上章回顾

- 文件测试常见的操作选项有哪些?
  - 语法:
    - test 测试符 参数
    - [测试符 参数]
  - 测试符
    - -d：是否是目录
    - -e：是否存在
    - -f：是否是文件
    - -r：是否具有读权限
    - -w：是否具有写权限
    - -x：是否具有执行权限
- 字符串测试常见的操作选项有哪些?
  - 语法:
    - [字符串 测试符 字符串]
  - 测试符
    - =内容相同
    - !=：内容不同
    - -z：内容为空（当作字符串来使用的，确认另一个字符串是否为空）
- if条件测试单分支与双分支的区别是什么?
  - 单分支

```
if [ 条件表达式 ];then
    执行命令块
    执行命令块
    执行命令块
fi
```

  - 双分支

```
if [ 条件表达式 ];then
    执行命令块
    执行命令块
    执行命令块
else
    执行命令块
    ....
```

```
fi
```

- 多分支

```
if [ 条件表达式 ];then
    执行命令块
    执行命令块
    执行命令块
elif [ 条件表达式 ];then
    执行命令块
    执行命令块
    执行命令块
elif [ 条件表达式 ];then
    执行命令块
    执行命令块
    执行命令块
....

else
    执行命令块
fi
```

- 逻辑测试有哪几种?
  - &&：（与）前面执行成功后才会执行后面的执行，必须满足所有条件 [test测试中需要使用 -a选项]
  - ||：（或）前面无法执行时，执行后面的指令，仅需要满足一个条件即可 [test测试中需要使用 -o选项]
  - !：（非）不是相同的值

本章内容

- 学会使用for、while循环语句
- 学会使用case分支语句

一、for循环语句

1、读取不同的变量值，用来逐个执行同一组命令

2、用户自定义一个变量名称，通过关键字 in 为该变量预先设置一个取值列表，多个值之间使用空格分隔

3、在do ..... done 之间的命令序列称之为循环体，其中执行语句需要引用之前设置的变量完成相应的任务

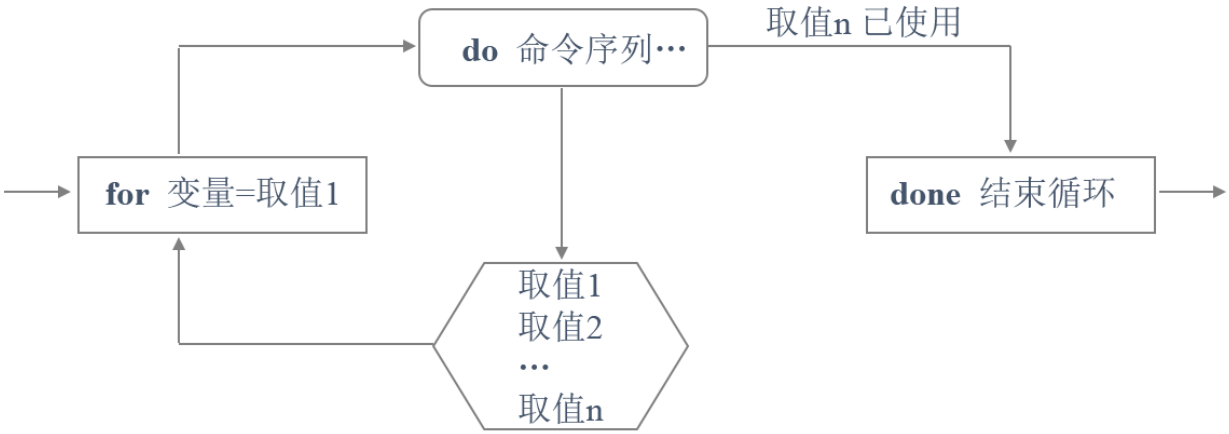
4、for 语句的执行流程：

1）首先将列表中的第 1 个值赋给变量，并执行一次do....done循环体中的命令序列

2）然后在取第 2 个值赋给变量，并且执行命令序列

3）直到序列中的所有值取完并且执行后，跳转到done，结束循环

5、for 语句结构



> 如果要显示shell执行过程，在首行添加 set -x

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

[root@localhost ~] vim test.sh
#!/bin/bash
for i in 1 2 3 4 5
        do
                echo "$(expr $i \* $i)"
        done
[root@localhost ~] ./test.sh
1
4
9
16
25
```

6、for 循环实例

- 根据姓名列表批量添加用户

