Shell条件测试和流程控制

授课目标

- 1.shell条件测试
- 2.熟悉shell中的常用语法
- (1)if (2)while (3)for (4)case (5)read
- (6)case (7)shift (8)continue (9)break
- 3.函数使用

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

条件测试操作

- 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
```

返回值为0，表示上一步测试的条件成立

如果测试的条件成立则输出“YES”

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

条件测试操作

- 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
```

如果登录用户数小于或等于10则输出 YES

```
[root@localhost ~]# df -hT | grep "/boot" | awk '{print $6}'
12%
[root@localhost ~]# BootUsage=`df -hT | grep "/boot" | awk '{print
    $6}' | cut -d "%" -f 1`
[root@localhost ~]# echo $BootUsage
12
[root@localhost ~]# [ $BootUsage -gt 95 ] && echo "YES"
```

如果/boot分区的磁盘使用率超过95%则输出 YES

条件测试操作

- 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
```

如果键入路径与指定的目录一致则输出 YES

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

如果当前的语言环境不是 en_US，则输出LANG变量的值

条件测试操作

- 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
```

如果发现用户不是 teacher
则提示：“Not teacher”

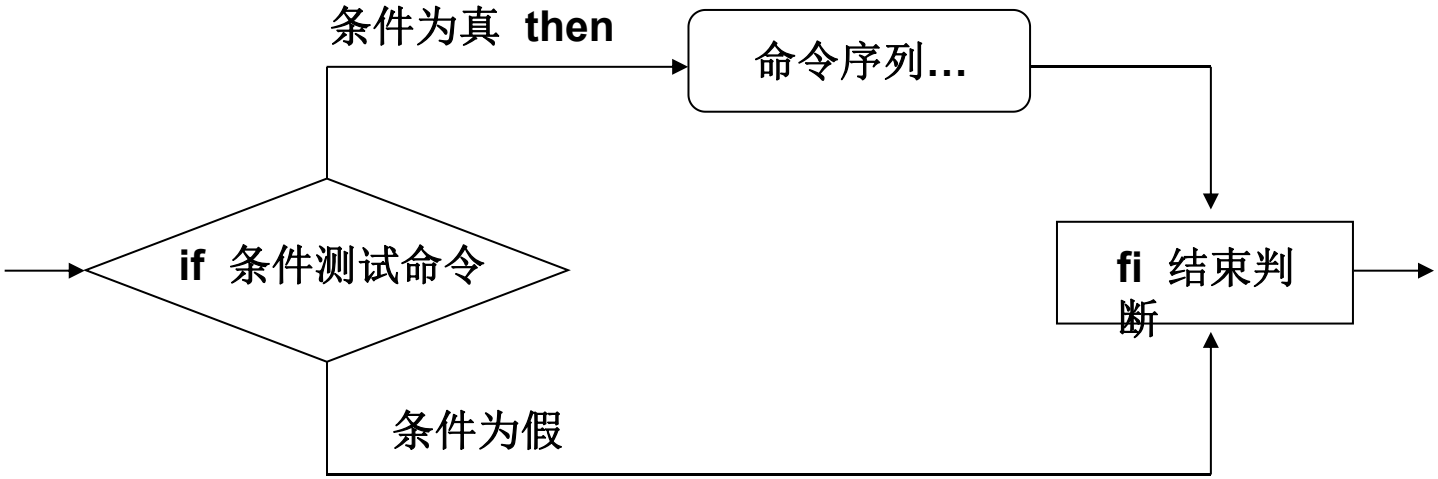
与上一命令行效果相同

if条件语句 -- 单分支

- 当“条件成立”时执行相应的操作

```
if 条件测试命令
   then 命令序列
fi
```

```
if 磁盘已用空间>80%
   then 报警
fi
```



