高效率移动

[编辑](javascript:;)

**1、在插入模式之外**

基本上来说，你应该尽可能少的呆在插入模式里面，因为在插入模式里面 VIM 就像一个“哑巴”[编辑器](http://baike.baidu.com/item/%E7%BC%96%E8%BE%91%E5%99%A8)一样。很多新手都会一直呆在插入模式里面，因为这样易于使用。但 VIM 的强大之处在于他的命令模式！你会发现，在你越来越了解 VIM 之后，你就会花越来越少的时间使用插入模式了。

**2、使用 h、j、k、l**

使用 VIM 高效率编辑的第一步，就是放弃使用[箭头](http://baike.baidu.com/item/%E7%AE%AD%E5%A4%B4" \t "_blank)键。使用 VIM，你就不用频繁的在箭头键和字母键之间移来移去了，这会节省你很多时间。当你在命令模式时，你可以用 h、j、k、l 来分别实现左、下、上、右箭头的功能。一开始可能需要适应一下，但一旦习惯这种方式，你就会发现这样操作的高效之处了。

在你编辑你的电子邮件或者其他有段落的文本时，你可能会发现使用方向键和你预期的效果不一样，有时候可能会一次跳过了很多行。这是因为你的段落在 VIM 看来是一个大的长长的行。这时你可以在按 h、j、k 或者 l 之前键入一个 g，这样 VIM 就会按屏幕上面的行如你所愿的移动了。

**3、在当前行里面有效的移动光标**

很多[编辑器](http://baike.baidu.com/item/%E7%BC%96%E8%BE%91%E5%99%A8)只提供了简单的命令来控制光标的移动（比如左、上、右、下、到行首/尾等）。VIM 则提供了很多强大的命令来满足你控制光标的欲望。当光标从一点移动到另外一点，在这两点之间的文本（包括这两个点）称作被“跨过”，这里的命令也被称作是 motion。（简单说明一下，后面会用到这个重要的概念）

**4、常用到的一些命令（motion）**

fx：移动光标到当前行的下一个 x 处。很明显，x 可以是任意一个字母，而且你可以使用 ; 来重复你的上一个 f 命令。

tx：和上面的命令类似，但是是移动到 x 的左边一个位置。（这真的很有用）

Fx：和 fx 类似，不过是往回找。使用 , 来重复上一个F命令。

Tx：和 tx 类似，不过是往回移动到 x 的右边一个位置。

b：[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)往前移动一个词。

w：光标往后移动一个词。

0：移动光标到当前行首。（是数字0）

^：移动光标到当前行的第一个字母位置。

$：移动光标到行尾。

)：移动光标到下一个句子。

( ：移动光标到上一个句子。

**5、在整个文件里面有效移动光标**

VIM 有很多命令，可以用来到达文件里面你想到达的地方。下面是一些在文件里面移动的命令：

<Ctrl-f>：向下移动一屏。

<Ctrl-d>：向下移动半屏。

<Ctrl-b>：向上移动一屏。

<Ctrl-u>：向上移动半屏。[5]

G：到文件尾

numG：移动[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)到指定的行（num）。（比如 10G 就是到第 10 行）

gg：到文件首

H：移动光标到屏幕上面

M：移动光标到屏幕中间

L：移动光标到屏幕下面

\*：读取光标处的字符串，并且移动光标到它再次出现的地方。

#：和上面的类似，但是是往反方向寻找。

/text：从当前[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)处开始搜索字符串 text，并且到达 text 出现的地方。必须使用回车来开始这个搜索命令。如果想重复上次的搜索的话，按 n移动到下个 text 处，N 移动到上一个 text 处 。

