

Shell脚本编程

一、Shell概述

Shell既是一种应用程序,又是一种程序设计语言。

1.1 作为应用程序

交互式地解释、执行用户输入的命令, 将用户的操作翻译成机器可以识别的语言, 完成相应功能。

称之为shell 命令解析器

shell 是用户和 Linux 内核之间的接口程序

用户在提示符下输入的命令都由 **shell** 先解释然后传给 **Linux** 核心, 它调用了系统核心的大部分功能来执行程序、并以并行的方式协调各个程序的运行。

shell 命令解释器

Linux 系统中提供了好几种不同的 shell 命令解释器, 如 sh、ash、bash 等。

默认使用 bash 作为默认的解释器 【终端的默认解释器】

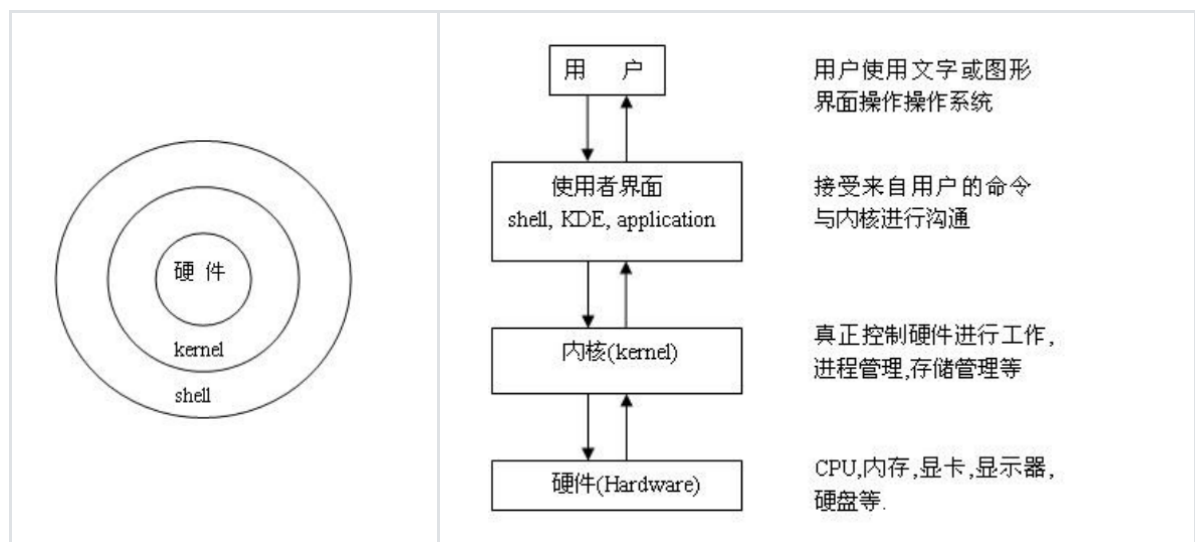
```
/bin/bash
```

```
$ bash
disen@qfxa:~$ ls
code  Desktop  Downloads  gcc.txt  Music  my2  Public  Templates
code2 Documents examples.desktop includes my  Pictures smb.conf Videos
disen@qfxa:~$
```

```
/bin/sh
```

```
disen@qfxa:~$ sh
$ ls
code  Desktop  Downloads  gcc.txt  Music  my2  Public  Templates
code2 Documents examples.desktop includes my  Pictures smb.conf Videos
$
```

shell 是用户跟内核通信方式的之一



1.2 作为程序设计语言

它定义了各种变量和参数，并提供了许多在高级语言中才具有的控制结构，包括循环和分支，完成类似于 windows 下批处理操作，简化我们对系统的管理与应用程序的部署称之为 shell 脚本。

c/c++等语言，属于编译性语言（编写完成后需要使用编译器完成编译、汇编、链接等过程变为二进制代码方可执行），**shell 脚本是一种脚本语言**，我们只需使用任意文本编辑器，按照语法编写相应程序，增加可执行权限，即可在安装 shell 命令解释器的环境下执行。

