shell脚本1部分

1、如何抒写一个脚本
2、运行脚本
3、shell中的符号使用
4、重定向
EOF的使用

shell脚本2部分

1、echo命令详解
2、read命令详解

变量

1、变量分类
2、定义变量
定义变量
读取变量内容
取消变量
设置全局变量
3、其他变量
readonly 只读变量
位置变量

数组

1、基本数组
数组语法
数组定义和读出
数组赋值
查看数组
2、关联数组

shell中的运算

1、数学比较运算
2、字符串比较运算
3、文件比较运算
4、逻辑运算
5、双小圆括号用法

if 判断语句

1、单步if语句
2、if-then-else双步语句

shell脚本1部分

shell脚本就是将多条命令聚集在一个文件中，自上而下的执行文件的命令。

1、如何抒写一个脚本

> shell 脚本的命名规范，能够快速定位到需要执行的脚本。
>
> shell脚本中填写"脚本信息"，方便后续修改，联系等。

```
# Author:
# Created Time:
# Version:
# Script Description:

//以上分别对应中文
# 作者信息
# 创建时间
# 版本
# 脚本描述
```

> shell 脚本中对难理解命令进行注释，方便后续使用一看便懂。

下面是一个完美脚本的内容：

```
#!/bin/bash
# 定义脚本执行环境


##脚本信息##

# Author: Hai ma ti
# Created Time: 2021/8/21 21:22
# Version: v1
# Script Description: create file

read -p "输入需要创建的文件名:" file

touch $file

if [ $? -ne 0 ];then
 echo "文件创建错误"

else
 echo "成功创建 $file 文件"

fi
```

2、运行脚本

第一种方式：脚本赋予执行权限运行，那上面的脚本举例：

```
[root@shell shells]# chmod +x test.sh
[root@shell shells]# ./test.sh
输入需要创建的文件名:qin
成功创建 qin 文件
```

第二种方式：使用解释器运行脚本

```
# 1、查看系统支持的解释器
[root@shell shells]# cat /etc/shells
/bin/sh
/bin/bash  #一般用这个就好了
/usr/bin/sh
/usr/bin/bash

# 2、使用bash解释器执行脚本
[root@shell shells]# bash test.sh
输入需要创建的文件名:qin
成功创建 qin 文件
```

3、shell中的符号使用

```
~                   //家目录   # cd ~ 代表进入当前用户家目录

!                   //执行历史命令  可配合history命令使用   "!!" 代表执行上一个命令

$                   //引用变量

+ - * / %           //加减乘除 配合 expr 使用

&                   //后台执行 配合nohap  和  jobs 一起使用

*                   // 是shell的通配符   匹配所有有
```

```
;                   //在shell中一行执行连个命令  # ifdown ens33;ifup ens33

|                   //管道符 上个命令的输出作为下个命令的输入

\                   //转义符

``                  //反引号  命令中执行命令

''                  //单引号 脚本中的字符串需要使用单引号和双引号引用起来，单引号不能解释变量

""                  //双引号 和单引号一样，双引号可以解释变量
```

演示以上难理解的几个符号使用：

```
# 1、!执行历史命令
[root@shell ~]# ls /etc/yum.repos.d/
CentOS-Base.repo  CentOS-Debuginfo.repo  CentOS-Media.repo    CentOS-Vault.repo
CentOS-CR.repo    CentOS-fasttrack.repo  CentOS-Sources.repo  CentOS-x86_64-kernel.repo
[root@shell ~]# pwd
/root

[root@shell ~]# !ls  #执行历史最后一个以ls 开头的命令
ls /etc/yum.repos.d/
CentOS-Base.repo  CentOS-Debuginfo.repo  CentOS-Media.repo    CentOS-Vault.repo
CentOS-CR.repo    CentOS-fasttrack.repo  CentOS-Sources.repo  CentOS-x86_64-kernel.repo

[root@shell ~]# pwd
/root
[root@shell ~]# !! #执行上次命令
pwd
/root
```

```
[root@shell ~]# history  #配合history命令使用
  171  pwd
  172  ls /etc/yum.repos.d/
  173  history

[root@shell ~]# !171
pwd
/root

# 2、加减乘除
[root@shell ~]# expr 1 + 1
2
[root@shell ~]# expr 100 - 1
99
[root@shell ~]# expr 100 /* 1  #需要注意乘号需要在Linux中是通配符需要使用'\'转义符进行转义
expr: syntax error
[root@shell ~]# expr 100 \* 1
100
[root@shell ~]# expr 100 / 50
2

# 3、&后台运行

[root@shell ~]# nohup yum install httpd -y &     #后台执行 前台命令使用ctrl+z可以实现相同效果
[1] 1866
[root@shell ~]# nohup: ignoring input and appending output to ‘nohup.out’

[root@shell ~]# jobs  #查看后台执行信息
[1]+  Running                 nohup yum install httpd -y &

# 4、单引号和双引号的区别

[root@shell ~]# echo '$USER' #单引号中不能引用变量
```

```
$USER
[root@shell ~]# echo "$USER" #双引号中可以引用变量
root
```

