

Bash脚本编程笔记

2020年7月9日 19:51

Bash简介

Bash (GNU Bourne-Again Shell) 是一个为GNU计划编写的Unix shell, 它是许多Linux平台默认使用的shell。

shell是一个命令解释器, 是介于操作系统内核与用户之间的一个绝缘层。准确地说, 它也是能力很强的计算机语言, 被称为解释性语言或脚本语言。它可以通过将系统调用、公共程序、工具和编译过的二进制程序“粘合”在一起建立应用, 这是大多数脚本语言的共同特征, 所以有时候脚本语言又叫做“胶水语言”。事实上, 所有的UNIX命令和工具再加上公共程序, 对于shell脚本来说, 都是可调用的。

一个例子 (hello.sh) :

脚本内容:

```
#!/bin/bash
# This is a comment
echo Hello World
```

注解:

Linux 系统根据 "#!" 及该字符串后面的信息确定该文件的类型, 可以通过 `man magic` 命令 及 `/usr/share/magic` 文件来了解这方面的更多内容。

`#!/bin/bash` 这一行是表示使用 `/bin/bash` 作为脚本的解释器, 这行要放在脚本的行首并且不要省略。BASH 这个程序一般是存放在 `/bin` 目录下, 如果 Linux 系统比较特别, `bash` 也有可能被存放在 `/sbin`、`/usr/local/bin`、`/usr/bin`、`/usr/sbin` 或 `/usr/local/sbin` 这样的目录下; 如果还找不到, 可以用 `locate bash`, `find / -name bash 2>/dev/null` 或 `whereis bash` 这三个命令找出 `bash` 所在的位置; 如果仍然找不到, 那就可能需要自己动手安装一个 BASH 软件包了。

脚本**正文**中以 `#` 开头的行都是注释语句, 这些行在脚本的实际执行过程中不会被执行。

第三行的 `echo` 语句的功能是把 `echo` 后面的字符串输出到标准输出中去。由于 `echo` 后跟的是 "Hello World" 这个字符串, 因此 "Hello World" 这个字符串就被显示在控制台终端的屏幕上了。需要注意的是 BASH 中的绝大多数语句结尾处都没有分号。

运行Bash脚本的方式:

```
# 使用shell来执行
$ sh hello.sh

# 使用bash来执行
$ bash hello.sh

使用.来执行
$ ./hello.sh

使用source来执行
$ source hello.sh

还可以赋予脚本所有者执行权限, 允许该用户执行该脚本
$ chmod u+rx hello.sh
$ ./hello.sh
```

使用重定向:

将hello world保存为一个文本:

脚本内容：

```
#!/bin/bash
echo "Hello World" > my.txt
```

使用脚本清除/var/log下的log文件：

脚本内容：

```
#!/bin/bash

# 初始化一个变量
LOG_DIR=/var/log

cd $LOG_DIR

cat /dev/null > wtmp

echo "Logs cleaned up."

exit
```

注解：

/dev/null 这个东西可以理解为一个黑洞，里面是空的。

特殊字符：

1、注释（#）：

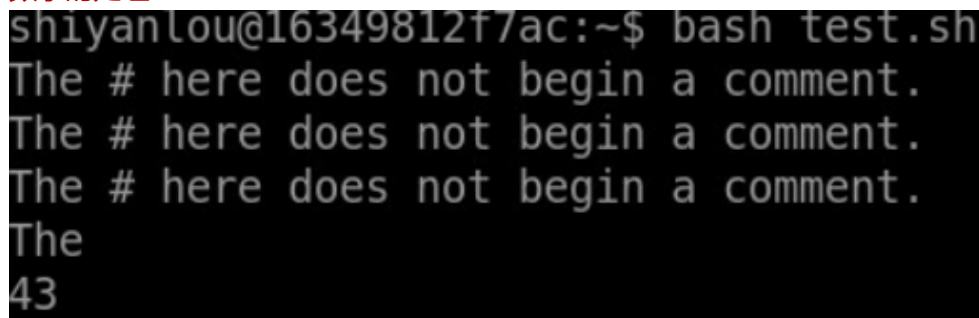
作用：

行首以 # 开头(除#!之外)的是注释。#!是用于指定当前脚本的解释器。在echo中转义的 # 是不能作为注释的。

示例：

```
#!/bin/bash

echo "The # here does not begin a comment."
echo 'The # here does not begin a comment.'
echo The \# here does not begin a comment.
echo The # 这里开始一个注释
echo $(( 2#101011 )) # 数制转换（使用二进制表示），不是一个注释，双括号表示对于数字的处理
```



```
shiyanlou@16349812f7ac:~$ bash test.sh
The # here does not begin a comment.
The # here does not begin a comment.
The # here does not begin a comment.
The
43
```

