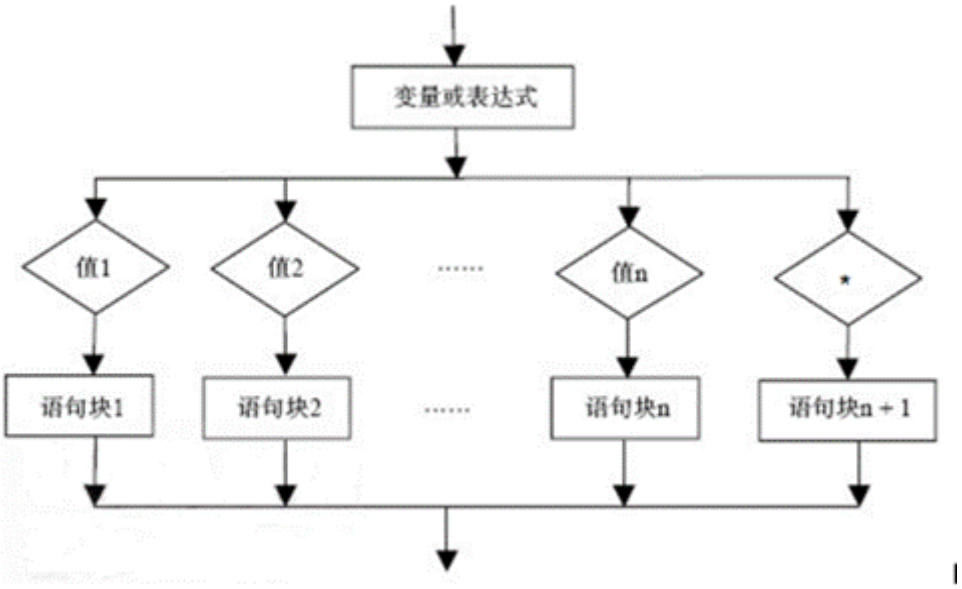
流程控制语句：case

控制语句：用来实现对程序流程的选择、循环、转向和返回等进行控制。
case是开关语句的一个组成部分；它是根据变量的不同进行取值比较，然后针对不同的取值分别执行不同的命令操作
适用于多分支，是一个多选择语句

语法格式：

```
case    变量或表达式    in
    变量或表达式1）
        命令序列1
        ；；
    变量或表达式2）
        命令序列2
        ；；
        ……
        *）
        默认命令序列
esac
```

case语句执行流程控制：



执行流程：
1、首先使用“变量或表达式”的值与值1进行比较，若取值相同则执行值1后的命令序列，直到遇见双分号“；；”后跳转至esac，表示分支结束；
2、若与值1不相匹配，则继续与值2进行比较，若取值相同则执行值2后的命令序列，直到遇见双分号“；；”后跳转至esac，表示结束分支。
3、依次类推，若找不到任何匹配的值，则执行默认模式“*）”后的命令序列，直到遇见esac后结束分支

注意事项：

- **“变量或表达式”后面必须为单词in，每一个“变量或表达式”的值必须以右括号结束。取值可以为变量或常数。匹配发现取值符合某一模式后，其间所有命令开始执行直至;;**
- **匹配中的值可以是多个值，通过 "|" 来分隔**

```
例子1：编写一个操作文件的脚本
[root@exercise1 opt]# vim case-1.sh    #插入以下内容
#!/bin/bash
cat <<eof
********************
**   1.backup     **
**    2.copy      **
**    3.quit      **
********************
eof
read -p "Input a choose:" OP
case $OP in
1|backup)
   echo "BACKUP......"
   ;;
2|copy)
   echo "COPY...."
   ;;
3|quit)exit
   ;;
   *)
echo error
esac
[root@exercise1 opt]# sh case-1.sh
```

```
**********
**   1.backup     **
**    2.copy      **
**    3.quit      **
**********
Input a choose:1
BACKUP......
[root@exercise1 opt]#
```

```
例子2：编写一个启动vsftpd服务脚本
[root@exercise1 opt]# vim case-2.sh   #插入以下内容
#!/bin/bash
case $1 in
start)
        systemctl start vsftpd
        systemctl status vfstpd
        ;;
stop)
        systemctl stop vsftpd
        ;;
restart)
        systemctl restart vsftpd
        ;;
reload)
        systemctl reload vsftpd
        ;;
status)
        systemctl status vsftpd
        ;;
        *)
        echo "请输入start|stop|restart|reload|status"
esac
[root@exercise1 opt]# sh !$ status
sh case-2.sh status
Unit vsftpd.service could not be found.
```