if条件语句 -- 单分支

- 应用示例：
  - ✓如果/boot分区的空间使用超过80%，输出报警信息

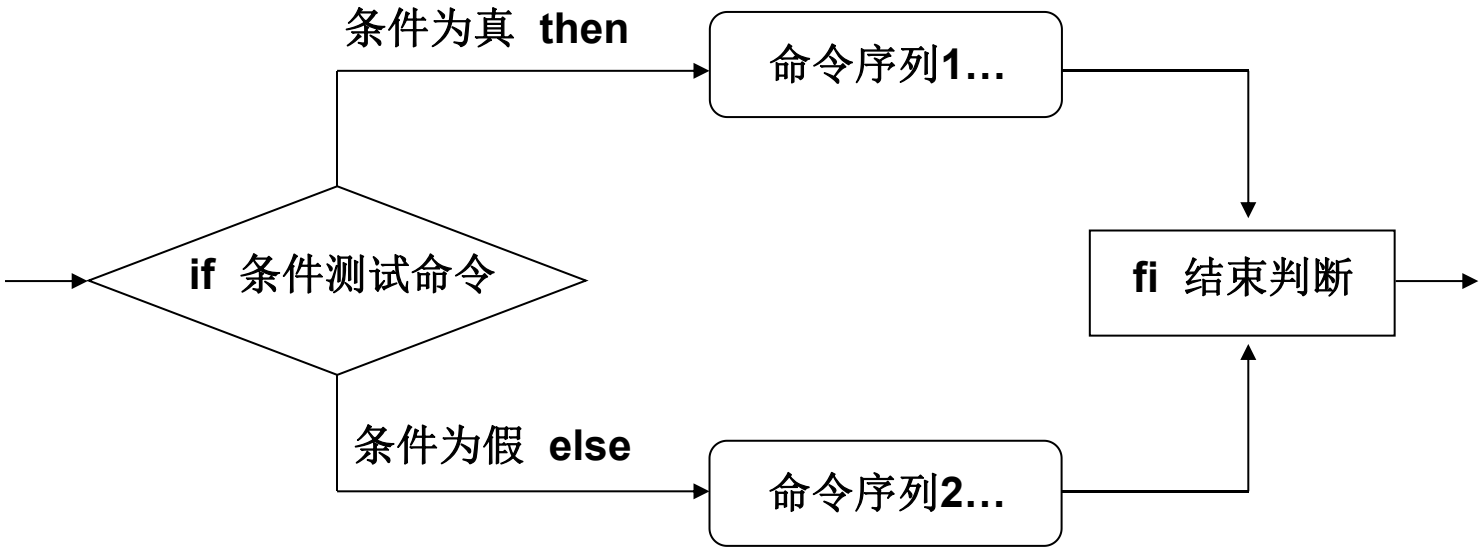
```
#!/bin/bash
RATE=`df -hT | grep "/boot" | awk '{print $6}' | cut -d "%" -f1 `
if  [  $RATE  -gt  80  ]
then
    echo "Warning,DISK is full!"
fi
```

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

```
if  条件测试命令
    then   命令序列1
    else   命令序列2
fi
```

```
if  3306端口是否在监听状态
    then   mysqld 服务已运行
    else   启动 mysqld 服务
fi
```



