Linux 小组活动之二

Vi 使用及 Shell 编程

报告人：阴小波 等

内容简介：

1、vi 及其使用（高兴誉、赵礼）
2、Shell 简述（刘歆）
3、变量及注释（刘歆）
4、最简单的例子（刘歆）
5、流程控制（阴小波）
6、函数（阴小波）
7、实例及其分析（阴小波）
8、感谢及答疑（阴小波）

Vi 简介

vi 是个可视化的编辑器(visual edit). vi 是 UNIX 下面的缺省编辑器。Vi 提供两种操作模式： 输入模式（insert mode），指令模式（command mode）.当使用者进入 vi 后，即处于指令模式下，此刻键入任何字皆被视为指令；a 或 i 键用来进入输入模式， 在输入模式下，每个按键代表输入字符，离开输入模式用<ESC>键。

vi 技巧

1. 文件加密 :X
然后系统会提示输入密码，输入密码后存盘退出。
下次编辑时请使用 vi -x filename 打开被加密的文件。

2. 上下两行合并 J
删除所有行 dG
交换两个字符位置 xp
上下两行调换 ddp
注：xp 和 ddp 应该理解为删除一个字符(x)和删除一行(dd),然后在 p(paste),这样便不需记新命令了.

3. 从当前位置删除到行尾 d$
从当前位置复制到行尾 y$
如果要粘贴到其他地方 p 就可以了

4. 文件移动
:line1,line2mline3: 将 line1--line2 之间的内容移到 line3 下 相当于 word 的剪切，
:line1,line2tline3: 将 line1--line2 之间的内容粘贴到 line3 下，相当于 word 的粘贴。
:n1,n2w filename：把行 N1-N2 之间内容写入新文件 filename 中。

5. 自动添加字符串 :ab string strings
例如 ":ab usa United States of America" 。
当你在文件里插入 usa 时 ，United States of America 就蹦出来了

6. 替换命令操作 :g/s1/s/s2/s3/g
第一个 g 表示对每一个包括 s1 的行都进行替换，第二个 g 表示对每一个包括 s1 的行的所有的 s2 都用 s3 替换，s 表示替换，s2 是要被替换的字符串，他可以和 s1 相同（如果相同的话用//代替），s3 是替换字符串
注：在整个文件中替换特定字符串 :%s/old_word/new_word/g

7. 寄存器操作
"?nyy：将当前行及其下 n 行的内容保存到寄存器？中，其中?为一个字母，n 为一个数字

"?nyw：将当前行及其下 n 个字保存到寄存器？中，其中?为一个字母，n 为一个数字

"?nyl：将当前行及其下 n 个字符保存到寄存器？中，其中?为一个字母，n 为一个数字

"?p：取出寄存器？中的内容并将其放到光标位置处。这里？可以是一个字母，也可以是一个数字

ndd：将当前行及其下共 n 行文本删除，并将所删内容放到 1 号删除寄存器中。

注：vi 还有有编号的删除缓冲区。撤消命令只能恢复上一次的删除，当且仅当这个删除是最后一次操作。但是，vi 可以把最近 9 次删除的内容都放入 9 个编号从 1 到 9 的缓冲区内。最近一次删除保存在缓冲区 1，次近的在缓冲区 2，依此类推。例如，"2p 可以放置第二个删除缓冲区中的内容。

8. 排序

如有一文件，内容为：

333
222
334
444
111
553
554
233

运行命令:%!sort 文件内容为：

111
222
233
333
334
444
553
554

9. vi 环境选项

noautoindent nomodelines noshowmode
autoprint nonumber noslowopen
noautowrite nonovice tabstop=8
nobeautify nooptimize taglength=0
directory=/var/tmp paragraphs=IPLPPPQPP LIpplpipnpbtags=tags /usr/lib/tags
noedcompatible prompt tagstack
noerrorbells noreadonly term=vt100
noexrc redraw noterse
flash remap timeout
hardtabs=8 report=5 ttytype=vt100
noignorecase scroll=11 warn
nolisp sections=NHSHH HUuhsh+c window=23
nolist shell=/bin/ksh wrapscan

magic shiftwidth=8 wrapmargin=0
mesg noshowmatch nowriteany

For C-Shell:
setenv EXINIT "set nu"
For Bourne or Korn Shell:
EXINIT="set nu"; export EXINIT
For Korn Shell Only (alternate method):
typeset -x EXINIT="set nu"
在 .profile 里设置 vi 的环境选项 , 以上均测试过

