测试和判断

1、测试

程序运行中经常需要根据实际情况来运行特定的命令或代码块。比如，判断某个文件或目录是否存在，如果不存在，则需要创建。例如：

[root@localhost ~]# ls /var/log/messages
/var/log/messages
[root@localhost ~]# echo $?
0　　//输出为0，则文件存在
[root@localhost ~]# ls /var/log/messages004
ls: 无法访问/var/log/messages004: 没有那个文件或目录
[root@localhost ~]# echo $?
2　　//输出为非0,则文件不存在

测试结构：

测试的第一种使用方式是直接使用test命令，格式：

test expression

其中expression是一个表达式，可以是算术比较、字符串比较、文本和文件属性比较等

第二种测试方式是使用”[“启动一个测试，再写expression，再以”]“结束测试。需要注意的是括号和表达式expression之间都有空格。推荐使用第二种方式。

[expression]

文件测试

shell中提供了大量的文件测试符，其格式如下：

test file_operator FILE

或者

[file_operator FILE]

其中file_operator是文件测试符，FILE是文件、目录（可以是文件或目录的全路径）

例如：判断文件是否存在

[root@localhost ~]# [-e /var/log/messages]
[root@localhost ~]# echo $?
0
[root@localhost ~]# [-e /var/log/messages5]
[root@localhost ~]# echo $?
1

表 14-1　文件测试符

文件测试	说　明
-b FILE	当文件存在且是个块文件时返回真，否则为假
-c FILE	当文件存在且是个字符设备时返回真，否则为假
-d FILE	当文件存在且是个目录时返回真，否则为假
-e FILE	当文件或者目录存在时返回真，否则为假
-f FILE	当文件存在且为普通文件时返回真，否则为假
-x FILE	当文件存在且为可执行文件时返回真，否则为假

文件测试	说　明
-w FILE	当文件存在且为可写文件时返回真，否则为假
-r FILE	当文件存在且为可读文件时返回真，否则为假
-l FILE	当文件存在且为连接文件时返回真，否则为假
-p FILE	当文件存在且为管道文件时返回真，否则为假
-s FILE	当文件存在且大小不为 0 时返回真，否则为假
-S FILE	当文件存在且为 socket 文件时返回真，否则为假
-g FILE	当文件存在且设置了 SGID 时返回真，否则为假
-u FILE	当文件存在且设置了 SUID 时返回真，否则为假
-k FILE	当文件存在且设置了 sticky 属性时返回真，否则为假
-G FILE	当文件存在且属于有效的用户组时返回真，否则为假
-O FILE	当文件存在且属于有效的用户时返回真，否则为假
FILE1 -nt FILE2	当 FILE1 比 FILE2 新时返回真，否则为假
FILE1 -ot FILE2	当 FILE1 比 FILE2 旧时返回真，否则为假

例如：

```
[root@sunday-test shell-script]# cat rwx.sh
#!/bin/bash
read -p "what file do you want to test?:" filename
if [ ! -e "$filename" ]; then
  echo "the file does not exist."
  exit 1
fi

if [ -r "$filename" ]; then
  echo "$filename is readable."
fi
if [ -w "$filename" ]; then
  echo "$filename is writeable."
fi
if [ -x "$filename" ]; then
```

```
  echo "$filename is executable."
fi
```

字符串测试

shell中的字符串比较主要有等于、不等于、大于、小于、是否为空等测试。

字符串测试	说　明
-z "string"	字符串 string 为空时返回真，否则为假
-n "string"	字符串 string 非空时返回真，否则为假
"string1" = "string2"	字符串 string1 和 string2 相同时返回真，否则为假
"string1" != "string2"	字符串 string1 和 string2 不相同时返回真，否则为假
"string1" > "string2"	按照字典排序，字符串 string1 排在 string2 之前时返回真，否则为假
"string1" < "string2"	按照字典排序，字符串 string1 排在 string2 之后时返回真，否则为假

测试str1是否为空：

```
[root@sunday-test ~]# str1=""
[root@sunday-test ~]# test -z "$str1"
[root@sunday-test ~]# echo $?
0
```

测试str1是否非空，非空则返回0

```
[root@sunday-test ~]# test -n "$str1"
[root@sunday-test ~]# echo $?
1
```

