第三章 vi使用与Shell编程

1. vi 使用

- 1.1 vi的启动

  vi [filenames]

  例：

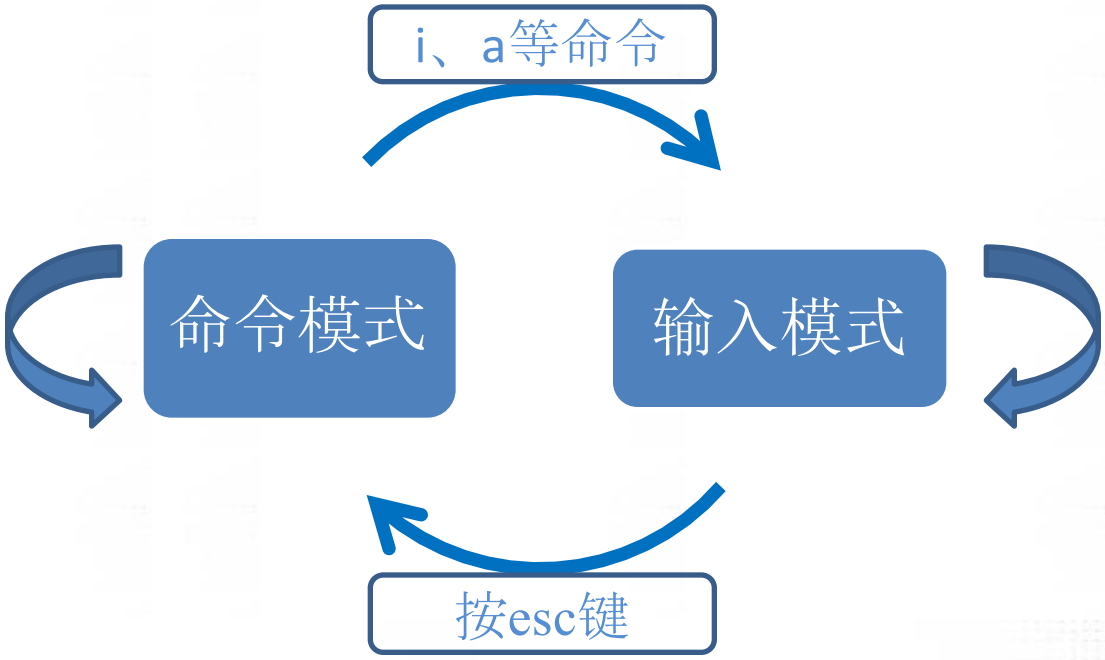
  $ vi abc.txt

  最常用的格式，vi后跟欲编辑的文件名

- vi有搜索命令，可以用来浏览文本文件，比more, less等更方便.

1. vi 使用

- 1.2 vi的工作方式
  - vi的工作方式分**命令模式**和**输入模式**。vi启动后就进入命令模式；



1. vi 使用

- 1.2 vi的工作方式
  - 处于**命令模式**时，用户键入的内容被当作vi的命令来解释，一般处于命令模式下按键无回显（以冒号打头的命令和查找命令除外）。编辑命令**i**，**a**等，可以从命令模式转到输入模式；
  - 处于**输入模式**时，用户键入的所有内容全部作为输入的正文内容，用户可以输入多行，每输入完一行后按回车键转入下一行，正文输入时有回显。输入完毕，按键盘左上角的**esc键**，返回到命令模式。

1. vi 使用

- 1.3 vi的编辑命令
  - 当vi处于命令模式时，用户的按键不回显，被解释成编辑命令，vi大约有100多个编辑命令。下面介绍的vi命令子集，足可以完成一般的编辑任务。

1. vi 使用

- 1.3 vi的编辑命令

  1.3.1 正文插入命令

  - 命令**i**，在当前光标处插入 (Insert) 正文段，进入输入模式，直至按**esc**键返回命令模式；
  - 命令**a**，在当前光标后追加(Append) 正文段，进入输入模式，直至按**esc**键返回命令模式；
  - 命令**o**，在当前行之下处插入 (Open) 新行，进入输入模式，直至按**esc**键返回命令模式；
  - 命令**O**，在当前行之上处插入 (Open) 新行，进入输入模式，直至按**esc**键返回命令模式。

1. vi 使用

- 1.3 vi的编辑命令

1.3.2 光标移动命令

– 单字符移动
  - h 光标左移一列
  - j 光标下移一行
  - k 光标上移一行
  - l 光标右移一列

– 多字符移动
  - 3h 光标左移3列
  - 10j 光标下移10行
  - 13k 光标上移13行
  - 20l 光标右移20列

1. vi 使用

- 1.3 vi的编辑命令

1.3.3 翻页命令

在vi中，把向文件尾方向定义为“向前”，向文件头方向定义为“向后”，这与许多人的习惯不同。

^B 向后翻页(Backward)