10. 重复上一次操作 .:
恢复上一次的操作 u
11. :! shell-command 执行 SHELL 命令的操作

举例

1．删除文件中的几行
到删除的开始处 ma
到删除的结束处 d’a
2．拷贝文件中的几行
到拷贝的开始处 ma
到拷贝的结束处 y’a
3．替换文件中所有匹配的字符
替换文件中的 lesson 为 course
s/lesson/course/g
注意：可能替换含有 lesson 的其他词
4．替换文件中一个区域的字符
start_line, end_line %s/lesson/course/g 或者
采用宏的方式
‘a,. %s/lesson/course/g #需要定义宏；’.’是指光标所在行
5．替换一个文件中某些词
比如替换 none 为 nothing
查找 /none
替换 cw，输入 nothing，按键盘 Esc
继续查找 n
替换 .
6．重复执行一个 shell 命令
编译一个程序
保存编辑后的文件 :w
编译文件 :!make
修改文件后，在保存 :w
再编译 :!!

7．在两个文件互拷贝

第一种方法：

编辑源文件　　vi source_file

到拷贝的部分开始处　　ma

到拷贝的部分结束处　　:.,’a w tempfile

退出当前文件　　:q

编辑目的文件　　vi dest_file

到拷贝的地方　　:r tempfile

第二种方法：

编辑源文件　　vi source_file

到拷贝的部分开始处　　ma

到拷贝的部分结束处　　“ay’a

不退出源文件编辑目的文件　:e dest_file

到拷贝的地方　　“ap

vi 命令一览表

1.进入 vi 的命令:

vi filename :打开或新建文件，并将光标置于第一行首
vi +n filename ：打开文件，并将光标置于第 n 行首
vi + filename ：打开文件，并将光标置于最后一行首
vi +/pattern filename：打开文件，并将光标置于第一个与 pattern 匹配的串处
vi -r filename：在上次正用 vi 编辑时发生系统崩溃，恢复 filename
vi filename....filename：打开多个文件，依次进行编辑

2.移动光标类命令:

h：光标左移一个字符
l ：光标右移一个字符
space：光标右移一个字符
Backspace：光标左移一个字符
k 或 Ctrl+p：光标上移一行
j 或 Ctrl+n ：光标下移一行
Enter ：光标下移一行
w 或 W ：光标右移一个字至字首
b 或 B ：光标左移一个字至字首
e 或 E ：光标右移一个字至字尾
) ：光标移至句尾
(：光标移至句首
)：光标移至段落开头
{：光标移至段落结尾
nG：光标移至第 n 行首
n+：光标下移 n 行
n-：光标上移 n 行
n$：光标移至第 n 行尾
H ：光标移至屏幕顶行
M ：光标移至屏幕中间行
L ：光标移至屏幕最后行
0：（注意是数字零）光标移至当前行首
$：光标移至当前行尾

3.屏幕翻滚类命令:

Ctrl+u：向文件首翻半屏
Ctrl+d：向文件尾翻半屏
Ctrl+f：向文件尾翻一屏
Ctrl＋b：向文件首翻一屏
nz：将第 n 行滚至屏幕顶部，不指定 n 时将当前行滚至屏幕顶部。

4.插入文本类命令:

i ：在光标前
I ：在当前行首
a：光标后
A：在当前行尾
o：在当前行之下新开一行
O：在当前行之上新开一行
r：替换当前字符
R：替换当前字符及其后的字符，直至按 ESC 键
s：从当前光标位置处开始，以输入的文本替代指定数目的字符
S：删除指定数目的行，并以所输入文本代替之
ncw 或 nCW：修改指定数目的字
nCC：修改指定数目的行

5.删除命令:

ndw 或 ndW：删除光标处开始及其后的 n-1 个字
do：删至行首
d$：删至行尾
ndd：删除当前行及其后 n-1 行
x 或 X：删除一个字符，x 删除光标后的，而 X 删除光标前的
Ctrl+u：删除输入方式下所输入的文本

