Shell 基础

2018年5月15日　　18:01

- 运行shell脚本有两种方式：

1、作为可执行程序

```
okay@okay-virtual-machine:~$ cd /home/okay/Decument
okay@okay-virtual-machine:~/Decument$ chmod +x ./test.sh
okay@okay-virtual-machine:~/Decument$ ./test.sh
Hello World !
```

2、作为解释器参数

```
okay@okay-virtual-machine:~$ cd /home/okay/Decument
okay@okay-virtual-machine:~/Decument$ /bin/sh test.sh
Hello World !
```

- shell变量 >>> 使用变量（在变量名前加$）

只读变量：readonly myUrl

删除变量：unset variable_name

变量类型：

1）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

3）shell变量：由shell程序设置的特殊变量（一部分是环境变量，一部分是局部变量）

拼接字符串 （echo $greeting $greeting_1）---获取字符串长度（echo ${#myUrl} ） ---提取子字符串（echo ${myUrl:1:4}）---查找子字符串（echo `expr index "$myUrl" is `）

```
okay@okay-virtual-machine:~/Decument$ /bin/sh Variable.sh
Variable.sh: 6: Variable.sh: Bad substitution
okay@okay-virtual-machine:~/Decument$ bash Variable.sh
unoo
okay@okay-virtual-machine:~/Decument$ bash Variable.sh
runoob is a great company
25
unoo
8
```

【其中当使用sh截取字符串时会出现问题，可以使用bash】

shell数组：

定义：数组名=(值1 值2 ... 值n)

使用@获取数组中的所有元素：echo ${array_name[@]}

获取数组的长度：length = ${#array_name[*//@]}

Shell注释： # （多行注释可以定义成一个函数）

- Shell传递参数

在执行Shell脚本时，向脚本传递参数，获取参数的格式：$n

（其中$0为执行的文件名）

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数组

array_name=(value1 ... valuen) //数组用括号表示，元素用“空格”分隔开

读取数组元素值：${array_name[index]}

获取数组长度：${#array_name[*//@]}

- Shell基本运算符

算数运算符、关系运算符、布尔运算符、字符串运算符、文件测试运算符

```
okay@okay-virtual-machine:~/Document$ /bin/sh test1.sh
两数之和为: 2+2
okay@okay-virtual-machine:~/Document$ /bin/sh test1.sh
两数之和为: 4
```

val=`expr 2 + 2` （要有空格，不然结果是2+2）

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！！

- 乘号(*)前边必须加反斜杠(\)才能实现乘法运算；

- if...then...fi 是条件语句

关系运算符：

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：

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：

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：

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 $a] 返回 true。
str	检测字符串是否为空，不为空返回 true。	[$a] 返回 true。

文件测试运算符：（用于检测Unix文件的各种属性）

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。

- shell echo命令（用于字符串的输出）

read：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

-e：开启转义

' '：原样输出字符串

- shell printf命令

printf命令：printf format-string（格式控制字符串） [arguments...]（参数列表）

%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）

%-4.2f指格式化为小数，其中.2指保留2位小数

转义字符：

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

- shell 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

代码中的 [ ]执行基本的算数运算： result=$[a+b] # 注意等号两边不能有空格

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

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

- shell流程控制（sh的流程控制不可为空）

if下面就加一个then

if then fi // if then else fi // if then elif then else fi

for循环：

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

while语句：

用于不断执行一系列命令（测试条件），也用于从输入文件中读取数据；

```
while condition
do
    command
done
```

let：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

无限循环： while： //while true // for（（；；））

until循环执行一系列命令直至条件为true时停止（和while相反）

```
until condition
do
    command
done
```

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，匹配成功，执行相应命令

```
case 值 in
模式1)
    command1
    command2
    ...
    commandN
    ;;
模式2)
    command1
    command2
    ...
    commandN
    ;;
esac
```

跳出循环：

break：允许跳出所有循环（终止执行后面的所有循环）

continue：不会跳出所有循环，仅仅跳出当前循环

esac：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函数

```
[ function ] funname [()]

{

    action;

    [return int;]

}
```

函数返回值在调用该函数后通过$?来获得

在函数体内部，通过$n的形式来获取参数的值，如$1表示第一个参数.....

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。

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
$$	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。
$-	显示Shell使用的当前选项，与set命令功能相同。
$?	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输入/输出重定向

大多数UNIX系统命令从你的终端接收输入并将所产生的输出发送回到终端。一个命令通常从一个叫标准输入的地方读取输入，默认情况下，即终端。同样，一个命令通常将其输出写入到标准输出，默认情况下，即终端

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（标准输入STDIN）、1（标准输出STDOUT）、2（标准错误输出STDERR）

重定向一般通过在命令间插入特定的符号来实现。

输出重定向：

command1 > file1　#执行command1，然后将输出的内容存入file1

如果要将新内容添加在文件末尾，使用 > >

输入重定向：

command1< file1

command1 < infile > outfile　#执行command1，从文件infile读取内容。然后将输出写入到outfile

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

```
command << delimiter
    document
delimiter
```

//将两个delimiter之间的内容（document）作为输入传递给command

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，那么将输出重定向到/dev/null //禁止输出的作用

- shell文件包含

```
. filename   # 注意点号(.)和文件名中间有一空格

或

source filename
```