4、重定向

//重定向符号

> >　　输出重定向到一个文件或设备 覆盖原来的文件
> >!　　输出重定向到一个文件或设备 强制覆盖原来的文件
> >>　　输出重定向到一个文件或设备 追加原来的文件
> <　　输入重定向到一个程序

//标准错误重定向符号

> 2>　　将一个标准错误输出重定向到一个文件或设备 覆盖原来的文件 b-shell
> 2>>　　将一个标准错误输出重定向到一个文件或设备 追加到原来的文件
> 2>&1　将一个标准错误输出重定向到标准输出 注释:1 可能就是代表 标准输出
> >&　　将一个标准错误输出重定向到一个文件或设备 覆盖原来的文件 c-shell
> |&　　将一个标准错误 管道 输送 到另一个命令作为输入

//命令重导向示例

在 bash 命令执行的过程中，主要有三种输出入的状况，分别是：

> 标准输入：代码为 0 ； 或称为 stdin ； 使用的方式为 <
>
> 标准输出：代码为 1 ； 或称为 stdout； 使用的方式为 1>
>
> 错误输出：代码为 2 ； 或称为 stderr； 使用的方式为 2

演示：

```
# 1、将ls 返回正确值追加到 log.txt文件   错误信息追加到log.err文件里面
[root@shell ~]# ls  1>> log.txt 2>> log.err

# 2、将ls 返回正确值追加到 log.txt文件   错误信息重定向到/dev/null(黑洞) 即丢弃错误信息的意思
[root@shell ~]# ls -al 1>> log.txt 2> /dev/null

# 3、将显示的数据，不论正确或错误均输出到 list.txt 当中
[root@shell ~]# ls -al 1> list.txt 2>&1
```

EOF的使用

```
<<EOF          //开始
....内容
EOF           //结束
```

演示:

```
# 1、EOF重定向覆盖
[root@shell shells]# cat > test.txt <<EOF
> test file
> hai mai ti 666
> EOF
[root@shell shells]# cat test.txt
test file
hai mai ti 666

# 2、EFO重定向追加
[root@shell shells]# cat >> test.txt <<EOF
> new file;
> EOF
[root@shell shells]# cat test.txt
test file
hai mai ti 666
new file;
```

```
# 3、交互式命令中使用EOF（针对/dev/sdb进行分区并挂载）
[root@shell shells]# cat fdisk.sh
#!/bin/bash
###分区###
fdisk /dev/sdb   <<EOF
n
p
1

+3G
w
EOF
###格式化###
mkdir /data
mkfs.xfs /dev/sdb1  >> /dev/null
mount /dev/sdb1  /data >> /dev/null

df -hT | grep "sdb1"
```

shell脚本2部分

1、echo命令详解

```
# 命令选项：
-n: 不换行

-e: 若出现一下字符，则特别加以处理，而不会将他当成一般字符输出：
\a: 发出警告声
\d: 删除前一个字符
\c: 最后不加换行符
\t: tab 键
\n: 空格键
```

```
# 1、-n不换行参数
[root@shell ~]# echo "date:"; date +%F
date:
2021-08-25
[root@shell ~]# echo  -n "date:"; date +%F
date:2021-08-25

# 2、\t tab键参数
[root@shell ~]# echo -e "\t\tqinziteng"
        qinziteng
# 3、\n 空格键参数
[root@shell ~]# echo -e "\n\n"  //这个选输出了三个空格，
原因是echo默认自带一个空格 可以使用-n参数去掉。



[root@shell ~]# echo  -n -e "\n\n"

# 4、\d 删除前一个字符
[root@shell ~]# echo -e "AC\b"
AC
```