应用场景：

帮助开发人员或系统管理员将复杂而又反复的操作放在一个文件中，通过简单的一步执行操作完成相应任务，从而解放他们的负担。

示例1：《linux 常用命令_练习.txt》

- (1)在/home/edu目录下建立一个名为cmd_test的目录。
- (2)进入cmd_test目录
- (3)将文件/lib/目录下所有以包含.so(*.so*)的文件复制到cmd_test目录下
- (4)将cmd_test目录下后缀为.so文件打包成so.tar.bz2
- (5)将cmd_test目录下后缀为.so的文件打包并压缩为so.tar.gz
- (6)在cmd_test目录下创建c_test文件夹
- (7)将so.tar.gz解压到c_test目录下
- (8)将so.tar.gz删除
- (9)修改so.tar.bz2的权限，要求仅文件的所有者拥有读、写、执行权限，其余人和同组人没有任何权限
- (10)使用 > 将 dev目录下的所有文件名保存在cmd_test下的dev.txt中
- (11)使用grep命令找出其中console出现的位置

shell脚本：

```
#!/bin/bash
mkdir /home/disen/cmd_test
echo "1-----mkdir-----end"
cd /home/disen/cmd_test
echo "2-----cd-----end"
cp /lib/*.so* ./
echo "3-----cp-----end"
tar jcvf so.tar.bz2 *.so*
echo "4-----tar jcvf -----end"
tar zcvf so.tar.gz *.so*
echo "5-----tar zcvf -----end"
mkdir c_test
echo "6----- mkdir c_test-----end"
tar zxvf so.tar.gz -C ./c_test
echo "7----- tar zxvf -----end"
rm so.tar.gz
echo "8---- rm-----end"
chmod 700 so.tar.bz2
echo "9----- chmod 700-----end"
ls /dev/ > dev.txt
echo "10-----ls /dev -----end"
```

```
grep console dev.txt -n
echo "11-----grep -----end"
```

设置test1.sh的脚本文件执行权限：

```
chmod +x test1.sh
```

执行脚本：

```
./test1.sh
```

示例2：

判断用户家目录下（~）下面有没有一个叫test的文件夹
如果没有，提示按y创建并进入此文件夹，按n退出
如果有，直接进入，提示请输入一个字符串，并按此字符串创建一个文件，如果此文件已存在，提示重新输入，重复三次自动退出，不存在创建完毕，退出

shell脚本如下：【了解】

```
#!/bin/bash
flag=1
if [ -e ~/test ]; then
    echo "the test dir is existed"
else
    echo "the test dir is not exist"
    while [ $flag ]
    do
        echo "press y to creat this dir and enter it,press n exit"
        read yn
        case $yn in
            y)
                echo "mkdir ~/test"
                mkdir ~/test
                flag=0
                ;;
            n)
                echo "end operate.."
                exit -1
                ;;
            *)
                echo "your input is error exit"
                echo "please input again"
                flag=1
                ;;
        esac
    done
fi

cd ~/test

for (( i=1; i<4 ; i++ ))
do
    echo "please input a string, we will creat it as a dir"
```

```

read name
if [ -e $name ];then
    echo "$i please input other name"
else
    echo "the $name is created"
    mkdir $name
    break
fi
done
echo "your times is over"
if [ $i -eq 3 ]; then
    echo "your times is over"
    exit 0
fi

```

```

disen@qfxa:~$ ./test2.sh
the test dir is existed
please input a string, we will creat it as a dir
a.txt
the a.txt is created
your times is over
disen@qfxa:~$ ./test2.sh
the test dir is existed
please input a string, we will creat it as a dir
a.txt
1 please input other name
please input a string, we will creat it as a dir
a.txt
2 please input other name
please input a string, we will creat it as a dir
a.txt
3 please input other name
your times is over

```

1.3 shell 脚本分类

1.3.1 系统进行调用

这类脚本无需用户调用，系统会在合适的时候调用，如：/etc/profile、~/.bashrc 等

/etc/profile:

此文件为系统的每个用户设置环境信息,当用户第一次登录时该文件被执行
系统的公共环境变量在这里设置
开始自启动的程序，一般也在这里设置

~/.bashrc

用户自己的家目录中的.bashrc
登录时会自动调用，打开任意终端时也会自动调用
这个文件一般设置与个人用户有关的环境变量，如交叉编译器的路径等等

1.3.2 手动进行调用

用户编写，需要手动调用的

无论是系统调用的还是需要我们自己调用的，其语法规则都一样。

二、shell 语法

2.1 shell 脚本的定义与执行

定义以开头：`#!/bin/bash`

```
#!/用来声明脚本由什么 shell 解释，否则使用默认 shell
```

单个"#"号代表**注释**当前行

三种执行方式：

- 1) `chmod +x test.sh`
`./test.sh` 增加可执行权限后执行
- 2) `bash test.sh` 直接指定使用 `bash` 解释 `test.sh`
- 3) `./test.sh` 使用当前 `shell` 读取解释 `test.sh`