定义非空字符串str2，并测试其是否为空

```
[root@sunday-test ~]# str2="hello"
[root@sunday-test ~]# [ -z "$str2" ]
[root@sunday-test ~]# echo $?
1
```

测试str2是否非空，非空返回0

```
[root@sunday-test ~]# [ -n "$str2" ]
[root@sunday-test ~]# echo $?
0
```

比较str1和str2是否相同

```
[root@sunday-test ~]# [ "str1" = "str2" ]
[root@sunday-test ~]# echo $?
1
[root@sunday-test ~]# [ "str1" != "str2" ]
[root@sunday-test ~]# echo $?
0
```

比较str1和str2的大小，注意的是>和<都需要转义

```
[root@sunday-test ~]# [ "str1" \> "str2" ]
[root@sunday-test ~]# echo $?
1
[root@sunday-test ~]# [ "str1" \< "str2" ]
[root@sunday-test ~]# echo $?
0
```

如果不想转义，也可以使用[[]]

```
[root@sunday-test ~]# [[ "str1" < "str2" ]]
[root@sunday-test ~]# echo $?
0
```

整数比较

整数测试是一种简单的算术运算，作用在于比较两个整数的大小关系，测试成立则返回0，否则返回非0值。格式：

test "num1" num_operator "num2"

或者

["num1" num_operator "num2"]

num_operator是整数测试符：

整数比较	说　明
"num1" -eq "num2"	如果 num1 等于 num2 则返回真，否则为假。其中 eq 为 equal
"num1" -gt "num2"	如果 num1 大于 num2 则返回真，否则为假。其中 gt 为 great than
"num1" -lt "num2"	如果 num1 小于 num2 则返回真，否则为假。其中 lt 为 less than
"num1" -ge "num2"	如果 num1 大于等于 num2 则返回真，否则为假。其中 ge 为 great equal
"num1" -le "num2"	如果 num1 小于等于 num2 则返回真，否则为假。其中 le 为 less equal
"num1" -ne "num2"	如果 num1 不等于 num2 则返回真，否则为假。其中 ne 为 not equal

```
[root@sunday-test ~]# a=10
[root@sunday-test ~]# b=12
[root@sunday-test ~]# [ $a -eq $b ]
[root@sunday-test ~]# echo $?
1
[root@sunday-test ~]# [ $a -gt $b ]
[root@sunday-test ~]# echo $?
1
[root@sunday-test ~]# [ $a -lt $b ]
[root@sunday-test ~]# echo $?
0
```

逻辑测试符和逻辑运算符

逻辑测试用于连接多个测试条件，并返回整个表达式的值。逻辑测试主要有逻辑非、逻辑与、逻辑或3种。逻辑测试符如下：

表 14-4　逻辑测试符

逻辑运算	说　明
! expression	如果 expression 为真，则测试结果为假
expression1 -a expression2	expression1 和 expression2 同时为真，则测试结果为真
expression1 -o expression2	expression1 和 expression2 只要有一个为真，则测试结果为真

逻辑非：

```
[root@sunday-test ~]# [ ! -e /var/log/messages ]
[root@sunday-test ~]# echo $?
1
```

逻辑与：

```
[root@sunday-test ~]# [ -e /var/log/messages -a -e /var/log/messages01 ]
[root@sunday-test ~]# echo $?
1
```

逻辑或：

```
[root@sunday-test ~]# [ -e /var/log/messages -o -e /var/log/messages01 ]
[root@sunday-test ~]# echo $?
0
```

shell中的逻辑运算符，也有逻辑非、逻辑与、逻辑或 3种

表 14-5　逻辑运算符

逻辑运算	说　明
!	逻辑非，对真假取反
&&	逻辑与，连接两个表达式，只有两个表达式为真结果才为真
||	逻辑或，连接两个表达式，只要有一个表达式为真结果就为真

逻辑非：

```
[root@sunday-test ~]# ! [ -e /var/log/messages ]
[root@sunday-test ~]# echo $?
1
```

逻辑与：

```
[root@sunday-test ~]# [ -e /var/log/messages ] && [ -e
/var/log/messages01 ]
[root@sunday-test ~]# echo $?
1
```

逻辑或：

[root@sunday-test ~]# [-e /var/log/messages] || [-e /var/log/messages01]

[root@sunday-test ~]# echo $?

