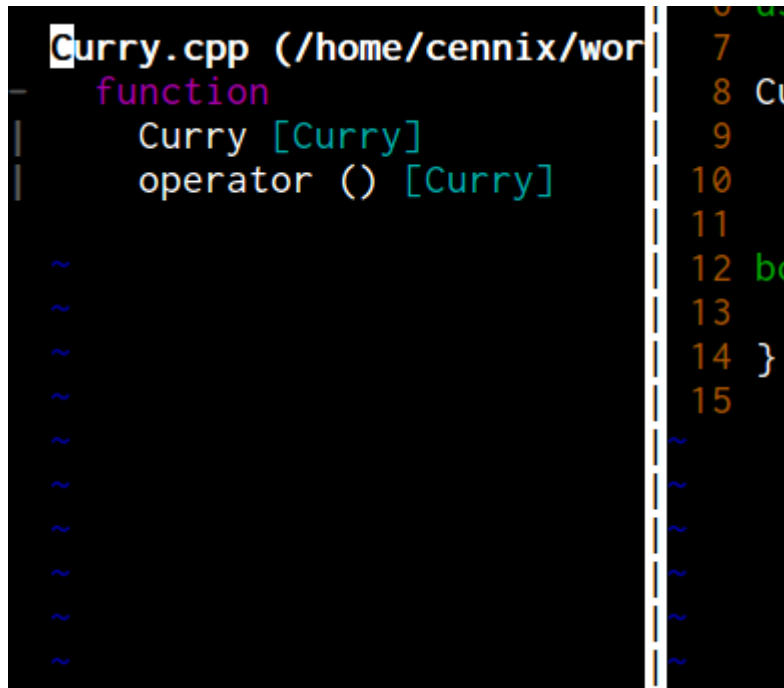


项目地址：

<https://github.com/vim-scripts/taglist.vim>

安装也非常简单，直接将压缩包内容放在.vim下即可，需要 ctags 软件的支持。

使用 TlistToggle 打开 tlist, TlistUpdate 更新 tlist。



在 vimrc 加入 `let Tlist_GainFocus_On_ToggleOpen=1`，在打开 tlist 的时候，自动将控制转移到 tlist 的分屏，减少按键。

### 三、 快捷键定义

在 IDE 中，很多事情往往只需要点一点按钮，前面我们已经说过，vim 是为了让工作更加有效率，所以 vim 的绑定快捷键是非常重要的，而且，vim 的快捷键更加灵活，可以自己定义，而且在不同的模式下，同一个快捷键具有不同的定义，这非常方便。

绑定模式如下：

**map noremap** 工作于任何模式

**vmap vnoremap** 工作于反白模式

**nmap nnoremap** 工作于常態模式

**imap inoremap** 工作于插入模式

部分按键表示如下：

**<Esc>** Esc 键

**<Tab>** Tab 键

**<Home>** Home 键

**<Del>** Del 键

**<CR>** Enter 键

**<BS>** Backspace 键

**<Up>** PageUp 键

**<F5>** F5 键

**<C-G>** Ctrl + G 键

这是几个常用的快捷键绑定命令，**nore** 表示防止重复绑定。示例：

**nnoremap <F2> :NerdTreeToggle<CR>**

`nnoremap` 表示在常态模式这个快捷键可用，`<F2>`表示绑定这个键，后面的字符串表示真正操作：在命令模式输入 `NerdtreeToggle`。这样按一下 `F2` 就调出 `nerdtree`，再按一下关掉 `nerdtree`，非常方便。同理，`tlist` 也可如此定义。

这个问题是解决了，那么编译的快捷键呢？`vim` 可没有这个命令。虽然 `vim` 没有，可是 `vim` 可以“借用”一下 `shell` 的命令 `g++`，比如：

```
nnoremap <F5> :! g++ main.cpp
```

在命令模式输入 `!`，表示后面跟的是 `shell` 的命令，当然，如果我们要编译 `User.cpp`，这可不方便，怎么办？`Makefile`，我们给 `make` 绑定一个快捷键就一了百了了。

