# 1、Shell 环境

1、Shell 编程跟 java、php 编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

2、Bash 也是大多数Linux 系统默认的 Shell。

在一般情况下，人们并不区分 Bourne Shell 和 Bourne Again Shell，所以，像 **#!/bin/sh**，它同样也可以改为 **#!/bin/bash**。

## #! 告诉系统其后路径所指定的程序即是解释此脚本文件的 Shell 程序。

# 2、第一个shell脚本

## 1、新建一个test.sh文件

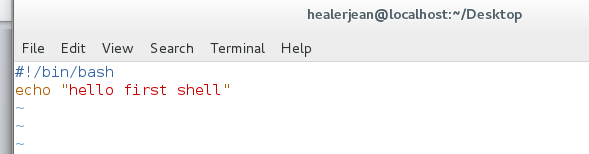
### vi testShell.sh



## 2、输入内容

### #!/bin/bash

### echo "hello first shell"



## 3、给用户赋予文件的执行权限

### [root@ Desktop]# chmod u+x testShell.sh

## 4、执行shell

### ./testShell.sh

## 4.2第二种执行方式

### /bin/bash testShell.sh



## 5、观察看到echo中的输出



# 3、参数传递

## 1、建立shellTags.sh文件

### vi shellTags.sh



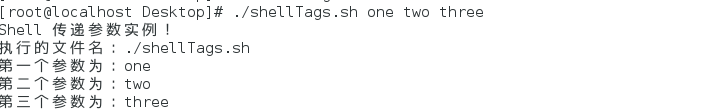
## 2、编辑文件中要被输入的参数

|  |
| --- |
| #!/bin/bash  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"; |

## 3、测试输入参数，执行

### []# ./shellTags.sh one two three

## 4、结果



## 5、解释：

### **$0** 为执行的文件名：

### $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……

# 4、变量

|  |
| --- |
| your\_name="Healerjean";  echo $your\_name; |

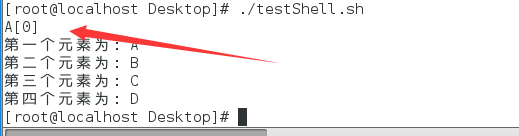


# 5、数组

## 1、定义数组

|  |
| --- |
| my\_array=(A B "C" D)echo $my\_array[0] //错误的指令echo "第一个元素为: ${my\_array[0]}"echo "第二个元素为: ${my\_array[1]}"echo "第三个元素为: ${my\_array[2]}"echo "第四个元素为: ${my\_array[3]}" |

## 2、执行



## 3、解释 一定要商大括号{}

# 6、变量的数值计算

## 1、解释：

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 是一款表达式计算工具，使用它能完成表达式的求值操作。

例如，两个数相加(**注意使用的是反引号 ` 而不是单引号 '**)：

## 2、代码，注意 单引号，和乘法有所不一样

|  |
| --- |
| a=10  b=20  val=`expr $a + $b` 加法+  echo "a + b : $val"  val=`expr $a - $b` 减法-  echo "a - b : $val"  val=`expr $a \\* $b` 乘法\\*  echo "a \* b : $val"  val=`expr $b / $a` 除法 、  echo "b / a : $val"  val=`expr $b % $a` 取余%  echo "b % a : $val"  if [ $a == $b ]  then  echo "a 等于 b"  fi  if [ $a != $b ]  then  echo "a 不等于 b"  fi |

# 7、函数

## 1、控制台输入参数

### 1、read为等待控制台输入参数

### 2、函数返回值在调用该函数后通过 $? 来获得。

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 !"

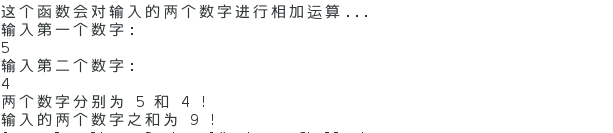
return $(($aNum+$anotherNum))

}

funWithReturn

echo "输入的两个数字之和为 $? !"

### 3、运行



## 2、内部传参

### 1、函数方法

|  |
| --- |
| funWithParam(){  echo "第一个参数为 $1 !"  echo "第二个参数为 $2 !"  echo "第十个参数为 $10 !"  echo "第十个参数为 ${10} !"  echo "第十一个参数为 ${11} !"  echo "参数总数有 $# 个!"  echo "作为一个字符串输出所有参数 $\* !"  }  funWithParam 1 2 3 4 5 6 7 8 9 34 73 |

### 2、运行结果

第一个参数为 1 !

第二个参数为 2 !

第十个参数为 10 !

第十个参数为 34 !

第十一个参数为 73 !

参数总数有 11 个!

作为一个字符串输出所有参数 1 2 3 4 5 6 7 8 9 34 73 !

### 3、解释:

$10 不能获取第十个参数，获取第十个参数需要${10}。当n>=10时，需要使用${n}来获取参数。

另外，还有几个特殊字符用来处理参数：

$10 可以用来在控制台进行输入时候可以使用，参考3，参数传递

# 8、字符表示

|  |  |
| --- | --- |
| **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](http://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。 |

## 举例

echo "数组的元素为: ${my\_array[\*]}"

数组的元素为: A B C D

# 9、>0 ,>1,>3

**0 是一个文件描述符，表示标准输入(stdin)**  
**1 是一个文件描述符，表示标准输出(stdout)**

**2 是一个文件描述符，表示标准错误(stderr)**

### [root@redhat box]# ls a.txt b.txt 1>file.out 2>file.err

### 执行后,没有任何返回值. 原因是, 返回值都重定向到相应的文件中了,而不再前端显示

### [root@redhat box]# cat file.out

#### a.txt

### [root@redhat box]# cat file.err

#### ls: b.txt: No such file or directory

## 2、一般来说, "1>" 通常可以省略成 ">".

### 即可以把如上命令写成: ls a.txt b.txt >file.out 2>file.err

## 3.、

## 1>&2 正确返回值传递给2输出通道 &2表示2输出通道  如果此处错写成 1>2, 就表示把1输出重定向到文件2中.  2>&1 错误返回值传递给1输出通道, 同样&1表示1输出通道.  举个例子.  [root@redhat box]# ls a.txt b.txt 1>file.out 2>&1  [root@redhat box]# cat file.out  ls: b.txt: No such file or directory  a.txt