shell逻辑控制

先前我们的shell脚本都是按照顺序，自上而下的依次处理。但是许多程序要求对于shell脚本可以进行逻辑流程控制，这类命令称之为 `结构化命令（structured command）`

结构化命令允许你修改程序执行的顺序。

if-then语句

语法

```
if command
then
  command
fi
```

如果你之前学过其他编程语言，if语句后面要求语句的结果得是 `True` 或 `False` 。

但是bash的if语句并不然。

bash的if语句会直接运行if后面的命令，如果该命令执行正确（状态码为0），处于then的命令就会被执行。否则就不会执行，或者执行其他逻辑的语句，最后到fi结束逻辑控制。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# testing the if statement
if pwd
then
  echo "It worked"
fi
[root@web01 ~]#
[root@web01 ~]# bash test.sh
/root
It worked
```

另一个案例

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# testing a bad command
if badcommand
then
  echo "It worked."
fi
echo "We are outside the if statement."
[root@web01 ~]#
[root@web01 ~]#
```

```
[root@web01 ~]# bash test.sh
test.sh: line 3: badcommand: command not found
We are outside the if statement.
```

这个案例就演示了if后面的命令非正确时，bash会跳过then的语句。

if-then-else语句

在这里，就增加了一个逻辑的选择

```
if command
then
  command
else
  command
fi
```

案例

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# testing the else section
testuser=someone
if grep $testuser /etc/passwd
then
  echo "The bash file for user $testuser are:"
  ls -a /home/$testuser/.b*
  echo
else
  echo "The user $testuser does not exist on this system."
  echo
fi

# 结果
[root@web01 ~]# bash test.sh
The user someone does not exist on this system.
```


可以看出语句翻译是

```
如果 ..
那么
  执行...
否则
  执行...
结束
```

嵌套if

其实就是比刚才多几个判断，来看超哥怎么写

```
1.创建一个测试的目录
[root@web01 ~]# mkdir /home/cccc

2.脚本
[root@web01 ~]# cat test.sh
#!/bin/bash
testuser=cccc

if grep $testuser /etc/passwd
then
  echo "The user $testuser exists on this system."
else
  echo "The user $testuser does not exist on this system."
  if ls -d /home/$testuser/
    then
      echo "However,$testuser has a directory."
  fi
fi

3.执行
[root@web01 ~]# bash test.sh
The user cccc does not exist on this system.
/home/cccc/
However,cccc has a directory.
```

这种写法问题在于代码不易阅读，很难理清逻辑。

elif

语法

```
if command1
then
  commands
elif command2
then
  more commands
fi

# 超哥翻译
如果 ...
  那么
    执行..
又如果...
  那么
    执行...
结束
```

案例，这样的逻辑，更适合阅读。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
testuser=cccc
if grep $testuser /etc/passwd
then
  "The user $testuser exists on this system."
elif ls -d /home/$testuser
then
  echo "The user $testuser does not exist on this system."
  echo "However,$testuser has a directory."
fi
[root@web01 ~]# bash test.sh
/home/cccc
The user cccc does not exist on this system.
However,cccc has a directory.
```

再来一个案例，用于判断 `cccc` 用户的两种情况

- 有家目录，用户信息被删除
- 没有家目录，用户也不存在

```
[root@web01 ~]# cat test.sh
#!/bin/bash
testuser=cccc
if grep $testuser /etc/passwd
then
  echo "The user $testuser exists on this system."
elif ls -d /home/$testuser
then
  echo "The user $testuser does not exist on this system."
  echo "However,$testuser has a directory."
else
  echo "The user $testuser does not exist on this system."
  echo "And $testuser does not have a directory."
fi

# 执行分别看结果
[root@web01 ~]# bash test.sh
/home/cccc
The user cccc does not exist on this system.
However,cccc has a directory.


# 删除家目录，再执行
[root@web01 ~]# rmdir /home/cccc/
[root@web01 ~]# bash test.sh
ls: cannot access /home/cccc: No such file or directory
The user cccc does not exist on this system.
And cccc does not have a directory.
```

test命令

test 命令最短的定义可能是评估一个表达式；如果条件为真，则返回一个 0 值。如果表达式不为真，则返回一个大于 0 的值 — 也可以将其称为假值。检查最后所执行命令的状态的最简便方法是使用 $? 值。