2、read命令详解

```
# 命令选项：
-p: 打印信息                    print[打印]
-t: 限定时间                    timeout[超时时间]
-s: 不显示输入的字符串
-n: 输入字符个数
```

```
# 1、 -s参数 不显示输入的字符
[root@shell ~]# cat read.sh
#!/bin/bash
echo -n "Passwd:"
read -s pwd
echo "password is:$pwd"
```

```
# 2、 -t限定时间
[root@shell ~]# cat read2.sh
#!/bin/bash
echo "请输入密码,3秒自动退出"
echo -n "Passwd:"
read -t 3 -s pwd

# 3、-n限定字符个数
#!/bin/bash
echo "请输入用户名,不得超过4个字符"
echo -n "Login:"
read -n 4
echo
echo "请输入密码,3秒自动退出"
echo -n "Passwd:"
read -t 3 -s pwd
echo

# 4、-p打印信息
//-p选项能实现和echo命令一样的效果  我们改一下上面的代码 用一行表示出来
//需要注意 read -p 之后在使用其他参数需要在后面添加即可!
[root@shell ~]# cat read4.sh
#!/bin/bash
echo  "请输入你的账号 不得超过4个字符"
read  -p  "Login:"  -n 4  login
echo
echo "请输入你的密码 不做操作3秒后自动退出"
read -p  "Passwd:"  -s -t 3 pwd
echo
echo "您的账号为$login 密码为: $pwd"
```

变量

1、变量分类

1. 本地变量：用户私有变量，只有本地用户可以使用，保存在家目录的 .bash_profile 和 .bashrc文件中。
2. 全局变量：所有用户都可以使用，保存在/etc/bashrc和/etc/profile文件中。
3. 用户自定义变量：比如脚本中的变量

2、定义变量

定义变量

```
# 变量格式：  变量名=值
```

```
[root@shell ~]# NAME=haimati
```

在shell中变量名与值之间不得有空格!

读取变量内容

```
# 读取变量内容符号：$
# 读取格式：$ 变量名
# 标准读取格式：$ {变量名}      //建议使用这款
```

```
[root@shell ~]# echo $NAME
haimati
```

取消变量

```
# unset 变量名
```

```
[root@shell ~]# unset NAME
[root@shell ~]# echo $NAME
```

设置全局变量

全局变量需要在 /etc/bashrc 或者 /etc/profile 文件中定义；

如果没有在配置文件中定义，重启系统后会消失！！！

```
# export 变量名=值
```

在添加变量时 前面添加export 即可，这样就是一个全局变量了，其他shell也能引用这个变量。

```
# root用户测试
[root@shell ~]# tail -1  /etc/profile
export USER1='zhangsan'
[root@shell ~]# source  /etc/profile
[root@shell ~]# echo $USER1
zhangsan

# zhangsan用户测试
[root@shell ~]# su - zhangsan
Last login: Tue Aug 31 20:52:16 CST 2021 on pts/1
[zhangsan@shell ~]$ echo $USER1
zhangsan
```

3、其他变量

readonly 只读变量

顾名思义，只读变量的值不可以被修改的。

```
# readonly 变量名：值
# declare -r 变量名：值
```

```
# 1、第一种声明只读变量
[zhangsan@shell ~]$ readonly test=t1
```

```
[zhangsan@shell ~]$ echo $test
t1

[zhangsan@shell ~]$ test=test1 # 修改变量会报错
-bash: test: readonly variable

#删除变量
[zhangsan@shell ~]$ unset t1


# 2、第二种声明只读变量
[zhangsan@shell ~]$ declare -r test2=t2
[zhangsan@shell ~]$ echo $test2
t2

# 这种方式只读变量不能使用unset命令删除，程序运行完成exit退出即可
```

```
# 3、打印只读变量
declare -p
readonly -p
```