三种执行方式不同点：

- 1) `./`和 `bash` 执行过程基本一致，后者明确指定 `bash` 解释器去执行脚本，脚本中`#!/`指定的解释器不起作用前者首先检测`#!/`，使用`#!/`指定的 `shell`，如果没有使用默认的 `shell`。
- 2) 用`./`和 `bash` 去执行会在后台启动一个新的 `shell` 去执行脚本
- 3) 用`./`去执行脚本不会启动新的 `shell`,直接由当前的 `shell` 去解释执行脚本。

如1: `t3.sh`

```
#!/bin/bash

# 注释的部分
echo "--t1.sh--"
ls ~/cmd_test -l
```

执行命令：

```
chmod +x t3.sh
./t3.sh
```

如2: `t4.sh`

```
# no bash interpret

echo "--create directory abc-"
mkdir abc
echo "--ok--"
```

执行命令：

```
bash t4.sh
sh t4.sh
. t4.sh
```

2.2 变量

2.2.1 自定义变量【重点】

定义语法:

```
变量名=变量值
```

引用语法:

```
$变量名
```

清除变量语法:

```
unset varname
```

如1: t5.sh

```
#!/bin/bash

age=20
name=disen
echo "$name $age"
```

执行: `. t5.sh`

```
disen@qfxa:~/shells$ . t5.sh
disen 20
```

变量的其它用法:

```
read string 从键盘输入一个字符串付给变量 string
readonly var=100 定义一个只读变量,只能在定义时初始化,以后不能改变,不能被清除
export var=300 使用export 说明的变量会被导出为环境变量,其它 shell 均可使用
【注意】必须使用 source 2_var.sh 才可以生效
```

如2: t6.sh

```
#!/bin/bash

echo "请输入您的年龄和姓名"
read age
read name

echo "姓名: $name, 年龄: $age"
```

```
disen@qfxa:~/shells$ . t6.sh
请输入您的年龄和姓名
20
jack
姓名: jack, 年龄: 20
disen@qfxa:~/shells$
```

如3: t7.sh

```
#!/bin/bash

readonly PI=3
echo $PI

x=1
x=2
echo $x
# 清除x变量之后，变量还可以再次读取，但变量的内容是空
unset x
echo $x

r=2
# shell进行算术运算时，bash/sh 不运行小数点计算
S=$(( PI*r ))
echo $S
```

```
disen@qfxa:~/shells$ bash t7.sh
3
2

12
```

注意事项:

- 1) 变量名只能包含英文字母下划线，不能以数字开头
- 2) 等号两边不能直接接空格符，若变量中本身就包含了空格，则整个字符串都要用双引号、或单引号括起来；双引号内的特殊字符可以保有变量特性，但是单引号内的特殊字符则仅为一般字符

如4: t8.sh

```
#!/bin/bash

msg="hi disen"
echo $msg
echo "$msg very good"
echo '$msg very good'
bye='bye bye'
echo "$bye"
```

```
disen@qfxa:~/shells$ . t8.sh
hi disen
hi disen very good
$msg very good
bye bye
```

2.2.2 环境变量

shell 在开始执行时就已经定义了一些和系统的工作环境有关的变量，我们在 shell 中可以直接使用\$name 引用。

在~/.bashrc 或/etc/profile 文件中使用 export 设置，传统上，所有**环境变量均为大写**。

env 命令可以查看所有环境变量。

常见环境变量：

```
HOME: 用于保存注册目录的完全路径名。
PATH: 用于保存用冒号分隔的目录路径名，shell 将按 PATH 变量中给出的顺序搜索这些目录，找到的第一个与命令名称一致的可执行文件将被执行。
      PATH=$HOME/bin:/bin:/usr/bin;
      export PATH
HOSTNAME: 主机名
SHELL: 默认的 shell 命令解析器
LOGNAME: 此变量保存登录名
PWD: 当前工作目录的绝对路径名
...
```

如1: t9.sh

```
#!/bin/bash

echo "your name is $USER"
echo "your home path is $HOME"
```