参数：

1. 关于某个文件名的『类型』侦测(存在与否)，如 test -e filename

-e 该『文件名』是否存在? (常用)
-f 该『文件名』是否为文件(file)? (常用)
-d 该『文件名』是否为目录(directory)? (常用)
-b 该『文件名』是否为一个 block device 装置?
-c 该『文件名』是否为一个 character device 装置?
-S 该『文件名』是否为一个 Socket 文件?
-p 该『文件名』是否为一个 FIFO (pipe) 文件?
-L 该『文件名』是否为一个连结档?

2. 关于文件的权限侦测，如 test -r filename

-r 侦测该文件名是否具有『可读』的属性?
-w 侦测该文件名是否具有『可写』的属性?
-x 侦测该文件名是否具有『可执行』的属性?
-u 侦测该文件名是否具有『SUID』的属性?
-g 侦测该文件名是否具有『SGID』的属性?
-k 侦测该文件名是否具有『Sticky bit』的属性?
-s 侦测该文件名是否为『非空白文件』?

3. 两个文件之间的比较，如： test file1 -nt file2

-nt (newer than)判断 file1 是否比 file2 新
-ot (older than)判断 file1 是否比 file2 旧
-ef 判断 file2 与 file2 是否为同一文件，可用在判断 hard link 的判定上。 主要意义在判定，两个文件是否均指向同一个 inode 哩!

4. 关于两个整数之间的判定，例如 test n1 -eq n2

-eq 两数值相等 (equal)
-ne 两数值不等 (not equal)
-gt n1 大于 n2 (greater than)
-lt n1 小于 n2 (less than)
-ge n1 大于等于 n2 (greater than or equal)
-le n1 小于等于 n2 (less than or equal)

5. 判定字符串的数据

test -z string 判定字符串是否为 0 ？若 string 为空字符串，则为 true
test -n string 判定字符串是否非为 0 ？若 string 为空字符串，则为 false。
注： -n 亦可省略
test str1 = str2 判定 str1 是否等于 str2 ，若相等，则回传 true
test str1 != str2 判定 str1 是否不等于 str2 ，若相等，则回传 false

```
6. 多重条件判定，例如： test -r filename -a -x filename

-a (and)两状况同时成立！例如 test -r file -a -x file，则 file 同时具有 r 与 x 权限时，才回传 true。
-o (or)两状况任何一个成立！例如 test -r file -o -x file，则 file 具有 r 或 x 权限时，就可回传 true。
! 反相状态，如 test ! -x file ，当 file 不具有 x 时，回传 true
```

test命令对于shell脚本是重要的命令，提供了在`if-then`语句里测试不同条件的路径。

我们来看用法，条件为真，返回`0`，条件不成立，返回`大于0`的值。

```
1.判断文件存在
[root@web01 ~]# test hello.sh
[root@web01 ~]# echo $?
0

2.判断目录
[root@web01 ~]# test -d data
[root@web01 ~]# echo $?
0

3.测试可写权限
[root@web01 ~]# test -w hello.sh
[root@web01 ~]# echo $?
0

4.测试执行权限
[root@web01 ~]# test -x hello.sh
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# echo $?
1

5.测试文件是否有内容，有则0，无则1
[root@web01 ~]# cat hello.sh
#!/bin/bash
echo 'hello 超哥，你讲的课真有意思'
[root@web01 ~]#
[root@web01 ~]# test -s hello.sh
[root@web01 ~]#
[root@web01 ~]# echo $?
0
```


结合控制语句

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# Testing the test command
my_var="Full"
if test $my_var
then
  echo "The $my_var expression returns a True."
```

```
else
  echo "The $my_var expression returns a False."
fi
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
The Full expression returns a True.