位置变量

shell中还有一些预先定义的特殊只读变量，它们的值只有在脚本运行时才能确定。

```
$0                          // 代表脚本本身名字

$1----9                     //第一个位置参数------第九个文章参数

$#                          //脚本参数的个数总和

$@                          //脚本的所有参数

$*                          //脚本的所有参数
```

//脚本演示

```
[root@shell shells]# cat test.sh
#!/bin/bash
echo "这个脚本的名字是: $0"
echo "参数一共有: $#"
echo "参数的列表是: $@"
echo "参数的列表是: $*"
echo "第一个位置参数是: $1"
echo "第二个位置参数是: $2"
echo "第三个位置参数是: $3"


# 执行后效果:
[root@shell shells]# bash test.sh  1 2 3
这个脚本的名字是: test.sh
参数一共有: 3
参数的列表是: 1 2 3
参数的列表是: 1 2 3
第一个位置参数是: 1
第二个位置参数是: 2
第三个位置参数是: 3
```

//上面说到位置变量是1-9 那第10个位置变量该怎么表示呢？这种情况就要用到花括号了。

```
[root@shell shells]# cat test.sh
#!/bin/bash
echo "这个脚本的名字是: $0"
echo "参数一共有: $#"
echo "参数的列表是: $@"
echo "参数的列表是: $*"
echo "第一个位置参数是: $1"
echo "第二个位置参数是: $2"
echo "第三个位置参数是: $3"
echo "第十二个位置参数是: ${12}"
```

```
[root@shell shells]# bash test.sh {1..13}
这个脚本的名字是: test.sh
参数一共有: 13
参数的列表是: 1 2 3 4 5 6 7 8 9 10 11 12 13
参数的列表是: 1 2 3 4 5 6 7 8 9 10 11 12 13
第一个位置参数是: 1
第二个位置参数是: 2
第三个位置参数是: 3
第十二个位置参数是: 12
```

数组

变量和数组的区别：一个变量只能定义一个值、一个数组可以定义多个值

1、基本数组

数组可以让用户一次赋予多个值，需要读取数据时只需要使用索引调用即可。

数组语法

```
#  数组名称=(元素1 元素2 元素3 ....)
```

数组定义和读出

```
#  ${数组名称[索引]}
#  索引默认是元素在数组中的编号，默认第一个从0开始
```

```
[root@shell ~]# ARRAY1=('a' 'b' 'c' 'd')
[root@shell ~]# echo ${ARRAY1[2]}
c          //第一个从0开始，打印出第2个，那就是c咯
```

数组赋值

1）一次赋多值

```
[root@shell ~]# ARRY2=('0' '1' '2' '3' '4' '5')
[root@shell ~]# echo ${ARRY2[0]}
```

2)一次性赋单个值

```
[root@shell ~]# ARRY2[7]='666'
[root@shell ~]# ARRY2[8]='777'
[root@shell ~]# echo ${ARRY2[7]}
666
[root@shell ~]# echo ${ARRY2[8]}
777
```

查看数组

```
# 1、查看系统中定义的数组
declare -a

# 2、查看数组第一个索引
[root@shell ~]# echo ${ARRY2[0]}

# 3、查看数组所有索引 @
[root@shell ~]# echo ${ARRY2[@]}
0 1 2 3 4 5 666 777

# 4、查看所有数组的索引值 !
[root@shell ~]# echo ${!ARRY2[@]}
0 1 2 3 4 5 7 8

# 5、从数组下标3开始
[root@shell ~]# echo ${ARRY2[@]:3}
3 4 5 666 777

# 6、从数组下标3开始,访问2个元素
```

```
[root@shell ~]# echo ${ARRY2[@]:3:2}
3 4
```

2、关联数组

基本数组和关联数组的区别是：基本数组的索引是从0 1 2 3 开始的，而关联数组可以修改索引名字。关联数组需要使用declare -A声明。

```
# declare -A 数组名称
```