如2: t10.sh 新增环境变量

新增环境变量可以在用户环境变量 (~/.bashrc) 或系统环境变量(/etc/profile)

```
source ~/.bashrc

source /etc/profile
```

```
fi

export ShellsPath=/home/disen/shells
```

```
#!/bin/bash

echo "your name is $USER"
echo "your home path is $HOME"
echo "your shells path is $ShellsPath"
```



```
disen@qfxa:~/shells$ . t10.sh
your name is disen
your home path is /home/disen
your shells path is /home/disen/shells
```

2.2.3 预设变量

`$#`: 传给 shell 脚本参数的数量
`$*`: 传给 shell 脚本参数的内容
`$1`、`$2`、`$3`、...、`$9`: 运行脚本时传递给其的参数，用空格隔开
`$?`: 命令执行后返回的状态
 " `$?` " 用于检查上一个命令执行是否正确(在 Linux 中，命令退出状态为 0 表示该命令正确执行，任何非 0 值表示命令出错)。
`$0`: 当前执行的进程名
`$$`: 当前进程的进程号
 " `$$` " 变量最常见的用途是用作临时文件的名称以保证临时文件不会重复

如1: t11.sh

```
#!/bin/bash
echo "$#"
echo "$*"
echo "$1,$2,$3"
echo $?
rm a.txt
echo $?
```

执行:

```
bash t11.sh 1 2 3
```

```
disen@qfxa:~/shells$ bash t11.sh 1 2 3
3
1 2 3
1,2,3
0
rm: cannot remove 'a.txt': No such file or directory
1
disen@qfxa:~/shells$
```

如2: t12.sh

```
echo "当前的进程名: $0"
echo "当前的进程号: $$"
```

```
disen@qfxa:~/shells$ bash t12.sh
当前的进程名: t12.sh
当前的进程号: 10552
```

2.2.4 脚本变量的特殊用法

"" (双引号)：包含的变量会被解释
'' (单引号)：包含的变量会当做字符串解释
`` (反引号)：反引号中的内容作为系统命令，并执行其内容，可以替换输出为一个变量
 \$ echo "today is `date`"
\
转义字符：同 c 语言 \n \t \r \a 等 echo 命令需加 -e 转义
(命令序列)：由子 shell 来完成,不影响当前 shell 中的变量，命令之间使用分号分隔。
{ 命令序列 }：在当前 shell 中执行，会影响当前变量，每个命令都要加分号";"

【扩展】date命令：显示年月日，时分秒

```
date +"%Y-%m-%d %H:%M:%S"
```

如: t13.sh

```
echo "current datetime is `date +"%Y-%m-%d %H:%M:%S"`"
```

```
echo "current datetime is `date +%Y-%m-%d\ %H:%M:%S`"
```

```
disen@qfxa:~/shells$ bash t13.sh
current datetime is 2023-08-14 11:48:21
```

如: t14.sh

```
name=Disen
echo "0 $name"

(name=Lucy; echo "1 $name")
echo 0:$name
{name=Jack; echo "2 $name";}
echo 0:$name
```

```
disen@qfxa:~/shells$ bash t14.sh
0 Disen
1 Lucy
0:Disen
2 Jack
0:Jack
```

2.3 条件测试语句

在写 shell 脚本时，经常遇到的问题就是判断字符串是否相等，可能还要检查文件状态或进行数字测试，只有这些测试完成才能做下一步动作。

test 命令：用于测试字符串、文件状态和数字

test 命令有两种格式：

- 1) test condition
- 2) [condition]

【注意】使用方括号时，条件两边加上空格

2.3.1 文件

文件测试：测试文件状态的条件表达式

-e 是否存在	-d 是目录	-f 是文件
-r 可读	-w 可写	-x 可执行
-L 符号连接	-c 是否字符设备	-b 是否块设备
-s 文件非空		

如1：t15.sh, 从命令行中读取一个文件路径，测试路径是否存在

```
#!/bin/bash
test -e $1
echo $?
```

如2：t15.sh, 从命令行读取一个参数为文件路径，测试它是具有可执行权限

```
#!/bin/bash
test -x $1
echo $?
```

2.3.2 字符串

语法：

```
test str_operator "str"
test "str1" str_operator "str2"
[ str_operator "str" ]
[ "str1" str_operator "str2"]
```

str_operator:

= 两个字符串相等	!= 两个字符串不相等
-z 空串	-n 非空串

如：从键盘输入两次口令，测试它们是否相等

```
#!/bin/bash

read pwd1
read pwd2
echo "$pwd1 is zero string?"
[ -z $pwd1 ]
echo $?
test $pwd1 = $pwd2
echo $?
```

```
disen@qfxa:~/shells$ bash t16.sh
123
123
123 is zero string?
1
0
```