若是删除my_var变量，该脚本，就False了。
```

简洁的测试方法

bash提供了可以不用test命令的写法，进行判断

```
# 注意空格
if [ 条件 ]
then
  commands
fi

在中括号里，写入测试条件。
```


数值比较

test命令常见用法是数值比较。

比　　较	描　　述
n1 -eq n2	检查n1是否与n2相等
n1 -ge n2	检查n1是否大于或等于n2
n1 -gt n2	检查n1是否大于n2
n1 -le n2	检查n1是否小于或等于n2
n1 -lt n2	检查n1是否小于n2
n1 -ne n2	检查n1是否不等于n2

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# Using numeric test evaluations
value1=10
value2=11
if [ $value1 -gt 5 ]
then
  echo "The test value $value1 is greater than 5"
fi
if [ $value1 -eq $value2 ]
then
  echo "The values are equal."
else
```

```
  echo "The values are different."
fi
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
The test value 10 is greater than 5
The values are different.
```

> 注意bash只能处理整数。

案例

-d 测试目录是否存在，在目录操作时，判断下是否存在是个好习惯。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
jump_directory=/home/cccc
if [ -d $jump_directory ]
then
  echo "The $jump_directory exists."
  cd $jump_directory
  ls
else
  echo "The $jump_directory does not exist."
fi
```


符合条件测试

> [条件] && [条件2]
>
> [条件] || [条件2]
>
> 布尔运算符

案例

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

if [ -d $HOME ] && [ -w $HOME/testing ]
then
  echo "The file exists and you can write to it."
else
  echo "I cannot wirte to the file."
fi
```

```
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
I cannot wirte to the file.

# 对文件创建，修改权限测试
```

双括号特性

###双小括号

bash支持双括号，写入高级数学表达式。

符　　号	描　　述
val++	后增
val--	后减
++val	先增
--val	先减
!	逻辑求反
~	位求反
**	幂运算
<<	左位移
>>	右位移
&	位布尔和
|	位布尔或
&&	逻辑和
||	逻辑或

可以在if语句里使用双括号，可以用在普通的命令。

```
# 10的平方是100，脚本

[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

val1=10
if (( $val1 ** 2 > 90 ))
then
  (( val2 = $val1**2 ))
  echo "The square of $val1 is $val2"
fi
[root@web01 ~]#
[root@web01 ~]#
```

```
[root@web01 ~]# bash test.sh
The square of 10 is 100

# 注意双括号里不需要转义，val2语句是赋值语句
```

双方括号

双方括号提供了针对字符串的高级特性，模式匹配，正则表达式的匹配。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

if [[ $USER == r* ]]
then
  echo "Hello $USER"
else
  echo "Sorry,I do not know you."
fi
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
Hello root
[root@web01 ~]#
```

在双中括号里，进行了`==`双等号，进行字符串匹配`r*`，也就找到了`root`。

case语句

当我们有一个场景，在计算后，要进行多个组的值判断，用一个low的办法写。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

if [[ $USER = "root" ]]
then
  echo "Welcome $USER"
  echo "Please enjoy your visit"
elif [ $USER = "chaoge" ]
then
  echo "Welcome $USER"
  echo "Please enjoy your visit"
elif [ $USER = "testing" ]
then
  echo "Special testing account"
elif [ $USER = "cc" ]
then
  echo "Do not forget to logout when you're done."
else
```

```
  echo "Sorry,you are not allowed here."
fi
[root@web01 ~]#

# 执行
[root@web01 ~]# bash test.sh
Welcome root
Please enjoy your visit
```

我们会发现，这样的写法，重复性太高，有点太啰嗦了。

有了case语句你就会发现，再也不需要写那么多elif不停的检查同一个变量 `$USER` 。

不low的写法

```
语法:
#! /bin/sh -

name=`basename $0 .sh`
case $1 in
 s|start)
        echo "start..."
        ;;
 stop)
        echo "stop ..."
        ;;
 reload)
        echo "reload..."
        ;;
 *)
        echo "Usage: $name [start|stop|reload]"
        exit 1
        ;;
esac
exit 0


# 语法

case "变量" in
值1)
  命令
  ;;
值2)
  命令2
  ;;
*)
  命令3
esac
```