2、分号（;）：

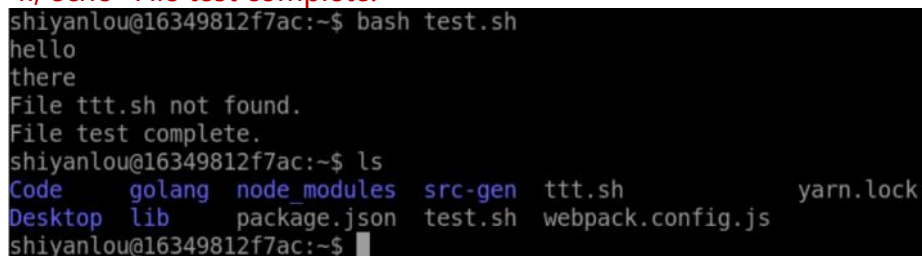
作用：

- 1) 使用分号（;）可以在同一行上写两个或两个以上的命令。
- 2) 终止case选项（双分号）。

示例:

1) :

```
#!/bin/bash
echo hello; echo there
filename=ttt.sh
if [ -e "$filename" ]; then # 注意: "if"和"then"需要分隔, -e用于判断文件是否存在
    echo "File $filename exists."; cp $filename $filename.bak
else
    echo "File $filename not found."; touch $filename
fi; echo "File test complete."
```



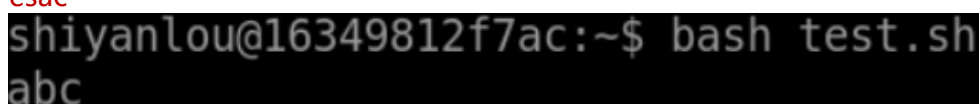
```
shiyanlou@16349812f7ac:~$ bash test.sh
hello
there
File ttt.sh not found.
File test complete.
shiyanlou@16349812f7ac:~$ ls
Code    go lang  node_modules  src-gen  ttt.sh      yarn.lock
Desktop lib     package.json  test.sh   webpack.config.js
shiyanlou@16349812f7ac:~$
```

2) :

```
#!/bin/bash
```

```
varname=b
```

```
case "$varname" in
    [a-z]) echo "abc";;
    [0-9]) echo "123";;
esac
```



```
shiyanlou@16349812f7ac:~$ bash test.sh
abc
```

3、点号 (.) :

作用: 等价于 source 命令。bash 中的 source 命令用于在当前 bash 环境下读取并执行脚本中的命令。

4、引号:

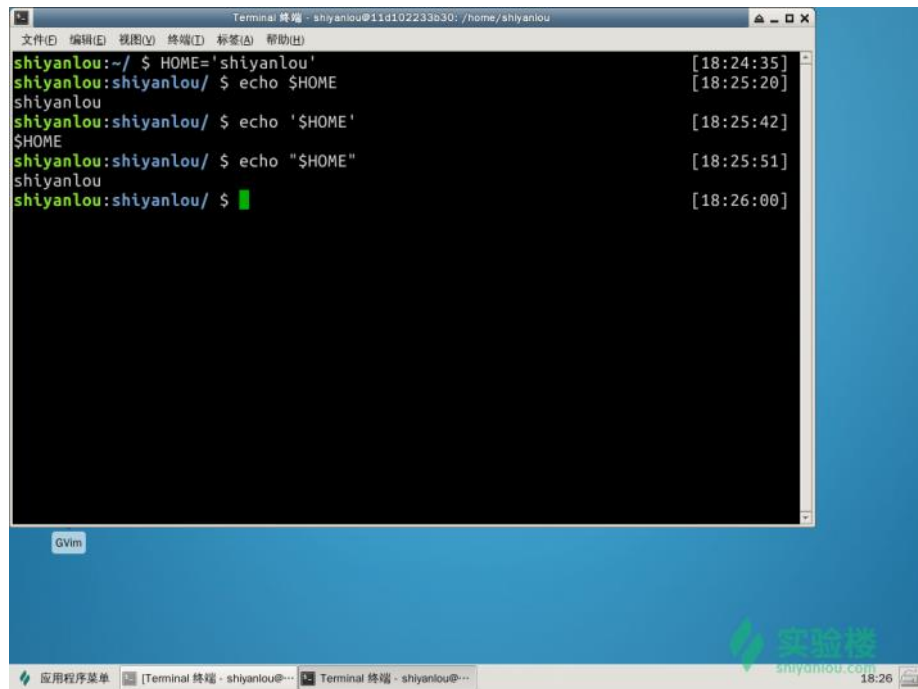
1) 双引号 ("):