？text：和上面类似，但是是反方向。

m{a-z}：在当前光标的位置标记一个书签，名字为 a-z 的单个字母。书签名只能是小写字母。你看不见书签的存在，但它确实已经在那里了。

`a：到书签 a 处。注意这个不是单引号，它一般位于大部分键盘的 1 的左边。

`.：到你上次编辑文件的地方。这个命令很有用，而且你不用自己去标记它。

%：在成对的括号等符号间移动，比如成对的 [ ] ， { }， ( ) 之间。将[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)放到任意符号上，然后通过 % 来移动到和这个符号匹配的符号上，% 还可以正确的识别括号的嵌套层数，总是移动到真正匹配的位置上。因此这个命令在编辑程序代码的时候非常有用，可以让你方便的在一段代码的头尾间移动。

高效的输入

[编辑](javascript:;)

**1、使用关键词自动完成**

VIM 有一个非常漂亮的关键词自动完成系统。这表示，你可以输入一个长词的一部分，然后按一下某个键，然后 VIM 就替你完成了这个长词的输入了。举个例子：你有一个变量名为 iAmALongAndAwkwardVarName 在你写的代码的[某个地方](http://baike.baidu.com/item/%E6%9F%90%E4%B8%AA%E5%9C%B0%E6%96%B9" \t "_blank)。也许你不想每回都自己一个一个字母的去输入它。

使用关键词自动完成功能，你只需要输入开始几个字母（比如 iAmAL），然后按 <C-N>（按住 Ctrl，再按 N）或者 <C-P>。如果 VIM 没有给出你想要的词，继续按，直到你满意为止，VIM 会一直循环它找到的匹配的字符串。

**2、聪明的进入插入模式**

很多新手进入插入模式都只是用 i。这样当然可以进入插入模式，但通常不是那么合适，因为 VIM 提供了很多进入插入模式的命令。下面是最常用的一些：

i：在当前字符的左边插入

I：在当前行首插入

a：在当前字符的右边插入

A：在当前行尾插入

o：在当前行下面插入一个新行

O：在当前行上面插入一个新行

c{motion}：删除 motion 命令跨过的[字符](http://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "_blank)，并且进入插入模式。比如：c$，这将会删除从[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)位置到行尾的字符并且进入插入模式。ct！，这会删除从光标位置到下一个叹号（但不包括），然后进入插入模式。被删除的字符被存在了剪贴板里面，并且可以再粘贴出来。

d{motion}：和上面差不多，但是不进入插入模式。

**3、有效的移动大段的文本**

使用可视选择（visual selections）和合适的选择模式

不像最初的 VI，VIM 允许你高亮（选择）一些文本，并且进行操作。这里有三种可视选择模式：

v：按[字符](http://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "_blank)选择。经常使用的模式，所以亲自尝试一下它。

V：按行选择。这在你想拷贝或者移动很多行的文本的时候特别有用。

<C-V>：按块选择。非常强大，只在很少的[编辑器](http://baike.baidu.com/item/%E7%BC%96%E8%BE%91%E5%99%A8" \t "_blank)中才有这样的功能。你可以选择一个矩形块，并且在这个矩形里面的文本会被高亮。

在选择模式的时候使用上面所述的方向键和命令（motion）。比如，vwww，会高亮[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)前面的三个词。Vjj 将会高亮当前行以及下面两行。

**4、在可视选择模式下剪切和拷贝**

一旦你高亮了选区，你或许想进行一些操作：

d：剪贴选择的内容到剪贴板。

y：拷贝选择的内容到剪贴板。

c：剪贴选择的内容到剪贴板并且进入插入模式。

在非可视选择模式下剪切和拷贝

如果你很清楚的知道你想拷贝或者剪切什么，那你根本就不需要进入可视选择模式。这样也会节省时间：

d{motion}：剪切 motion 命令跨过的字符到剪贴板。比如，dw 会剪切一个词而 dfS 会将从当前[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)到下一个 S 之间的字符剪切至剪贴板。