^F 向前翻页(Forward)

^U 向上翻半页(Up)

^D 向下翻半页(Down)

2^B 向后翻2页(Backward)

5^F 向前翻5页(Forward)

字母之前的“^”表示Ctrl键

可以实现翻多页

1. vi 使用

- 1.3 vi的编辑命令

1.3.4 将光标移至当前行首^

1.3.5 将光标移至当前行尾$

1.3.6 移到右一个单词 w W

1.3.7 移到左一个单词 b B

w、W、b、B也可以使用5w、10W、3b、13B形式的命令

小写命令的w和b，以非字母、数字、下划线之外的所有字符作为“单词”分界符。

大写命令的W和B，以空白符作为“单词”分界符。

1. vi 使用

- 1.3 vi的编辑命令

1.3.8 将光标移动到指定行

:123 将光标定位到第123行

:$ 将光标定位到文件末尾

:$-10 将光标定位到文件倒数第10行

:.10 将光标向下移10行

“$”代表文件末尾而“.”代表当前行；都可以使用“+”或“-”进行相对的位移光标（“+”可以省略）。

1. vi 使用

- 1.3 vi的编辑命令

1.3.9 括号匹配命令%

先把光标移到一个大括号（或括号，或方括号）上，按%键，则光标自动定位到与它配对的那一个括号，对编写和检查C语言的源程序非常有用。

1. vi 使用

- 1.3 vi的编辑命令

1.3.10 删除命令

x	删除光标所在的字符
5x	删除光标所在开始的5个字符
dd	删除当前行
4dd	删除当前行开始的4行
d$	从当前光标处删除到行尾
d^	从当前光标处删除到行首
dw	删除一个单词

1. vi 使用

- 1.3 vi的编辑命令

1.3.11 字符替换命令

r　　替换光标处字符的命令

例：

ra

rarbrc

R　　替换多个字符的命令

例：

Rabc

然后按“esc”键

1. vi 使用

- 1.3 vi的编辑命令

1.3.12 取消和重复命令

u　　取消上次的命令(undo)

.　　重复执行上次的命令

1. vi 使用

- 1.3 vi的编辑命令

1.3.13 段落的删除、复制、粘贴和移动命令

dd 行删除命令(delete)

例：:11,13dd

删除11至13行

co 段落的复制命令(copy)

例：:11,13co15

复制11至13行到15行后

m 段落的移动命令(move)

例：:11,13m15

移动11至13行到15行后

1. vi 使用

- 1.3 vi的编辑命令

1.3.13 剪贴板功能

d 行删除命令(delete)

例：:11,13d

删除11至13行

y 抽取命令(yank)

例：:11,13y

复制11至13行用法同“d”命令

p 粘贴命令(paste)

例：p

将“d”或“y”操作的行粘贴到当前光标处

1. vi 使用

- 1.3 vi的编辑命令

1.3.14 查找命令

/　　查找“/”后面跟的内容

例：

/abc

在文章中查找“abc”