```
[root@localhost ~] vim /root/user.txt
zhangsan
lisi
wangwu
[root@localhost ~] vim test.sh
```

```
#!/bin/bash
ULIST=$(cat /root/user.txt)
for UNAME in $ULIST
        do
                useradd $UNAME
                echo "123456" | passwd --stdin $UNAME &> /dev/null
        done
[root@localhost ~] tail -3 /etc/passwd

zhangsan:x:1000:1000::/home/zhangsan:/bin/bash
lisi:x:1001:1001::/home/lisi:/bin/bash
wangwu:x:1002:1002::/home/wangwu:/bin/bash
```

- 批量删除用户

```
[root@localhost ~]# ./test.sh
#!/bin/bash
ULIST=$(cat /root/user.txt)
for UNAME in $ULIST
        do
                userdel -r  $UNAME &> /dev/null
        done
[root@localhost ~] tail -3 /etc/passwd
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
chrony:x:998:996::/var/lib/chrony:/sbin/nologin
```

- 根据IP地址列表检查主机状态

```
[root@localhost ~] vim /root/ipadds.txt
192.168.1.1
192.168.1.2
192.168.1.3

[root@localhost ~] vim test.sh

#!/bin/bash
HLIST=$(cat /root/ipadds.txt)
for IP in $HLIST
        do
```

```
                ping -c 3 -i 0.2 -w 3 $IP &> /dev/null
                if [ $? -eq 0 ]
                        then
                                echo "Host $IP ip up"
                        else
                                echo "Host $IP ip down"
                fi
        done
[root@localhost ~]# ./test.sh
Host 192.168.1.1 ip down
Host 192.168.1.2 ip down
Host 192.168.1.3 ip down

#方法二：

#!/bin/bash
ip=192.168.112
for ((i=125;i<=130;i++))
        do
                ping -c 3 -i 0.2 -w 3 $ip.$i &> /dev/null
                if [ $? -eq 0 ]
                        then
                                echo "Host $ip.$i ip up"
                        else
                                echo "Host $ip.$i ip down"
                fi
        done
[root@localhost ~]# ./test.sh
Host 192.168.112.125 ip down
Host 192.168.112.126 ip down
Host 192.168.112.127 ip down
Host 192.168.112.128 ip up
Host 192.168.112.129 ip down
Host 192.168.112.130 ip down
```

- 打印1-5

```
#!/bin/bash
for ((i=1;i<=5;i++))
        do
                echo $i
```

```
        done
[root@localhost ~] ./test.sh
1
2
3
4
5
```

- 打印1 - 100的和

```
#!/bin/bash

sum=0
for ((i=1;i<=100;i++))
        do
                sum=$(expr $sum + $i)
        done
echo "1-100的和是：$sum"
[root@localhost ~]# ./test.sh
1-100的和是：5050
```

- 1-100的奇数和