y{motion}：和上面类似，不过是拷贝。

c{motion}：和 d{motion} 类似，不过最后进入插入模式。

dd：剪切当前行。

yy：拷贝当前行。

cc：剪切当前行并且进入插入模式。

D：剪切从光标位置到行尾到剪贴板。

Y：拷贝当前行。

C：和 D 类似，最后进入插入模式。

x：剪切当前字符到剪贴板。

s：和x类似，不过最后进入插入模式。

**5、粘贴**

粘贴很简单，按 p。

**6、使用多重剪贴板**

很多[编辑器](http://baike.baidu.com/item/%E7%BC%96%E8%BE%91%E5%99%A8)都只提供了一个剪贴板。VIM 有很多。剪贴板在 VIM 里面被称为寄存器（Registers）。你可以列出当前定义的所有寄存器名和它们的内容，命令为“:reg”。最好使用小写字母来作为寄存器的名称，因为大写的有些被 VIM 占用了。

使用[寄存器](http://baike.baidu.com/item/%E5%AF%84%E5%AD%98%E5%99%A8)的命令为双引号 “。

比如：我们要拷贝当前行到寄存器 k。你应该按 “kyy。（你也可以使用 V”ky。为什么这样也可以呢？）现在当前行应该已经存在了寄存器 k 里面直到你又拷贝了一些东西进入寄存器 k。你可以使用命令 “kp 来粘贴寄存器 k 里面的内容到你想要的位置。

**7、避免重复**

令人惊奇的 . 命令

在 VI 里面，输入 . (小数点符号），将会重复你输入的上一个命令。比如，你上个命令为“dw”（删除一个词），VI 将会接着再删除一个词。

**8、使用数字**

使用数字也是 VIM 强大的而且很节省时间的重要特性之一。在很多 VIM 的命令之前都可以使用一个数字，这个数字将会告诉 VIM 这个命令需要执行几次。比如：

3j 将会把[光标](http://baike.baidu.com/item/%E5%85%89%E6%A0%87" \t "_blank)向下移动三行。

10dd 将会删除十行。

y3t″ 将会拷贝从当前光标到第三个出现的引号之间的内容到剪贴板。

数字是扩展 motion 命令作用域非常有效的方法。

高效读代码

[编辑](javascript:;)

建议在vim上安装Taglist+cscope+c[tags](http://baike.baidu.com/item/tags)[插件](http://baike.baidu.com/item/%E6%8F%92%E4%BB%B6)有利于高效读代码，有利于学习！[6]

**1、安装**

对于小编我来说我使用的是Ubuntu，安装软件一般使用的命令是apt-get install \*\*\*!

对于vim+Taglist+cscope+ctags组合[编辑器](http://baike.baidu.com/item/%E7%BC%96%E8%BE%91%E5%99%A8" \t "_blank)的创建，vim , cscope , ctags安装起来比较方便使用上述命令即可安装！

（对于不同的Linux发行版本安装软件的命令不太一样，需要根据自己的安装版稍微修改，比如说Fedora即使用yum命令安装）

对于Taglist使用这个方法安装不了,我们需要先下载，然后安装完成：

首先上网下载Taglist插件，下载完成后解压，再将文件下的taglist.vim使用cp命令拷贝到HOME/.vim/plugin文件夹下（cp -r taglist.vim ~/.vim/plugin)

**2、vim的配置**

vim是一个非常好用的编辑工具,以下介绍几个常用的底行模式命令：

（1）. 设置缩进

:set smartindent // 设置缩进

:set smartindent shiftwidth=4 // C语言自动缩进，缩进值为4个字符宽度

（2）. 显示光标行列信息

:set ruler // 显示光标信息（右下角）

（3）. 查看制表符

:set list // 显示制表符，以^符号显示制表符的位置

（4）. 查看行数信息

:set number //显示行号（左边）

（5）. 关键字高亮

:syntax on //关键字高亮

（6）. 多文件编辑 ，在编写程序时有可能需要同时编辑多个文件，在这种情况下，需要如下设置

: split two.c // 多文件编辑

当然，如果不嫌麻烦，可以在每次打开Vim时 输入如上命令，如我经常用的是:set number :set ruler :syntax on :set smartindent 命令，如果将平时在底行模式下输入的命令直接写入配置文件，则以后使用Vim时不用再次输入这些命令。