作用: "STRING" 将会阻止 (解释) STRING 中大部分特殊的字符。

2) 单引号 ('):

作用: 'STRING' 将会阻止 STRING 中所有特殊字符的解释, 这是一种比使用"更强烈的形式。

示例 (同样是\$HOME,单引号会直接认为是字符, 而双引号认为是一个变量) :



```
Terminal 终端 - shiyanlou@11d102233b30: /home/shiyanlou
文件(F) 编辑(E) 视图(V) 终端(T) 标签(S) 帮助(H)
shiyanlou:~/ $ HOME='shiyanlou' [18:24:35]
shiyanlou:shiyanlou/ $ echo $HOME [18:25:20]
shiyanlou
shiyanlou:shiyanlou/ $ echo '$HOME' [18:25:42]
$HOME
shiyanlou:shiyanlou/ $ echo "$HOME" [18:25:51]
shiyanlou
shiyanlou:shiyanlou/ $ [18:26:00]
```

5、斜线和反斜线：

1) 斜线 (/)：

作用：

文件名路径分隔符。分隔文件名不同的部分（如/home/bozo/projects/Makefile）。也可以用来作为除法算术操作符。注意在linux中表示路径的时候，许多个/跟一个/是一样的。/home/shiyanlou等同于///home///shiyanlou。

2) 反斜线 (\)：

作用：

一种对单字符的引用机制。\X 将会“转义”字符X。这等价于"X"，也等价于'X'。\\通常用来转义双引号（"）和单引号（'），这样双引号和单引号就不会被解释成特殊含义了。

符号 说明

\n 表示新的一行

\r 表示回车

\t 表示水平制表符

\v 表示垂直制表符

\b 表示后退符

\a 表示"alert"(蜂鸣或者闪烁)

\0xx 转换为八进制的ASCII码, 等价于0xx

\" 表示引号字面的意思

转义符也提供续行功能，也就是编写多行命令的功能。

每一个单独行都包含一个不同的命令，但是每行结尾的转义符都会转义换行符，这样下一行会与上一行一起形成一个命令序列。

6、反引号 (`)：

作用：反引号中的命令会优先执行。

7、冒号 (:) :

作用:

- 1) 空命令。等价于“NOP” (no op, 一个什么也不干的命令)。也可以被认为与shell的内建命令true作用相同。“:”命令是一个bash的内建命令, 它的退出码 (exit status) 是 (0) 。
- 2) 变量扩展/子串替换。在与>重定向操作符结合使用时, 将会把一个文件清空, 但是并不会修改这个文件的权限。如果之前这个文件并不存在, 那么就创建这个文件。在与>>重定向操作符结合使用时, 将不会对预先存在的目标文件 (: >> target_file)产生任何影响。如果这个文件之前并不存在, 那么就创建它。“:”还用来在 /etc/passwd 和 \$PATH 变量中做分隔符。

8、问号 (?) :

作用:

测试操作符。在一个双括号结构中, ? 就是C语言的三元操作符。

示例:

```
#!/bin/bash
```

```
a=10  
(( t=a<50?8:9 ))  
echo $t
```

```
shianlou@16349812f7ac:~$ bash test.sh  
8
```

9、美元符号 (\$) :

作用:

用作变量替换 (取变量保存的值) :

示例:

```
#!/bin/bash
```

```
var1=5  
var2=23skidoo
```

```
echo $var1    # 5  
echo $var2    # 23skidoo
```

```
shianlou@16349812f7ac:~$ bash test.sh  
5  
23skidoo  
shianlou@16349812f7ac:~$
```

10、小括号 (()) :

作用:

- 1) 命令组。在括号中的命令列表, 将会作为一个子 shell 来运行。在括号中的变量, 由于是在子shell中, 所以对于脚本剩下的部分是不可用的。父进程, 也就是脚本本身, 将不能够读

