第十天　Shell 编程

一、正则表达式
二、Shell 编程中常用命令
三、shell 条件测试
四、熟悉 shell 中的常用语法（流程控制）
　　(1)if (2)while (3)for (4)case
五、熟悉 shell 中函数的使用

一．正则表达式

1　^
#只匹配行首
2　$
#只匹配行尾
3　*
#匹配 0 个或者多个单字符
4　[]
#只匹配[]内字符，可以是一个单字符，也可以是字符序列，可以使用*表示[]内字符序列范围，如用[1-5]代替[12345]
5　\
#只用来屏蔽一个元字符的特殊含义

6　.
#只匹配任意单字符
7　pattern\{n\}
#匹配 n 次 pattern
8　pattern\{n,\}
#匹配 n 次以上 pattern
9　pattern\{n,m\}
#匹配 n 到 m 次 pattern

11　^只允许在一行的开始匹配字符或单词
^d　　筛选出以 d 开头的文件属性
12　^ $
#匹配空行
13　^.$
#匹配包含一个字符的行
14　kkk$
#匹配以 kkk 结尾的所有字符
15　*\.pas
#匹配以*.pas 结尾的所有字符或文件

16　a\{2\}b
#a 出现两次，aab
17　a\{4,\}b

#a 至少出现 4 次，aaaab,aaaaab ..
18　a\{2,4\}
#a 出现次数范围 2-4 次
19　[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}
#匹配所有 ip 地址

二. shell 编程中常用命令

行提取命令

grep 选项：　　-v　　-n　　-i

grep　"[^a-z]oo"　aa　(文件名)
oo 前不是小写字母的行匹配。　注意：和开头没有关系
grep　“\.$”aa
匹配以.结尾的行
grep　"^[^A-Za-z]"　aa
匹配不以字母开头的行　　注意：所有字母不能这样写　A-z
grep　“^$”　aa
匹配空白行
grep　"oo*" aa
匹配最少一个 o
grep　“g.*d”　aa
匹配 g 开头，d 结尾，中间任意字符 gd

列提取命令

awk　'条件　{动作}'
last | awk '{printf $1 "\t" $3 "\n"}'
提取 last 显示结果的第一和第三列

last | grep "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}"|awk '{printf $1 "\t" $3 "\n"}'
在 last 中提取包含 ip 的行，在行中提取第一和第三列

awk 内置变量　　FS　指定分隔符
more /etc/passwd | awk 'BEGIN {FS=":"} {printf $1 "\t" $3 "\n"}'
读取 passwd 文件，以":"为分隔符，截取第一和第三列
BEGIN　在截取前使分隔符生效。如果没有 BEGIN，那么第一行自定义的分隔符不生效

1　last　> file
#把 last 命令结果保存在 file 文件中
2　awk　‘{print $0　“\n”}' file
#查找出 file 文件中的每 1 列
3　awk '{print $1"\t"$7　“\n”}' file

#查找出 file 文件中的第 1 列和第 7 列

cut

cut　-d　“分隔符”　-f　提取列　文件名

more /etc/passwd | grep "/bin/bash" | cut -d ":" -f 1,3
提取 passwd 文件中可以登录的用户的用户名和 UID

输出命令

echo　-e　“输出内容”

-e　识别格式化打印内容

echo　-e　“1\t2\t3”　打印 tab 键

echo　-e　"\e[1;31m　this is red text \e[0m"　输出红色字体
\e[　格式
1;31m　指定颜色
0m　恢复颜色（重置）
30m=黑色，31m=红色，32m=绿色，33m=黄色，34m=蓝色，35m=洋红，36m=青色，37=白色

条件测试操作

test 命令
用途：测试特定的表达式是否成立，当条件成立时，命令执行后的返回值为 0，否则为其他数值

格式：test　条件表达式　[　条件表达式　]

常见的测试类型
测试文件状态
字符串比较
整数值比较
逻辑测试

测试文件状态

格式：[　操作符　文件或目录　]

常用的测试操作符
-d：测试是否为目录（Directory）
-e：测试目录或文件是否存在（Exist）
-f：测试是否为文件（File）
-r：测试当前用户是否有权限读取（Read）

