shell学习

awk sed grep三个指令的使用

2019年7月25日

14:44

sh -x shell\_name shell文件的调试

命令 sh -x file.sh 提供跟踪执行信息，将执行的每一条命令和结果依次打印出来；

每行前面的+号后面的内容就是要执行的命名，多个加号代表命令的嵌套

行前面没有加号，则代表执行结果的输出

**shell数组**

bash支持一维数组（不支持多维数组），并且没有限定数组的大小。

类似于 C 语言，数组元素的下标由 0 开始编号。获取数组中的元素要利用下标，下标可以是整数或算术表达式，其值应大于或等于 0。

**定义数组**

在 Shell 中，用括号来表示数组，数组元素用"空格"符号分割开。定义数组的一般形式为：

数组名=(值1 值2 ... 值n)

例如：

array\_name=(value0 value1 value2 value3)

或者

array\_name=(

value0

value1

value2

value3

)

还可以单独定义数组的各个分量：

array\_name[0]=value0

array\_name[1]=value1

array\_name[n]=valuen

可以不使用连续的下标，而且下标的范围没有限制。

**读取数组**

读取数组元素值的一般格式是：

${数组名[下标]}

例如：

valuen=${array\_name[n]}

使用 @ 符号可以获取数组中的所有元素，例如：

echo ${array\_name[@]}

**获取数组的长度**

获取数组长度的方法与获取字符串长度的方法相同，例如：

# 取得数组元素的个数

length=${#array\_name[@]}

# 或者

length=${#array\_name[\*]}

# 取得数组单个元素的长度

lengthn=${#array\_name[n]}

cat/etc/euleros-latest|awk -F**'=''$1~/eulerversion/{print$2}'**|xargs

|管道，将cat到的内容输送到后面，awk分割，以=为分隔符，$1~**/eulerversion/{print$2},** 获取分割后第一部分为eulerversion的第二部分。xargs将结果写在一行。

**shell脚本文件**

* 1. test.sh
  2. test.php

shell指令的执行(chmod +x test.sh使其有执行权限）

* 1. ./test.sh
  2. sh test.sh

读取变量

read name

shell test命令

shell中的test命令用于检查某个条件是否成立，它可以进行数值、字符和文件三个方面的测试

if test ${num1} -eq ${num2}

then

echo '两个数相等'

else

echo '两个数不相等'

fi

chmod命令

* + u 表示该文件的拥有者，g 表示与该文件的拥有者属于同一个群体(group)者，o 表示其他以外的人，a 表示这三者皆是。
  + + 表示增加权限、- 表示取消权限、= 表示唯一设定权限。
  + r 表示可读取，w 表示可写入，x 表示可执行，X 表示只有当该文件是个子目录或者该文件已经被设定过为可执行。

其他参数说明：

* + -c : 若该文件权限确实已经更改，才显示其更改动作
  + -f : 若该文件权限无法被更改也不要显示错误讯息
  + -v : 显示权限变更的详细资料
  + -R : 对目前目录下的所有文件与子目录进行相同的权限变更(即以递回的方式逐个变更)
  + --help : 显示辅助说明
  + --version : 显示版本

来自 <<https://www.runoob.com/linux/linux-comm-chmod.html>>

其中a,b,c各为一个数字，分别表示User、Group、及Other的权限。

**r=4，w=2，x=1**

* + 若要rwx属性则4+2+1=7；
  + 若要rw-属性则4+2=6；
  + 若要r-x属性则4+1=5。

chmod命令

chmod 参数 文件名

可以使用如下方法直接设置文件的权限

1.直接设置代表权限的数字  
chmod 777（对应权限的数字） 文件名  
2.给文件添加或减去某些权限  
 chmod u+w 文件名 给user用户加上w权限  
 chmod g-r 文件名 将group中的r权限去掉  
 chmod a+x 文件名 给user、group、other全部加上x权限

来自 <<https://www.cnblogs.com/xs104/p/4510114.html>>

**shell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：

your\_name="runoob.com"

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

* + 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
  + 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）。

**使用变量**

只要在变量名前面加美元符合即可$

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，推荐给所有的变量都加上花括号，这是一个很好的编程习惯。

**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。

例：

value=3

readonly value

**删除变量**

使用unset命令可以删除变量

unset variable\_name

变量删除后不能再次被使用，unset命令不能删除只读变量

变量类型

运行shell时，会同时存在三种变量：

* + **1) 局部变量** 局部变量在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动的程序不能访问局部变量。
  + **2) 环境变量** 所有的程序，包括shell启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候shell脚本也可以定义环境变量。
  + **3) shell变量** shell变量是由shell程序设置的特殊变量。shell变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了shell的正常运行