取在子进程中创建的变量，也就是在子shell 中创建的变量。

2) 初始化数组。

示例：

1) :

```
#!/bin/bash
```

```
a=123
```

```
( a=321; )
```

```
echo "$a" #a的值为123而不是321，因为括号将判断为局部变量
```

```
shiyanolou@16349812f7ac:~$ bash test.sh
123
```

2) :

```
#!/bin/bash
```

```
arr=(1 4 5 7 9 21)
```

```
echo ${arr[3]} # get a value of arr
```

```
shiyanolou@16349812f7ac:~$ bash test.sh
7
```

11、大括号 ({}):

作用：

- 1) 文件名扩展。注意：此时在大括号中，不允许有空白，除非这个空白被引用或转义。
- 2) 代码块，又被称为内部组，这个结构事实上创建了一个匿名函数（一个没有名字的函数）。然而，与“标准”函数不同的是，在其中声明的变量，对于脚本其他部分的代码来说还是可见的。

示例：

1) :

```
#!/bin/bash
```

```
if [ ! -w 't.txt' ];
```

```
then
```

```
    touch t.txt
```

```
fi
```

```
echo 'test text' >> t.txt
```

```
cp t.{txt,back}
```

```
shiyanolou@16349812f7ac:~$ bash test.sh
```

```
shiyanolou@16349812f7ac:~$ ls
```

```
Code    golang  node_modules  src-gen  test.sh  t.txt                                yarn.lock
```

```
Desktop lib    package.json  t.back  ttt.sh  webpack.config.js
```

```
shiyanolou@16349812f7ac:~$ cat t.txt
```

```
test text
```

```
shiyanolou@16349812f7ac:~$ cat t.back
```

```
test text
```

2) :

```
#!/bin/bash
```

```
a=123  
{ a=321; }  
echo "a = $a"
```

```
shiyamlou@16349812f7ac:~$ bash test.sh  
a = 321
```

12、中括号 ([]):

作用:

- 1) 条件测试表达式放在[]中。
- 2) 在一个array结构的上下文中, 中括号用来引用数组中每个元素的编号。

示例:

1) :

```
#!/bin/bash
```

```
a=5  
if [ $a -lt 10 ]  
then  
    echo "a: $a"  
else  
    echo 'a>=10'  
fi
```

```
$ bash test24.sh  
a: 5
```

2) :

```
#!/bin/bash
```

```
arr=(12 22 32)  
arr[0]=10  
echo ${arr[0]}
```

```
$ bash test25.sh  
10
```

13、尖括号 (< 和 >):

作用:

重定向。**test.sh > filename**: 重定向test.sh的输出到文件 filename 中。如果 filename 存在的话, 那么将会被覆盖。**test.sh &> filename**: 重定向 test.sh 的 stdout (标准输出) 和 stderr (标准错误) 到 filename 中。**test.sh >&2**: 重定向 test.sh 的 stdout 到 stderr 中。**test.sh >> filename**: 把 test.sh 的输出追加到文件 filename 中。如果filename不存在的话, 将会被创建。

14、竖线 (|):

作用: 管道。分析前边命令的输出, 并将输出作为后边命令的输入。

15、破折号 (-):

作用：

- 1) 选项，前缀。在所有的命令内如果想使用选项参数的话,前边都要加上“-”。
- 2) 用于重定向stdin或stdout。

16、波浪号 (~)：

作用：~ 表示 home 目录。

变量：

概念：

变量的名字就是变量保存值的地方。引用变量的值就叫做变量替换。如果 variable 是一个变量的名字，那么 \$variable 就是引用这个变量的值，即这变量所包含的数据。variable 事实上只是 {variable} 的简写形式。在某些上下文中 variable 可能会引起错误，这时候就需要用 {variable} 了，加花括号可以帮助解释器识别变量的边界。

定义变量：

定义变量时，变量名不加美元符号 (\$，PHP语言中变量需要)，如：myname="shianlou"。

注意：变量名和等号之间不能有空格。

同时，变量名的命名须遵循如下规则：

首个字符必须为字母 (a-z, A-Z)。

中间不能有空格，可以使用下划线 (_)。

不能使用标点符号。

不能使用bash里的关键字（可用help命令查看保留关键字）。

除了直接赋值，还可以用语句给变量赋值，如：for file in `ls /etc`

只读变量：

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变。

示例：

```
#!/bin/bash
myUrl="http://www.shianlou.com"
readonly myUrl
myUrl="http://www.shianlou.com"
```