0

2、判断

有了测试，就要有获得测试结果的机制，并根据测试结果运行不同的代码段，实现程序的流程控制。

if判断结构

if是最简单的判断语句，可以针对测试结果做相应处理：如果测试为真则运行相关代码，其语法结构如下：

```
if  expression; then
        command
fi
```

如果expression测试返回真，则执行command。如果要执行的不止一条命令，则不同命令之间用换行符隔开，如下所示：

```
if  expression; then
        command1
        command2
        ....
fi
```

例如：

```
[root@sunday-test shell-script]# cat score01.sh
#!/bin/bash
echo -n "please input a score:"
read SCORE
if [ "$SCORE" -lt 60 ]; then
        echo "C"
fi
if [ "$SCORE" -lt 80 -a "$SCORE" -ge 60 ]; then
        echo "B"
fi
if [ "$SCORE" -ge 80 ]; then
       echo "A"
fi
```

if/else判断结构

如果if后的判断成立，则执行then后面的内容；否则执行else后面的内容。语法结构如下：

```
if  expression; then
          command
else
          command
fi
```

例如：检查文件是否存在

```
[root@sunday-test shell-script]# cat check_file.sh
#!/bin/bash
FILE=/var/log/messages
#FILE=/var/log/messages01

if [ -e $FILE ];then
     echo "$FILE exists"
else
     echo "$FILE not exist"
fi
```

if/elif/else判断结构

可以代替if嵌套，语法结构如下：

```
if  expression1;then
        command1
elif  expression2; then
         command2
elif   expression3 ; then
         command3
...
fi
```

例如：

```
[root@sunday-test shell-script]# cat score02.sh
#!/bin/bash
echo -n "please input a score:"
read SCORE
```

```
if [ "$SCORE" -lt 60 ]; then
        echo "C"
elif  [ "$SCORE" -lt 80 -a "$SCORE" -ge 60 ]; then
        echo "B"
else
      echo "A"
fi
```

case判断结构

和if/elif/else判断结构一样，case判断结构也可以用于多种可能情况下的分支选择。

语法：

```
case VAR in
var1)  command1 ;;
var2)  command2 ;;
var3)  command3 ;;
...
*)  command4 ;;
esac
```

其原理为从上到下依次比较VAR和var1、var2、var3的值是否相等，如果匹配则执行其后面的命令语句，都不匹配，则匹配最后的默认*，执行其后面的默认命令。要注意的是，case判断结构中的var1、var2、var3等这些值只能是常量或正则表达式。

检测当前操作系统

```
[root@sunday-test shell-script]# cat os_type.sh
#!/bin/bash
OS=`uname -s`
case "$OS" in
FreeBSD)
   echo  “this is FreeBSD”  ;;
SunOS)
   echo "this is Solaris" ;;
Darwin)
   echo  "this is Mac OSX" ;;
AIX)
   echo "this is AIX" ;;
Minix)
   echo "this is Minix" ;;
```

```
    Linux)
        echo  "this is Linux" ;;
    *)
        echo  "Failed to identify this OS" ;;
    esac
```

下面的脚本可用于检测用户的输入中是否含有大写字母、小写字母或者数字，这里case匹配的值是正则表达式。

```
    [root@sunday-test shell-script]# cat detect_input.sh
    #!/bin/bash
    read -p "Give me a word: " input
    echo -en "You gave me some "
    case $input in
          *[[:lower:]]*)  echo -en "Lowercase" ;;
          *[[:upper:]]*)  echo -en "Uppercase" ;;
          *[[:digit:]]*)  echo -en  "Numerical" ;;
           *)  echo "unknown input." ;;
    esac
```

循环

shell中的循环主要有for、while、until、select几种

for循环：是最常见的循环结构。for循环是一种运行前测试语句，也就是在运行任何循环体之前先要判断循环条件是否成立，只有在条件成立的情况下才会运行循环体，否则将退出循环。每完成一次循环后，在进行下一次循环之前都会再次进行测试。

