

## Linux vim与gcc使用

1. 学习yum工具，进行软件安装
2. 掌握vim编辑器使用，学会vim的简单配置
3. 掌握gcc/g++编译器的使用，并了解其过程，原理

### Linux开发工具

- IDE例子

version 1.1  
April 1st, 06  
翻译:2006-5-22

### vi / vim 键盘图

Esc 命令模式															
~ 转换大小写 跳转到标注	! 外部过滤器	@ 运行宏	# prev ident	\$ 行末	% 括号匹配	^ "软"行首	& 重复:s	* next ident	( 句首	) 下一句首	"soft" bol down	+ 后一行首			
1	2	3	4	5	6	7	8	9	0	"硬"行首	- 前一行首	= 自动格式化			
Q 切换到ex模式	W 下一单词	E 词尾	R 替换模式	T back 'till	Y 拷贝行	U 撤销行内命令	I 到行首插入	O 分段(前)	P 粘贴(前)	{ 段首	}	段尾			
q 录制宏	w 下一单词	e 词尾	r 替换字符	t 'till	y 拷贝1,3	u 撤销命令	i 插入模式	o 分段(后)	p 粘贴(后)	[ 杂项	]	杂项			
A 在行末附加	S 删除行并插入	D 删除至行末	F 行内字符反向查找	G 文尾/行号	H 屏幕顶行	J 合并两行	K 帮助	L 屏幕底行	:	ex 命令	" 寄存器标识	' 行首/列			
a 附加	s 删除字符并插入	d 删除1,3	f 行内字符查找	g 附加命令	h ←	j ↓	k ↑	l →	:	重复 u/T/t/F	' 跳转到标注的行首	\ 未使用!			
Z 退出4	X 退格	C 修改至行末	V 可视行模式	B 前一单词	N 查找上一处	M 屏幕中间行	< 反缩进3	> 缩进3	?	向前搜索					
Z 附加命令5	x 删除(字符)	c 修改1,3	v 可视模式	b 前一单词	n 查找下一处	m 设置标注	, u/T/t/F	.	重复命令	/	向后搜索				

**动作** 移动光标，或者定义操作的范围

**命令** 直接执行的命令，  
红色命令进入编辑模式

**操作** 后面跟随表示操作范围的指令

**extra** 特殊功能，需要额外的输入

**q.** 后跟字符参数

**w,e,b命令**

小写(b): `quux(foo, bar, baz);`

大写(B): `quux(Foo, Bar, Baz);`

**主要ex命令:**

`:w` (保存), `:q` (退出), `:q!` (不保存退出)

`:e f` (打开文件 f),

`:%s/x/y/g` ('y' 全局替换 'x'),

`:h` (帮助 in vim), `:new` (新建文件 in vim)

**其它重要命令:**

**CTRL-R:** 重复 (vim),

**CTRL-F/-B:** 上翻/下翻,

**CTRL-E/-Y:** 上滚/下滚,

**CTRL-V:** 块可视模式 (vim only)

**可视模式:**

漫游后对选中的区域执行操作 (vim only)

**备注:**

(1) 在 拷贝/粘贴/删除 命令前使用 "x (x=a..z,") 使用命令的寄存器(剪贴板)" (如: "ay\$ 拷贝剩余的行内容至寄存器 'a')"

(2) 命令前添加数字 多遍重复操作 (e.g.: 2p, d2w, 5i, d4j)

(3) 重复本字符在光标所在行执行操作 (dd = 删除本行, >> = 行首缩进)

(4) ZZ 保存退出, ZQ 不保存退出

(5) zt: 移动光标所在行至屏幕顶端, zb: 底端, zz: 中间

(6) gg: 文首 (vim only), gf: 打开光标处的文件名 (vim only)

## Linux编辑器-vim使用

vi/vim的区别简单点来说，它们都是多模式编辑器，不同的是vim是vi的升级版本，它不仅兼容vi的所有指令，而且还有一些新的特性在里面。例如语法加亮，可视化操作不仅可以在终端运行，也可以运行于x window、mac os、windows。我们课堂上，统一按照vim来进行讲解。