6.搜索及替换命令:

/pattern：从光标开始处向文件尾搜索 pattern
?pattern：从光标开始处向文件首搜索 pattern
n：在同一方向重复上一次搜索命令
N：在反方向上重复上一次搜索命令
：s/p1/p2/g：将当前行中所有 p1 均用 p2 替代
：n1,n2s/p1/p2/g：将第 n1 至 n2 行中所有 p1 均用 p2 替代
：g/p1/s//p2/g：将文件中所有 p1 均用 p2 替换

7.选项设置:

all：列出所有选项设置情况
term：设置终端类型
ignorance：在搜索中忽略大小写
list：显示制表位(Ctrl+I)和行尾标志（$）
number：显示行号
report：显示由面向行的命令修改过的数目
terse：显示简短的警告信息
warn：在转到别的文件时若没保存当前文件则显示 NO write 信息

nomagic：允许在搜索模式中，使用前面不带“\”的特殊字符
nowrapscan：禁止 vi 在搜索到达文件两端时，又从另一端开始
mesg：允许 vi 显示其他用户用 write 写到自己终端上的信息

8.最后行方式命令:

：n1,n2 co n3：将 n1 行到 n2 行之间的内容拷贝到第 n3 行下
：n1,n2 m n3：将 n1 行到 n2 行之间的内容移至到第 n3 行下
：n1,n2 d ：将 n1 行到 n2 行之间的内容删除
：w ：保存当前文件
：e filename：打开文件 filename 进行编辑
：x：保存当前文件并退出
：q：退出 vi
：q!：不保存文件并退出 vi
：!command：执行 shell 命令 command
：n1,n2 w!command：将文件中 n1 行至 n2 行的内容作为 command 的输入并执行之，若不指定 n1，n2，则表示将整个文件内容作为 command 的输入
：r!command：将命令 command 的输出结果放到当前行

9.寄存器操作:

"?nyy：将当前行及其下 n 行的内容保存到寄存器？中，其中?为一个字母，n 为一个数字
"?nyw：将当前行及其下 n 个字保存到寄存器？中，其中?为一个字母，n 为一个数字
"?nyl：将当前行及其下 n 个字符保存到寄存器？中，其中?为一个字母，n 为一个数字
"?p：取出寄存器？中的内容并将其放到光标位置处。这里？可以是一个字母，也可以是一个数字
ndd：将当前行及其下共 n 行文本删除，并将所删内容放到 1 号删除寄存器中。

Shell 编程

如何开始:

如同其他语言一样，通过我们使用任意一种文字编辑器，比如 nedit、kedit、emacs、vi 等来编写我们的 shell 程序。程序必须以下面的行开始（必须方在文件的第一行）：

```
#!/bin/sh
```

符号#!用来告诉系统它后面的参数是用来执行该文件的程序。在这个例子中我们使用/bin/sh 来执行程序。当编辑好脚本时，如果要执行该脚本，还必须使其可执行。要使脚本可执行：

```
chmod +x filename
```

然后，您可以通过输入： ./filename 来执行您的脚本。

注释:

在进行 shell 编程时，以#开头的句子表示注释，直到这一行的结束。我们真诚地建议您在程序中使用注释。如果您使用了注释，那么即使相当长的时间内没有使用该脚本，您也能在很短的时间内明白该脚本的作用及工作原理。

变量:

在其他编程语言中您必须使用变量。在 shell 编程中，所有的变量都由字符串组成，并且您不需要对变量进行声明。要赋值给一个变量，您可以这样写：

```
变量名=值
```

取出变量值可以加一个美元符号（$）在变量前面，如：

```
$HOME
```

四、实例：

```
#!/bin/sh
a="hello world"
echo "A is:"
echo $a
```

在您的编辑器中输入以上内容，然后将其保存为一个文件 first。之后执行

```
chmod +x first
```

使其可执行，最后输入./first 执行该脚本。这个脚本将会输出：

```
A is:
hello world
```

有许多变量是系统自动设定的，这将在后面使用这些变量时进行讨论。如果您需要处理数学表达式，那么您需要使用诸如 expr 等程序（见下面）。除了一般的仅在程序内有效的 shell 变量以外，还有环境变量。由 export 关键字处理过的变量叫做环境变量。我们不对环境变量进行讨论，因为通常情况下仅仅在登录脚本中使用环境变量。