-w：测试当前用户是否有权限写入（Write）
-x：测试当前用户是否可执行（Excute）该文件
-L：测试是否为符号连接（Link）文件

```
[root@localhost ~]# [    -d /etc/vsftpd    ]
[root@localhost ~]# echo $?
0
[root@localhost ~]# [    -d /etc/hosts    ]
[root@localhost ~]# echo $?
1

[root@localhost ~]# [ -e /media/cdrom ] && echo "YES"
YES
[root@localhost ~]# [ -e /media/cdrom/Server ] && echo "YES”
[root@localhost ~]#
```

整数值比较

格式：[　整数 1　操作符　整数 2　]

常用的测试操作符
-eq：等于（Equal）
-ne：不等于（Not Equal）
-gt：大于（Greater Than）
-lt：小于（Lesser Than）
-le：小于或等于（Lesser or Equal）
-ge：大于或等于（Greater or Equal）

```
[root@localhost ~]# who | wc -l
5
[root@localhost ~]# [ `who | wc -l` -le 10 ] && echo "YES"
YES

[root@localhost ~]# df -hT | grep "/boot" | awk '{print $6}'
18%
[root@localhost ~]# BootUsage=`df -hT | grep "/boot" | awk '{print $6}' | cut -d "%" -f 1`
[root@localhost ~]# echo $BootUsage
18
[root@localhost ~]# [ $BootUsage -gt 95 ] && echo "YES"
```

字符串比较

格式：[　字符串 1　==　字符串 2]
　　[　字符串 1　!=　字符串 2]
　　[　-z　字符串　]

常用的测试操作符

==：字符串内容相同
!=：字符串内容不同，！号表示相反的意思
-z：字符串内容为空

[root@localhost ~]# read -p "Location：" FilePath
Location：/etc/inittab
[root@localhost ~]# [$FilePath == "/etc/inittab"] && echo "YES"
YES

[root@localhost ~]# [$LANG != "en.US"] && echo $LANG
zh_CN.UTF-8

逻辑测试

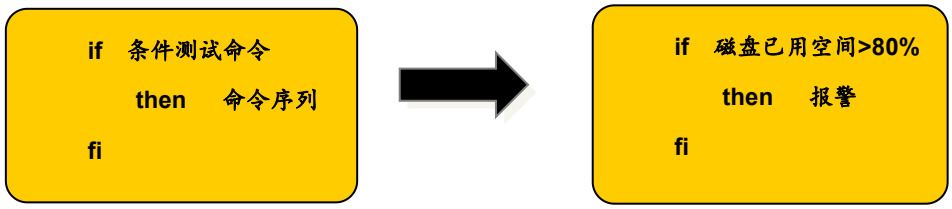
格式：[表达式 1] 操作符 [表达式 2] ...

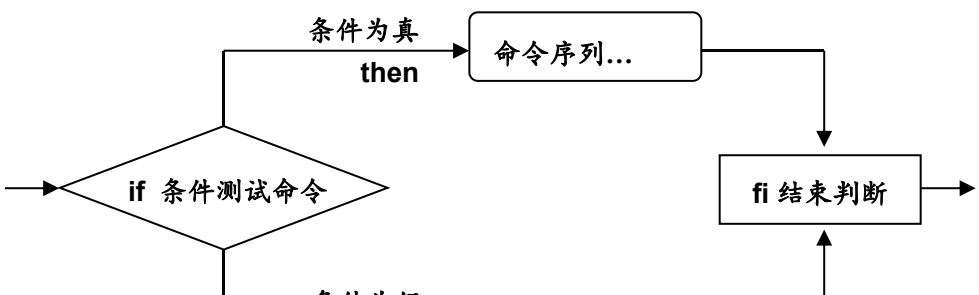
常用的测试操作符
-a 或&&：逻辑与，“而且”的意思
#前后两个表达式都成立时整个测试结果才为真，否则为假
-o 或||：逻辑或，“或者”的意思
#操作符两边至少一个为真时，结果为真，否则结果为假
!：逻辑否
#当指定的条件不成立时，返回结果为真