运行脚本，结果如下：

```
/bin/sh: NAME: This variable is read only.
```

特殊变量：

1、局部变量：这种变量只有在代码块或者函数中才可见，如小括号里的变量。

2、环境变量：

这种变量将影响用户接口和 shell 的行为。在通常情况下，每个进程都有自己的“环境”，这个环境是由一组变量组成的，这些变量中存有进程可能需要引用的信息。在这种情况下，shell 与一个一般的进程没什么区别。

3、位置参数：

从命令行传递到脚本的参数：0, 1, 2, 3...

0就是脚本文件自身的名字，0就是脚本文件自身的名字，1 是第一个参数，2 是第二个参数，2是第二个参数，3 是第三个参数，然后是第四个。9 之后的位置参数就必须用大括号括起来了，比如，9之后的位置参数就必须用大括号括起来了，比如{10}，{11}，{12}。

\$# : 传递到脚本的参数个数
\$* : 以一个单字符串显示所有向脚本传递的参数。与位置变量不同,此选项参数可超过 9个
\$\$: 脚本运行的当前进程 ID号
\$! : 后台运行的最后一个进程的进程 ID号
\$@ : 与\$*相同,但是使用时加引号,并在引号中返回每个参数
\$: 显示shell使用的当前选项,与 set命令功能相同
\$? : 显示最后命令的退出状态。0表示没有错误,其他任何值表明有错误。

示例:

```
#!/bin/bash

# 作为用例, 调用这个脚本至少需要10个参数, 比如:
# bash test.sh 1 2 3 4 5 6 7 8 9 10
MINPARAMS=10

echo

echo "The name of this script is \"$0\"."

echo "The name of this script is \"`basename $0`\"."

echo

if [ -n "$1" ]          # 测试变量被引用.
then
echo "Parameter #1 is $1" # 需要引用才能够转义"#"
fi

if [ -n "$2" ]
then
echo "Parameter #2 is $2"
fi

if [ -n "${10}" ] # 大于$9的参数必须用{}括起来.
then
echo "Parameter #10 is ${10}"
fi

echo "-----"
echo "All the command-line parameters are: "$*"

if [ $# -lt "$MINPARAMS" ]
then
echo
echo "This script needs at least $MINPARAMS command-line arguments!"
fi

echo

exit 0
```

```
$ bash test30.sh 1 2 10

The name of this script is "test.sh".
The name of this script is "test.sh".

Parameter #1 is 1
Parameter #2 is 2
-----
All the command-line parameters are: 1 2 10

This script needs at least 10 command-line arguments!
```

基本运算符：

1、算术运算符：

运算符	说明
+	加法
-	减法
*	乘法
/	除法
%	取余
=	赋值
==	相等。用于比较两个数字，相同则返回 true。
!=	不相等。用于比较两个数字，不相同则返回 true。

示例：

```
#!/bin/bash

a=10
b=20

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"

if [ $a == $b ]
then
    echo "a == b"
fi
if [ $a != $b ]
then
    echo "a != b"
```

fi

```
$bash test.sh
a + b : 30
a - b : -10
a * b : 200
b / a : 2
b % a : 0
a != b
```

注:

原生bash不支持简单的数学运算，但是可以通过其他命令来实现，例如 `awk` 和 `expr`，`expr` 最常用。`expr` 只能用于整数计算，可以使用 `bc` 或者 `awk` 进行浮点数运算。

`expr` 是一款表达式计算工具，使用它能完成表达式的求值操作。

注意使用的反引号（`esc`键下边）

表达式和运算符之间要有空格 `$a + $b` 写成 `$a+$b` 不行

条件表达式要放在方括号之间，并且要有空格 `[$a == $b]` 写成 `[$a==$b]` 不行

乘号（`*`）前边必须加反斜杠（`\`）才能实现乘法运算

2、关系运算符:

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

运算符	说明
<code>-eq</code>	检测两个数是否相等，相等返回 <code>true</code> 。
<code>-ne</code>	检测两个数是否相等，不相等返回 <code>true</code> 。
<code>-gt</code>	检测左边的数是否大于右边的，如果是，则返回 <code>true</code> 。
<code>-lt</code>	检测左边的数是否小于右边的，如果是，则返回 <code>true</code> 。
<code>-ge</code>	检测左边的数是否大于等于右边的，如果是，则返回 <code>true</code> 。
<code>-le</code>	检测左边的数是否小于等于右边的，如果是，则返回 <code>true</code> 。

示例:

```
#!/bin/bash

a=10
b=20

if [ $a -eq $b ]
then
    echo "$a -eq $b : a == b"
else
    echo "$a -eq $b: a != b"
fi
```

```
$bash test2.sh
10 -eq 20: a != b
```

3、逻辑运算符:

运算符	说明
<code>&&</code>	逻辑的 AND
<code> </code>	逻辑的 OR

4、字符串运算符:

运算符	说明
=	检测两个字符串是否相等，相等返回 true
!=	检测两个字符串是否相等，不相等返回 true
-z	检测字符串长度是否为 0，为 0 返回 true
-n	检测字符串长度是否为 0，不为 0 返回 true
str	检测字符串是否为空，不为空返回 true

5、文件测试运算符：

操作符	说明
-e	文件存在
-a	文件存在，这个选项的效果与 -e 相同。但是它已经被“弃用”了，并且不鼓励使用。
-f	表示这个文件是一个一般文件（并不是目录或者设备文件）
-s	文件大小不为零
-d	表示这是一个目录
-b	表示这是一个块设备（软盘，光驱，等等）
-c	表示这是一个字符设备（键盘，modem，声卡，等等）
-p	这个文件是一个管道
-h	这是一个符号链接
-L	这是一个符号链接
-S	表示这是一个 socket
-t	文件（描述符）被关联到一个终端设备上，这个测试选项一般被用来检测脚本中的 <code>stdin([-t 0])</code> 或者 <code>stdout([-t 1])</code> 是否来自于一个终端
-r	文件是否具有可读权限（指的是正在运行这个测试命令的用户是否具有读权限）
-w	文件是否具有可写权限（指的是正在运行这个测试命令的用户是否具有写权限）
-x	文件是否具有可执行权限（指的是正在运行这个测试命令的用户是否具有可执行权限）
-g	set-group-id(sgid)标记被设置到文件或目录上
-k	设置粘贴位
-O	判断你是否是文件的拥有者
-G	文件的group-id是否与你的相同
-N	从文件上一次被读取到现在为止，文件是否被修改过
f1 -nt f2	文件f1比文件f2新
f1 -ot f2	文件f1比文件f2旧
f1 -ef f2	文件f1和文件f2是相同文件的硬链接
!	“非”，反转上边所有测试的结果（如果没给出条件，那么返回真）

示例：

```
#!/bin/bash

file="/home/shiyanlou/test.sh"
if [ -r $file ]
then
    echo "The file is readable"
else
    echo "The file is not readable"
fi
```

```

if [ -e $file ]
then
    echo "File exists"
else
    echo "File not exists"
fi

```

```

The file is readable
File exists

```

流程控制：

1、if else:

1) if:

```

if condition
then
    command1
    command2
    ...
    commandN
fi

```

2) if else:

```

if condition
then
    command1
    command2
    ...
    commandN
else
    command
fi

```

3) if-elif-else:

```

if condition1
then
    command1
elif condition2
then
    command2
else
    commandN
fi

```

if else语句与test命令结合使用的示例：

```

num1=$((2*3))
num2=$((1+5))
if test $[num1] -eq $[num2]
then
    echo 'Two numbers are equal!'
else
    echo 'The two numbers are not equal!'
fi

```

输出结果：

```

Two numbers are equal!

```

2、for 循环:

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

3、while 语句:

```
while condition
do
    command
done
```

一个Bash let示例:

```
#!/bin/bash
int=1
while(( $int<=5 ))
do
    echo $int
    let "int++"
done
```

一个读取键盘信息的示例:

```
echo 'press <CTRL-D> exit'
echo -n 'Who do you think is the most handsome: '
while read MAN
do
    echo "Yes! $MAN is really handsome"
done
```

注: `echo -n`命令可以使输入在echo输出的同一行。

4、无限循环:

```
while :
do
    command
done
或者
while true
do
    command
done
或者
for ((;;))
```