案例

```
[root@web01 ~]# cat test.sh
```

```
#!/bin/bash
# 超哥带你学shell

case $USER in
root | chaoge)
  echo "Welcome,$USER"
  echo "Please enjoy your visit"
;;
cc)
  echo "Welcome cc"
;;
yuchao)
  echo "Welcome yuchao"
;;
*)
  echo "Sorry,you are not allowed here"
;;
esac

[root@web01 ~]# bash test.sh
Welcome,root
Please enjoy your visit
```

这样的case语句，比起写多个if-elif是简单的多了。

for命令

for语句提供了重复一些过程的作用，也就是循环执行一组命令，直到某个特定条件结束。

```
bash提供for命令，允许你遍历一组值
语法
for var in list
do
  commands
done
```

for循环在list变量值里，反复迭代，第一次迭代，使用第一个值，第二次用第二个值，以此类推，直到所有元素都过一遍。

案例

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for test in yuchao apple orange potato
do
  echo "The next is $test"
done
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]#
```

```
[root@web01 ~]# bash test.sh
The next is yuchao
The next is apple
The next is orange
The next is potato
```

for循环的坑

```
1.看案例
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for test in I don't know if this'll work
do
  echo "word: $test"
done
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
word: I
word: dont know if thisll
word: work
```


本意我们是想以空格区分，循环出每个单词，这里shell被单引号给误导了，这里得进行转义，解决办法

- 转义符
- 双引号

```
# 双引号
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for test in I "don't" know if "this'll" work
do
  echo "word: $test"
done

# 转义符
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for test in I don\'t  know if this\'ll work
do
  echo "word: $test"
done
```

空格坑

for默认循环是以空格区分，如果你的数据包含了空格，那又该怎么办

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for test in China New York
do
  echo "Now going to $test"
done

[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
Now going to China
Now going to New
Now going to York
```

这不是我们要的结果，用双引号解决这个问题

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for test in China "New York"
do
  echo "Now going to $test"
done

[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
Now going to China
Now going to New York
```


从变量遍历

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell
list="yuchao apple orange potato"
for state in $list
do
  echo "Hello $state?"
done
[root@web01 ~]#
[root@web01 ~]#
```

```
[root@web01 ~]#
[root@web01 ~]# bash test.sh
Hello yuchao?
Hello apple?
Hello orange?
Hello potato?
```

从文件里遍历

```
1.准备文件
[root@web01 ~]# cat test.txt
yuchao
apple
orange
potato

2.for脚本
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

file="./test.txt"

for state in $(cat $file)
do
  echo "Hello $state"
done
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
Hello yuchao
Hello apple
Hello orange
Hello potato
```


##通配符遍历

可以使用for命令自动遍历目录中的文件，得使用 通配符 支持匹配。

遍历判断文件/目录

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for file in /opt/*
do
  if [ -d "$file" ]
```

```
  then
    echo "$file is a directory"
  elif [ -f "$file" ]
  then
    echo "$file is a file"
  fi
done
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
/opt/2020-07-22-16-48-00-docker-guacamole-v2.1.0.tar.gz is a file
/opt/apache-tomcat-9.0.36.tar.gz is a file
/opt/guacamole is a directory
/opt/jumpserver is a directory
```

for循环在/opt目录下挨个遍历，因为`*`就已经匹配了所有的内容，然后`-d`参数进行判断目录，以此进行逻辑判断，到底是目录或是其他。

遍历多个目录

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for file in /home/* /root/*.txt
do
  if [ -d "$file" ]
  then
    echo "$file is a directory"
  elif [ -f "$file" ]
  then
    echo "$file is a file"
  fi
done
```


