# Shell脚本知识点

# 基础

## 1.1 文件格式

文件后缀为sh，并且文件第一行为#!/bin/bash

## 1.2 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**

使用一个定义过的变量，只要在变量名前面加美元符号即可，

your\_name="qinjx"

echo $your\_name

echo ${your\_name}

推荐给所有变量加上花括号，这是个好的编程习惯。

### **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。

readonly name;

### **删除变量**

使用 unset 命令可以删除变量。

unset name;

变量被删除后不能再次使用。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

## 1.3 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；
* 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
* 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。

## 1.4 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。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[@]}

### **获取数组中的所有元素**

使用@ 或 \* 可以获取数组中的所有元素

### **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]}

### 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，

## 1.5 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**

以下实例我们向脚本传递三个参数，并分别输出，其中 **$0** 为执行的文件名：

#!/bin/bash# author:菜鸟教程# url:www.runoob.com

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";

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" \t "/home/zgj/文档\\x/_blank)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。 |

$\* 与 $@ 区别：

* 相同点：都是引用所有参数。
* 不同点：只有在双引号中体现出来。假设在脚本运行时写了三个参数 1、2、3，，则 " \* " 等价于 "1 2 3"（传递了一个参数），而 "@" 等价于 "1" "2" "3"（传递了三个参数）。

## 1.6 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。

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 '**)：

*#!/bin/bash*  
  
val=**`expr** 2 + 2**`**  
**echo** "两数之和为 : $val"

执行脚本，输出结果如下所示：

两数之和为 : 4

两点注意：

* 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。
* 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

下表列出了常用的算术运算符，假定变量 a 为 10，变量 b 为 20：

|  |  |  |
| --- | --- | --- |
| **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 ]**。

***注意：***

* 乘号(\*)前边必须加反斜杠(\)才能实现乘法运算；
* 在 MAC 中 shell 的 expr 语法是：**$((表达式))**，此处表达式中的 "\*" 不需要转义符号 "\" 。

### 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不相等，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。 |
| $ | 检测字符串是否不为空，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：

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

### Linux let 命令

et 命令是 BASH 中用于计算的工具，用于执行一个或多个表达式，变量计算中不需要加上 $ 来表示变量。如果表达式中包含了空格或其他特殊字符，则必须引起来。

以下实例计算 a 和 b 两个表达式，并输出结果：

#!/bin/bash

let a=5+4let b=9-3

echo $a $b

以上实例执行结果为：

9 6

## 1.7 Shell echo命令

### **1.显示普通字符串:**

echo "It is a test"

这里的双引号完全可以省略，以下命令与上面实例效果一致：

echo It is a test

### **2.显示转义字符**

echo "\"It is a test\""

结果将是:

"It is a test"

同样，双引号也可以省略

### **3.显示变量**

read 命令从标准输入中读取一行,并把输入行的每个字段的值指定给 shell 变量

#!/bin/sh

read name

echo "$name It is a test"

以上代码保存为 test.sh，name 接收标准输入的变量，结果将是:

[root@www ~]# sh test.sh

OK #标准输入

OK It is a test #输出

### **4.显示换行**

echo -e "OK! \n" # -e 开启转义

echo "It is a test"

输出结果：

OK!

It is a test

### **5.显示不换行**

#!/bin/sh

echo -e "OK! \c" # -e 开启转义 \c 不换行

echo "It is a test"

输出结果：

OK! It is a test

### **6.显示结果定向至文件**

echo "It is a test" > myfile

### **7.原样输出字符串，不进行转义或取变量(用单引号)**

echo '$name\"'

输出结果：

$name\"

### **8.显示命令执行结果**

echo `date`

**注意：** 这里使用的是反引号 **`**, 而不是单引号 **'**。

结果将显示当前日期

Thu Jul 24 10:08:46 CST 2014

## 1.8 Shell printf 命令

使用 printf 的脚本比使用 echo 移植性好。

printf 使用引用文本或空格分隔的参数，外面可以在 printf 中使用格式化字符串，还可以制定字符串的宽度、左右对齐方式等。默认 printf 不会像 echo 自动添加换行符，我们可以手动添加 \n。

printf 命令的语法：

printf format-string [arguments...]

**参数说明：**

* **format-string:** 为格式控制字符串
* **arguments:** 为参数列表。

接下来,我来用一个脚本来体现printf的强大功能：

#!/bin/bash# author:菜鸟教程# url:www.runoob.com

printf "%-10s %-8s %-4s\n" 姓名 性别 体重kg

printf "%-10s %-8s %-4.2f\n" 郭靖 男 66.1234

printf "%-10s %-8s %-4.2f\n" 杨过 男 48.6543

printf "%-10s %-8s %-4.2f\n" 郭芙 女 47.9876

执行脚本，输出结果如下所示：

姓名 性别 体重kg

郭靖 男 66.12

杨过 男 48.65

郭芙 女 47.99

%s %c %d %f都是格式替代符

%-10s 指一个宽度为10个字符（-表示左对齐，没有则表示右对齐），任何字符都会被显示在10个字符宽的字符内，如果不足则自动以空格填充，超过也会将内容全部显示出来。

