# Linux菜鸟笔记

# Linux Shell菜鸟笔记

## [Shell 传递参数](http://www.runoob.com/linux/linux-shell-passing-arguments.html)

./test.sh param1 param2 param3

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，表示获取第n（从1开始）个参数。

$0获取的是执行的脚本文件名称。

$# 获取传递到脚本的参数个数。

**$\* 与 $@ 区别？**

$$ 与 $!

$-

$?

|  |
| --- |
| 在为shell脚本传递的参数中如果包含空格，应该使用单引号或者双引号将该参数括起来，以便于脚本将这个参数作为整体来接收。  ./test.sh “a b c” def >> $1就是”a b c”，$2就是def |
| 在使用参数前，先校验参数是否存在，以减少错误发生：  if [ -n "$1" ]; then  echo "包含第一个参数"  else  echo "没有包含第一参数"  fi |

## [Shell数组](http://www.runoob.com/linux/linux-shell-array.html)

Bash Shell 只支持一维数组（不支持多维数组），初始化时不需要定义数组大小。

数组元素的下标由0开始。

|  |
| --- |
| # 定义数组会用到 **小括号 与 空格**  array\_name=(value0 value1 value2 ... valueN)  # 读取数组元素值 ： **${数组名[下标]}**  valuen=${array\_name[index]} |

|  |
| --- |
| # 我们还可以通过单独定义数组的各个分量来定义数组  # (可以不使用连续的下标，而且下标的范围没有限制)：  arr[0]=art  arr[3]=dash  arr[5]=ether  # 读取数组元素值：${数组名[下标]}  echo ${arr[0]} # art  echo ${arr[2]} # 无内容  echo ${arr[3]} # dash  # 使用@或\*可以获取数组中的所有元素  echo ${arr[@]} # art dash ether  echo ${arr[\*]} # art dash ether  # 相应的取得数组中所有元素的个数  echo ${#arr[@]} # 3  echo ${#arr[\*]} # 3  # 取得数组指定位置元素的长度  echo ${#arr[3]} # 4 |

遍历数组：

|  |
| --- |
| my\_arry=(a b c,d abc)  echo "-------FOR循环遍历输出数组--------"  for i in ${my\_arry[@]}; do  echo $i  done  echo "-------WHILE循环输出---------"  j=0  while [ $j -lt ${#my\_arry[@]} ] # 注意空格  do  echo ${my\_arry[$j]}  let j++  done |
| A=elem1  B=elem2  my\_array=($A $B C D) # 数组元素的值可以引用变量  for i in ${my\_array[@]}; do  echo $i  done |

## [Shell运算符](http://www.runoob.com/linux/linux-shell-basic-operators.html)

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 用于完成表达式的求值操作。

|  |
| --- |
| # 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  # 完整的表达式要被 **反引号`** 所包含【不是单引号'】  val=`expr 2 + 2`  echo "两数之和为 : $val" # 两数之和为 : 4  # 算术运算符 +, -, \\*, /, % 【注意乘号前需要\】  a=10  b=20  echo `expr $a + $b` # 30  #if...then...fi ==与!=  #注意：条件表达式要放在方括号之间，并且要有空格，例如: [$a==$b] 是错误的，必须写成 [ $a == $b ]  if [ $a == $b ]  then  echo "==, a 等于 b"  fi  if [ $a != $b ]  then  echo "!=, a 不等于 b"  fi |

|  |
| --- |
| # 布尔运算符 -a 与 -o 与 ! 优先级为："!"最高，"-a"次之，"-o"最低  echo -e "-a测试：$a 小于 100 且 $b 大于 15 : \c" # -e开启转义，\c表示不换行  if [ $a -lt 100 -a $b -gt 15 ]  then echo " 返回 true"  else echo " 返回 false"  fi  # [ ! $a -lt 100 -a ! $b -gt 15 ]  # 逻辑运算符 && 与 ||  # 注意这里用的 [[ ... ]]  echo -e "&&测试：$a 小于 100 且 $b 大于 100 : \c"  if [[ $a -lt 100 && $b -gt 100 ]]  then echo "返回 true"  else echo "返回 false"  fi |

|  |  |
| --- | --- |
| 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。  -eq, -ne, -gt, -lt, -ge, -le |
| 字符串测试 | 1. 两个字符串是否相等 =与!= 2. 检测字符串长度是否为0 -z与-n 3. 检测字符串是否为空 |
| 文件测试 | 1. 检测文件是否是块设备文件 2. 检测文件是否是字符设备文件 3. 检测文件是否是目录 4. 检测文件是否是普通文件（既不是目录，也不是设备文件） 5. 检测文件是否设置了 SGID 位 6. 检测文件是否设置了粘着位(Sticky Bit) 7. 检测文件是否设置了 SUID 位 8. 检测文件是否是有名管道 9. 检测文件是否可读 10. 检测文件是否可写 11. 检测文件是否可执行 12. 检测文件是否为空（文件大小是否大于0） 13. 检测文件（包括目录）是否存在 |

笔记部分没有看…

## [Echo命令](http://www.runoob.com/linux/linux-shell-echo.html)

笔记部分也有用

## [printf 命令](http://www.runoob.com/linux/linux-shell-printf.html)

## [test命令](http://www.runoob.com/linux/linux-shell-test.html)

## [Shell 流程控制](http://www.runoob.com/linux/linux-shell-process-control.html)

### if

If … then … [elif … then …] [else …] fi

注意：sh的流程控制不可为空，如果else分支没有语句执行，就不要写这个else。

有时候需要将if语句写成一行，可查原文看如何解决。

if else语句经常与test命令结合使用，如下所示：

num1=$[2\*3]

num2=$[1+5]

if test $[num1] -eq $[num2]

then

echo '两个数字相等!'

else

echo '两个数字不相等!'

fi

### for

in列表可以包含替换、字符串和文件名。

in列表是可选的，如果不用它，for循环使用命令行的位置参数。

### While, [let](http://www.runoob.com/linux/linux-comm-let.html)

While可用于从输入文件中读取数据。

while循环可用于读取键盘信息。

### 无限循环

### Until

### Case

### 跳出循环break、continue

## [Shell函数](http://www.runoob.com/linux/linux-shell-func.html)

## [Shell 输入/输出重定向](http://www.runoob.com/linux/linux-shell-io-redirections.html)

## [Shell 文件包含](http://www.runoob.com/linux/linux-shell-include-file.html)

可以封装一些公用代码到一个独立的脚本文件中，在其它脚本文件中如果需要用到公用代码，就包含一下该文件。

***注：****被包含的文件 test1.sh 不需要可执行权限。*