2.3.3 数字

语法:

```
test num1 num_operator num2
[ num1 num_operator num2 ]
```

num_operator:

-eq 数值相等	-ne 数值不相等
-gt 数 1 大于数 2	-ge 数 1 大于等于数 2
-lt 数 1 小于数 2	-le 数 1 小于等于数 2

如: 从命令读取两个数, 测试它们是否相等

```
#!/bin/bash

echo "$1 equal $2? "
test $1 -eq $2
echo $?

echo "$1 not equal $2? "
test $1 -ne $2
echo $?

echo "$1 > $2? "
test $1 -gt $2
echo $?

echo "$1 >= $2? "
test $1 -ge $2
echo $?

echo "$1 < $2? "
test $1 -lt $2
echo $?

echo "$1 <= $2? "
test $1 -le $2
echo $?
```

```

disen@qfxa:~/shells$ bash t17.sh 5 8
5 equal 8?
1
5 not equal 8?
0
5 > 8?
1
5 >= 8?
1
5 < 8?
0
5 <= 8?
0

```

2.3.4 复合测试

命令执行控制：

```

command1 && command2  左边命令执行成功(即返回 0) shell 才执行右边的命令
command1 || command2  左边的命令未执行成功(即返回非 0) shell 才执行右边的命令

```

如：测试a.txt是否存在，如果存在，则写入good内容。

```

test -e a.txt && echo "good" >> a.txt
echo $?

```

```

disen@qfxa:~/shells$ bash t18.sh
0
disen@qfxa:~/shells$ cat a.txt
good

```

如：从命令行中获取一个文件的路径，测试它是否存在，如果存在，则测试它是否文件，如果为文件，则查看文件中是否包含 disen的内容，并显示disen所在的行号。

```

test -e $1 && test -f $1 && grep disen $1 -n
echo $?

```

```

disen@qfxa:~/shells$ bash t19.sh a.txt
1:disen
4:disen666
0
disen@qfxa:~/shells$

```

如：从命令行获取一个文件路径，如果文件不存在，则创建

```

[ -e $1 ] || touch $1
echo $?

```

```
disen@qfxa:~/shells$ bash t20.sh a1.txt
0
disen@qfxa:~/shells$ ls -l
total 88
-rw-rw-r-- 1 disen disen    0 8月  14 14:51 a1.txt
drwxrwxr-x 2 disen disen 4096 8月  14 10:24 abc
```

多重条件判定：

-a	<p>(and)两状况同时成立！</p> <p><code>test -r file -a -x file</code></p> <p>file 同时具有 r 与 x 权限时，才为 true.</p>
-o	<p>(or)两状况任何一个成立！</p> <p><code>test -r file -o -x file</code></p> <p>file 具有 r 或 x 权限时，就传回 true.</p>
!	<p>相反状态</p> <p><code>test ! -x file</code></p> <p>当 file 不具有 x 时，回传 true.</p>

如：从命令行读取一个文件路径，测试它即为可读、可写，也为可执行

```
[ -e $1 ] && [ -r $1 -a -w $1 -a -x $1 ]
echo $?
```

```
disen@qfxa:~/shells$ bash t21.sh t6.sh
1
disen@qfxa:~/shells$ bash t21.sh t3.sh
0
disen@qfxa:~/shells$
```

2.4 控制语句

shell中包含的控制语句： if case for while until break

2.4.1 if 控制语句

语法1：

```
if [ 条件 ]; then
    执行程序
fi
```

如: 设计svim 编辑器的脚本，功能自动添加shell脚本的第一行和执行权限， 将并 svim创建/usr/bin/svim的连接（软）

```
#!/bin/bash
# 验证是否提供了文件路径
if [ $# -eq 1 ]
then
    echo "命令行参数有效"
fi
```

将then 和 条件 放在同一行，则需要 ; 分隔

```
#!/bin/bash
# 验证是否提供了文件路径
if [ $# -eq 1 ] ;then
    echo "命令行参数有效"
fi
```

语法2:

```
if [条件 1]; then
    执行第一段程序
else
    执行第二段程序
fi
```

如: svim

```
#!/bin/bash
# 验证是否提供了文件路径
if [ $# -eq 1 ] ;then
    echo "命令行参数有效"
else
    echo "必须提供一个文件名，格式 svim filepath.sh"
fi
```