循环语句

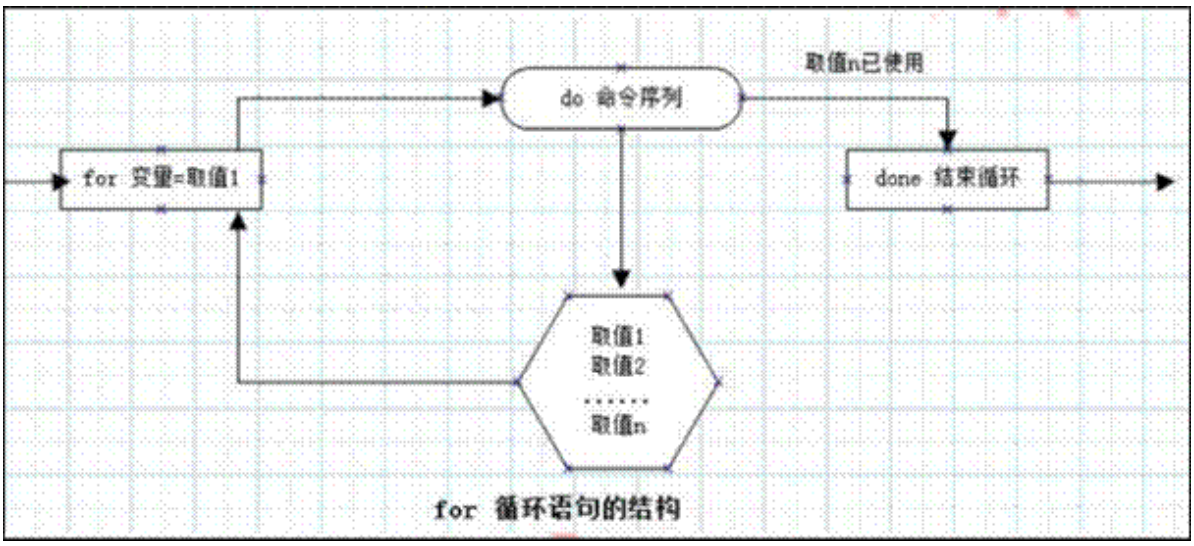
for-do-done

语法格式：

```
for var in  list
do
    commands
done

或：

for var in  list ;  do
    commands
done
```



取值列表有多种取值方式

```
一、可以直接读取in后面的值，默认以空格做分隔符
[root@exercise1 opt]# vim for-1.sh
#!/bin/bash
for var in a1 b1 c1 d1                    #in 后接内容，也可以接命令，正则
do
        echo the text is $var
done

[root@exercise1 opt]# sh !$
sh for-1.sh
the text is a1
```

```
the text is b1
the text is c1
the text is d1
[root@exercise1 opt]#
```

二、列表中的复杂值，可以使用引号' "或转义字符""来加以约束

[root@exercise1 opt]# vim for-2.sh

#!/bin/bash

for var in a1 b1 "c1 d1" e2 "hello world"

do

echo the text is $var

done

[root@exercise1 opt]# sh !$

sh for-2.sh

the text is a1

the text is b1

the text is c1 d1

the text is e2

the text is hello world

[root@exercise1 opt]#

[root@exercise1 opt]# vim for-3.sh

#!/bin/bash

for var in a1 b\'1 "c1 d1" e2 "hello world" l\'s a22 #用转义符去掉特殊含义

do

echo the text is $var

done

[root@exercise1 opt]# sh for-3.sh

the text is a1

the text is b'1

the text is c1 d1

the text is e2

the text is hello world

the text is l's

```
the text is a22
[root@exercise1 opt]#
```

三、从变量中取值

```
[root@exercise1 opt]# vim for-4.sh
#!/bin/bash
list="a1 b1 c1 d1"
for i in listdoechothisisai
done
[root@exercise1 opt]# sh !$
sh for-4.sh
this is a a1
this is a b1
this is a c1
this is a d1
```

四、从命令中取值

```
[root@exercise1 opt]# vim for-5.sh   #以空格做分隔符
#!/bin/bash
for   i   in    `cat    /etc/hosts`
do
   echo    "$i"
done
```