```
#!/bin/bash
sum=0
for ((i=1;i<=100;i++))
        do
                if [ `expr $i % 2` -eq 1 ]
                        then
                                sum=$(expr $sum + $i)
                fi
        done
echo "1-100的奇数和是：$sum"
[root@localhost ~]# ./test.sh
1-100的奇数和是：2500
```

二、while循环语句

1、相比for循环语句，while更适合在控制循环次数、操作对象按数字顺序编号和在特定条件下重复操作等情况使用。

2、使用whlie语句时，可以根据特定的条件反复执行一个命令序列，直到该条件不再满足时为止

3、在脚本中，为了避免死循环导致后面的命令无法执行的情况，需要在命令序列中添加测试条件的语句，便于在适当的时候使测试条件不再成立，从而结束死循环

4、while执行流程

1）首先判断while后的条件测试操作结果，如果条件成立则执行do....done循环体中的命令序列

2）返回的while再次判断条件测试结果，如果依旧成立则继续执行，依次循环

3）直到while的条件测试结果不在成立，最后跳转到done语句，表示循环结束

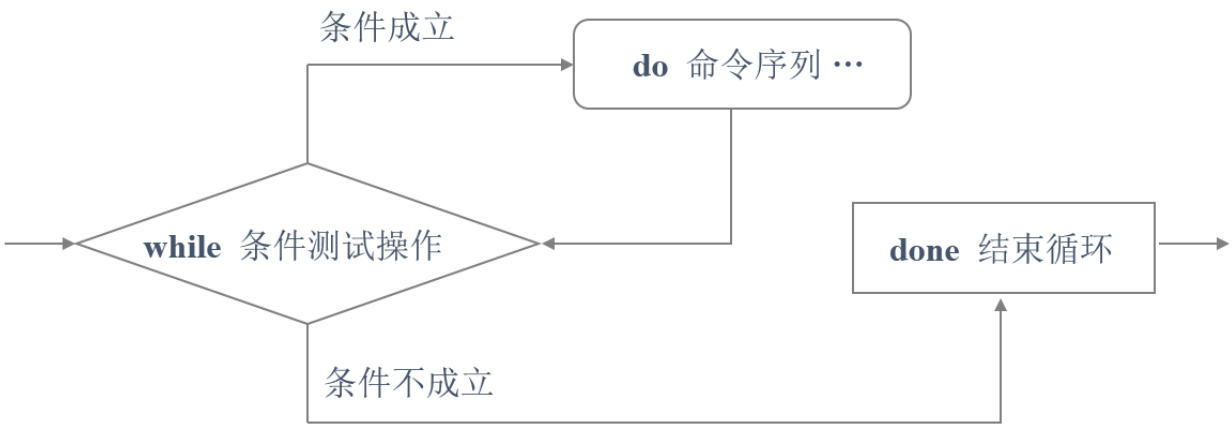
5、在while循环中，除了针对条件测试以外，还可以使用true和false进行判断

1）使用true作为条件时，表示条件永远是成立的，如果没有exit指令，则会一直循环下去

2）使用false作为条件时，则循环体永远不会被执行

3）这两个还可以在 if 中使用

6、while语句结构




```
#!/bin/bash
while   条件测试操作
do
    命令序列
done
```

7、while案例

- 1-100的求和

```
#!/bin/bash
sum=0
i=1
while [ $i -le 100 ]
        do
                sum=$(expr $sum + $i)
                let i++
done
```

```
echo "1-100的和是：$sum"
[root@localhost ~]# ./test.sh
1-100的和是：5050
```

- 批量创建有规律的用户

```
#!/bin/bash
prefix="user"
i=1
while [ $i -le 7 ]
        do
                useradd $prefix$i
                echo "123456" | passwd --stdin $prefix$i &> /dev/null
                let i++
done
[root@localhost ~] tail -7 /etc/passwd
user1:x:1000:1000::/home/user1:/bin/bash
user2:x:1001:1001::/home/user2:/bin/bash
user3:x:1002:1002::/home/user3:/bin/bash
user4:x:1003:1003::/home/user4:/bin/bash
user5:x:1004:1004::/home/user5:/bin/bash
user6:x:1005:1005::/home/user6:/bin/bash
user7:x:1006:1006::/home/user7:/bin/bash

```

- 猜数字游戏

> 在bash环境中默认存在 $RANDOM 变量，会随机输出0 - 32767的数字