1）一次赋单个值

```
# 数组名[索引名]=变量值
```

```
[root@shell ~]# declare -A T1    #声明 T1这个数组
[root@shell ~]# T1[name]="zhangsan"
[root@shell ~]# T1[age]=19
[root@shell ~]# echo ${T1[@]}
zhangsan 19
```

2）一次赋多个值

```
# 数组名([索引名]=变量值  [索引名]=变量值...)
```

```
[root@shell ~]# T1=([info]='ok' [home]='beijing')
[root@shell ~]# echo ${T1[@]}
beijing ok
```

shell中的运算

1、数学比较运算

```
# 运算符解释：

-eq                 等于
-ne                 不等于
-gt                 大于
-lt                 小于
-ge                 大于等于
-le                 小于等于
```

可以使用test命令测试 返回值0为真 非0为假

```
[root@shell ~]# test 1 -eq 1;echo $?
0
[root@shell ~]# test 1 -lt 1;echo $?
1
[root@shell ~]# test 1 -le 1;echo $?
0
```

2、字符串比较运算

```
# 运算符解释：注意字符串一定别忘记使用引号引起来！

==                      等于
!=                      不等于
-n                      是否不为空
-z                      是否为空
```

使用test命令测试

```
# 1、检查当前用户是否是root
[root@shell ~]# echo $USER
root
[root@shell ~]# test  $USER == 'root' ;echo $?
0

# 2、是否不为空
[root@shell ~]# test -n 'root' ;echo $?
0

# 3、是否为空
[root@shell ~]# test -z 'root' ;echo $?
1
```

3、文件比较运算

```
-e                  检查文件是否存在（目录、文件都可以，只有它存在）
-d                  检查文件是否存在，且为目录
-f                  检查文件是否存在，且为文件
-r                  检查文件是否存在，并有r可读权限
-w                  检查文件是否存在，并有w可写权限
-x                  检查文件是否存在，并有w可写权限
-O                  检查文件是否存在并，且被当前用户所拥有
-G                  检查文件是否存在，并且被当前用户所组
file1 -nt file2     检查file1是否比file2新
file1 -ot file2     检查file1是否比file2旧
```

使用test命令测试

```
# 1、检查文件是否存在
[root@shell ~]# test -e /etc/passwd;echo $?
0

# 2、检查目录是否存在
[root@shell ~]# test -d /etc/passwd;echo $?
```

```
1

# 3、检查文件是否存在
[root@shell ~]# test -f /etc/passwd;echo $?
0

# 4、检查/etc/passwd是否比/etc/shadow新
[root@shell ~]# test /etc/passwd -nt /etc/shadow ;
echo $?
1

# 5、检查/etc/passwd是否比/etc/shadow旧
[root@shell ~]# test /etc/passwd -ot /etc/shadow ;
echo $?
0
```

4、逻辑运算

```
&&                  逻辑与运算
||                  逻辑或运算
!                   逻辑非运算

逻辑运算注意事项：
逻辑与、逻辑或  运算都需要两个条件，逻辑非运算只能一个条件

口诀：              逻辑与运算              真真为真   真假
为假    假假为假
                    逻辑或运算              真真为真   真假
为真    假假为假
                    逻辑非运算              非假为真   非真
为假
```

5、双小圆括号用法

(())可以在里面进行数学自增减运算，(())里面可自动引用变量不需要使用$应用。

```
# 1、不使用$符号引用变量
((a=1+6))                    将1+6的值赋予给a变量
((b=a-1))                    a变量的值-1 赋值给b变量
((c=a+b))                    c变量的值等于ab变量相加的值

# 2、使用$符号引用变量
a=((1+6))
b=((a-1))
c=((a+b))

# 3、多个表达式同时进行运算
((a=3+5,b=a+10))

# 4、进行逻辑运算,在if语句中经常被使用
((a>7 && b==c))

# 5、需要立即输出表达式
echo $((a+10))
```

if 判断语句

1、单步if语句

适用范围：只需要一步判断

```
# 语句格式：

if [ condition ]
  then
    commands
fi
```

该语句翻译成汉语大致意思如下：

```
假如  [ 条件为真 ]
  那么
    执行什么操作
结束
```

演示：

```
[root@shell shell_if]# cat if1.sh
#!/bin/bash

####  假如/tmp/abc目录夹不存在 则创建此目录 ####
 eraffif [ ! -d /tmp/abc ];then
  mkdir /tmp/abc
fi eraff
```