自定义shell分隔符

默认情况下，shell会以空格、制表符tab 、换行符\n做为分隔符。\
通过IFS来自定义为分隔符指定单个字符做分隔符：\
IFS=:   #以：冒号做分隔符\
IFS=' '   #以空格做分隔符\
可以指定多个\
如IFS=\n:;"   #这个赋值会将\、n、冒号、分号和双引号作为字段分隔符。

注：$'\n'与\n时的区别\
IFS=\n   #将字符\和字符n作为IFS的分隔符。\
IFS='#39;\n'   #真正的使用换行符做为字段分隔符。

注：分隔符会消失，并且没有先后顺序

```
例子1：指定以\n回车做为for语句的分隔符
[root@exercise1 ~]# vim /opt/for-6.sh    #插入以下内容
#!/bin/bash
IFS=$'\n'
for i in `cat /etc/hosts`
do
        echo "$i"
done
[root@exercise1 ~]# sh !$
sh /opt/for-6.sh
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

例子2：以：冒号做分隔符
[root@exercise1 ~]# vim /opt/for-7.sh #插入以下内容
#!/bin/bash
IFS=:
list=`head -1 /etc/passwd`
for i in $list do echo i
done
[root@exercise1 ~]# sh !$
sh /opt/for-7.sh
root
x
0
0
root
/root
/bin/bash
[root@exercise1 ~]#

C语言风格的for

语法格式：

```
for ((i=0;i<10;i++))    #c语言风格：先赋值，再定范围，后自增长或自减
do
    commmands
done
```

```
例子1：单个变量。输出1到10之间的数字
[root@exercise1 ~]# vim /opt/for-8.sh    #插入以下内容
#!/bin/bash
for (( i=1 ; i<=10 ; i++ ))
do
    echo num is $i
done

[root@exercise1 ~]# sh !$
sh /opt/for-8.sh
num is 1
num is 2
num is 3
num is 4
num is 5
num is 6
num is 7
num is 8
num is 9
num is 10
[root@exercise1 ~]#
```

注：

互动：i++这一条语句在for循环体中哪个位置执行?

```
for((i=1;i<=10;))#i=1只赋值一次。然后执行i<=10
do
    echo num is $i
    i=$(($i+1))   #i++在这里执行。当for循环体中所有命令执行完后，再执行i++
done
```

```
例子2：多个变量。同时输出1-9的升序和降序
[root@exercise1 ~]# vim /opt/for-9.sh
#!/bin/bash
for ((a=1,b=9 ; a<10 ; a++,b--))
do
    echo num is $a - $b
done
[root@exercise1 ~]# sh !$
sh /opt/for-9.sh
num is 1 - 9
num is 2 - 8
num is 3 - 7
num is 4 - 6
num is 5 - 5
num is 6 - 4
num is 7 - 3
num is 8 - 2
num is 9 - 1
[root@exercise1 ~]#
```

例子3：seq是连续之意，代表后面接的两个数值是一直连续的

```
[root@exercise1 ~]# vim /opt/for-10.sh
#!/bin/bash
for  i  in $(seq 1 10)                #seq 10 也是相同的结果
   do
   echo num is $i
done
[root@exercise1 ~]# sh /opt/for-10.sh
num is 1
```

num is 2
num is 3
num is 4
num is 5
num is 6
num is 7
num is 8
num is 9
num is 10

---

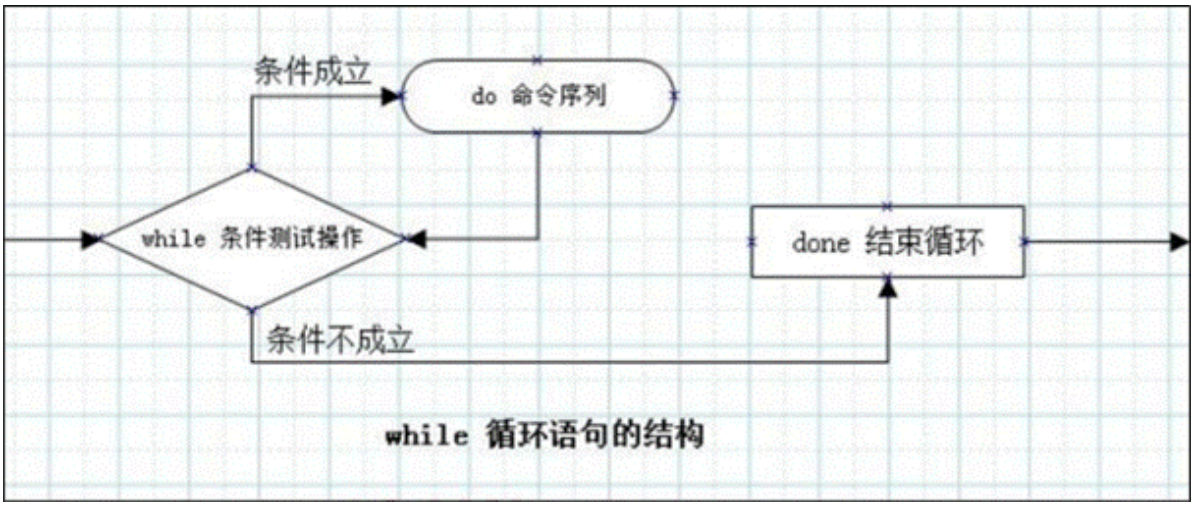
while循环语句和循环嵌套

while-do-done

重复测试指令的条件，只要条件成立就反复执行对应的命令操作，直到命令不成立或为假；

语法格式如下：