[root@localhost ~]# echo $USER
root
[root@localhost ~]# [$USER != "teacher"] && echo "Not teacher"
Not teacher
[root@localhost ~]# [$USER = "teacher"] || echo "Not teacher"
Not teacher

if 条件语句 -- 单分支

当“条件成立”时执行相应的操作



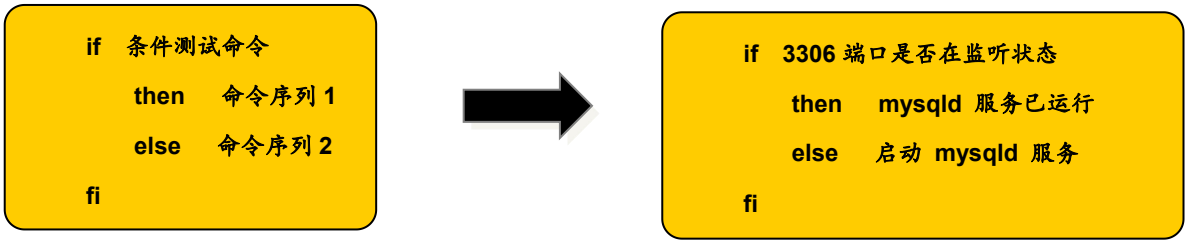


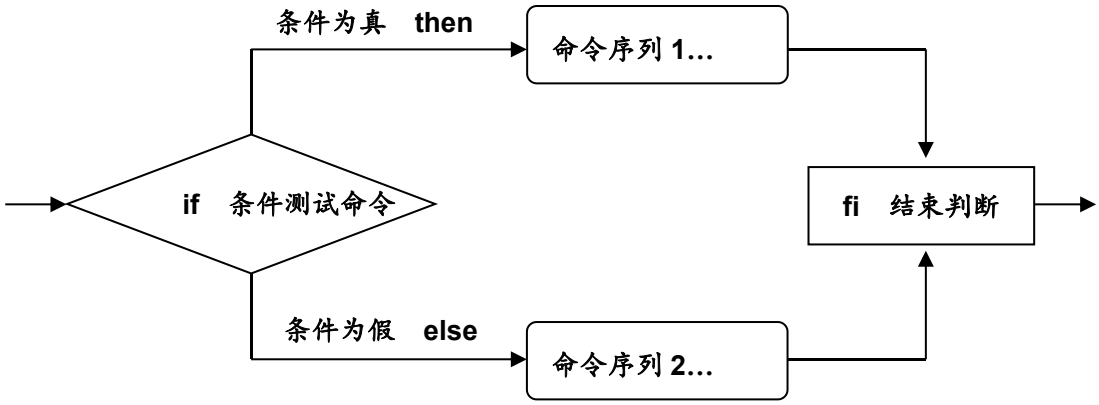
应用示例：
如果/boot 分区的空间使用超过 80%，输出报警信息

```
#!/bin/bash
RATE=`df -hT | grep "/boot" | awk '{print $6}' | cut -d "%" -f1 `
if  [  $RATE  -gt  80  ]
then
      echo "Warning,DISK is full!"
fi
```