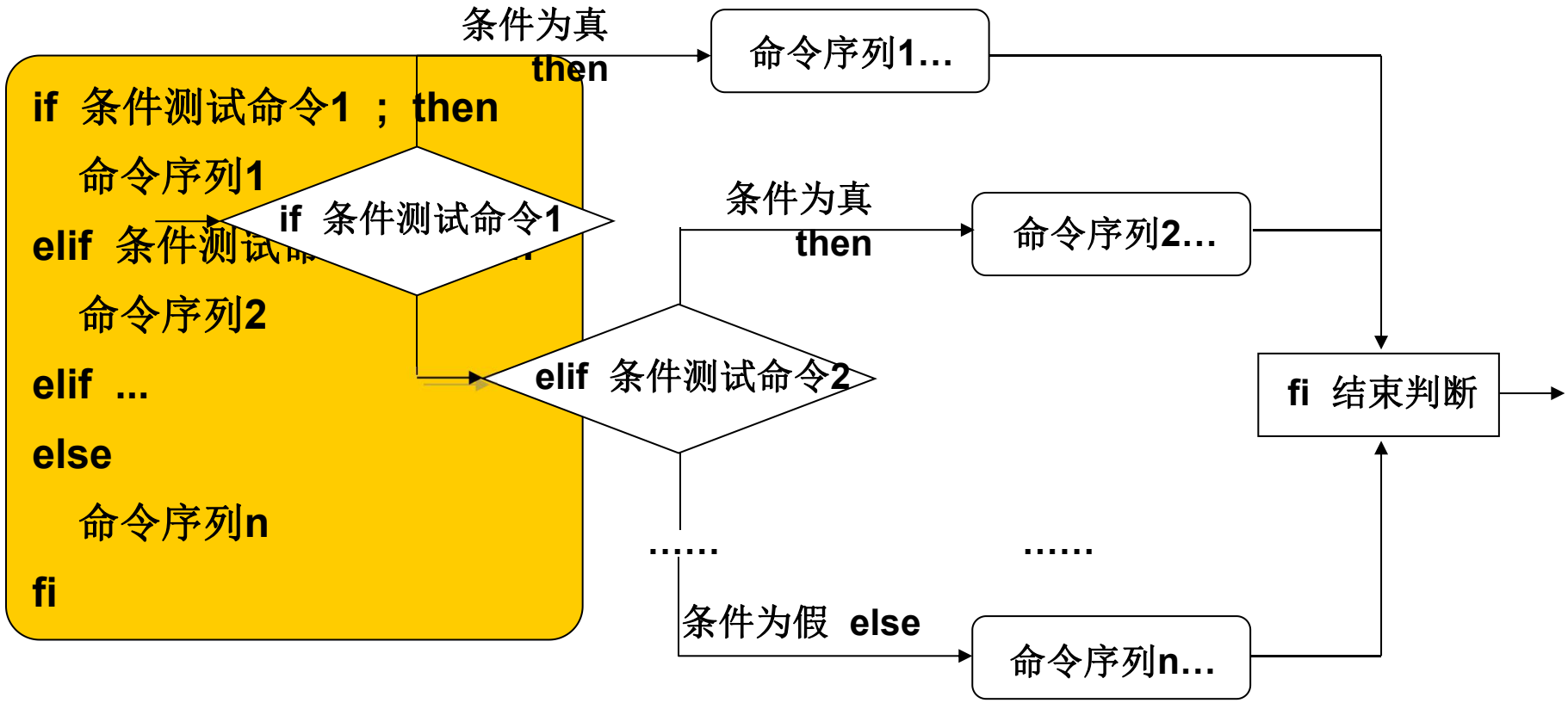
if条件语句 -- 双分支

- 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
service mysqld status &> /dev/null
if  [  $?  -eq  0  ]
    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