5、until 循环:

```
until condition
do
    command
done
```

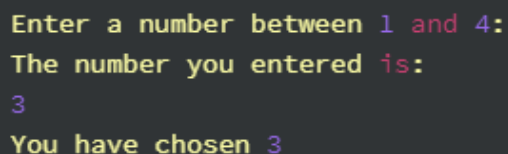
6、case:

```
case 值 in
模式1)
    command1
    command2
    ...
    commandN
;;
模式2)
    command1
    command2
    ...
    commandN
;;
esac
```

注：取值后面必须为单词in，每一模式必须以右括号结束。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;;。取值将检测匹配的每一个模式。一旦模式匹配，则执行完匹配模式相应命令后不再继续其他模式。如果无一匹配模式，使用星号*捕获该值，再执行后面的命令。case的语法和C family语言差别很大，它需要一个esac（就是case反过来）作为结束标记，每个case分支用右圆括号，用两个分号表示break。

一个示例：

```
echo 'Enter a number between 1 and 4:'
echo 'The number you entered is:'
read aNum
case $aNum in
    1) echo 'You have chosen 1'
    ;;
    2) echo 'You have chosen 2'
    ;;
    3) echo 'You have chosen 3'
    ;;
    4) echo 'You have chosen 4'
    ;;
    *) echo 'You did not enter a number between 1 and 4'
    ;;
esac
```



```
Enter a number between 1 and 4:
The number you entered is:
3
You have chosen 3
```

7、break和continue:

break命令允许跳出所有循环（终止执行后面的所有循环）。continue命令与break命令类似，只有一点差别，它不会跳出所有循环，仅仅跳出当前循环。

函数：

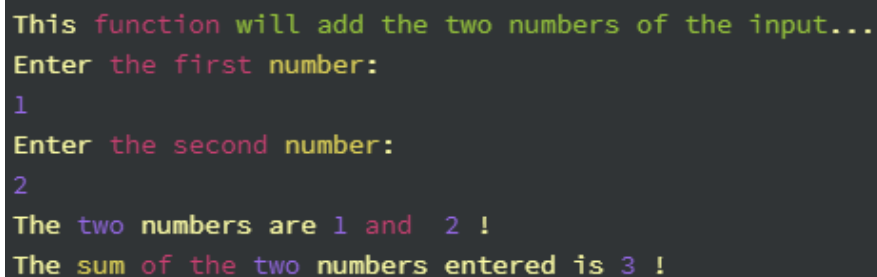
定义格式:

```
[ function ] funname [()]  
  
{  
  
    action;  
  
    [return int;]  
  
}
```

注: 可以带`function fun()` 定义, 也可以直接`fun()` 定义, 不带任何参数。参数返回, 可以显示加:
`return` 返回, 如果不加, 将以最后一条命令运行结果, 作为返回值。 `return`后跟数值`n(0-255)`。
所有函数在使用前必须定义。

示例1:

```
#!/bin/bash  
funWithReturn(){  
    echo "This function will add the two numbers of the input..."  
    echo "Enter the first number: "  
    read aNum  
    echo "Enter the second number: "  
    read anotherNum  
    echo "The two numbers are $aNum and $anotherNum !"  
    return $((aNum+anotherNum))  
}  
funWithReturn  
echo "The sum of the two numbers entered is $? !"
```



```
This function will add the two numbers of the input...  
Enter the first number:  
1  
Enter the second number:  
2  
The two numbers are 1 and 2 !  
The sum of the two numbers entered is 3 !
```

注: 函数返回值在调用该函数后通过 `$?` 来获得。

示例2:

```
#!/bin/bash  
funWithParam(){  
    echo "The first parameter is $1 !"  
    echo "The second parameter is $2 !"  
    echo "The tenth parameter is $10 !"  
    echo "The tenth parameter is ${10} !"  
    echo "The eleventh parameter is ${11} !"  
    echo "The total number of parameters is $# !"  
    echo "Outputs all parameters as a string $* !"  
}  
funWithParam 1 2 3 4 5 6 7 8 9 34 73
```



```
The first parameter is 1 !  
The second parameter is 2 !  
The tenth parameter is 10 !  
The tenth parameter is 34 !  
The eleventh parameter is 73 !  
The total number of parameters is 11 !  
Outputs all parameters as a string 1 2 3 4 5 6 7 8 9 34 73 !
```

注：10不能获取第十个参数，获取第十个参数需要{10}。当 $n \geq 10$ 时，需要使用 $\${n}$ 来获取参数。