首先，试着在Vi的底行模式下输入如下命令，如果不支持的话，去下载一个Vim的全版本，sudo apt-get install vim-full。

以下是修改配置文件的方法：

（1）. 找到Vimrc文件。

（2） 查看文件的权限：

ls -l vimrc

shisir@shisir-desktop:/usr/share/vim$ cd /etc/vim

shisir@shisir-desktop:/etc/vim$ ls

gvimrc vimrc vimrc.tiny

shisir@shisir-desktop:/etc/vim$ ls -l vimrc

-r--r--r-- 1 root root 2492 2009-04-10 08:08 vimrc

（3） 如果没有写权限的话，修改权限，使我们可以修改该配置文件

sudo chmod a+w vimrc

（4）根据个人需要将上述命令写入配置文件

比如，我个人的设置是： syntax on , set ruler ,set number , set smartindent shiftwidth=4

下面贴上我的vimrc文件的配置信息：

"显示行号

set number

"检测文件的类型

filetype on

"记录历史的行数

set history=1000

"自动对齐

set autoindent

set cindent

"智能选择对齐方式

set smartindent

"tab为4个空格

set tabstop=4

"当前行之间交错时使用4个空格

set shiftwidth=4

"设置匹配模式，输入左括号会出现右括号

set showmatch

"编辑时显示光标状态

set ruler

"快速匹配

set incsearch

"修改文件自动备份

if has("vms")

set nobackup

else

set backup

endif

"""""""""""""设置开启ctags"""""""""""""

set[tags](http://baike.baidu.com/item/tags)=/home/zhouyl/linux-3.4.7/tags;"这边设置的是你使用ctags -R的目录，后面有介绍！你根据自己的使用修改这个路径

set autochdir

"""""""""""""""""Taglist设置"""""""""""""""""

""let Tlist\_Auto\_Open = 1

let Tlist\_C[tags](http://baike.baidu.com/item/tags)\_Cmd = '/usr/bin/ctags'

let Tlist\_Show\_One\_File = 1

let Tlist\_Exit\_OnlyWindow = 1

""""""""""""""""""""""cscope设置""""""""""""""""""

set cscopequickfix=s-,c-,d-,i-,t-,e-

if has("cscope")

set csprg=/usr/bin/cscope

set csto=1

set cst

set nocsverb

" add any database in current directory

if filereadable("cscope.out")

cs add cscope.out

endif

set csverb

endif

因为我的文档中的原配置文件已经有了语法高亮：

" Vim5 and later versions support syntax highlighting. Uncommenting the next

" line enables syntax highlighting by default.

if has("syntax")

syntax on

endif

所以我就没写了，您如果vimrc文件里面没有此项建议自己加上syntax on 设置语法高亮

PS：在配置文件中"符号后的内容是注释，而且在配置文件中是不需要添加符号的 ，这里关于cscope、c[tags](http://baike.baidu.com/item/tags)和Taglist的设置可以先写上，后面将对这三个[插件](http://baike.baidu.com/item/%E6%8F%92%E4%BB%B6" \t "_blank)有介绍。好了，去开一个Vim，编辑一个程序看看，是不是感觉很不一般呢？

好了，下面顺便演示下，智能补全的用法，我分四种情况：1、前面申明过的变量或者函数名的补全；2、结构体等自建结构的补全；3、STL等标准库的补全；4、括号、引号等自动匹配补全。

对于第一种情况，写程序的过程中，直接用CTRL+p即可选择，对于第二种情况，如果想申明了结构体T，想在输入T.时弹出成员供选择，只需在用之前按一下：CTRL+F12，然后效果如图：