%-4.2f 指格式化为小数，其中.2指保留2位小数。

### printf的转义序列

|  |  |
| --- | --- |
| **序列** | **说明** |
| \a | 警告字符，通常为ASCII的BEL字符 |
| \b | 后退 |
| \c | 抑制（不显示）输出结果中任何结尾的换行字符（只在%b格式指示符控制下的参数字符串中有效），而且，任何留在参数里的字符、任何接下来的参数以及任何留在格式字符串中的字符，都被忽略 |
| \f | 换页（formfeed） |
| \n | 换行 |
| \r | 回车（Carriage return） |
| \t | 水平制表符 |
| \v | 垂直制表符 |
| \\ | 一个字面上的反斜杠字符 |
| \ddd | 表示1到3位数八进制值的字符。仅在格式字符串中有效 |
| \0ddd | 表示1到3位的八进制值字符 |

## 1.9 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。

### 数值测试

|  |  |
| --- | --- |
| **参数** | **说明** |
| -eq | 等于则为真 |
| -ne | 不等于则为真 |
| -gt | 大于则为真 |
| -ge | 大于等于则为真 |
| -lt | 小于则为真 |
| -le | 小于等于则为真 |

实例演示：

num1=100

num2=100if test $[num1] -eq $[num2]then

echo '两个数相等！'else

echo '两个数不相等！'fi

输出结果：

两个数相等！

### 代码中的 [] 执行基本的算数运算，如：

#!/bin/bash

a=5

b=6

result=$[a+b] # 注意等号两边不能有空格

echo "result 为： $result"

结果为:

result 为： 11

### 字符串测试

|  |  |
| --- | --- |
| **参数** | **说明** |
| = | 等于则为真 |
| != | 不相等则为真 |
| -z 字符串 | 字符串的长度为零则为真 |
| -n 字符串 | 字符串的长度不为零则为真 |

实例演示：

num1="ru1noob"

num2="runoob"if test $num1 = $num2then

echo '两个字符串相等!'else

echo '两个字符串不相等!'fi

输出结果：

两个字符串不相等!

### 文件测试

|  |  |
| --- | --- |
| **参数** | **说明** |
| -e 文件名 | 如果文件存在则为真 |
| -r 文件名 | 如果文件存在且可读则为真 |
| -w 文件名 | 如果文件存在且可写则为真 |
| -x 文件名 | 如果文件存在且可执行则为真 |
| -s 文件名 | 如果文件存在且至少有一个字符则为真 |
| -d 文件名 | 如果文件存在且为目录则为真 |
| -f 文件名 | 如果文件存在且为普通文件则为真 |
| -c 文件名 | 如果文件存在且为字符型特殊文件则为真 |
| -b 文件名 | 如果文件存在且为块特殊文件则为真 |

实例演示：

cd /binif test -e ./bashthen

echo '文件已存在!'else

echo '文件不存在!'fi

输出结果：

文件已存在!

另外，Shell还提供了与( -a )、或( -o )、非( ! )三个逻辑操作符用于将测试条件连接起来，其优先级为："!"最高，"-a"次之，"-o"最低。例如：

cd /binif test -e ./notFile -o -e ./bashthen

echo '至少有一个文件存在!'else

echo '两个文件都不存在'fi

输出结果：

至少有一个文件存在!

## 1.10 Shell 流程控制

### if

if 语句语法格式：

if condition

then

command1

command2

...

commandN

fi

### **if else**

if else 语法格式：

if condition

then

command1

command2

...

commandN

else

Command

fi

### **if else-if else**

if else-if else 语法格式：

if condition1

then

command1

elif

condition2

then

command2

else

commandN

fi

### for 循环

for循环一般格式为：

for var in item1 item2 ... itemN

do

command1

command2

...

commandN

done

### while 语句

while循环用于不断执行一系列命令，也用于从输入文件中读取数据；命令通常为测试条件。其格式为：

while condition

do

Command

done

### **无限循环**

无限循环语法格式：

while :do

commanddone

或者

while truedo

commanddone

或者

for (( ; ; ))

### until 循环

until 循环执行一系列命令直至条件为 true 时停止。

until 循环与 while 循环在处理方式上刚好相反。

一般 while 循环优于 until 循环，但在某些时候—也只是极少数情况下，until 循环更加有用。

until 语法格式:

until conditiondo

commanddone

condition 一般为条件表达式，如果返回值为 false，则继续执行循环体内的语句，否则跳出循环。

### 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。

### 跳出循环

在循环过程中，有时候需要在未达到循环结束条件时强制跳出循环，Shell使用两个命令来实现该功能：break和continue。

## 1.11 Shell 函数

说明：

* 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

下面的例子定义了一个函数并进行调用：

#!/bin/bash# author:菜鸟教程# url:www.runoob.com

demoFun(){

echo "这是我的第一个 shell 函数!"