n　　向后查找

N　　向前查找

“n”和“N”命令必须在“/”命令之后执行

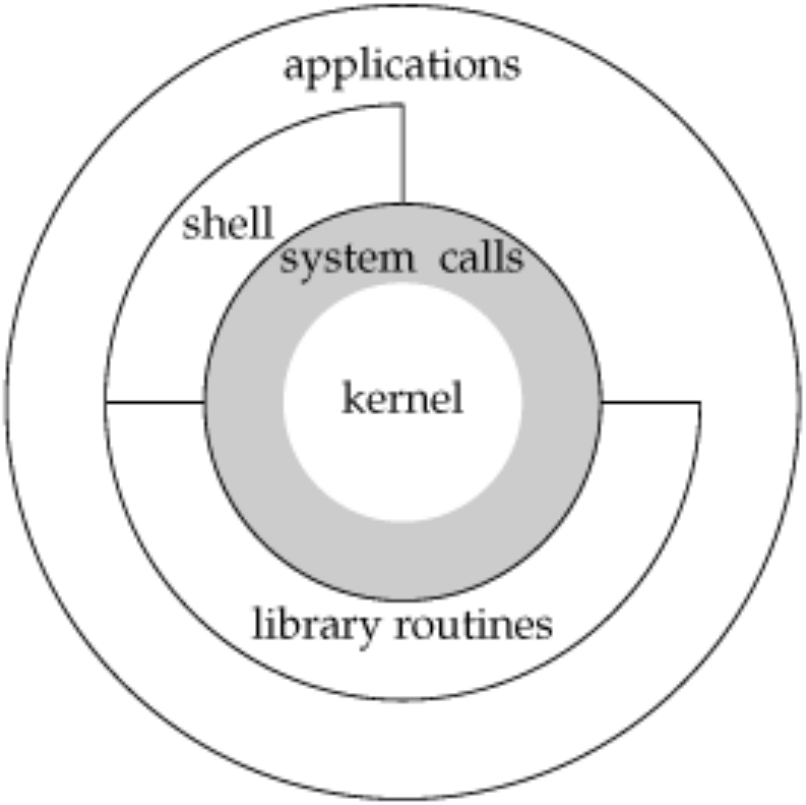
1. vi 使用

- 1.4 vi的文件命令

ZZ	保存文件并退出vi编辑状态
:wq	保存文件并退出vi编辑状态
:w	只保存文件而不退出vi编辑状态
:q！	不保存文件而强行退出vi编辑状态
:r filename	读入filename文件内容到当前行

2. Shell 编程

- 2.1 什么是Shell
  - Shell是一个命令行解释器，为系统解释用户的操作命令；
  - Shell是用户使用UNIX系统的桥梁；
  - Shell既是一种命令语言，又是一种程序设计语言；



2. Shell 编程

- 2.2 什么是Shell编程
  - Shell编程是利用判断、流程控制等方法把多个Shell命令有机的组织成Shell脚本；
  - Shell脚本类似于Windows系统中的批处理程序，通过执行Shell脚本来完成一系列Shell命令；

例：

```
#!/bin/sh
#########################
# Name: echohello.sh
# Usage: print 'Hello, World'
# Author: Gene
# Date: 2005-03-18
#########################
echo 'Hello, World!'
```

2. Shell 编程

- 2.3 Shell脚本的执行
  - 脚本文件本身是一个文本文件，不可能直接执行。
  - 当脚本文件具有可执行属性，用户将它执行的时候，系统会启动shell程序文件/bin/sh，运行/bin/sh文件中的CPU指令来解释执行脚本文件中的命令。
  - 脚本文件的第一个命令需要指明Shell命令解释程序：#!/bin/sh，“#!”必须出现在本文的最开头。

2. Shell 编程

- 2.3 Shell脚本的执行
  - 三种方法可以执行脚本文件

例：

$sh < echohello.sh

$sh echohello.sh

$./echohello.sh

需要为echohello.sh脚本文件赋予“可执行”属性

2. Shell 编程

- 2.3 Shell脚本的执行
  - 用vi编辑了Shell脚本之后，由于Shell脚本没有“可执行”属性，所以还无法执行；

```
[root@localhost UnixProgramming]# ls -al echohello.sh
-rw-r--r-- 1 root root 204 2008-08-29 15:54 echohello.sh
```

  - 运行chmod命令为“echohello.sh”Shell脚本赋予“可执行”属性

    例：chmod a+x echohello.sh

```
[root@localhost UnixProgramming]# chmod a+x echohello.sh
[root@localhost UnixProgramming]# ls -al echohello.sh
-rwxr-xr-x 1 root root 204 2008-08-29 15:54 echohello.sh
```

2. Shell 编程

- 2.4 Shell的变量的定义、赋值与引用
  - 变量的定义与赋值

    变量名以字母开头,由字母、数字及下划线组成;
    变量名可以包含数字，但不能以数字打头。

    例： ux=hello
  - 变量的引用

    在变量名前加“$”

    例： echo $ux

    例： echo ${ux}world 或 echo “$ux”world

2. Shell 编程

- 2.4 Shell的变量的定义、赋值与引用
  - 转义字符“\”

例： echo $ux

结果为：hello

例： echo \$ux

结果为： $ux

  - 清除变量unset

例： unset ux

2. Shell 编程

- 2.5 Shell的变量中三种引号的作用
  - 单引号(’ ’)：屏蔽任意字符的特殊含义；
  - 双引号(“ ”)：屏蔽任意字符的特殊含义，除了$、` 、\；
  - 反引号(` `)：（一般在键盘最左上角esc键下方）其间的命令可作为执行结果进行赋值，与( )的功能一样；