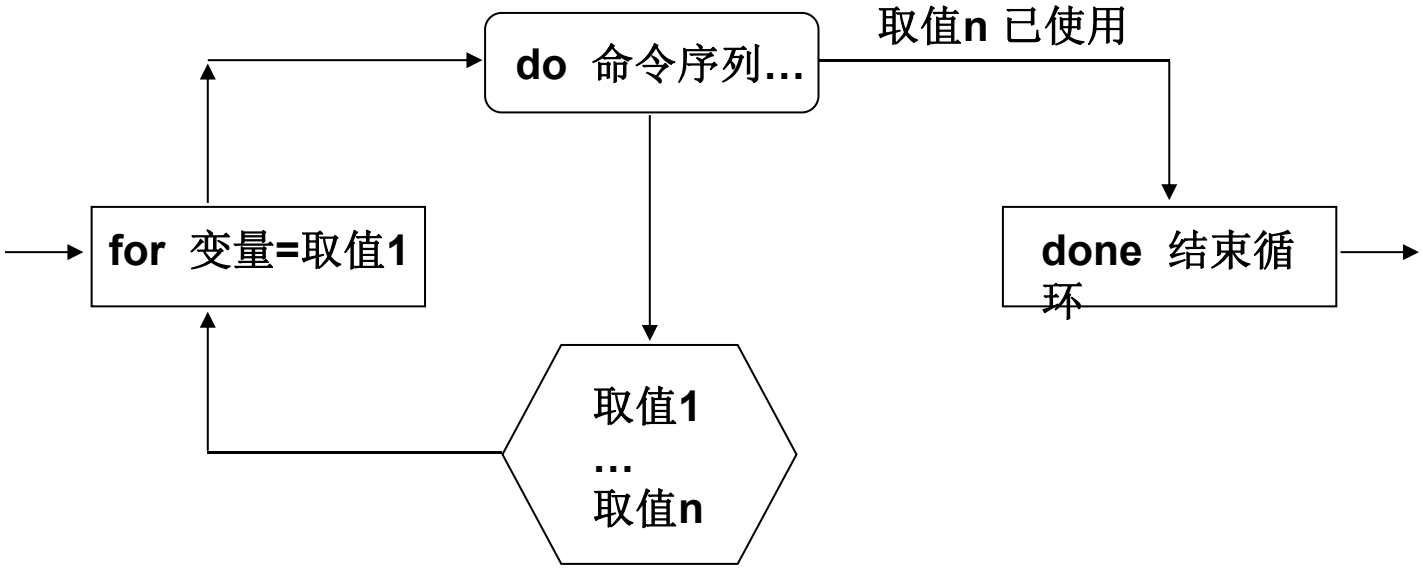


for循环语句

- 根据变量的不同取值，重复执行一组命令操作

```
for 变量名 in 取值列表
do
    命令序列
done
```

```
for 收件人 in 邮件地址列表
do
    发送邮件
done
```



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
```

验证脚本执行结果

for循环语句

- 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

获得使用bash作为登录Shell的用户名列表

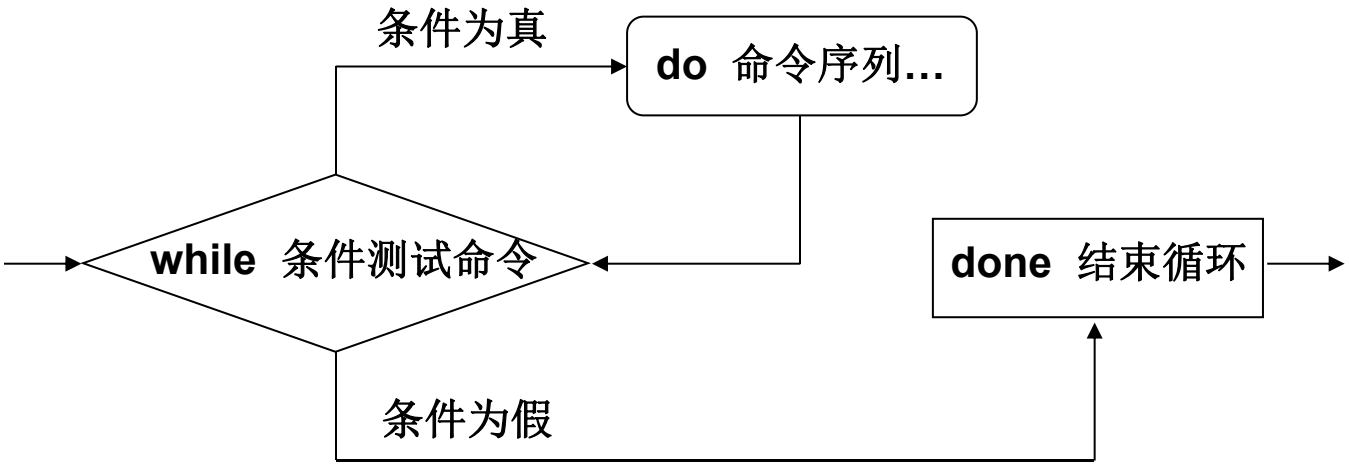
```
#!/bin/bash
DIR="/opt"
LMT=100
ValidUsers=`grep "/bin/bash" /etc/passwd | cut -d ":" -f 1`
for UserName  in  $ValidUsers
do
    Num=`find $DIR -user $UserName | wc -l`
    if  [  $Num  -gt  $LMT  ]  ;  then
        echo "$UserName have $Num files."
    fi
done
```

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

```
while 命令或表达式
do
        命令列表
done
```

```
while 可用内存<100MB
do
        获取可用内存数
done
```



while循环语句

- 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
```