## 1. vim的基本概念

课堂上我们讲解vim的三种模式(其实有好多模式，目前掌握这3种即可),分别是命令模式 (command mode)、插入模式 (Insert mode) 和底行模式 (last line mode)，各模式的功能区分如下：

- 正常/普通/命令模式(Normal mode)

控制屏幕光标的移动，字符、字或行的删除，移动复制某区段及进入Insert mode下，或者到 last line mode

- 插入模式(Insert mode)

只有在Insert mode下，才可以做文字输入，按「ESC」键可回到命令行模式。该模式是我们后面用的最频繁的编辑模式。

- 末行模式(last line mode)

文件保存或退出，也可以进行文件替换，找字符串，列出行号等操作。在命令模式下，*shift+:* 即可进入该模式。要查看你的所有模式：打开vim，底行模式直接输入

`:help vim-modes`

我这里一共有12种模式:six BASIC modes和six ADDITIONAL modes.

## 2. vim的基本操作

- 进入vim,在系统提示符号输入vim及文件名称后，就进入vim全屏幕编辑画面:

- `$ vim test.c`
- 不过有一点要特别注意，就是你进入vim之后，是处于[正常模式]，你要切换到[插入模式]才能够输入文字。
- [正常模式]切换至[插入模式]
  - 输入a
  - 输入i
  - 输入o
- [插入模式]切换至[正常模式]
  - 目前处于[插入模式]，就只能一直输入文字，如果发现输错了字,想用光标键来回移动，将该字删除，可以先按一下「ESC」键转到[正常模式]再删除文字。当然，也可以直接删除。
- [正常模式]切换至[末行模式]
  - 「shift + ;」，其实就是输入「:」
- 退出vim及保存文件,在[正常模式]下，按一下「:」冒号键进入「Last line mode」,例如:
  - :w (保存当前文件)
  - :wq (输入「wq」,存盘并退出vim)
  - :q! (输入q!,不存盘强制退出vim)

### 3. vim正常模式命令集 [重要点进行讲解]

- 插入模式
  - 按「i」切换进入插入模式「insert mode」，按“i”进入插入模式后是从光标当前位置开始输入文件；[重要]
  - 按「a」进入插入模式后，是从目前光标所在位置的下一个位置开始输入文字；
  - 按「o」进入插入模式后，是插入新的一行，从行首开始输入文字。
- 从插入模式切换为命令模式
  - 按「ESC」键。 [重要]
- 移动光标
  - vim可以直接用键盘上的光标来上下左右移动，但正规的vim是用小写英文字母「h」、「j」、「k」、「l」，分别控制光标左、下、上、右移一格
  - 按「G」：移动到文章的最后 [重要]
  - 按「\$」：移动到光标所在行的“行尾” [重要]
  - 按「^」：移动到光标所在行的“行首” [重要]
  - 按「w」：光标跳到下个字的开头 [重要]
  - 按「e」：光标跳到下个字的字尾
  - 按「b」：光标回到上个字的开头 [重要]
  - 按「#l」：光标移到该行的第#个位置，如：5l,56l
  - 按「gg」：进入到文本开始 [重要]
  - 按「shift + g」：进入文本末端
  - 按「ctrl」+「b」：屏幕往“后”移动一页
  - 按「ctrl」+「f」：屏幕往“前”移动一页
  - 按「ctrl」+「u」：屏幕往“后”移动半页
  - 按「ctrl」+「d」：屏幕往“前”移动半页

- 删除文字

- 「x」：每按一次，删除光标所在位置的一个字符 [重要]
- 「#x」：例如，「6x」表示删除光标所在位置的“后面（包含自己在内）”6个字符
- 「X」：大写的X，每按一次，删除光标所在位置的“前面”一个字符
- 「#X」：例如，「20X」表示删除光标所在位置的“前面”20个字符
- 「dd」：删除光标所在行 [重要]
- 「#dd」：从光标所在行开始删除#行 [重要]

- 复制

- 「yw」：将光标所在之处到字尾的字符复制到缓冲区中。
- 「#yw」：复制#个字到缓冲区
- 「yy」：复制光标所在行到缓冲区。 [重要]
- 「#yy」：例如，「6yy」表示拷贝从光标所在的该行“往下数”6行文字。
- 「p」：将缓冲区内的字符贴到光标所在位置。注意：所有与“y”有关的复制命令都必须与“p”配合才能完成复制与粘贴功能。 [重要]