带列表的for循环

用于执行一定次数的循环（循环次数等于列表元素个数），其语法结构如下：

```
for  VARIABLE  in   (list)
do
            command
done
```

例如：

```
[root@sunday-test shell-script]# cat fruit01.sh
#!/bin/bash
for FRUIT in apple  orange banana pear
do
      echo "$FRUIT is john's favorite"
```

done
echo "no more fruits"
执行结果如下：
[root@sunday-test shell-script]# ./fruit01.sh
apple is john's favorite
orange is john's favorite
banana is john's favorite
pear is john's favorite
no more fruits
为了便于修改，可以先定义变量，然后用变量替代列表
[root@sunday-test shell-script]# cat fruit02.sh
#!/bin/bash
fruits="apple orange banana pear"

for FRUIT in ${fruits}
do
        echo "$FRUIT is john's favorite"
done
echo "no more fruits"
列表也可以是数字：
[root@sunday-test shell-script]# cat for_list01.sh
#!/bin/bash
for VAR in 1 2 3 4 5
do
     echo "Loop $VAR times"
done
执行结果：
[root@sunday-test shell-script]# ./for_list01.sh
Loop 1 times
Loop 2 times
Loop 3 times
Loop 4 times
Loop 5 times
数字多了以后，可以写成这样：
[root@sunday-test shell-script]# cat for_list02.sh

```
#!/bin/bash
for VAR in {1..10}
do
     echo "Loop  $VAR  times"
done
```

还可以使用seq命令结合命令替换的方式生成列表。

```
[root@sunday-test ~]# cat for_list03.sh
#!/bin/bash
sum=0
for VAR in `seq 1 100`
#for VAR in $(seq 1 100)
do
   let "sum+=${VAR}"
done
echo "Total: ${sum}"
```

运行结果：

```
[root@sunday-test shell-script]# ./for_list03.sh
Total: 5050
```

还可以使用“步长”计算1到100的奇数和

```
[root@sunday-test shell-script]# cat for_list04.sh
#!/bin/bash
sum=0
for VAR in `seq 1 2 100`
#for VAR in $(seq 1 2 100)
do
   let "sum+=${VAR}"
done
echo "Total: ${sum}"
```

运行结果：

```
[root@sunday-test shell-script]# ./for_list04.sh
Total: 2500
```

列表也可以是命令的输出

```
[root@sunday-test shell-script]# cat for_list05.sh
#!/bin/bash
for VAR in $(ls)
```

```
do
        ls -l $VAR
done
```

类C的for循环

格式如下：

```
for  ((expression1；expression2; expression3))
do
          command
done
```

其中，expression1为初始化语句，一般用作变量定义和初始化；expression2为判断表达式，用于测试表达式返回值并以此控制循环，返回值为真则循环继续，返回值为假则退出循环；expression3用于变量值修改，从而影响expression2的返回值，并以此影响循环行为。

例如：

```
[root@sunday-test shell-script]# cat c_for01.sh
#!/bin/bash
for ((i=1;i<=10;i++))
do
   echo -n "$i "
done
```

还可以初始化多个值，例如：

```
[root@www shell-script]# cat c_for02.sh
#!/bin/bash
for ((i=1,j=100;i<=10;i++,j--))
do
    echo "i=$i  j=$j"
done
```

运行结果：

```
[root@sunday-test shell-script]# ./c_for02.sh
i=1  j=100
i=2  j=99
i=3  j=98
i=4  j=97
i=5  j=96
i=6  j=95
```

```
i=7  j=94
i=8  j=93
i=9  j=92
i=10  j=91
```

for的无限循环

```
[root@sunday-test shell-script]# cat c_for3.sh
#!/bin/bash
for ((i=0;i<1;i+=0))
do
    echo "infinite loop"
done
```

while循环

和for循环一样，while循环也是一种运行前测试语句，语法更简单：

```
while  expression
do
        command
done
```

首先while将测试expression的返回值，如果返回值为真则执行循环体，返回值为假则不执行循环。循环完成后进入下一次循环之前将再次测试。

如果已知循环次数，可以用计数的方式控制循环，即设定一个计数器，在达到规定的循环次数后退出循环。

例如：

```
[root@sunday-test shell-script]# cat while01.sh
#!/bin/bash
CONTER=5
while [ $CONTER -gt 0 ]
do
   echo -n "$CONTER "
   let "CONTER-=1"
done
echo
```