2、if-then-else双步语句

```
# 语句格式：

if [ condition ]
  then
    commands
else
    commands
fi
```

翻译中文大致意思为：

```
如果 [ 条件为真 ]
  那么
    执行什么操作
否则
    执行什么操作
结束
```

演示：

```
[root@shell shell_if]# cat if2.sh
#!/bin/bash

#### 登入用户是root输出 管理员好!
#### 登入用户是普通账号输出 guest你好!

if [ $USER == 'root' ];then
  echo "管理员好!"
else
  echo "guest你好!"
fi
```

3、if-then-elif-else 多步语句

```
# 语句个事:

if [ condition1 ]
  then
    commands
elif [ condition2 ]
    commands
......

else
    commands
fi
```

翻译大致为:

```
假如 [ 条件1为真 ]
  那么
    执行什么操作

假如 [ 条件2为真 ]
  那么
    执行什么操作
.....

否则
  执行什么操作
结束
```

演示：

```
[root@shell shell_if]# cat if3.sh
#!/bin/bash

#### 判断两个整数的关系 ####

if [ $1 -eq $2 ];then
    echo "$1 = $2"

elif [ $1 -ge $2 ];then
    echo "$1 > $2"

else
    echo "$1 < $2"

fi
```

4、嵌套if

演示：

```
[root@shell shell_if]# cat if4.sh
#!/bin/bash

#### 判断两个整数的关系 ####

if [ $1 -eq $2 ];then
   echo "$1=$2"

else
  if [ $1 -gt $2 ];then
      echo "$1>$2"
  else
      echo "$1<$2"
  fi
fi
```

5、if 小技巧

1）条件符号使用双圆括号，可以在条件中植入数学表达式

演示：

```
[root@shell shell_if]# cat if5.sh
#!/bin/bash
if (( 100%3+1>10 ));then
    echo "大于10"
else
    echo "小于10"
fi
```

2）条件符号使用双中括号，可以在条件中使用通配符

演示：

```
[root@shell shell_if]# cat if6.sh
#!/bin/bash

for i in zi cc zz tt qq
  do
    if [[ $i == z* ]];then
      echo $i
    fi
done
```

for循环语句

很多人将for循环叫做 “条件循环”因为for是按条件循环的。

1、for语法

```
for  var  in value1 value2 ....
    do
        commands
done
```

演示： 倒计时

```
[root@shell shell_for]# cat for1.sh
#!/bin/bash
for i in `seq 9 -1 1`
    do
        echo $i
    sleep 1
done
```

2、 类C语言for语法

C语言格式for循环格式

返回值为真则循环继续，返回值为假则退出循环>

```
for  ((变量;条件;自增减运算))
    do
        commands
done
```

演示： 倒计时

```
[root@shell shell_for]# cat for2.sh
#!/bin/bash
for (( i=10;i>0;i-- ))
    do
        echo $i
    sleep 1
done


#!/bin/bash
for (( n=10;n>0;n-- ))
        do
                echo "$n"
done
```

//双变量使用'，'逗号隔开即可 如下：

```
[root@shell shell_for]# cat for3.sh
#!/bin/bash
for (( n=10,m=0;n>0,m<10;n--,m++ ))
    do
        echo -e "\t$n\t\t$m"
done
```

3、无限循环

```
[root@shell shell_for]# cat for4.sh
#!/bin/bash
for ((;;))   #((;;)) 表示无线循环
    do
    echo "无限循环"
done
```

循环控制语句

1、sleep N 脚本执行休眠时间

```
# 语法 sleep 休眠时间
```

演示1：倒计时

```
[root@shell shell_for]# cat sleep.sh
#!/bin/bash
echo  "10秒倒计时:"
echo
for (( i=10;i>0;i-- ))
    do
        echo $i
    sleep 1
done
```

