# 一、Shell变量

for file in `ls /etc`

for file in $(ls /etc)

## 1.使用变量

your\_name="qinjx"

echo $your\_name

echo ${your\_name}

## 2.只读变量

#!/bin/bash

myUrl="http://www.google.com"

readonly myUrl

## 3.删除变量

unset myUrl

unset命令不能删除只读变量

# 二、字符串

## 1.获取字符串长度

string=”abcd”

echo ${#string} #输出4

## 2.提取子字符串

string=”runoob is a great site”

echo ${string:1:4} #输出 unoo

## 3.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 ,`反引号

# 三、Shell 数组

## 1.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。

数组名=(值1 值2 ... 值n)

array\_name=(value0 value1 value2 value3)

## 2.读取数组

${数组名[下标]}

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[@]}

## 3.获取数组的长度

length=${#array\_name[@]} #取得数组元素的个数

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

lengthn=${#array\_name[n]} #取得数组单个元素的长度

# 四、Shell传递参数

|  |  |
| --- | --- |
| 参数处理 | 说明 |
| $n | $0 为当前执行的文件名  n代表一个数字, 1执行脚本的第一个参数,  当n>=10时，需要使用${n}来获取参数。 |
| $# | 传递到脚本的参数个数(不包括程序本身) |
| $\* | 以一个单字符串显示所有向脚本传递的参数。(不包括程序本身) 如"$\*"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。 |
| $UID | 当前用户的ID |
| $PWD | 当前所在的目录 |

# 五、Shell基本运算符

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。

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"

## 1.算术运算符

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。 |

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 "\" 。

## 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。

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。 |

## 3.布尔运算符

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。 |

## 4.逻辑运算符

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 |

## 5.字符串运算符

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。 |

## 6.文件测试运算符

|  |  |  |
| --- | --- | --- |
| 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。 |

-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。

# 六、Shell echo命令

## 1.显示普通字符串

echo “It is a test”

echo It is a test

## 2.显示转义字符

echo “\”It is a test\””

结果 “It is a test”

双引号可以省略

## 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 激活转义

echo “It is a test”

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

# 七、Shell printf命令

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。 %s %c %d %f都是格式替代符

## 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 |

# 八、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。

## 1.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 |

## 2.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 |

## 3.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 |

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

# 九、Shell流程控制

## 1.If else

### 1)if

if condition ;then

commandN

fi

例：if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi

### 2)if else

if condition ;then

commandN

else

command

fi

### 3)if else-if else

if condition1 ;then

command1

elif condition2

then

command2

else

commandN

fi

## 2.for循环

for var in item1 item2 ... itemN

do

command1

command2

...

commandN

done

例：

for loop in 1 2 3 4 5

do

echo "The value is: $loop"

done

## 3.while语句

while condition

do

command

done

例：

#!/bin/bash

int=1

while(( $int<=5 ))

do

echo $int

let "int++"

done

## 4.until循环

until 循环执行一系列命令直至条件为 true 时停止。

until 语法格式:

until condition

do

command

done

## 5.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。

break命令允许跳出所有循环（终止执行后面的所有循环）。

continue命令它不会跳出所有循环，仅仅跳出当前循环。

# 十、Shell函数

[ function ] funname [()]

{

action;

[return int;]

}

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)

## 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](#_Shell传递参数) 的形式来获取参数的值，

# 十一、Shell输出/输出重定向

|  |  |
| --- | --- |
| **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。 |

注意：文件描述符 0 通常是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）。

## 1.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。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。

## 2.输入重定向

语法为：

command1 < file1

需要从键盘获取输入的命令会转移到文件读取内容。

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。

## 3.Here Document

Here Document 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。

# 十二、Shell文件包含

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。*