可以做个猜字游戏：

```
[root@sunday-test shell-script]# cat while02.sh
#!/bin/bash
```

```
PRE_SET_NUM=8
echo "input a number between 1 and 10"
while read GUESS
do
    if [ $GUESS -eq $PRE_SET_NUM ];then
        echo "you get the right number "
        exit
    else
        echo "wrong,try again"
    fi
done
```

使用while按行读取文件

按行读取文件是while的一个非常经典的用法，常用于处理格式化数据。例如有个文件如下：

```
[root@sunday-test shell-script]# cat student_info.txt
John    30    Boy
Sue     28    Girl
Wang    25    Boy
Xu      23    Girl
[root@sunday-test shell-script]# cat while03.sh
#!/bin/bash
cat student_info.txt | while read LINE
do
        NAME=`echo $LINE | awk '{print $1}'`
        AGE=`echo $LINE | awk '{print $2}'`
        Sex=`echo $LINE | awk '{print $3}'`
        echo "my name is $NAME,I'm $AGE years old, I'm a $Sex"
done
```

运行结果：

```
[root@sunday-test shell-script]# ./while03.sh
my name is John,I'm 30 years old, I'm a Boy
my name is Sue,I'm 28 years old, I'm a Girl
my name is Wang,I'm 25 years old, I'm a Boy
my name is Xu,I'm 23 years old, I'm a Girl
```

while死循环：

方法1：

```
while   ((1))
do
    command
done
```

方法2：

```
while  true
do
   command
done
```

方法3:

```
while  :
do
    command
done
```

可以利用while的无限循环实时的监控系统进程，以保证系统中的关键应用一直处于运行状态

```
[root@sunday-test shell-script]# cat while04.sh
#!/bin/bash
while true
do
    HTTPD_STATUS=$(service httpd status | grep running)
    if [ -z "$HTTPD_STATUS" ];then
        echo "HTTPD is stopped,try to restart"
        service httpd  restart
    else
         echo "HTTPD is running, wait 5 sec until next check"
    fi
    sleep 5
done
```

until循环

until循环也是运行前测试，但是until采用的是测试假值的方式，当测试结果为假时才继续执行循环体，直到测试为真才会停止循环。语法：

```
until  expression
do
```

```
        command
    done
```

下面的实例使用until同时计算1到100的和以及1到100的奇数和。

```
[root@sunday-test shell-script]# cat until01.sh
#!/bin/bash
sum01=0
sum02=0
i=1
until [ $i -gt 100 ]
do
    let "sum01+=i"
    let "j=i%2"
    if [ $j -ne 0 ];then
        let  "sum02+=i"
    fi
    let "i+=1"
done
echo $sum01
echo $sum02
```

运行结果：

```
[root@sunday-test shell-script]# ./until01.sh
5050
2500
```

until的无限循环：

方式1：

```
until  ((0))
do
    command
done
```

方式二：

```
until  flase
do
   command
done
```

select循环

select是一种菜单扩展循环方式，其语法和带列表的for循环非常类似，基本结构如下：

```
select  MENU  in (list)
do
    command
done
```

当程序运行到select语句时，会自动将列表中的所有元素生成为可用1、2、3等数选择的列表，并等待用户输入。用户输入并回车后，select可判断输入并执行后续命令。如果用户在等待输入的光标后直接按回车键，select将不会退出而是再次生成列表等待输入。例如：

```
[root@sunday-test shell-script]# cat select01.sh
#!/bin/bash
echo "which cat do you prefer?"
select CAR in Benz Audi VolksWagen
do
    break
done
echo "You chose $CAR"
```

运行结果如下：

```
[root@sunday-test shell-script]# bash select01.sh
which cat do you prefer?
1) Benz
2) Audi
3) VolksWagen
#?                          //直接回车，从新生成列表
1) Benz
2) Audi
3) VolksWagen
#? 2                //选择2，程序会退出select并继续执行后面的语句。
You chose Audi
```