演示2：监控主机存活

```
[root@shell ~]# cat ping.sh
#!/bin/bash

for ((;;))
    do
        ping -c2 $1 &>/dev/null
        if [ $? -eq 0 ];then
            echo -e "`date +%F-%H-%M-%S` is
\033[032m up \033[0m"
```

```
        else
            echo -e "`date +%F-%H-%M-%S`  is
\033[031m down \033[0m"
        fi

        #定义脚本节奏 5秒一次   生产环境不要这么搞!
         sleep 5
done
```

2、continue 跳过某次循环

看下面一段代码，输出1-9 但是使用continue 跳过输出5

```
[root@shell ~]# cat continue.sh
#!/bin/bash

for i in {1..10}
    do
        if [ $i -eq 5 ];then
            continue
            # if判断当循环到5时，到此结束了，可以开始下次循
环了。
        fi
        echo $i
        sleep 0.3
done
```

3、break 跳出循环

演示：输入Q则跳出循环，否则无限循环

```
[root@shell ~]# cat break.sh
#!/bin/bash

for ((;;))
    do
        read -p "输入一个字母:" ch
        if [ $ch == 'Q' ];then
                #终止本次循环
                break
        else
             echo "您输入的字母是:$ch"
        fi
done
```

//如果循环多层嵌套，循环从里往外排序 0-N，如果想跳出某层循环使用 break N

```
# break N
```

如下:

```
[root@shell ~]# cat break2.sh
#!/bin/bash

for (( i=1;i<100;i++ ))
    do
      echo "外循环 $i"
      for ((;;))
          do
              echo "内循环 $i"
          # 跳出外循环
              break 2
          done
done
```

while循环语句

while循环和for循环语法上都几乎一致，都是当条件true时进行循环，当条件为fake时终止循环。

很多小伙伴不知道什么时候使用for 什么时候使用while循环，根据场景而定吧，当明确知道要循环多少次的时候使用 for 循环，当不明确要循环多少次时，使用while循环。

1、while循环语法

```
while [ condition ] #注意，条件为真while循环，为假终止循环。
do

        commands
done
```

2、基本语法练习

演示：语法练习1

```
# 当输入值大于1时条件为真 则一直循环下去，当输入值小于0 条件为假 终止循环。
[root@shell while]# cat while1.sh
#!/bin/bash
read -p "NUM:" num1

while [ $num1 -gt 0 ]
  do
echo "大于"
sleep 1
done
```

演示：语法练习2

```
# 当输入用户不是root 条件为真一直循环下去直到 输入用户是root 为止。
[root@shell while]# cat while2.sh
#!/bin/bash
read -p "login:" account
while [ $account != 'root' ]
  do
read -p "login:" account
done
```

演示：语法练习3

```
# 丈母娘选婿 脚本中涉及到 || 逻辑或运算  真真为真 真假为真 假假为假
[root@shell while]# cat while3.sh
#!/bin/bash

#1、姑娘带回来第一个男朋友

read -p "money:" money
read -p "car:" car
read -p "house" house

#2、第一个男朋友不满足，进行循环，开始选择模式

while [ $money -lt 100000 ] || [ $car -lt 1 ] || [ $house -lt 1 ]
#逻辑或运算 真真为真 真假为真 假假为假 如果这些条件有一个为真则进入循环。
  do
    echo "不同意"

    read -p "money:" money
    read -p "car:" car
    read -p "house" house
done
```

```
echo "同意啦"
```

演示：语法练习4

```
# 输入Q退出
[root@shell while]# cat while4.sh
#!/bin/bash
read -p "请输入一个char:" char

while [ $char != 'Q' ]
do
read -p "请输入一个char:" char
done
```

3、嵌套 循环控制练习

演练：嵌套if语句

```
# 打印1-9 当数值为5时停止循环
[root@shell while]# cat whilet1.sh
#!/bin/bash
i=1
while [ $i -lt 10 ]
do
    echo $i
    if [ $i -eq 5 ];then
        break
    fi
    i=$((i+1))
done
```