[](http://baike.baidu.com/pic/VIM/60410/0/6609c93d70cf3bc799a94271d100baa1cc112a85?fr=lemma&ct=single)

**3、Taglist插件的使用**

TagList[插件](http://baike.baidu.com/item/%E6%8F%92%E4%BB%B6" \t "_blank),是一款基于c[tags](http://baike.baidu.com/item/tags),在vim代码窗口旁以分割窗口形式(如上图所示）显示当前的代码结构概览,增加代码浏览的便利程度的vim插件.

在Vim中加载代码文件后，可以使用以下命令控制taglist

:TlistOpen 打开并将输入焦点至于标签列表窗口,效果如上图所示

:TlistClose 关闭标签列表窗口

:TlistToggle 切换标签列表窗口状态(打开←→关闭)，标签列表窗口是否获得焦点取决于其他配置

ctl-w + w 或ctl-w + 方向键 窗口切换（taglist本质上是一个vim分隔窗口，因此可以使用ctl-w系列快捷键对窗口进行切换操作)

在Taglist窗口主要有以下操作

按F1:打开帮助

回车键:跳到光标所在的标记的定义处(如将光标移到main函数,按回车键)

o:新建一个窗口,跳到标记定义处

p:预览标记定义(仍然在taglist窗口)

空格:显示标记的原型(如函数原型)

u:更新标记列表(比如源文件新增了一个函数,并在保存后,可在taglist窗口按u)

s:选择排序字段(暂时我也不知道什么意思)

d:删除光标所在的taglist文件(如用vi打开了两个文件f1.c,f2.c可以删除f1.c的标记)

x:n放大/缩小taglist窗口

+:展开(指标记)

-:折叠

\*:全部展开

=:全部折叠

[[:将光标移到前一个文件的起点

]]:将光标移到后一个文件的起点

q:退出taglist窗口

F1:关闭帮助

**4、ctags插件的使用**

VIM简单易用，大家只要看看VIM自带的vitutor，大约只要20分钟就能掌握VIM的用法,上面也简单介绍了点vim的用法。但是对于Linux下的C/C++的 程序员，使用VIM+C[tags](http://baike.baidu.com/item/tags)的组合来写程序也许是最佳的选择。^\_^

尽管ctags也可以支持其它[编辑器](http://baike.baidu.com/item/%E7%BC%96%E8%BE%91%E5%99%A8" \t "_blank)，但是它正式支持的只有VIM。并且VIM中已经默认安装了Ctags，它可以帮助程序员很容易地浏览源代码。

熟练的使用ctags仅需记住下面七条命令：（很简单吧，呵呵）

1． $ ctags–R \* ($ 为Linux系统Shell提示符)

2. $ vi –t tag (请把tag替换为您欲查找的变量或函数名)

3． ：ts (ts 助记字：[tags](http://baike.baidu.com/item/tags" \t "_blank)list, “:”开头的命令为VI中命令行模式命令)

4． ：tp (tp 助记字：tagspreview)---此命令不常用，可以不用记

5． ：tn (tn 助记字：tagsnext) ---此命令不常用，可以不用记

6． Ctrl + ] 跳到光标所在函数或者结构体的定义处

7． Ctrl + T 返回查找或跳转

下面我们逐条的解释上面的命令：

“$ ctags–R \*”：“-R”表示递归创建，也就包括源代码根目录（当前目录）下的所有子目录。“\*”表示所有文件。这条命令会在当前目录下产生一个“[tags](http://baike.baidu.com/item/tags)”文件， 当用户在当前目录中运行vi时，会自动载入此tags文件。Tags文件中包括这些对象的列表：