完成svim的功能:

```
#!/bin/bash
# 验证是否提供了文件路径
if [ $# -eq 1 ] ;then
    if [ ! -e $1 ] ;then
        echo "#!/bin/bash" > $1
        echo "" >> $1
        chmod +x $1
    fi
    vi +2 $1
else
    echo "必须提供一个文件名，格式 svim filepath.sh"
fi
```

添加svim的可执行权限:

```
chmod +x svim
```

创建软链接

```
sudo ln -s /home/disen/shells/svim /usr/bin/svi
```

使用:

```
svi t22.sh
```

```
disen@qfxa:~/shells$ svi t22.sh
disen@qfxa:~/shells$ ./t22.sh
good svim
```

语法3:

```
if [ 条件1 ]; then
    执行第一段程序
elif [条件2 ]; then
    执行第二段程序
else
    执行第三段程序
fi
```

如:

```
#!/bin/bash

echo "please input y/n"
read yn
if [ $yn = "y" ]; then
    echo "you input yes..."
elif [ $yn = "n" ]; then
    echo "exit..."

else
    echo "you input is invalid argument "
fi
```



```

disen@qfxa:~/shells$ ./t23.sh
please input y/n
x
you input is invalid argument
disen@qfxa:~/shells$ ./t23.sh
please input y/n
y
you input yes...
disen@qfxa:~/shells$ ./t23.sh
please input y/n
n
exit...
disen@qfxa:~/shells$

```

【扩展】read命令

```
read [-p "提示信息"] 变量名
```

如：

```
read -p "x:" x
```

2.4.2 case 控制语句

语法：

```

case $变量名称 in
    “第一个变量内容”)
        程序段一
        ;;
    “第二个变量内容”)
        程序段二
        ;;
    *)
        其它程序段
        exit 1 或 ;;
esac

```

如：

```

#!/bin/bash

echo "please input y/n"
read yn

case $yn in
    y)
        echo "you input yes..."
        ;;
    n)
        echo "exit..."

```

```
;;
*)
    echo "you input is invalid argument "
    exit 1
esac
```

【扩展】多个变量值可以通过 `|` 进行连接，进行穿透效果。

```
#!/bin/bash

echo "please input y/n"
read yn

case $yn in
    y|Y|yes|Yes|YES)
        echo "you input yes..."
        ;;
    n|N|no|No|NO)
        echo "exit..."
        ;;
    *)
        echo "you input is invalid argument "
        exit 1
esac
```

```
disen@qfxa:~/shells$ ./t25.sh
please input y/n
yes
you input yes...
disen@qfxa:~/shells$ ./t25.sh
please input y/n
Y
you input yes...
disen@qfxa:~/shells$ ./t25.sh
please input y/n
YES
you input yes...
disen@qfxa:~/shells$ ./t25.sh
please input y/n
NO
exit...
```

2.4.3 for 控制语句

语法1:

```
for (( 初始值; 限制值; 执行步阶 ))
do
    程序段
done
```

declare 是 bash 的一个内建命令，可以用来声明 shell 变量、设置变量的属性。
declare 也可以写作 typeset。

declare -i s 代表强制把 s 变量当做 int 型参数运算。

如： 求100以内所有5的倍数的和

```
#!/bin/bash

declare -i total=0
for((i=5;i <= 100; i++)) ;do
    # shell中进行算术运算及关系比较，最好使用 (( 算术相关表达式 ))
    if (( i%5 == 0 )); then
        total+=i
    fi
done

echo "$total"
```

```
disen@qfxa:~/shells$ ./t26.sh
1050
```

如： 键盘输入一个数值，验证它是否为质数

```
#!/bin/bash

declare -i n
read -p "请输入一个整数： " n

for((i=2; i < n; i++)); do
    if (( n % i == 0 )); then
        # break 结束循环
        break
    fi
done
# i 变量可以在for循环外部使用
if [ $i -eq $n ]; then
    echo "$n 是质数"
else
    echo "$n 不是质数"
fi
```

语法2:

```
for var in con1 con2 con3 ...
do
    程序段
done
```

如： 查看所有.sh文件，统计它们的文件大小

```
declare -i total=0
for line in `ls *.sh`
do
    total+=`ls -ls $line|awk '{print $5}`
done
echo "所有sh文件的总大小为: $total 字节"
```

```
disen@qfxa:~/shells$ ./t29.sh
所有sh文件的总大小为: 4316 字节
disen@qfxa:~/shells$
```