```
#!/bin/bash
NUM=$(expr $RANDOM % 1000)
times=0
echo "数字范围在1 - 1000之间"
        while true
                do
                        read -p "请输入你猜的数字：" int
                                let times++
                                if [ $int -eq $NUM ]; then
                                        echo "猜中了，数字是：$NUM"
                                        echo "总共猜了$times 次"
```

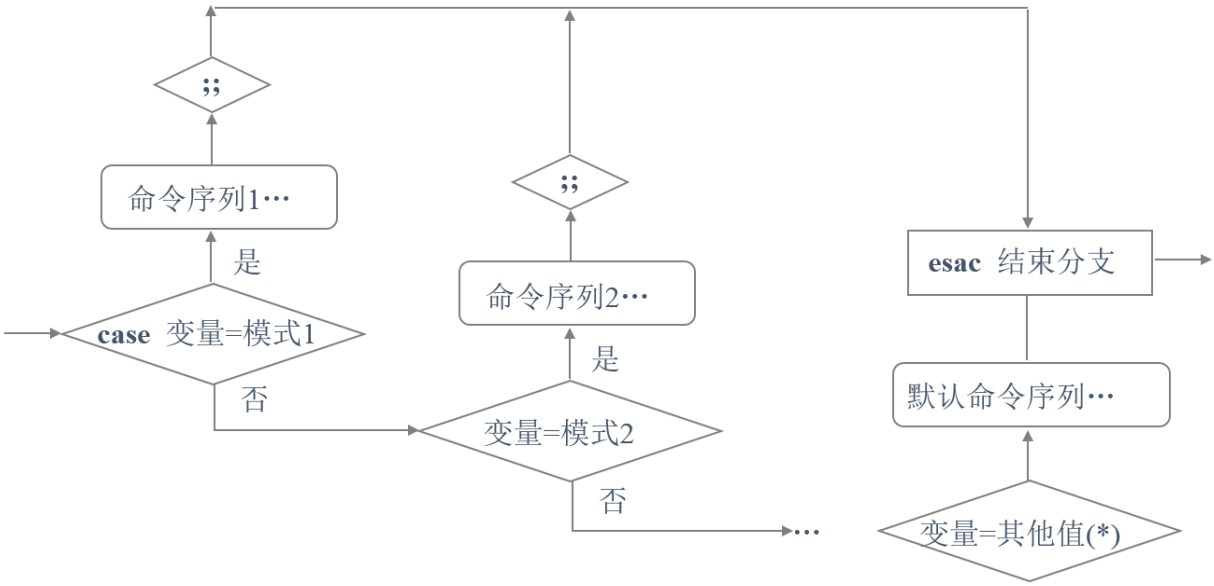
```
                exit 0

            elif [ $int -gt $NUM ]; then
                echo "太高了"
            elif [ $int -lt $NUM ]; then
                echo "太低了"
            else [ $times -eq 11 ]
                shutdown -h now
            fi
        done
```

三、case分支语句

1、与 if 不同的是，case分支可以针对不同的变量进行取值，然后执行不同的命令序列

2、if 语句永远是从if - elif - else顺序执行，意味着从上到下依次判断，而case则是直接判断对应的条件执行命令块



3、击键类型识别

，提示用户输入一个字符

，判断出该字符是字母、数字

```
#!/bin/bash
read -p "请输入一个字符，并按Enter键确认：" KEY
case "$KEY" in
  [a-z]|[A-Z])
      echo "您输入的是 字母。"
      ;;
  [0-9])
      echo "您输入的是 数字。"
      ;;
  *)
```

```
        echo "您输入的是 空格、功能键或其他控制字符。"
esac
```

4、编写自动启动httpd服务的系统服务脚本

- 使用start、stop、restart等参数来控制服务
- 服务控制指令通过位置变量$1传入
- 能够通过chkconfig命令来管理此服务

```
[root@localhost ~] cat /etc/init.d/myprog
#!/bin/bash
# chkconfig: - 90 10
case "$1" in
start)
    ... 正在启动XX服务
    ;;
stop)
    ... 正在停止XX服务
    ;;
status)
    portnu=$(取出端口号)
        if [ $portnu -eq 80 ]
        then
            echo "vsftpd服务已经启动。"
        else
            echo "vsftpd服务已经停止。"
        fi
        ;;
restart)
    $0 stop
    $0 start
    ;;
*)
    echo "用法: $0 {start|stop|restart|status}"
    ;;
esac
```

作业：　-

1. for语句
    a. 1 - 100的偶数求和

b. 1 - 100的能被5整除所有数之和

c. 检查192.168.0.100 - 192.168.0.130之间的主机能否通信（不能使用列表文件）

2. while语句

a. /data目录下创建20个name(n).txt文件

b. 删除data目录下刚才创建的20个文件

c. 10 - 50之间的数字求和

3. case语句

a. 使用脚本直接安装apche服务，安装完成后使用脚本控制服务的状态

b. 使用case脚本控制编译安装的apache程序执行启动、关闭、查看状态等操作；