用#define定义的宏

枚举型变量的值

函数的定义、原型和声明

名字空间（namespace）

类型定义（typedefs）

变量（包括定义和声明）

类（class）、结构（struct）、枚举类型（enum）和联合（union）

类、结构和联合中成员变量或函数

VIM用这个“[tags](http://baike.baidu.com/item/tags)”文件来定位上面这些做了标记的对象。

剩下的命令就是定位这些对象的方法：

“$ vi –t tag” ：在运行vim的时候加上“-t”参数，例如：

[/usr/src]$ vim -t main

这个命令将打开定义“main”（变量或函数或其它）的文件，并把光标定位到这一行。如果这个变量或函数有多处定义，在VI命令行模式“：ts”命令就能 列出一个列表供用户选择。 “：tp”为上一个tag标记文件，“：tn”为下一个tag标记文件。当然，若当前tags文件中用户所查找的变量或函数名只有一 个，“:tp,:tn”命令不可用。 ：（

最方便的方法是把光标移到变量名或函数名上，然后按下“ Ctrl+] ”，这样就能直接跳到这个变量或函数定义的源文件中，并把光标定位到这一行。用 “ Ctrl+T ”可以退回原来的地方。即使用户使用了N次“Ctrl+]”查找了N个变量，按N次“Ctrl+t”也能回到最初打开的文件，它会按原路返 回 。 ^\_^

注意：运行vim的时候，必须在“[tags](http://baike.baidu.com/item/tags)”文件所在的目录下运行。否则，运行vim的时候还要用“:settags=”命令设定“tags”文件的路 径，这样vim才能找到“tags”文件。在完成编码时，可以手工删掉tags文件（帚把不到，灰尘不会自己跑掉^\_^）。

**5、cscope插件的使用**

Cscope是一个类似 ctags的工具。 你可以把它想作是超过频的 ctags，因为它功能比ctags 强大很多。 在 Vim 里，通过 cscope 查询结果来跳转就象跳转到其他的标签完全一样；它被保存在标签栈里。这样你就可以象使用 c[tags](http://baike.baidu.com/item/tags)一样在函数等等之间便捷的跳转[7]  。

在VIM中使用cscope非常简单，首先调用“cscope add”命令添加一个cscope数据库，然后就可以调用“cscope find”命令进行查找了。VIM支持8种cscope的查询功能，如下：例如，我们想在代码中查找调用work()函数的函数，我们可以输入：“:cs find c work”，回车后发现没有找到匹配的功能，可能并没有函数调用work()。我们再输入“:cs find s work”，查找这个符号出现的位置，vim列出了这个符号出现的所有位置。我们还可以进行字符串查找，它会双引号或单引号括起来的内容中查找。还可以输入一个正则表达式，这类似于egrep程序的功能。

在源代码目录下打开vim。要使用cscope查找就必须加载cscope.out文件.在vim命令行下执行：(按照上述方法配置好的vimrc中有这一步）

:cs add cscope.out

在vim命令行下执行：

:cs help

cscope commands:

add : Add a new database (Usage: add file|dir [pre-path] [flags])

find : Query for a pattern (Usage: find c|d|e|f|g|i|s|t name)

c: Find functions calling this function //查找调用本函数的函数

d: Find functions called by this function //查找本函数调用的函数

e: Find this egrep pattern //查找egrep模式，相当于egrep功能，但查找速度快多了

f: Find this file //查找并打开文件，类似vim的find功能

g: Find this definition //查找函数、宏、枚举等定义的位置，类似c[tags](http://baike.baidu.com/item/tags)的功能

i: Find files #including this file //查找包含本文件的文件

s: Find this C symbol //查找C语言符号，即查找函数名、宏、枚举值等出现的地方

t: Find assignments to //查找指定的字符串

help : Show this message (Usage: help)

VIM命令

kill : Kill a connection (Usage: kill #)

reset: Reinit all connections (Usage: reset)

show : Show connections (Usage: show)

例如：使用cscope查找do\_fork函数的定义：

在vim命令行下执行： :cs f g do\_fork