}

echo "-----函数开始执行-----"

demoFun

echo "-----函数执行完毕-----"

输出结果：

-----函数开始执行-----

这是我的第一个 shell 函数!

-----函数执行完毕-----

下面定义一个带有return语句的函数：

#!/bin/bash# author:菜鸟教程# url:www.runoob.com

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 $? !"

输出类似下面：

这个函数会对输入的两个数字进行相加运算...

输入第一个数字: 1

输入第二个数字: 2

两个数字分别为 1 和 2 !

输入的两个数字之和为 3 !

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。

### 函数参数

在Shell中，调用函数时可以向其传递参数。在函数体内部，通过 $n 的形式来获取参数的值，例如，$1表示第一个参数，$2表示第二个参数...

带参数的函数示例：

#!/bin/bash# author:菜鸟教程# url:www.runoob.com

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

输出结果：

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 !

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

## 1.12 Shell 输入/输出重定向

重定向命令列表如下：

|  |  |
| --- | --- |
| **命令** | **说明** |
| command > file | 将输出重定向到 file。 |
| command < file | 将输入重定向到 file。 |
| command >> file | 将输出以追加的方式重定向到 file。 |
| n > file | 将文件描述符为 n 的文件重定向到 file。 |
| n >> file | 将文件描述符为 n 的文件以追加的方式重定向到 file。 |
| n >& m | 将输出文件 m 和 n 合并。 |
| n <& m | 将输入文件 m 和 n 合并。 |
| << tag | 将开始标记 tag 和结束标记 tag 之间的内容作为输入。 |

### 输出重定向

重定向一般通过在命令间插入特定的符号来实现。特别的，这些符号的语法如下所示:

command1 > file1

上面这个命令执行command1然后将输出的内容存入file1。

注意任何file1内的已经存在的内容将被新内容替代。如果要将新内容添加在文件末尾，请使用>>操作符。

### 输入重定向

和输出重定向一样，Unix 命令也可以从文件获取输入，语法为：

command1 < file1

这样，本来需要从键盘获取输入的命令会转移到文件读取内容。

注意：输出重定向是大于号(>)，输入重定向是小于号(<)。

### 重定向深入讲解

一般情况下，每个 Unix/Linux 命令运行时都会打开三个文件：

* 标准输入文件(stdin)：stdin的文件描述符为0，Unix程序默认从stdin读取数据。
* 标准输出文件(stdout)：stdout 的文件描述符为1，Unix程序默认向stdout输出数据。
* 标准错误文件(stderr)：stderr的文件描述符为2，Unix程序会向stderr流中写入错误信息。

默认情况下，command > file 将 stdout 重定向到 file，command < file 将stdin 重定向到 file。

如果希望 stderr 重定向到 file，可以这样写：

$ command 2 > file

如果希望 stderr 追加到 file 文件末尾，可以这样写：

$ command 2 >> file

**2** 表示标准错误文件(stderr)。

如果希望将 stdout 和 stderr 合并后重定向到 file，可以这样写：

$ command > file 2>&1

或者

$ command >> file 2>&1

如果希望对 stdin 和 stdout 都重定向，可以这样写：

$ command < file1 >file2

command 命令将 stdin 重定向到 file1，将 stdout 重定向到 file2。

### EOF

EOF 是 Shell 中的一种特殊的重定向方式，用来将输入重定向到一个交互式 Shell 脚本或程序。

它的基本的形式如下：

command << delimiter

document

delimiter

它的作用是将两个 delimiter 之间的内容(document) 作为输入传递给 command。

*注意：*

* 结尾的delimiter 一定要顶格写，前面不能有任何字符，后面也不能有任何字符，包括空格和 tab 缩进。
* 开始的delimiter前后的空格会被忽略掉。

### /dev/null 文件

如果希望执行某个命令，但又不希望在屏幕上显示输出结果，那么可以将输出重定向到 /dev/null：

$ command > /dev/null

/dev/null 是一个特殊的文件，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。但是 /dev/null 文件非常有用，将命令的输出重定向到它，会起到"禁止输出"的效果。

如果希望屏蔽 stdout 和 stderr，可以这样写：

$ command > /dev/null 2>&1

***注意：****0 是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）。*

## 1.13 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

### **实例**

创建两个 shell 脚本文件。

test1.sh 代码如下：

#!/bin/bash

# author:菜鸟教程

# url:www.runoob.com

url="http://www.runoob.com"

test2.sh 代码如下：

#!/bin/bash# author:菜鸟教程

# url:www.runoob.com

#使用 . 号来引用test1.sh 文件

. ./test1.sh

# 或者使用以下包含文件代码

# source ./test1.sh

echo "菜鸟教程官网地址：$url"

接下来，我们为 test2.sh 添加可执行权限并执行：

$ chmod +x test2.sh

$ ./test2.sh

菜鸟教程官网地址：http://www.runoob.com

***注：****被包含的文件 test1.sh 不需要可执行权限。*