执行 let i++ 也可以
使变量i的值递增1

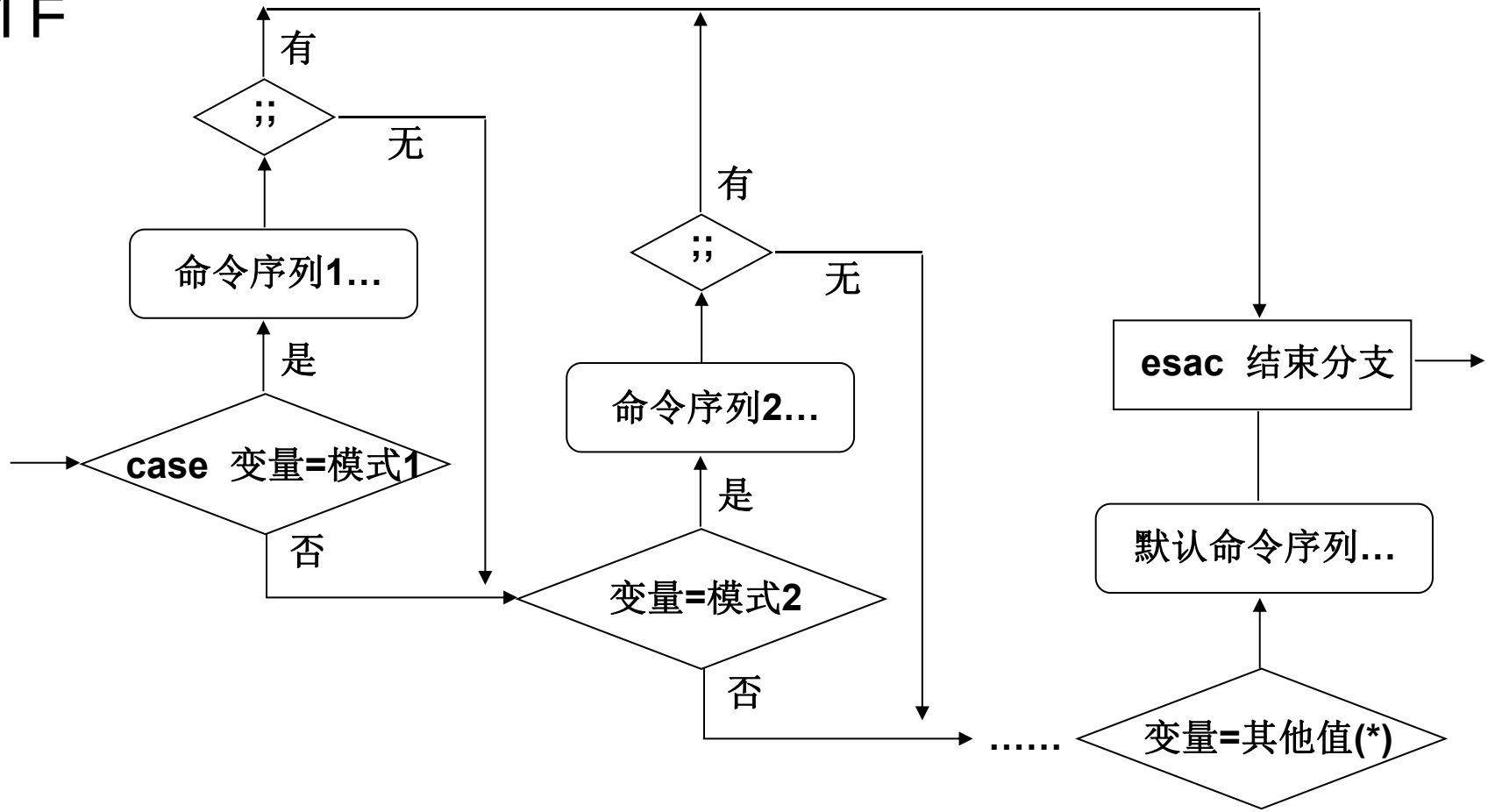
while循环语句

- 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



case多重分支语句

- 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
```

case多重分支语句

- 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
```