- 替换

- 「r」：替换光标所在处的字符。
- 「R」：替换光标所到之处的字符，直到按下「ESC」键为止。 [重要]

- 撤销上一次操作

- 「u」：如果您误执行一个命令，可以马上按下「u」，回到上一个操作。按多次“u”可以执行多次回复。 [重要]
- 「ctrl + r」：撤销的恢复 [重要]

- 更改

- 「cw」：更改光标所在处的字到字尾处
- 「c#w」：例如，「c3w」表示更改3个字

- 跳至指定的行

- 「ctrl」+「g」列出光标所在行的行号。 [重要]
- 「#G」：例如，「15G」，表示移动光标至文章的第15行行首。

## 4. vim末行模式命令集

在使用末行模式之前，请记住先按「ESC」键确定您已经处于正常模式，再按「:」冒号即可进入末行模式。

- 列出行号

- 「set nu」：输入「set nu」后，会在文件中的每一行前面列出行号。 [重要]

- 跳到文件中的某一行

- 「#」：「#」号表示一个数字，在冒号后输入一个数字，再按回车键就会跳到该行了，如输入数字15，再回车，就会跳到文章的第15行。

- 查找字符

- 「/关键字」：先按「/」键，再输入您想寻找的字符，如果第一次找的关键字不是您想要的，可以一直按「n」会往后寻找到您要的关键字为止。



- 「?关键字」：先按「?」键，再输入您想寻找的字符，如果第一次找的关键字不是您想要的，可以一直按「n」会往前寻找到您要的关键字为止。
- 问题：/ 和 ? 查找有和区别？操作实验一下

- 保存文件 [重要]

- 「w」：在冒号输入字母「w」就可以将文件保存起来

- 离开vim [重要]

- 「q」：按「q」就是退出，如果无法离开vim，可以在「q」后跟一个「!」强制离开vim。
- 「wq」：一般建议离开时，搭配「w」一起使用，这样在退出的时候还可以保存文件。

## 5. vim操作总结

- 三种模式

- 正常模式
- 插入模式
- 底行模式

- 我们一共有12种总模式，大家下来可以研究一下
- vim操作

- 打开，关闭，查看，查询，插入，删除，替换，撤销，复制等等操作。

- 练习：当堂口头模式切换练习

## 6. 简单vim配置 [选学]

### 自动配置 (推荐)

<https://github.com/askunix/VimForCpp>

保证自己联网的情况下，命令行直接运行,一键部署

```
curl -sLf https://gitee.com/HGtz2222/VimForCpp/raw/master/install.sh -o ./install.sh &&
bash ./install.sh
```

### 配置文件的位置

- 在目录 /etc/ 下面，有个名为vimrc的文件，这是系统中公共的vim配置文件，对所有用户都有效。
- 而在每个用户的主目录下，都可以自己建立私有的配置文件，命名为：“.vimrc”。例如，/root目录下，通常已经存在一个.vimrc文件,如果不存在，则创建之。
- 切换用户成为自己执行 su，进入自己的主工作目录,执行 cd ~
- 打开自己目录下的.vimrc文件，执行 vim .vimrc

### 常用配置选项,用来测试

- 设置语法高亮: syntax on
- 显示行号: set nu
- 设置缩进的空格数为4: set shiftwidth=4

### 使用插件

要配置好看的vim，原生的配置可能功能不全，可以选择安装插件来完善配置，保证用户是你要配置的用户，接下来：

- 安装TagList插件,下载taglist\_xx.zip ,解压完成，将解压出来的doc的内容放到~/.vim/doc, 将解压出来的plugin下的内容拷贝到~/.vim/plugin
- 在~/.vimrc 中添加: `let Tlist_Show_One_File=1 let Tlist_Exit_Onlywindow=1 let Tlist_Use_Right_Window=1`
- 安装文件浏览器和窗口管理器插件: WinManager
- 下载winmanager.zip, 2.X版本以上的
- 解压winmanager.zip, 将解压出来的doc的内容放到~/.vim/doc, 将解压出来的plugin下的内容拷贝到~/.vim/plugin
- 在~/.vimrc 中添加 `let g:winManagerwindowLayout='FileExplorer|TagList nmap wm :WMToggle<cr>`
- 然后重启vim,打开~/XXX.c或~/XXX.cpp, 在normal状态下输入"wm", 你将看到上图的效果。 更具体移步: [点我](#), 其他手册, 请执行 `vimtutor` 命令。