通过上面的实例，可以知道select有判断用户输入的功能，所以select经常和case语句合并使用。下面的例子使用select确认用户的输入并交由case处理，之后将根据不同输入执行不同代码段。代码中使用了“|”符，表示选择Saturday和Sunday的效果是一致的。

```
[root@sunday-test shell-script]# cat select02.sh
#!/bin/bash
```

```
select DAY in Mon  Tue Wed Thu Fri Sat Sun
do
        case $DAY in
        Mon)  echo "Today is Monday" ;;
        Tue)  echo "Today is Tuesday" ;;
        Wed)  echo "Today is Wednesday" ;;
        Thu)  echo "Today is Thursday" ;;
        Fri)  echo  "Today is Friday" ;;
        Sat|Sun)  echo "You can have a rest today" ;;
        *)  echo  "Unknown input, exit now"  &&  break ;;
        esac
done
```

运行结果：

```
[root@sunday-test shell-script]# ./select02.sh
1) Mon
2) Tue
3) Wed
4) Thu
5) Fri
6) Sat
7) Sun
#? 4
Today is Thursday
#? 5
Today is Friday
#? 10
Unknown input, exit now
```

嵌套循环

是指一个循环语句中的循环体是另外一个循环体。

使用for的嵌套打印九九乘法表：

```
[root@sunday-test shell-script]# cat nesting01.sh
#!/bin/bash
for ((i=1;i<=9;i++))
do
   for ((j=1;j<=9;j++))
```

```
        do
          let "multi=$i * $j"
          echo -n "$i*$j=$multi "
        done
        echo
    done
```

结果如下：

```
[root@sunday-test shell-script]# bash nesting01.sh
1*1=1 1*2=2 1*3=3 1*4=4 1*5=5 1*6=6 1*7=7 1*8=8 1*9=9
2*1=2 2*2=4 2*3=6 2*4=8 2*5=10 2*6=12 2*7=14 2*8=16 2*9=18
3*1=3 3*2=6 3*3=9 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27
4*1=4 4*2=8 4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72
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
```

也可以使用while来做：

```
[root@sunday-test shell-script]# cat nesting02.sh
#!/bin/bash
i=1
while [ $i -le 9 ]
do
    j=1
    while  [ $j -le 9 ]
    do
      let "multi=$i*$j"
      echo -n "$i*$j=$multi  "
      let "j+=1"
    done
    echo
    let "i+=1"
done
```

循环控制

break语句

break用于终止当前整个循环体。一般都是和if判断语句一起使用，当if条件满足时使用break终止循环

修改九九乘法表：

```
[root@sunday-test shell-script]# cat break01.sh
#!/bin/bash
for ((i=1;i<=9;i++))
do
   for ((j=1;j<=9;j++))
     do
       if [ $j -le $i ]; then
          let "multi=$i * $j"
          echo -n "$i*$j=$multi "
       else
            break
       fi
     done
     echo
done
```

运行结果如下：

```
[root@sunday-test shell-script]# bash break01.sh
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
```

continue语句

continue语句用于结束当前循环转而进入下一次循环，注意：这是和break不同的地方，continue并不会终止当前的整个循环体，它只是提前结束本次循环，而循环体还将继续执行；而break则会结束整个循环体。

例如：

```
[root@sunday-test shell-script]# cat continue01.sh
```

```
#!/bin/bash
for ((i=1;i<=100;i++))
do
  if ! (($i%2));then
      continue
  fi
  echo -n "$i "
done
echo
```

运行结果：

```
[root@sunday-test shell-script]# bash continue01.sh
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57
59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

如果将continue换成break

```
[root@sunday-test shell-script]# cat continue02.sh
#!/bin/bash
for ((i=1;i<=100;i++))
do
  if ! (($i%2));then
      break
  fi
  echo -n "$i "
done
echo
```

运行结果则会为：

```
[root@sunday-test shell-script]# bash continue02.sh
1
```

补充：调试脚本

1、bash 命令

语法

sh [-nvx] 脚本名

常用选项

-n：不执行脚本，仅检查语法。没有语法问题不显示任何内容，、有问题提示报错

-v：执行脚本时，先显示脚本内容，然后执行脚本。存在错误时，给出错误提示

-x：将执行的脚本内容输出到屏幕上

2、set命令

set -x：开启调节模式

set +x：关闭调节模式