五、Shell 命令和流程控制

在 shell 脚本中可以使用三类命令：

1)Unix 命令:

虽然在 shell 脚本中可以使用任意的 unix 命令，但是还是由一些相对更常用的命令。这些命令通常是用来进行文件和文字操作的。

常用命令语法及功能

echo "some text": 将文字内容打印在屏幕上

ls: 文件列表

wc -l file 计算文件行数

wc -w file 计算文件中的单词数

wc -c file: 计算文件中的字符数

cp sourcefile destfile: 文件拷贝

mv oldname newname : 重命名文件或移动文件

rm file: 删除文件

grep 'pattern' file: 在文件内搜索字符串比如：grep 'searchstring' file.txt

cut -b colnum file: 指定欲显示的文件内容范围，并将它们输出到标准输出设备比如：输出每行第 5 个到第 9 个字符 cut -b5-9 file.txt 千万不要和 cat 命令混淆，这是两个完全不同的命令

cat file.txt: 输出文件内容到标准输出设备（屏幕）上

file somefile: 得到文件类型

read var: 提示用户输入，并将输入赋值给变量

sort file.txt: 对 file.txt 文件中的行进行排序

expr: 进行数学运算 Example: add 2 and 3expr 2 "+" 3

find: 搜索文件比如：根据文件名搜索 find . -name filename -print

tee: 将数据输出到标准输出设备(屏幕) 和文件比如：somecommand | tee outfile

sed: Sed 是一个基本的查找替换程序。可以从标准输入（比如命令管道）读入文本，并将结果输出到标准输出(屏幕)。该命令采用正则表达式(见参考)进行搜索。不要和 shell 中的通配符相混淆。比如：将 linuxfocus 替换为 LinuxFocus ：cat text.file | sed 's/linuxfocus/LinuxFocus/' > newtext.file

2) 概念: 管道, 重定向和 backtick

这些不是系统命令，但是他们真的很重要。

管道（|）将一个命令的输出作为另外一个命令的输入。

grep "hello" file.txt | wc -l

在 file.txt 中搜索包含有"hello"的行并计算其行数。在这里 grep 命令的输出作为 wc 命令的输入。当然您可以使用多个命令。

重定向：将命令的结果输出到文件，而不是标准输出（屏幕）。

> 写入文件并覆盖旧文件

>> 加到文件的尾部，保留旧文件内容。

反短斜线：使用反短斜线可以将一个命令的输出作为另外一个命令的一个命令行参数。

命令：find . -mtime -1 -type f -print

用来查找过去 24 小时（-mtime -2 则表示过去 48 小时）内修改过的文件。如果您想将所有查找到的文件打一个包，则可以使用以下脚本：

```
#!/bin/sh
# The ticks are backticks (`) not normal quotes ('):
tar -zcvf lastmod.tar.gz `find . -mtime -1 -type f -print`
```

3) 流程控制

a. "if" 表达式 如果条件为真则执行 then 后面的部分：

```
if ....; then
   ....
elif ....; then
   ....
else
   ....
fi
```

大多数情况下，可以使用测试命令来对条件进行测试。比如可以比较字符串、判断文件是否存在及是否可读等等…通常用" [ ] "来表示条件测试。注意这里的空格很重要。要确保方括号的空格。

[-f "somefile"] ：判断是否是一个文件

[-x "/bin/ls"] ：判断/bin/ls 是否存在并有可执行权限

[-n "$var"] ：判断$var 变量是否有值

["$a" = "$b"] ：判断$a 和$b 是否相等

*）执行 man test 可以查看所有测试表达式可以比较和判断的类型。

直接执行以下脚本：

```
#!/bin/sh
if [ "$SHELL" = "/bin/bash" ]; then
   echo "your login shell is the bash (bourne again shell)"
else
   echo "your login shell is not bash but $SHELL"
fi
```