## 参考资料

[Vim从入门到生逼\(vim from zero to hero\)](#)

## Linux编译器-gcc/g++使用

### 1. 背景知识

1. 预处理（进行宏替换）
2. 编译（生成汇编）
3. 汇编（生成机器可识别代码）
4. 连接（生成可执行文件或库文件）

### 2. gcc如何完成

格式 `gcc [选项] 要编译的文件 [选项] [目标文件]`

#### 预处理(进行宏替换)

- 预处理功能主要包括宏定义,文件包含,条件编译,去注释等。
- 预处理指令是以#号开头的代码行。
- 实例: `gcc -E hello.c -o hello.i`

- 选项“-E”，该选项的作用是让 gcc 在预处理结束后停止编译过程。
- 选项“-o”是指目标文件，“.i”文件为已经过预处理的C原始程序。

## 编译（生成汇编）

- 在这个阶段中,gcc 首先要检查代码的规范性、是否有语法错误等,以确定代码的实际要做的工作,在检查无误后,gcc 把代码翻译成汇编语言。
- 用户可以使用“-S”选项来进行查看,该选项只进行编译而不进行汇编,生成汇编代码。
- 实例: `gcc -S hello.i -o hello.s`

## 汇编（生成机器可识别代码）

- 汇编阶段是把编译阶段生成的“.s”文件转成目标文件
- 读者在此可使用选项“-c”就可看到汇编代码已转化为“.o”的二进制目标代码了
- 实例: `gcc -c hello.s -o hello.o`

## 连接（生成可执行文件或库文件）

- 在成功编译之后,就进入了链接阶段。
- 实例: `gcc hello.o -o hello`

## 在这里涉及到一个重要的概念:函数库

- 我们的C程序中,并没有定义“printf”的函数实现,且在预编译中包含的“stdio.h”中也只有该函数的声明,而没有定义函数的实现,那么,是在哪里实现“printf”函数的呢?
- 最后的答案是:系统把这些函数实现都被做到名为 libc.so.6 的库文件中去了,在没有特别指定时,gcc 会到系统默认的搜索路径“/usr/lib”下进行查找,也就是链接到 libc.so.6 库函数中去,这样就能实现函数“printf”了,而这也正是链接的作用

## 函数库一般分为静态库和动态库两种。

- 静态库是指编译链接时,把库文件的代码全部加入到可执行文件中,因此生成的文件比较大,但在运行时也就不再需要库文件了。其后缀名一般为“.a”
- 动态库与之相反,在编译链接时并没有把库文件的代码加入到可执行文件中,而是在程序执行时由运行时链接文件加载库,这样可以节省系统的开销。动态库一般后缀名为“.so”,如前面所述的 libc.so.6 就是动态库。gcc 在编译时默认使用动态库。完成了链接之后,gcc 就可以生成可执行文件,如下所示。 `gcc hello.o -o hello`
- gcc默认生成的二进制程序,是动态链接的,这点可以通过 `file` 命令验证。

## gcc选项

- -E 只激活预处理,这个不生成文件,你需要把它重定向到一个输出文件里面
- -S 编译到汇编语言不进行汇编和链接
- -c 编译到目标代码
- -o 文件输出到 文件
- -static 此选项对生成的文件采用静态链接
- -g 生成调试信息。GNU 调试器可利用该信息。
- -shared 此选项将尽量使用动态库,所以生成文件比较小,但是需要系统由动态库。
- -O0
- -O1
- -O2

- -O3 编译器的优化选项的4个级别，-O0表示没有优化,-O1为缺省值，-O3优化级别最高
- -w 不生成任何警告信息。
- -Wall 生成所有警告信息。

## gcc选项记忆

- esc,iso例子