```
while 条件
do
命令
done
```

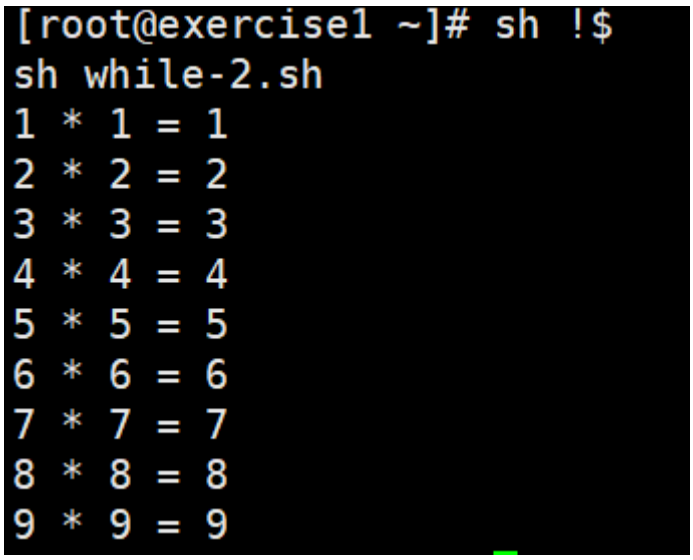


注意：避免陷入死循环while true

```
例子1：降序输出10到1的数字
[root@exercise1 ~]# vim while-1.sh
```

```
#!/bin/bash
var=10
while [ $var -gt 0 ]
do
    echo $var
    var=$[$var-1]
done
[root@exercise1 ~]# sh while-1.sh
10
9
8
7
6
5
4
3
2
1
[root@exercise1 ~]#
```

```
例子2：两数相乘
[root@exercise1 ~]# vim while-2.sh
#!/bin/bash
num=1
while [ $num -lt 10 ]
do
    sum=$(( $num * $num ))
    echo "$num * $num = $sum"
    ((num++))
    # let num++
done
```



自增操作　let var++ 相当于((var++))

自减操作　let var--　相当于((var--))

until:直到某个条件成立才结束循环

语句格式:

```
until 条件
do
    commands
done

例子1:
[root@exercise1 opt]# vim until.sh
#!/bin/bash
a=15
until [  $a -le 10 ]
do
    echo $a
    let a--
done
[root@exercise1 opt]# sh !$
sh until.sh
15
14
```

```
13
12
11
[root@exercise1 opt]#
```

例子2:
until false　　#死循环
do
　commands
done

嵌套循环

```
例子1：批量添加5个用户a.txt
[root@exercise1 ~]# vim /opt/a.txt   #添加5个用户
ab
cd
ef
gh
ij
[root@exercise1 ~]# vim /opt/for-adduser.sh
#!/bin/bash
a=`cat /opt/a.txt`
for name in $a
#for name in $(cat /opt/a.txt)
do
        id $name &> /dev/null
        if [ $? -ne 0 ];then
                useradd $name
                echo "123456" |passwd --stdin $name   &> /dev/null
                echo "user $name created"
        else
                echo "user $name is exist"
        fi
done
[root@exercise1 ~]# sh /opt/for-adduser.sh
```