变量$SHELL 包含了登录 shell 的名称，我们和/bin/bash 进行了比较。

b. case 表达式可以用来匹配一个给定的字符串，而不是数字。

```
case ... in
...) do something here ;;
esac
```

让我们看一个例子。 file 命令可以辨别出一个给定文件的文件类型，比如：

```
file lf.gz
```

这将返回：

```
lf.gz: gzip compressed data, deflated, original filename,
last modified: Mon Aug 27 23:09:18 2001, os: Unix
```

我们利用这一点写了一个叫做 smartzip 的脚本，该脚本可以自动解压 bzip2, gzip 和 zip 类型的压缩文件：

```
#!/bin/sh
ftype=`file "$1"`
case "$ftype" in
"$1: Zip archive"*)
    unzip "$1" ;;
"$1: gzip compressed"*)
    gunzip "$1" ;;
```

```
"$1: bzip2 compressed"*)
    bunzip2 "$1" ;;
*) error "File $1 can not be uncompressed with smartzip";;
esac
```

您可能注意到我们在这里使用了一个特殊的变量$1。该变量包含了传递给该程序的第一个参数值。也就是说，当我们运行：

```
smartzip articles.zip
```

$1 就是字符串 articles.zip

c. while-loop 将运行直到表达式测试为真。will run while the expression that we test for is true. 关键字"break" 用来跳出循环。而关键字”continue”用来不执行余下的部分而直接跳到下一个循环。

d. for-loop 表达式查看一个字符串列表（字符串用空格分隔）然后将其赋给一个变量：

```
for var in ....; do
  ....
done
```

在下面的例子中，将分别打印 ABC 到屏幕上：

```
#!/bin/sh
for var in A B C ; do
  echo "var is $var"
done
```

######引号#######

在向程序传递任何参数之前，程序会扩展通配符和变量。这里所谓扩展的意思是程序会把通配符（比如*）替换成合适的文件名，它变量替换成变量值。为了防止程序作这种替换，您可以使用引号：让我们来看一个例子，假设在当前目录下有一些文件，两个 jpg 文件，mail.jpg 和 tux.jpg。

```
#!/bin/sh
echo *.jpg
```

这将打印出"mail.jpg tux.jpg"的结果。

引号（单引号和双引号）将防止这种通配符扩展：

```
#!/bin/sh
echo "*.jpg"
echo '*.jpg'
```

这将打印"*.jpg" 两次。

Here documents

当要将几行文字传递给一个命令时，here documents 一种不错的方法。对每个脚本写一段帮助性的文字是很有用的，此时如果我们有那个 here documents 就不必用 echo 函数一行行输出。 一个 "Here document" 以 << 开头，***后面接上一个字符串，这个字符串还必须出现在 here document 的末尾***。下面是一个例子，在该例子中，我们对多个文件进行重命名，并且使用 here documents 打印帮助：

```
#!/bin/sh
```

```
# we have less than 3 arguments. Print the help text:
if [ $# -lt 3 ] ; then
cat <<HELP
ren -- renames a number of files using sed regular expressions
USAGE: ren 'regexp' 'replacement' files...
EXAMPLE: rename all *.HTM files in *.html:
  ren 'HTM$' 'html' *.HTM
HELP
  exit 0
fi
OLD="$1"
NEW="$2"
# The shift command removes one argument from the list of
# command line arguments.
shift
shift
# $* contains now all the files:
for file in $*; do
    if [ -f "$file" ] ; then
      newfile=`echo "$file" | sed "s/${OLD}/${NEW}/g"`
      if [ -f "$newfile" ]; then
        echo "ERROR: $newfile exists already"
      else
        echo "renaming $file to $newfile ..."
        mv "$file" "$newfile"
      fi
    fi
done
```