##While循环

while循环命令可以理解是`if-then`和`for`循环的混杂体。

```
语法
while test command
do
  other command
done
```

while语句的`test command`和`if-then`语句格式一样，可以使用任何的普通bash shell命令。

注意的是while的`test command`的退出状态码，必须随着循环里的命令改变，否则状态码如果不变化，循环会不停止的继续下去。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
```

```
# 超哥带你学shell

var1=10
while [ $var1 -gt 0 ]
do
  echo $var1
  var1=$[ $var1 -1 ]
done
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
10
9
8
7
6
5
4
3
2
1
```

只要条件成立，while命令就会不断的循环下去，直到条件不成立，结束循环。

多个测试命令

while可以写入多个测试命令，只有最后一个测试命令的退出状态码会被决定是否退出循环。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

var1=10
while echo $var1
  [ $var1 -ge 0 ]
do
  echo "This is inside the loop"
  var1=$[ $var1 -1 ]
done
```

注意写法，换行，多个测试命令要单独的出现在一行。

这里是先显示var1变量的值，第二个测试是方括号里判断var1是否大于0

在循环体内部，先打印消息，然后进行变量计算。

因此脚本执行结果是

```
[root@web01 ~]# bash test.sh
10
```

```
This is inside the loop
9
This is inside the loop
8
This is inside the loop
7
This is inside the loop
6
This is inside the loop
5
This is inside the loop
4
This is inside the loop
3
This is inside the loop
2
This is inside the loop
1
This is inside the loop
0
This is inside the loop
-1
```

unitl命令

until和while是相反的语意，until命令要求你指定一个 返回非零退出码的测试命令 。

只有退出状态码不是0，bash才会执行循环的命令。

```
语法：
until test commands
do
  other commands
done
```

until也支持多个测试命令，只有最后一个决定bash是否执行其他命令。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

var1=100
until [ $var1 -eq 0 ]
do
  echo $var1
  var1=$[ $var1 -25 ]
done

[root@web01 ~]#
[root@web01 ~]#
```

```
[root@web01 ~]#
[root@web01 ~]# bash test.sh
100
75
50
25
```

多条件until

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# 超哥带你学shell

var1=100

until echo $var1
  [ $var1 -eq 0 ]
do
  echo "Inside the loop: $var1"
  var1=$[ $var1 - 25 ]
done
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# bash test.sh
100
Inside the loop: 100
75
Inside the loop: 75
50
Inside the loop: 50
25
Inside the loop: 25
0
```


##嵌套循环

也就是在循环体内，再写上循环语句命令，使用嵌套循环要小心了。

###C语言风格的for循环

案例

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash

for (( i=1;i<=10;i++ ))
do
  echo "The next number is $i"
done
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
```

```
The next number is 1
The next number is 2
The next number is 3
The next number is 4
The next number is 5
The next number is 6
The next number is 7
The next number is 8
The next number is 9
The next number is 10

for循环通过定义好的变量i，进行迭代的使用。
```

###for使用多个变量

C语言风格的for循环允许迭代多个变量，循环会单独的处理每个变量，可以为每个变量定义不同的迭代过程。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash

for (( a=1,b=10;a<=10;a++,b-- ))
do
  echo "$a - $b"
done
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
1 - 10
2 - 9
3 - 8
4 - 7
5 - 6
6 - 5
7 - 4
8 - 3
9 - 2
10 - 1
```


每次迭代，a在增加的同时，减小了b变量。

###嵌套循环案例

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash

for (( a = 1;a <=3;a++ ))
do
  echo "Starting loop $a:"
  for (( b = 1; b <=3; b++ ))
  do
    echo "  Inside loop:$b"
  done
```