演练：嵌套if语句

```
# 打印1-9 当数值为5时 跳过当前循环
[root@shell while]# cat whilet2.sh
#!/bin/bash
```

```
i=0
while [ $i -lt 10 ]
 do
    i=$((i+1))

    if [ $i -eq 5 ];then

    continue

    fi
    echo $i
done
```

演练：嵌套for 打印99乘法表

```
[root@shell while]# cat while_for_99.sh
#!/bin/bash
n=1
while [ $n -lt 10 ];do
    for ((m=1;m<=$n;m++));do
    echo -n -e "$m * $n =$((n*m))\t"
done
echo
n=$((n+1))
done
```

演练：嵌套while 打印99乘法表

```
[root@shell while]# cat while_while_99.sh
#!/bin/bash
n=1
while [ $n -lt 10 ];do
m=1
    while [ $m -le $n ];do
        echo -n -e  "$m * $n =$((m*n))\t"
        m=$((m+1))
    done
    echo
    n=$((n+1))
done
```

until语句

until和while正好相仿，until是条件为假开始循环，条件为真停止循环

```
#语法:
until [ condition ] # 条件为假循环，  为真停止循环
do

        commands
done
```

演示：打印10-20数字

```
[root@shell until]# cat until1.sh
#!/bin/bash
# 打印10-20数字
int_num=10
until [ $int_num -gt 20 ];do
    echo "$int_num"
    int_num=$((int_num+1))
   done
```

case多条件分支语句

case和if一样也是判断语句，和if相比case更适合做多条件判断。

case 判断语法

```
case  变量  in

条件1)
    执行条件1代码块
;;

条件2)
    执行条件2代码块
;;
.........
*)
    条件不是1和2 执行的代码块
;;
esac
```

注意：每个代码块执行完成后;;表示代码块执行结束 ，case结尾要以倒过来写esac，来结束。

演示：丈母娘家串门

```
[root@shell until]# cat case.sh
#!/bin/bash

read -p "丈母娘家串门，开门的是:" N

case $N in

伯母|丈母娘)
    echo "伯母好"
    echo "伯母辛苦了"
```

```
;;

伯父|老丈人)
        echo "伯父好"
        echo "伯父真帅"
;;

奶奶|老奶奶)
        echo "奶奶好"
        echo "奶奶  奶奶吉祥"
;;

*)
    echo "脚本$0 可选值为  伯母|伯父|奶奶"
esac
```

select循环

语法:

```
#!/bin/bash
select 变量名 in [ 菜单列表]
do
        指令1....
done
```

```
echo "What is your favourite OS?"
select name in "Linux" "Windows" "Mac OS" "UNIX"
"Android"
do
    case $name in
        "Linux")
            echo "Linux是一个类UNIX操作系统，它开源免费，
运行在各种服务器设备和嵌入式设备。"
            break
            ;;
```

```
        "Windows")
            echo "Windows是微软开发的个人电脑操作系统，它是闭源收费的。"
            break
            ;;
        "Mac OS")
            echo "Mac OS是苹果公司基于UNIX开发的一款图形界面操作系统，只能运行与苹果提供的硬件之上。"
            break
            ;;
        "UNIX")
            echo "UNIX是操作系统的开山鼻祖，现在已经逐渐退出历史舞台，只应用在特殊场合。"
            break
            ;;
        "Android")
            echo "Android是由Google开发的手机操作系统，目前已经占据了70%的市场份额。"
            break
            ;;
        *)
            echo "输入错误，请重新输入"
    esac
done
```

函数

函数听起来很高大上，其实so easy，就是将脚本中的代码模块化，起个函数名字，之后可以多个调用这个函数。

由于函数是将代码模块化，出现问题了也容易排查。

函数语法

```
# 1、语法1
function 函数名字 {
```

```
代码块

}

# 2、语法2

函数名字（）{

代码块

}

# 调用函数：  使用函数名字调用，可反复调用
# 注意：  代码块中涉及到位置变量  调用函数时要在函数名称后面加上 $1 $2 等。
```