其中，文档类型有如下表示方法：

d - 目录，例如上表档名为『.gconf』的那一行；

- - 文档，例如上表档名为『install.log』那一行；

l  - 连结档(link file)；

b - 装置文件里面的可供储存的接口设备(可随机存取装置)；

c - 装置文件里面的串行端口设备，例如键盘、鼠标(一次性读取装置)。

权限由rwx三个字幕表示，分别表示为可读、可写、可执行，如果没有该权限，则用“ - ”表示，**对于目录来说，必须有x权限，否则无法读取目录内容**

如果文件名前面有“ . ”,则表示这个文档或目录是隐藏的

来自 <<https://www.cnblogs.com/xs104/p/4510114.html>>

**shell注释**

以 # 开头的行就是注释，会被解释器忽略。

通过每一行加一个 # 号设置多行注释，像这样：

#--------------------------------------------

# 这是一个注释

# author：菜鸟教程

# site：www.runoob.com

# slogan：学的不仅是技术，更是梦想！

#--------------------------------------------

##### 用户配置区 开始 #####

#

#

# 这里可以添加脚本描述信息

#

#

##### 用户配置区 结束 #####

如果在开发过程中，遇到大段的代码需要临时注释起来，过一会儿又取消注释，怎么办呢？

每一行加个#符号太费力了，可以把这一段要注释的代码用一对花括号括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行，达到了和注释一样的效果。

**多行注释**

多行注释还可以使用以下格式：

:<<EOF

注释内容...

注释内容...

注释内容...

EOF

EOF 也可以使用其他符号:

:<<'

注释内容...

注释内容...

注释内容...

'

:<<!

注释内容...

注释内容...

注释内容...

!

**shell字符串**

字符串是shell编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。单双引号的区别跟PHP类似。

**单引号**

str='this is a string'

单引号字符串的限制：

单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；（体现在变量的拼接）

单引号字串中不能出现单独一个的单引号（对单引号使用转义符后也不行），但可成对出现，作为字符串拼接使用。

**双引号**

your\_name='runoob'

str="Hello, I know you are \"$your\_name\"! \n"

echo -e $str

输出结果为：

Hello, I know you are "runoob"!

双引号的优点：

双引号里可以有变量（体现在变量的拼接）

双引号里可以出现转义字符

**拼接字符串**

your\_name="runoob"

# 使用双引号拼接

greeting="hello, "$your\_name" !"

greeting\_1="hello, ${your\_name} !"

echo $greeting $greeting\_1

# 使用单引号拼接

greeting\_2='hello, '$your\_name' !'

greeting\_3='hello, ${your\_name} !'

echo $greeting\_2 $greeting\_3

输出结果为：

hello, runoob ! hello, runoob !

hello, runoob ! hello, ${your\_name} !

**获取字符串长度**

string="abcd"

echo ${#string} #输出 4

**提取子字符串**

以下实例从字符串第 2 个字符开始截取 4 个字符：

string="runoob is a great site"

echo ${string:1:4} # 输出 unoo

**查找子字符串**

查找字符 i 或 o 的位置(哪个字母先出现就计算哪个)：

string="runoob is a great site"

echo `expr index "$string" io` # 输出 4

注意： 以上脚本中 ` 是反引号，而不是单引号 '，不要看错了哦。

**shell基本运算符**

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

例：两个数相加(注意使用的是反引号 ` ，在ESC下面，而不是单引号 ')

**val=`expr 2 + 2`**

**echo "两数之和为 : $val"**

**注：**

* 1. **表达式和运算符之间要有空格，例如 2+2 是不对的，必须写成 2 + 2，这与我们熟悉的大多数编程语言不一样。**
  2. **完整的表达式要被 ` ` 包含，注意这个字符不是常用的单引号，在 Esc 键下边。**

**算术运算符**

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| + | 加法 | `expr $a + $b` 结果为 30。 |
| - | 减法 | `expr $a - $b` 结果为 -10。 |
| \* | 乘法 | `expr $a \\* $b` 结果为  200。 |
| / | 除法 | `expr $b / $a` 结果为 2。 |
| % | 取余 | `expr $b % $a` 结果为 0。 |
| = | 赋值 | a=$b 将把变量 b 的值赋给 a。 |
| == | 相等。用于比较两个数字，相同则返回 true。 | [ $a == $b ] 返回 false。 |
| != | 不相等。用于比较两个数字，不相同则返回 true。 | [ $a != $b ] 返回 true。 |