```
done
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Starting loop 1:
  Inside loop:1
  Inside loop:2
  Inside loop:3
Starting loop 2:
  Inside loop:1
  Inside loop:2
  Inside loop:3
Starting loop 3:
  Inside loop:1
  Inside loop:2
  Inside loop:3
```

##循环处理文件数据

遍历读取文件内的数据，一般用到两个知识点：

- 嵌套循环
- 修改IFS环境变量

修改IFS环境变量，是因为文件里的数据都是单独的每一行

IFS字段分隔符

IFS是一个特殊的环境变量，叫做内部字段分隔符，IFS环境变量定义了bash shell用作字段分隔符的一系列字符，bash shell将如下字段当作分隔符

- 空格
- 制表符
- 换行符

案例

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash

IFS.OLD=$IFS
IFS=$'\n'
for entry in $(cat /etc/passwd)
do
  echo "Value in $entry ."
  IFS=:
  for value in $entry
  do
    echo "  $value"
  done
done
```

这里修改了两个不同的IFS变量，来进行解析文件数据

- 第一个IFS=$'\n'，用于解析文件的每一行
- 第二个IFS=: ，是修改IFS的值为冒号，然后解析每一行单独的值。

控制循环

刚才我们学习的都是循环后自动结束，也是有办法手动结束的。

- break
- continue

break用于强制退出任意类型的循环。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell
for var1 in `seq 10`
do
  if [ $var1 -eq 5 ]
  then
    break
  fi
  echo "Iteration number: $var1"
done
echo "The for loop is completed"

[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
The for loop is completed
```


break也适用于while，untile循环。

终止内部循环break

跳出多个循环时，break会自动终止你在的最内层循环。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for (( a = 1; a < 4 ; a++ ))
do
  echo "Outer loop: $a"
  for (( b = 1; b < 100; b++ ))
```

```
  do
    if [ $b -eq 5 ]
    then
      break
    fi
    echo "  Inner loop: $b"
  done
done
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Outer loop: 1
  Inner loop: 1
  Inner loop: 2
  Inner loop: 3
  Inner loop: 4
Outer loop: 2
  Inner loop: 1
  Inner loop: 2
  Inner loop: 3
  Inner loop: 4
Outer loop: 3
  Inner loop: 1
  Inner loop: 2
  Inner loop: 3
  Inner loop: 4
```

外层循环是执行结束了，内部循环到了5立即结束了。

终止外层循环break

语法

```
break n
n表示跳出的循环层级，默认是1，下一层就是2
```

案例

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for (( a = 1; a < 4; a++ ))
do
  echo "Outer loop: $a"
  for (( b = 1;b < 100; b++ ))
  do
    if [ $b -gt 4 ]
    then
      break 2
    fi
```

```
    echo "  Inner loop: $b"
  done
done

[root@chaogelinux shell]# bash test.sh
Outer loop: 1
  Inner loop: 1
  Inner loop: 2
  Inner loop: 3
  Inner loop: 4
```

当内层`$b`等于4的时候，立即停止外层，也就结束了。

continue

continue用于跳过某次循环，当条件满足时，然后继续循环。

```
[root@chaogelinux shell]# cat test.sh
#!/bin/bash
# 超哥带你学shell

for (( var1 = 1;var1<15;var1++ ))
do
  if [ $var1 -gt 5 ] && [ $var1 -lt 10 ]
  then
    continue
  fi
  echo "Iteration number: $var1"
done
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Iteration number: 1
Iteration number: 2
Iteration number: 3
Iteration number: 4
Iteration number: 5
Iteration number: 10
Iteration number: 11
Iteration number: 12
Iteration number: 13
Iteration number: 14
```


当条件是大于5且小于10的时候，shell会执行continue命令，跳过此阶段的循环条件。

处理循环的输出

在shell脚本里，循环输出后的结果，可以进行重定向输出

```
[root@chaogelinux shell]# cat test.sh
```

```
#!/bin/bash
# 超哥带你学shell

for (( a = 1;a<10;a++ ))
do
  echo "The number is $a"
done > test.txt
echo "Finished"

[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]#
[root@chaogelinux shell]# bash test.sh
Finished
[root@chaogelinux shell]#
[root@chaogelinux shell]# cat test.txt
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9
```