shift迁移语句

- 用于迁移位置变量，将 $1~$9 依次向左传递

1.例如，若当前脚本程序获得的位置变量如下：

- $1=file1、$2=file2、$3=file3、$4=file4

2.则执行一次shift命令后，各位置变量为：

- $1=file2、$2=file3、$3=file4

3.再次执行shift命令后，各位置变量为：

- $1=file3、$2=file4

- 应用示例：
  - ✓通过命令行参数传递多个整数值，并计算总和

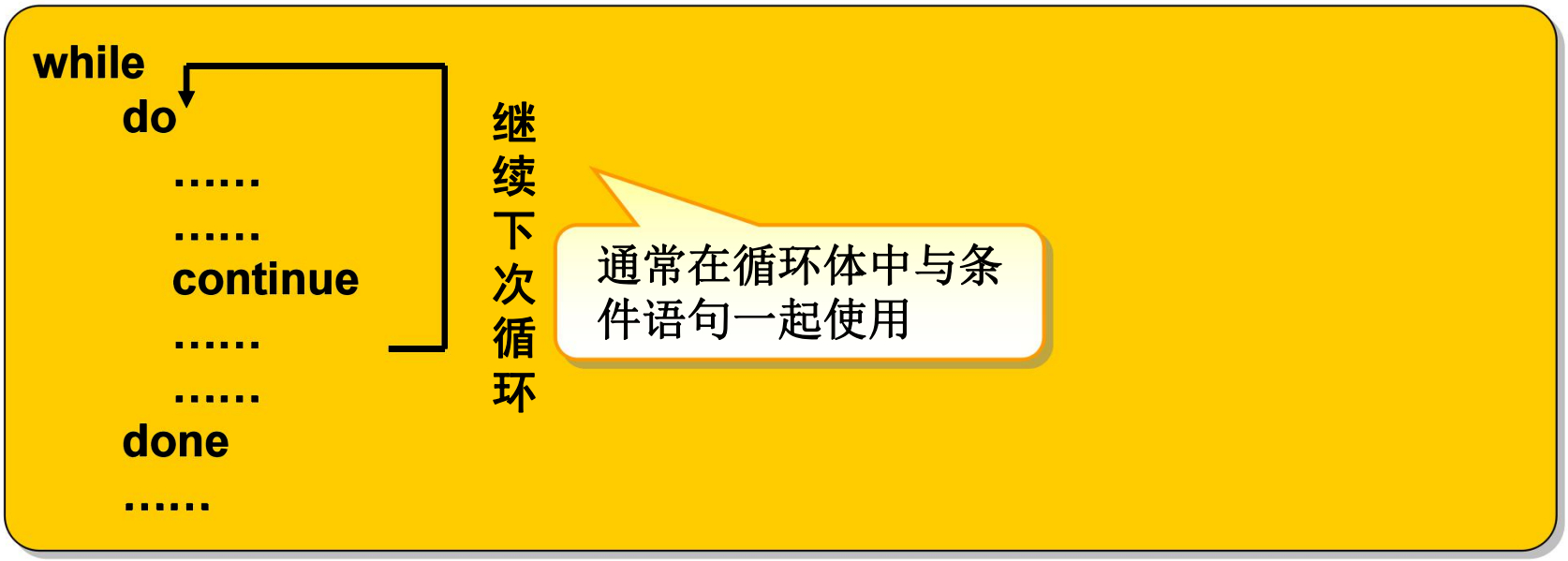
```
[root@localhost ~]# vi showday.sh
#!/bin/bash
Result=0
while  [  $#  -gt  0  ]
do
    Result=`expr $Result + $1`
    shift
done
echo "The sum is : $Result"
```

```
[root@localhost ~]# ./sumer.sh  12  34  56
The sum is : 102
```

验证脚本执行结果

- break语句
  - ✓在for、while、until等循环语句中，用于跳出当前所在的循环体，执行循环体后的语句

- continue
  - ✓在for、while、until等循环语句中，用于跳过循环体内余下的语句，重新判断条件以便执行下一次循环



- 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
}
```

- 或者

```
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
```

验证脚本执行结果

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

云知梦

云知梦，只为有梦想的人