**注意：**条件表达式要放在方括号之间，并且要有空格，例如: **[$a==$b]** 是错误的，必须写成 **[ $a == $b ]**。乘号(\*)前边必须加反斜杠(\)才能实现乘法运算；

**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。

下表列出了常用的关系运算符，假定变量 a 为 10，变量 b 为 20：

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| -eq | 检测两个数是否相等，相等返回 true。 | [ $a -eq $b ] 返回 false。 |
| -ne | 检测两个数是否不相等，不相等返回 true。 | [ $a -ne $b ] 返回 true。 |
| -gt | 检测左边的数是否大于右边的，如果是，则返回 true。 | [ $a -gt $b ] 返回 false。 |
| -lt | 检测左边的数是否小于右边的，如果是，则返回 true。 | [ $a -lt $b ] 返回 true。 |
| -ge | 检测左边的数是否大于等于右边的，如果是，则返回 true。 | [ $a -ge $b ] 返回 false。 |
| -le | 检测左边的数是否小于等于右边的，如果是，则返回 true。 | [ $a -le $b ] 返回 true。 |

**布尔运算符**

下表列出了常用的布尔运算符，假定变量 a 为 10，变量 b 为 20：

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| ! | 非运算，表达式为 true 则返回 false，否则返回 true。 | [ ! false ] 返回 true。 |
| -o | 或运算，有一个表达式为 true 则返回 true。 | [ $a -lt 20 -o $b -gt 100 ] 返回 true。 |
| -a | 与运算，两个表达式都为 true 才返回 true。 | [ $a -lt 20 -a $b -gt 100 ] 返回 false。 |

**逻辑运算符**

以下介绍 Shell 的逻辑运算符，假定变量 a 为 10，变量 b 为 20:

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| && | 逻辑的 AND | [[ $a -lt 100 && $b -gt 100 ]] 返回 false |
| || | 逻辑的 OR | [[ $a -lt 100 || $b -gt 100 ]] 返回 true |

**字符串运算符**

下表列出了常用的字符串运算符，假定变量 a 为 "abc"，变量 b 为 "efg"：

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| = | 检测两个字符串是否相等，相等返回 true。 | [ $a = $b ] 返回 false。 |
| != | 检测两个字符串是否相等，不相等返回 true。 | [ $a != $b ] 返回 true。 |
| -z | 检测字符串长度是否为0，为0返回 true。 | [ -z $a ] 返回 false。 |
| -n | 检测字符串长度是否为0，不为0返回 true。 | [ -n "$a" ] 返回 true。 |
| $ | 检测字符串是否为空，不为空返回 true。 | [ $a ] 返回 true。 |

**文件测试运算符**

文件测试运算符用于检测 Unix 文件的各种属性。

属性检测描述如下：

|  |  |  |
| --- | --- | --- |
| **操作符** | **说明** | **举例** |
| -b file | 检测文件是否是块设备文件，如果是，则返回 true。 | [ -b $file ] 返回 false。 |
| -c file | 检测文件是否是字符设备文件，如果是，则返回 true。 | [ -c $file ] 返回 false。 |
| -d file | 检测文件是否是目录，如果是，则返回 true。 | [ -d $file ] 返回 false。 |
| -f file | 检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。 | [ -f $file ] 返回 true。 |
| -g file | 检测文件是否设置了 SGID 位，如果是，则返回 true。 | [ -g $file ] 返回 false。 |
| -k file | 检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。 | [ -k $file ] 返回 false。 |
| -p file | 检测文件是否是有名管道，如果是，则返回 true。 | [ -p $file ] 返回 false。 |
| -u file | 检测文件是否设置了 SUID 位，如果是，则返回 true。 | [ -u $file ] 返回 false。 |
| -r file | 检测文件是否可读，如果是，则返回 true。 | [ -r $file ] 返回 true。 |
| -w file | 检测文件是否可写，如果是，则返回 true。 | [ -w $file ] 返回 true。 |
| -x file | 检测文件是否可执行，如果是，则返回 true。 | [ -x $file ] 返回 true。 |
| -s file | 检测文件是否为空（文件大小是否大于0），不为空返回 true。 | [ -s $file ] 返回 true。 |
| -e file | 检测文件（包括目录）是否存在，如果是，则返回 true。 | [ -e $file ] 返回 true。 |

其他检查符：

* 1. **-S**: 判断某文件是否 socket。
  2. **-L**: 检测文件是否存在并且是一个符号链接。

shell传递参数

我们可以在执行 Shell 脚本时，向脚本传递参数，脚本内获取参数的格式为：$n。n 代表一个数字，1 为执行脚本的第一个参数，2 为执行脚本的第二个参数，以此类推……