```
24
25 注：&>是正确和错误的信息都重定向到/dev/null里面
```

```
[root@exercise1 ~]# sh /opt/for-adduser.sh
user ab created
更改用户 ab 的密码 。
passwd：所有的身份验证令牌已经成功更新。
user cd created
更改用户 cd 的密码 。
passwd：所有的身份验证令牌已经成功更新。
user ef created
更改用户 ef 的密码 。
passwd：所有的身份验证令牌已经成功更新。
user gh created
更改用户 gh 的密码 。
passwd：所有的身份验证令牌已经成功更新。
user ij created
[root@exercise1 ~]# 更改用户 ij 的密码 。
passwd：所有的身份验证令牌已经成功更新。
```

例子2：打印九九乘法表（三种方法）

```
1 * 1 = 1
2 * 1 = 2  2 * 2 = 4
3 * 1 = 3  3 * 2 = 6  3 * 3 = 9
4 * 1 = 4  4 * 2 = 8  4 * 3 = 12  4 * 4 = 16
5 * 1 = 5  5 * 2 = 10  5 * 3 = 15  5 * 4 = 20  5 * 5 = 25
6 * 1 = 6  6 * 2 = 12  6 * 3 = 18  6 * 4 = 24  6 * 5 = 30  6 * 6 = 36
7 * 1 = 7  7 * 2 = 14  7 * 3 = 21  7 * 4 = 28  7 * 5 = 35  7 * 6 = 42  7 * 7 = 49
8 * 1 = 8  8 * 2 = 16  8 * 3 = 24  8 * 4 = 32  8 * 5 = 40  8 * 6 = 48  8 * 7 = 56  8 * 8 = 64
9 * 1 = 9  9 * 2 = 18  9 * 3 = 27  9 * 4 = 36  9 * 5 = 45  9 * 6 = 54  9 * 7 = 63  9 * 8 = 72  9 * 9 = 81
```

提示：

echo -n 不换行输出

```
1  [root@home opt]# cat g.sh
2  #!/bin/bash
3  for i in aa bb cc dd
4  do
5      echo  $i
6  done
7  [root@home opt]# sh g.sh
8  aa
9  bb
10 cc
11 dd
```

```
12
13 [root@home opt]# cat g.sh
14 #!/bin/bash
15 for i in aa bb cc dd
16 do
17     echo -n  $i
18 done
19 [root@home opt]# sh g.sh
20 aabbccdd
```

注:

外层循环循环行，内层循环循环列

规律:内层循环的变量<=外层循环的变量

用于产生从某个数到另外一个数之间的所有整数

```
1 例3：监控服务自动启动并记录
2 [root@exercise1 opt]# vim /opt/test.sh    #得有nmap才行
3 #!/bin/bash
4 while true
5 do
6 port=$(nmap -sT 192.168.201.135 | grep tcp | grep http
  | awk '{print $2}')
7 if [ "$port" == "open" ]
8 then
9 echo "$(date) httpd is ok!" >> /tmp/autostart-acc.log
```

```
 else
 /usr/bin/systemctl start httpd &> /dev/null
 echo "$(date) restart httpd!!" >> /tmp/autostart-
err.log
 fi
 sleep 5 #检测的间隔时间为 5 秒。
 done
```

wait:循环完成，才执行命令

实战-10个shell脚本实战

实战1-将/opt目录下所有的日志文件全自动打包

实战2-找出192.168.245.128-138网段中，服务器已经关机的IP地址

实战3.判断用户输入的数据类型（字母、数字或其他）

实战4.左下三角形（使用循环语句）

```
*
**
***
****
*****
******
```

实战5.左上三角形（使用循环语句）

```
*******
******
*****
****
***
**
*
```

实战6.右下三角形（使用循环语句）

```
      *
     **
    ***
   ****
  *****
 ******
*******
```

实战7.右上三角形（使用循环语句）

```
********
 *******
  ******
   *****
    ****
     ***
      **
       *
```

实战8.编写一个脚本，计算100以内所有能被3整除数字的和

实战9.编写nginx启动脚本

实战10. 编写脚本192.168.125.0网段中哪些主机是处于开机状态，哪些主机是处于关机状态（两种方法）