例：echo ‘$ux’　　结果为：$ux

echo “$ux”　　结果为：hello

echo `$ux`　　结果为：报错，找不到$ux这个命令

2. Shell 编程

- 2.6 系统默认的内置变量
  - $# 传递到脚本的参数个数；
  - $* 以一个单字符串显示所有向脚本传递的参数；
  - $$ 脚本运行的当前进程ID号；
  - $! 后台运行的最后一个进程的进程ID号；
  - $@ 与$*相同，但是以多个字符串显示所有向脚本传递的参数，每个字符串为一个参数；
  - $- 显示shell使用的当前选项，与set命令功能相同；
  - $? 显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误；

2. Shell 编程

- 2.6 系统默认的内置变量

例：

```
#!/bin/sh
##################################################
[root@localhost UnixProgramming]# ./var.sh para1 para2 para3
There are 3 parameters
The parameters are para1 para2 para3
The Shell's PID is 4161
Test finished!
##################################################

echo There are $# parameters
echo The parameters are $@
echo The Shell\'s PID is $$
echo Test finished!
```

2. Shell 编程

- 2.7 测试文件状态
  - test condition
  - [condition]

“[”或“]”与条件“condition”之间必须有空格

- 测试内容：
  - -d 目录
  - -s 文件非空
  - -f 正规文件
  - -w 可写
  - -L 符号链接
  - -u 文件有suid设置
  - -r 可读
  - -x 可执行

2. Shell 编程

- 2.7 测试文件状态

- 最简单的条件判断
  - cmd1 && cmd2

    若cmd1执行成功（返回码为0）则执行cmd2，否则不执行cmd2。
  - cmd1 || cmd2
  - 若cmd1执行失败（返回码不为0）则执行cmd2，否则不执行cmd2。

- 2.7 测试文件状态
  - 测试文件状态时使用的逻辑操作符

    -a：逻辑与，操作符两边均为真，结果为真；只要有一边为假，结果为假；两边均为假，结果为假；

    -o：逻辑或，操作符两边为假，结果为假；只要有一边为真，结果为真；两边均为真，结果为真；

    !：逻辑否，条件为假，结果为真。

    例：**[-r** main.c **-a -f** main.c **]** && echo main.c is a file

    [-w main.c **-o -r** main.c **]** && echo Get it

2. Shell 编程

- 2.7 测试文件状态

例：

测试多个文件状态的逻辑操作命令

```
[ -r  main.c -a -f  main.c ] && echo main.c is a readable file

[ -w  main.c -o -r  main.c ] && echo mai.c is a normal file

[ ! -d main -a -x main] || echo  main is an executable file
```

“!”与“[”之间与“-x”之间都必须有空格

2. Shell 编程

- 2.8 测试字符串
  - **test op** string
    [op string **]**
  - **test** string1 **op** string2
    [string1 **op** string2 **]**

测试内容**op**：

=	两个字符串相同
!=	两个字符串不相同
-z	空串

注意：不是“==”

2. Shell 编程

- 2.8 测试数值
  - **test** number1 **op** number2

    [number1 **op** number2 **]**

    测试内容**op**：

    -eq 数值相等 (=)　　**-ne** 数值不相等 (!=)

    -gt 前者大于后者 (>)　　**-lt** 前者小于后者 (<)

    -le 前者小于等于后者 (≤)　　**-ge** 前者大于等于后者 (≥)

2. Shell 编程

- 2.8 测试数值

```
[root@localhost chapter3]# more test.sh
#!/bin/sh
#################################################
# Name: test.sh
# Usage: Test the file, string and number
# Author: Gene
# Date: 2008-07-29
#################################################

x="005"
y=5

[ $x = $y ]
echo $?

[ $x -eq $y ]
echo $?
```

2. Shell 编程

- 2.8 { }与( )的使用
  - 当使用 && 或 || 时，需要在条件分支中完成多个动作，执行若干个命令，就需要使用类似复合语句的构造，在shell中使用大括号。
  - 书写规则1：

```
[ -f main.c ] && { pwd; ls; rm main.c -f;}
```

必须有空格

必须有“;”

  - 书写规则2：

```
[ -f main.c ] && {
pwd
ls
rm main.c -f
}
```

回车

( )的使用没有这么多限制；但{ }的执行效率高

2. Shell 编程

- 2.8 expr计算表达式的值
  - B-shell本身没有提供数学运算和字符串运算的能力，所有这些运算都是借助于命令expr完成的；
  - expr支持算术运算(+、-、*、/)，取余数(%)，以及数值比较的关系运算(<、<=、=、!=、>=、>)；
  - expr的运算优先级和C语言一样：乘除法优先级最高，其次加减法，然后是关系运算。关系运算的结果是expr打印1 (关系成立)或者0(关系不成立)；也可以使用括号。