实例：（其中 $0 为执行的文件名）

./test.sh中的内容

echo "Shell 传递参数实例！";

echo "执行的文件名：$0";

echo "第一个参数为：$1";

echo "第二个参数为：$2";

echo "第三个参数为：$3";

执行脚本：./test.sh 1 2 3

输出结果：

Shell 传递参数实例！

执行的文件名：./test.sh

第一个参数为：1

第二个参数为：2

第三个参数为：3

另外，还有几个特殊字符用来处理参数：

|  |  |
| --- | --- |
| **参数处理** | **说明** |
| $# | 传递到脚本的参数个数 |
| $\* | 以一个单字符串显示所有向脚本传递的参数。  如"$\*"用「"」括起来的情况、以"$1 $2 … $n"的形式输出所有参数。 |
| $$ | 脚本运行的当前进程ID号 |
| $! | 后台运行的最后一个进程的ID号 |
| $@ | 与$\*相同，但是使用时加引号，并在引号中返回每个参数。  如"$@"用「"」括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。 |
| $- | 显示Shell使用的当前选项，与[set命令](https://www.runoob.com/linux/linux-comm-set.html)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 |

**shell echo命令**

* 1. 显示普通字符串

echo "It is a test"

或 echo It is a test

* 1. 显示转移字符

echo "\"It is a test\""

（双引号可以省略）

* 1. 显示变量

上面有显示

* 1. 显示换行

echo -e "OK! \n"#-e 开始转义

echo "It is a test"

* 1. 显示不换行

echo -e "OK! \c" #-e 开始转义 \c不换行

* 1. 显示结果定向至文件

echo "It is a test">myfile

* 1. 原样输出字符串，不进行转义或取变量（用单引号）
  2. 显示命令执行结果

echo `data` #（反引号而不是单引号）

结果将显示当前日期

**shell流程控制**

**if**（经常与test连用）

if condition

then

command1

command2

command3

fi

**if else**

if condition

then

command1

command2

else

command

fi

**if else if else**

if condition

then

command1

elif condition2

then

command2

else

commandN

fi

**for循环**

for var in item1 item2 … itemN

do

command1

command2

……

commandN

done

**while语句**

while condition

do

command

done

**无限循环**

while :

do

command

done

或者

while true

do

command

done

**until循环**

until循环执行一系列命令直至条件为true时停止

until command

do

command

done

**case**

Shell case语句为多选择语句。可以用case语句匹配一个值与一个模式，如果匹配成功，执行相匹配的命令。case语句格式如下：

case 值 in

模式1)

command1

command2

...

commandN

;;

模式2）

command1

command2

...

commandN

;;

esac

case工作方式如上所示。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。

例：

echo '输入 1 到 4 之间的数字:'

echo '你输入的数字为:'

read aNum

case $aNum in

1) echo '你选择了 1'

;;

2) echo '你选择了 2'

;;

3) echo '你选择了 3'

;;

4) echo '你选择了 4'

;;

\*) echo '你没有输入 1 到 4 之间的数字'

;;

esac

**跳出循环**

**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

**esac**

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

shell文件包含

和其他语言一样，Shell 也可以包含外部脚本。这样可以很方便的封装一些公用的代码作为一个独立的文件。

Shell 文件包含的语法格式如下：

. filename # 注意点号(.)和文件名中间有一空格

或

source filename

**shell指令总结：**

* + 在shell中，如果变量被声明成整数，把一个结果不是整数的表达式赋值给它时就会变成0
  + Bash并不内置对浮点数的支持，所以会得到错误的信息，变量会变成0
  + 单引号（''）双引号（""）反引号（``）的区别

单引号：单引号括起来的字符都作为普通字符出现

双引号：除"$", "\", "`"几个字符仍是特殊字符并保留其特殊功能外，其余字符仍作为普通字符对待.

反引号：反引号括起来的字符串 被shell解释成命令

* + 全局变量

在脚本中定义的变量全是全局变量，起作用域为从被定义的地方开始一直到shell脚本结束或者被显示地删除.在函数内部定义的变量默认情况下也是全局变量，其作用域为从函数被调用时执行变量定义的地方开始，一直到shell脚本结束或者被显示地删除为止.