if 条件语句 -- 双分支

当“条件成立”、“条件不成立”时执行不同操作





应用示例：

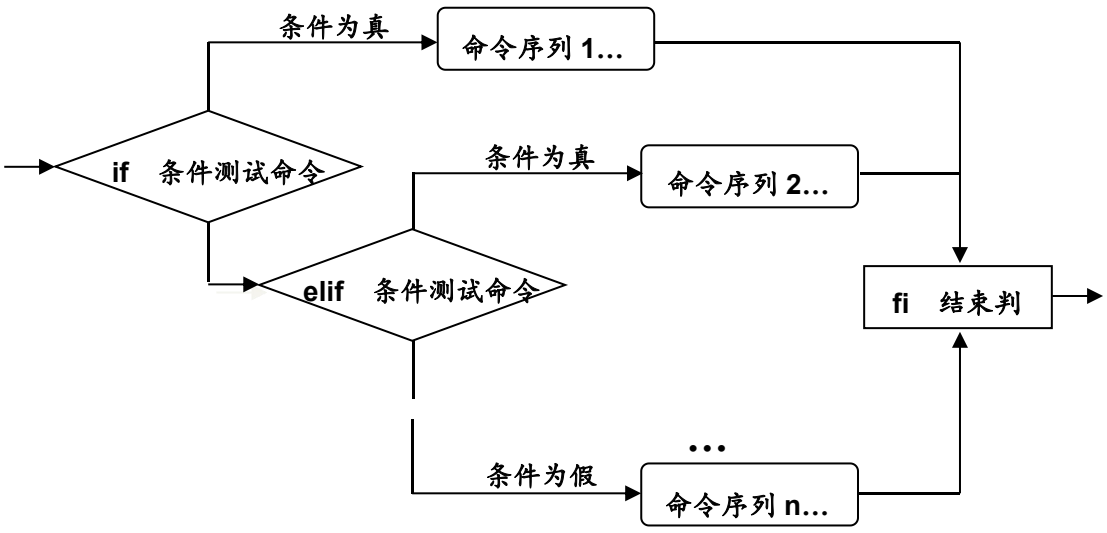
判断 mysqld 是否在运行，若已运行则输出提示信息，否则重新启动 mysqld 服务

```
#!/bin/bash
TEST=`/usr/bin/pgrep mysqld `
if  [  “$TEST”  !=  “”  ]
      then
            echo   "mysqld service is running."
      else
            /etc/init.d/mysqld   restart
fi
```

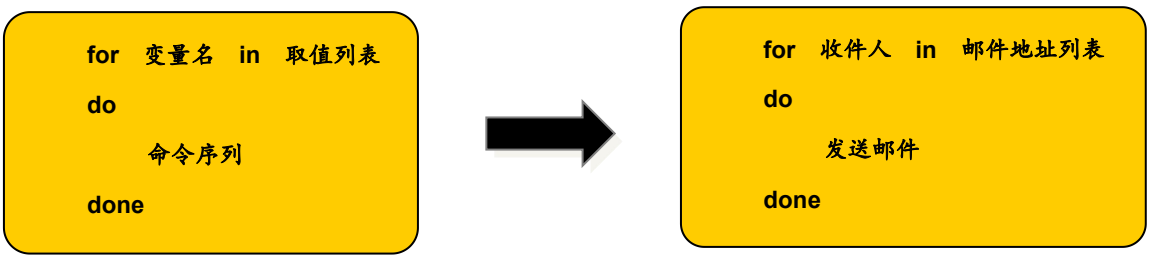
if 条件语句 -- 多分支

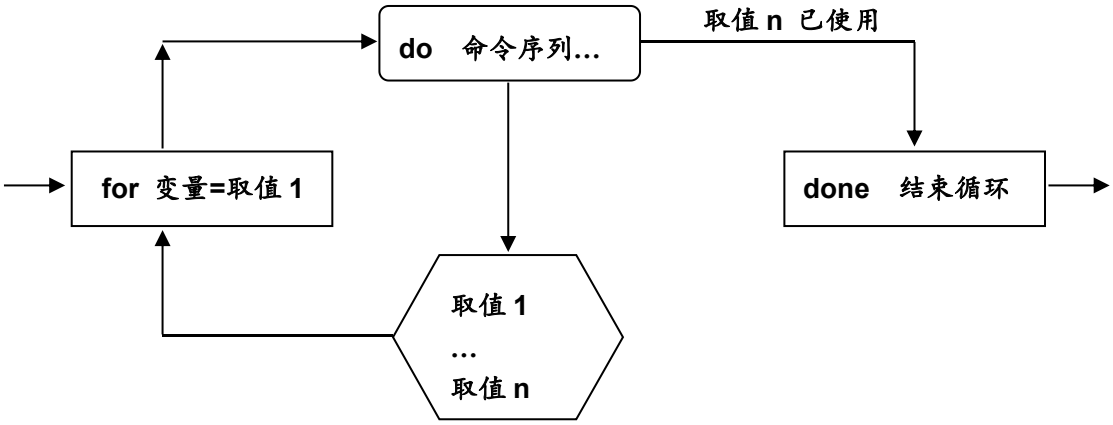
相当于 if 语句嵌套，针对多个条件执行不同操作

```
if  条件测试命令 1  ;  then
      命令序列 1
elif  条件测试命令 2  ;  then
      命令序列 2
elif  ...
else
      命令序列 n
fi
```