这是一个复杂一些的例子。让我们详细讨论一下。第一个 if 表达式判断输入命令行参数是否小于 3 个（特殊变量$# 表示包含参数的个数）。如果输入参数小于 3 个，则将帮助文字传递给 cat 命令，然后由 cat 命令将其打印在屏幕上。打印帮助文字后程序退出。 如果输入参数等于或大于 3 个，我们就将第一个参数赋值给变量 OLD，第二个参数赋值给变量 NEW。下一步，我们使用 shift 命令将第一个和第二个参数从参数列表中删除，这样原来的第三个参数就成为参数列表$*的第一个参数。然后我们开始循环，命令行参数列表被一个接一个地被赋值给变量$file。接着我们判断该文件是否存在，如果存在则通过 sed 命令搜索和替换来产生新的文件名。然后将反短斜线内命令结果赋值给 newfile。这样我们就达到了我们的目的：得到了旧文件名和新文件名。然后使用 mv 命令进行重命名。

六、函数

如果您写了一些稍微复杂一些的程序,您就会发现在程序中可能在几个地方使用了相同的代码，并且您也会发现，如果我们使用了函数，会方便很多。一个函数是这个样子的：

```
functionname()
{
# inside the body $1 is the first argument given to the function
```

```
# $2 the second ...
body
}
```

您需要在每个程序的开始对函数进行声明，在后面的例子中再详细介绍。

七、实例

现在我们来讨论编写一个脚本的一般步骤。任何优秀的脚本都应该具有帮助和输入参数。并且写一个伪脚本（framework.sh），该脚本包含了大多数脚本都需要的框架结构，是一个非常不错的主意。这时候，在写一个新的脚本时我们只需要执行一下 copy 命令：cp framework.sh myscript，然后再插入自己的函数。

让我来看一个具体的例子：**二进制到十进制的转换**

脚本 b2d 将二进制数（比如 1101）转换为相应的十进制数。这也是一个用 expr 命令进行数学运算的例子：

```
#!/bin/sh
# vim: set sw=4 ts=4 et:
help()
{
  cat <<HELP
b2h -- convert binary to decimal

USAGE: b2h [-h] binarynum

OPTIONS: -h help text

EXAMPLE: b2h 111010
will return 58
HELP
  exit 0
}

error()
{
    # print an error and exit
    echo "$1"
    exit 1
}

lastchar()
{
    # return the last character of a string in $rval
    if [ -z "$1" ]; then
        # empty string
        rval=""
        return
```

```
    fi
    # wc puts some space behind the output this is why we need sed:
    numofchar=`echo -n "$1" | wc -c | sed 's/ //g' `
    # now cut out the last char
    rval=`echo -n "$1" | cut -b $numofchar`
}

chop()
{
    # remove the last character in string and return it in $rval
    if [ -z "$1" ]; then
        # empty string
        rval=""
        return
    fi
    # wc puts some space behind the output this is why we need sed:
    numofchar=`echo -n "$1" | wc -c | sed 's/ //g' `
    if [ "$numofchar" = "1" ]; then
        # only one char in string
        rval=""
        return
    fi
    numofcharminus1=`expr $numofchar "-" 1`
    # now cut all but the last char:
    rval=`echo -n "$1" | cut -b 0-${numofcharminus1}`
}

while [ -n "$1" ]; do
case $1 in
    -h) help;shift 1;; # function help is called
    --) shift;break;; # end of options
    -*) error "error: no such option $1. -h for help";;
    *) break;;
esac
done

# The main program
sum=0
weight=1
# one arg must be given:
[ -z "$1" ] && help
binnum="$1"
binnumorig="$1"
```

```
while [ -n "$binnum" ]; do
    lastchar "$binnum"
    if [ "$rval" = "1" ]; then
        sum=`expr "$weight" "+" "$sum"`
    fi
    # remove the last position in $binnum
    chop "$binnum"
    binnum="$rval"
    weight=`expr "$weight" "*" 2`
done

echo "binary $binnumorig is decimal $sum"
#
```

该脚本使用的算法是利用十进制和二进制数权值 (1,2,4,8,16,..)，比如二进制"10"可以这样转换成十进制： 0 * 1 + 1 * 2 = 2

为了得到单个的二进制数我们是用了 lastchar 函数。该函数使用 wc -c 计算字符个数，然后使用 cut 命令取出末尾一个字符。Chop 函数的功能则是移除最后一个字符。

我们希望您现在可以开始写您自己的 shell 脚本，希望您玩得开心。

谢谢大家

2005 年 4 月 7 日