vim的使用

如果没有vim，先下载：

```
sudo yum install vim
```

vim：就是一款编辑器，只负责写代码
gcc：就是一款编译器，只负责程序的编译
gdb：就是一款调试器，只负责程序的调试

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：
vim是一款多模式的编辑器
1.命令模式（默认），输入的任何内容基本都被当作命令

> 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

2.插入模式

> 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。

3.底行模式

> 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

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 光标后移
> • 输入i 不变
> • 输入o 新起一行

[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,想用光标键往回移动，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］ shift+4
- 按「^」：移动到光标所在行的“行首”［重要］ shift+6
- 按「w」：光标跳到下个字的开头［重要］（向后找）
- 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］（向前找）
- 按「#l」：光标移到该行的第#个位置，如：51,561
- 按［gg］：进入到文本开始［重要］
- 按［shift＋g］：进入文本末端 n+shift+g 任意行切换
- 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］= ctrl + x n+dd 剪切n行
- 「#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 + z
- 「ctrl + r」: 撤销的恢复［重要］ = ctrl + y

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
- shift + ~ ：更改光标所在处的字大小写

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］

文件操作

> • : vs +「文件名」 打开另一个文件
> • ctrl + ww/ctrl +wl两个桌面的切换
> • :!gcc「文件名」 不退出vim，直接进行编译
> • :man n「命令名」查看第n行的命令的帮助文档

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。

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

命令配置

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

gcc/g++的使用

1. 背景知识

c文件 --> 可执行程序 --> 程序的翻译（编译）

1. 预处理（进行宏替换）

a.头文件展开

```
gcc -E bianyi.c -o mybin.i 把临时结果放到mybin.i
```

b.去注释

c.宏替换

d.条件编译

2. 编译（生成汇编）

把C变成汇编语言

```
gcc -S bianyi.c -o mybin.s
或gcc -S mybin.i -o mybin.s
```

3. 汇编（生成机器可识别代码）

汇编变成.o二进制文件，不可执行（windows系统下的.obj文件）

```
gcc -c bianyi.c -o mybin.o
或gcc -c mybin.s -o mybin.o
```

```
od mybin.o 以八进制的形式显示mybin.o文件
```

4. 链接（生成可执行文件或库文件）

引入在代码中使用的第三方库 -- c库

```
gcc bianyi.c -o mybin
```

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“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,而这也就是链接的作用

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。