## 四、 自动补全

我们将要介绍几种补全方式。

### 1. 后缀补全

第一次接触这个功能，我也觉得有点牛头不对马嘴，为什么我在 `cout.`后面出来一大堆不相关的东西？其实这个后缀补全并不了解语义，只是简单进行模式匹配，但是这是一个非常有用的功能，比如，`POSIX` 的 `pthread` 的结构体，函数名字都很长了（比如 `pthread_barrierattr_getpshared`，快赶上 `java` 了，这种东西手写简直要人命，语法树的补全又只能补全类型方法），所以真的相当节省力气。

```

10 pthread
11 pthread_attr_t
12 pthread_barrier_t
13 pthread_barrierattr_t
14 # pthread_cleanup_pop
15 # pthread_cleanup_pop_restore_np
16 # pthread_cleanup_push
17 # pthread_cleanup_push_defer_np
18 # pthread_cond_t
19 pthread_condattr_t
20 # pthread_equal
21 pthread_key_t
22 / pthread_mutex_t
23 pthread_mutexattr_t
24 pthread_once_t
25 i pthread_rwlock_t
26 { pthread_rwlockattr_t
27 pthread_setcancelstate
28 pthread_spinlock_t
29 pthread_t
30 pthread_barrier_wait /usr/include/pthread.h
31 pthread_create /usr/include/pthread.h
32 pthread_exit /usr/include/pthread.h
33 pthread
34
35 return EXIT_SUCCESS;
36 } /* end function main */
37
- Keyword completion (^N^P) match 6 of 134

```

在配置文件添加如下：

```

inoremap <expr> <C-n> pumvisible() ? '<C-n>' :
    \ '<C-n><C-r>=pumvisible() ? "\<lt>Down>" : ""<CR>'

set completeopt=menuone,longest

```

输入几个首字母，按下 **ctrl + n**，或者 **ctrl + p**，你会发现很多东西他都帮你列出来了。按下 **Ctrl + v**，你会有惊喜，比如，**ctrl + v** 再 **ctrl + f** 文件名补全，有了 **vim**，妈妈再也不用担心我 **include** 错文件。

## 2. 智能补全

记得我说的 `ctags` 吗? `omni-complete`, 智能补全要用到他。我并不打算详细讲这个东西, 首先我写 `c` 或者 `python` 的时候并不怎么用到这个东西, 而写 `c++` 的时候的体会是, 这个东西实用度尚可, 略麻烦, 只是简单介绍, 反倒是后缀补全最实用。

项目地址: `Omni complete`,

<https://github.com/vim-scripts/OmniCppComplete>。

## 3. Clang complete, YouCompleteMe

这两个东西真的像 IDE 的补全了, 利用语法树进行补全。但是前提略麻烦, 首先要依赖 `clang` 编译器, 其次 `vim` 必须 7.3 以上, 必须支持 `python` (通常来说, `ubuntu` 通过 `apt-get` 安装的 `vim` 都是支持, 可以通过 `vim -version` 检查)。

项目地址:

`clang complete`: [https://github.com/Rip-Rip/clang\\_complete](https://github.com/Rip-Rip/clang_complete)

`YouCompleteMe`: <https://github.com/Valloric/YouCompleteMe>

这里介绍一下 `clang complete`。

安装 `clang` 编译器:

```
sudo apt-get install clang
```

```
sudo apt-get install libclang1 libclang-common-dev libclang-dev
```

安装成功后,

```
git clone https://github.com/Rip-Rip/clang_complete.git
```

```
cd clang_complete
```

```
make install
```

即可，反安装 `make uninstall` 即可，详细配置参考 `doc`（不配置也可以用了）。

```
using namespace std;

#include "../Curry.h"

int main(int argc, char *argv[])
{
    Curry c("Curry");
    c.
        ~Curry      ~ void ~Curry()
        r operator=  f Curry & operator=(const Curry &)
    } / operator() f bool operator()()
        m_s         m std::string m_s
        Curry       t Curry::
```