- 2.8 expr计算表达式的值

例1：求 “a*(b+c)”

```
a=1
b=2
c=3
x=`expr $a \* \( $b + $c \)`
```

注意：在expr表达式中，变量与转义符、操作符之间用空格分隔

例2：在例1的基础上判断x是否大于20

```
[ `expr $x \> 20` = 0 ] && echo ‘$x = ’ $x is less than 20
```

2. Shell 编程

- 2.9 条件结构**if-then-elif-fi**

语法1

```
if  条件1; then
    命令1
elif 条件2; then
    命令2
else
    命令3
fi
```

语法2

```
if  条件1
then  命令1
elif  条件2
then  命令2
else
    命令3
fi
```

条件与“then”在一行的，条件之后必须加“;”

2. Shell 编程

- 2.9 条件结构**if-then-elif-fi**

```
[root@localhost chapter3]# more if.sh
#!/bin/sh
#################################################
# Name: if.sh
# Usage: Test if-then-elif-fi
# Author: Gene
# Date: 2008-07-29
#################################################

LOG=./error.log
date >> $LOG

if [ -r errfile ]
then
        cat errfile >> $LOG
        rm errfile
else
        echo "No error" >> $LOG
fi
```

2. Shell 编程

- 2.10 case结构

语法

```
case 条件 in
条件1)
        命令1
;;
条件2)
        命令2
;;
esac
```

可以是多个命令串

“esac”是“case”的反写

2. Shell 编程

- 2.10 case结构

一定要加上双引号，如果在引用这个脚本文件时没有携带任何参数，那么$1就会是空字符串，这种情况下省略了双引号就会导致case行语法错误。

```
case "$1" in
START|start)
        echo Program started!
;;
STOP|stop)
        echo Program stopped!
;;
RESTART|restart)
        echo Program restarted!
;;
*)
        echo 'Usage:case.sh [start|stop|restart]'
;;
esac
```

2. Shell 编程

- 2.11 **while**循环结构

语法

```
while 条件
do
        命令1
        命令2
        ……
done
```

2. Shell 编程

- 2.11 **while**循环结构

例：

```
#!/bin/sh
#########################
# Name: while.sh
# Usage: Test while-do-done
# ……
#########################
a=10
while [ $a -gt 0 ]
do
    echo '$a = ' $a
    a=`expr $a - 1`
    sleep 1
done
```

```
[root@localhost chapter3]# ./while.sh
$a =  10
$a =  9
$a =  8
$a =  7
$a =  6
$a =  5
$a =  4
$a =  3
$a =  2
$a =  1
```

2. Shell 编程

- 2.12 for循环结构

语法

```
for name in word1 word2 …
do
        命令1
        命令2
        ……
done
```

循环控制变量

循环条件表格：每一次循环name取表格中的一个值

2. Shell 编程

- 2.12 for循环结构

例：

```
#!/bin/sh
######################
# Name: for.sh
# Usage: Test for-do-done
# ……
######################
for file in ./*
do
    echo $file
done
```

打印当前目录下的所有文件

```
[root@localhost chapter3]# ./for.sh
./bracket.sh
./case.sh
./echohello.sh
./expr.sh
./for.sh
./if.sh
./test.sh
./var.sh
./while.sh
```

2. Shell 编程

- 2.13 **break**、**continue**和**exit**的使用

  - 命令**break**、**continue**用在循环结构for和while中使用，与C语言中的beak和continue流程控制功能类似。break退出循环；continue退出当前循环进入下一次循环。

  - **exit**命令用来终止Shell程序；exit后面的参数，就是Shell脚本程序结束的返回值。

2. Shell 编程

- 2.14 Shell函数

语法

```
name() { cmd1; cmd2; ...;}
```

或

```
name() {
        cmd1
        cmd2
        ......
}
```

2. Shell 编程

- 2.14 Shell函数
  - 在调用函数时，引用函数的名字，可以附加上0到多个参数，在函数体内部以位置变量$1，$2，…或$*，$@方式引用函数的参数。
  - 函数体内部可以使用内部命令return，使函数有返回码，返回码0代表成功，非零表示失败。
  - 函数体内一个函数不能调用它自己。shell函数不允许递归调用。

2. Shell 编程

- 2.14 Shell函数

例：

```
# ……
######################
sum(){
    ret=`expr $1 + $2`
}

if [ $# -lt 2 ]
then
    echo Please input two parameters
else
    sum $1 $2
    echo "$1 + $2 = $ret"
fi
```

函数头

函数体

调用函数