* + 局部变量

在shell语言中，可以在函数内部通过local关键字定义局部变量，另外函数的参数也是局部变量。

* + 系统变量

shell语言的系统变量主要在对参数判断和命令返回值判断时使用，包括脚本和函数的参数，以及脚本和函数的返回值。

* + 系统常用变量

$n: n是一个整数（1-9），表示参数的位置，例如$1表示第1个参数，$2表示第2个参数等。

$#：命令行参数的个数

$0：当前shell脚本的名称

$?：上一次命令的返回状态或者函数的返回值命令的返回状态比如ls /temp/log若是没有log那么$?为2，若是有值为0。

$\*: 以“参数1 参数2 。。。。”构成字符串并返回。

$@：以“参数1”“参数2” 。。。的形式返回每个参数。

$$：返回本程序的进程ID(PID)。

* + $(())、$()、${}的区别

在bash中$()与反引号都是用来作命令替换的。建议使用$()

一般情况下，$var与${var}是没有区别的，但是用${}会比较精确的界定变量名称的范围。

$(())用来做算数运算

* + 反斜杠

用于将一些shell中的特殊字符转换为普通字符，例如"$"或者"'"等。

* + 中间的等号"="称为赋值符号，赋值符号的左右两边不能直接跟空格，否则shell会将其视为命令。
  + 一个没有初始化的shell变量被认为是一个空字符串。
  + 清除变量后，其所代表的值也会消失，输出空。
  + 测试命令（test 和 [）test string（判断指定的字符串是否为空）只能使用test命令来测试是否为空串，而不能使用方括号的方式来测试。
  + 在if语句中，如果使用条件测试，可以使用test命令，也可以使用方括号[]
  + 还有一个特殊的命令，称为空命令，其表示方法是一个冒号":",该命令不做任何事情，但是它的退出状态永远是0
  + 可以使用&&来替代if语句

eg: test "$(whoami)" != "root" && (echo you are using a no-privileged)

exit 1

首先用test条件测试，若是条件为真，那么就会执行&&后面的语句

通配符类型

\*：匹配任意长度的任意字符

？：匹配任意单个字符

[]：匹配指定范围内的任意单个字符

[^]：匹配指定范围之外的任意单个字符

[:space:]：空白字符

[:punct:]：标点符号

[:lower:]：小写字母

[:upper:]：大写字母

[:alpha:]：大小写字母

[:digit:]：数字

[:alnum:]：数字和大小写字母

* + 在shell中可以使用两种语法来表示不同的进制，首先是增加前缀，例如以0开头的文字表示八进制，以0x开头的数字表示16进制；第二种使用"#"，例如2#1000表示二进制，8#42表示8进制
  + 当测试使用"-n"或者"-z"这种方式判断变量是否为空时，"[]"与"[[]]"有区别使用"[]"时需要在变量的外侧加上双引号,与test命令的用法完全相同，使用"[[]]"时则不用。
  + 使用"[[]]"是不能使用-a -o的，只能使用"&&"和"||"来链接各个条件
  + ((exp))结构扩展并计算一个算术表达式的值如果表达式的结果为0，那么返回的退出状态码为1，或者 是"假"，而一个非零值的表达式所返回的退出状态码将为 0，或者是"true"。若是逻辑判断，表达式exp为真则为1,假则为0。
  + continue和break后面接数字表示退出几层循环
  + 我们定义了一个别名ls，而shell中已经存在一个名称为ls的命令。但是在别名 和命令同时存在的情况下，shell会优先使用用户定义的别名
  + shell函数的参数以空格隔开，如果某个参数中含有空格，则应该使用引号将其 引用起来
  + 位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1，原来的 $5现在变成$2等等，原来的$1、$2、$3丢弃，$0不移动。不带参数的shift命令 相当于shift 1
  + 函数库文件的调用

. filename

库文件可以是相对路径，也可以是绝对路径，原点命令和库文件名之间有一个空格

shell函数

[ function ] funname [()]

{

action;

[return int;]

}

1、可以带function fun() 定义，也可以直接fun() 定义,不带任何参数。

2、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）

例：

demoFun(){

echo "这是我的第一个 shell 函数!"

}

echo "-----函数开始执行-----"

demoFun

echo "

例：

funWithReturn(){

echo "这个函数会对输入的两个数字进行相加运算..."

echo "输入第一个数字: "

read aNum

echo "输入第二个数字: "

read anotherNum

echo "两个数字分别为 $aNum 和 $anotherNum !"

return $(($aNum+$anotherNum))

}

funWithReturn

echo "输入的两个数字之和为 $? !"

函数返回值在调用该函数后通过 $? 来获得。

注意：所有函数在使用前必须定义。这意味着必须将函数放在脚本开始部分，直至shell解释器首次发现它时，才可以使用。调用函数仅使用其函数名即可