for 循环语句





应用示例 1：
依次输出 3 条文字信息，包括一天中的“Morning”、“Noon”、“Evening”字串

```
[root@localhost ~]# vi showday.sh
#!/bin/bash
for TM in "Morning" "Noon" "Evening"
do
    echo "The $TM of the day."
done

[root@localhost ~]# sh showday.sh
The Morning of the day.
```

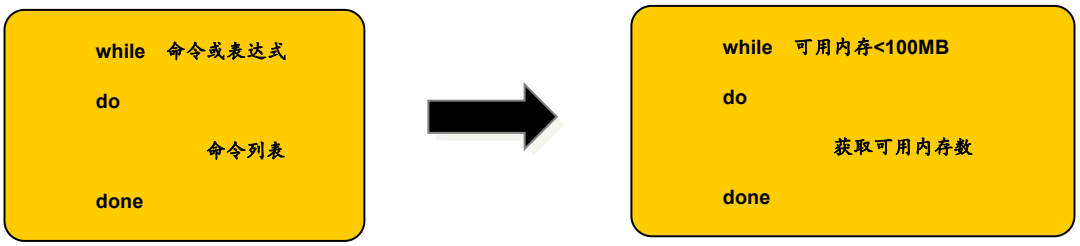
The Noon of the day.
The Evening of the day

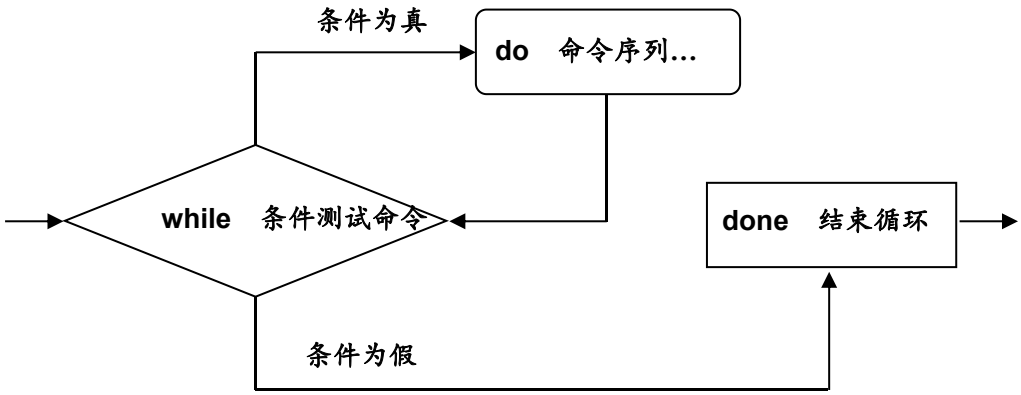
应用示例 2：
对于使用“/bin/bash”作为登录 Shell 的系统用户，检查他们在“/opt”目录中拥有的子目录或文件数量，如果超过 100 个，则列出具体个数及对应的用户帐号

```
 #!/bin/bash
DIR="/opt"
LMT=100
ValidUsers=`grep "/bin/bash" /etc/passwd | cut -d ":" -f 1`
for UserName    in    $ValidUsers
do
      Num=`find $DIR -user $UserName | wc -l`
      if    [    $Num    -gt    $LMT    ]    ;    then
            echo "$UserName have $Num files."
      fi
done
```

while 循环语句

重复测试指定的条件，只要条件成立则反复执行对应的命令操作





应用示例 1：
批量添加 20 个系统用户帐号， 用户名依次为“stu1”、“stu2”、……、“stu20”

这些用户的初始密码均设置为“123456”