2.4.4 while 控制语句

语法:

```
while [ condition ]
do
    程序段
done
```

如: 聊天小程序, 输入exit退出程序, 其它输入的都按原样输出

```
read -p ">" cmd
while [ $cmd != "exit" ] ;do
    echo "$cmd"
    read -p ">" cmd
done
```

```
disen@qfxa:~/shells$ ./t30.sh
>good
good
>yes
yes
>no
no
>bye
bye
>exit
```

2.4.5 until 控制语句

语法:

```
until [ condition ]
do
    程序段
done
```

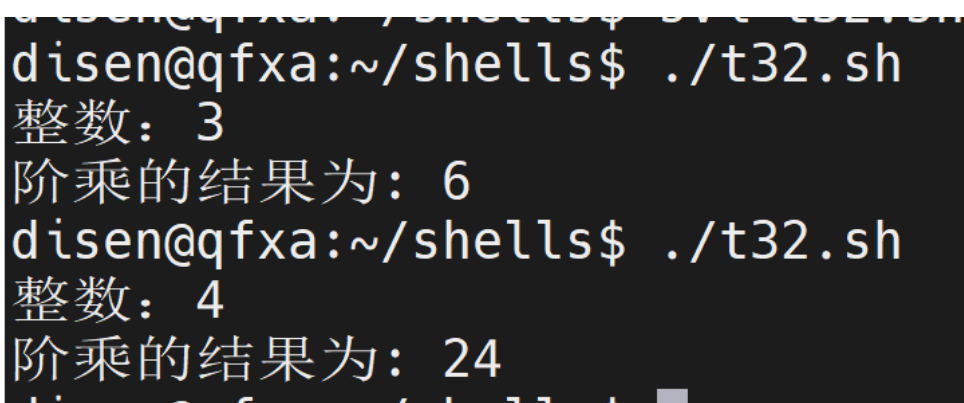
这种方式与 while 恰恰相反, 当 condition 成立的时候退出循环, 否则继续循环。

如：

```
read -p ">" cmd
until [ $cmd == "exit" ] ;do
    echo "$cmd"
    read -p ">" cmd
done
```

如：键盘输入一个整数，计算它的阶乘

```
declare -i ret=1
declare -i n
read -p "整数: " n
until [ $n -eq 1 ]; do
    ret=ret*n
    n=n-1
done
echo "阶乘的结果为: $ret"
```



The screenshot shows a terminal window with the following text:

```
disen@qfxa:~/shells$ ./t32.sh
整数: 3
阶乘的结果为: 6
disen@qfxa:~/shells$ ./t32.sh
整数: 4
阶乘的结果为: 24
```

2.4.6 break和continue

break 命令允许跳出循环，通常在进行一些处理后退出循环或 case 语句

continue 命令类似于 break 命令，只有一点重要差别，它不会跳出循环，只是跳过这个循环步骤。

2.5 函数【了解】

有些脚本段间互相重复，如果能只写一次代码块而在任何地方都能引用那就提高了代码的可重用性。

shell 允许将一组命令集或语句形成一个可用块，这些块称为 shell 函数。

定义函数的两种格式：

```
函数名(){
    命令 ...
}
```

或

```
function 函数名(){  
    命令 ...  
}
```

调用函数的格式为：

```
函数名 param1 param2.....
```

使用参数同在一般脚本中使用特殊变量

\$1, \$2 ...\$9 一样

函数可以使用 return 提前结束并带回返回值. return 0 无错误返回, return 1 有错误返回。

如：设计shell函数，完成传入的参数1(整数)是否为质数，返回0表示真，返回1表示假。

```
#!/bin/bash  
function isZs(){  
    declare -i n  
    # $1 是函数调用时的第一个参数  
    n=$1  
    for((i=2; i< n; i++)) ;do  
        if (( n % i == 0)); then  
            break  
        fi  
    done  
  
    if [ $i -eq $n ] ;then  
        return 0  
    fi  
  
    return 1  
}  
  
# $1 是命令行的参数  
if isZs $1 ; then  
    echo "$1 是质数"  
fi
```

```
disen@qfxa:~/shells$ ./t33.sh 29  
29 是质数  
disen@qfxa:~/shells$ ./t33.sh 28  
disen@qfxa:~/shells$
```