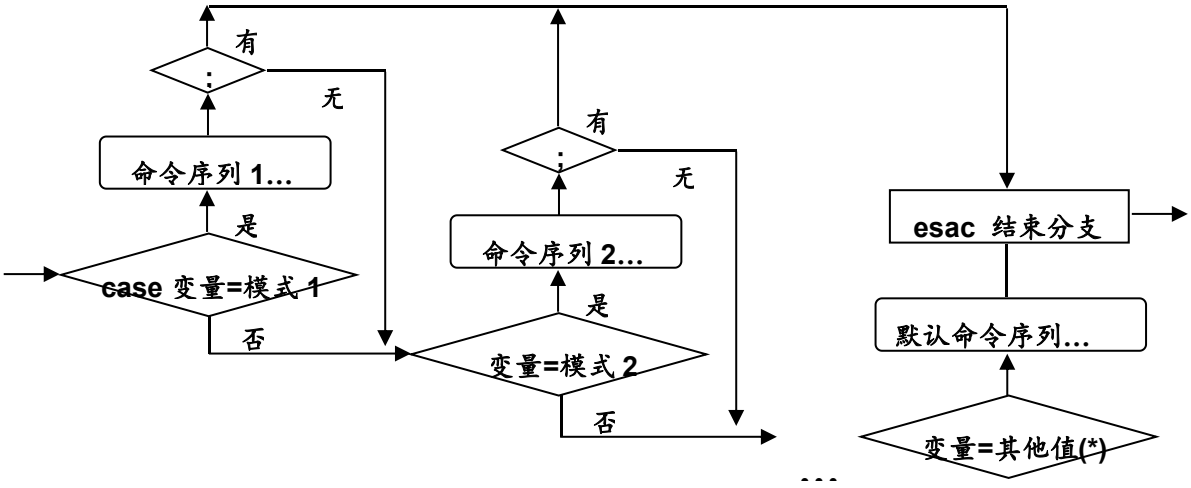
```
#!/bin/bash
i=1
while   [   $i   -le   20   ]
do
      useradd stu$i
      echo "123456" | passwd --stdin stu$i &> /dev/null
      i=`expr $i + 1`
done
```

应用示例 2：
批量删除上例中添加的 20 个系统用户帐号

```
#!/bin/bash
i=1
while   [   $i   -le   20   ]
do
      userdel -r stu$i
      i=`expr $i + 1`
done
```

case 多重分支语句

根据变量的不同取值，分别执行不同的命令操作



应用示例 1：
编写脚本文件 mydb.sh，用于控制系统服务 mysqld
当执行 ./mydb.sh start 时，启动 mysqld 服务
当执行 ./mydb.sh stop 时，关闭 mysqld 服务
如果输入其他脚本参数，则显示帮助信息

```
#!/bin/bash
case    $1    in
    start)
        echo   "Start MySQL service."
        ;;
    stop)
        echo   "Stop MySQL service."
        ;;
    *)
        echo   "Usage：$0   start|stop"
        ;;
esac
```

应用示例 2：
提示用户从键盘输入一个字符，判断该字符是否为字母、数字或者其它字符，并输出相应的提示信息

```
#!/bin/bash
read   -p   "Press some key, then press Return:“   KEY
case   "$KEY“   in
  [a-z]|[A-Z])
      echo "It's a letter."
      ;;
  [0-9])
      echo "It's a digit."
      ;;
  *)
      echo "It's function keys、Spacebar or other keys. "
esac
```

Shell 函数应用

Shell 函数概述
在编写 Shell 脚本程序时，将一些需要重复使用的命令操作，定义为公共使用的语句块，即可称为函数
合理使用 Shell 函数，可以使脚本内容更加简洁，增强程序的易读性，提高执行效率

定义新的函数

```
function 函数名 {
        命令序列
    }

函数名() {
        命令序列
    }
```

调用已定义的函数

函数名

向函数内传递参数

函数名　参数 1　参数 2　...

应用示例：
在脚本中定义一个加法函数，用于计算 2 个整数的和
调用该函数计算（12+34）、（56+789）的和

```
#!/bin/bash
adder() {
    echo `expr $1 + $2`
}
adder 12 34
adder 56 789
```

```
[root@localhost ~]# bash adderfun.sh
46
845
```