Shell编程

# ShellScript

Shell本身是一个用C语言编写的程序，是用户使用Linux的一个工具，用户在Linux下敲的命令都是shell命令，通过shell解释后作用于Linux内核；shell也是一种编程语言，有特定的语法格式，我们编写的shell脚本（shell script）需要通过解释器来执行脚本（通常会在脚本前指定解释器#!/bin/sh，或运行时指定解释器：/bin/sh test.sh）然后作用于Linux；所以说shell脚本是一种解释性的编程语言。

Shell脚本中的注释：shell中可以用#开头表示单行的注释，也可以用以下三种方式进行多行的注释：

:<<EOF :<<**’** :<<!

注释内容… 注释内容… 注释内容…

EOF **‘** !

# Shell变量

## 变量的定义和使用

1. 定义变量：变量只能是下划线数字和字母的组合，且必须是下划线或字母开头；
2. 变量赋值：变量的定义可以直接给出变量名即可，可以不用赋值也可以直接显式赋值也可以用语句赋值（如循环体的变量）；
3. 变量引用：变量的的引用需要使用**$**符号，$符号后面连续的字符串在脚本中被视为一个变量的引用，通过**{}**可以在引用变量时给出变量的边界，以便正确使用变量；
4. 只读变量：当变量定义后，可以使用readonly 变量名关键字声明变量为只读变量，只读变量声明后，该变量就不能被更改；
5. 删除变量：使用unset 变量名，可以删除一个变量，只读变量不能被删除；

## 变量的类型

### 环境变量

Linux的环境变量在etc/profile 以及~/.bashrc中定义，供所有程序使用，必要的时候shell脚本也可以定义环境变量；

### 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，其他的shell程序不能访问局部变量；

### Shell变量

Shell里设置了的一些变量，包括环境变量和一些局部变量；

## Shell字符串

### 字符串使用规则

Shell编程中字符串可以使用单引号或双引号表示，但是单引号字符串中的任何字符都按原样输出，所以使用$也被当做一个字符而不是对变量的引用，并且单引号不能单独出现，只能成对出现表示一串字符；

双引号字符串中可以出现变量的引用以及字符的转义(\c表示不换行)单双引号表示的字符串均可直接拼接形成新的字符。

### 字符串的操作

1. 字符串拼接：字符串可以直接拼接在一起形成新的字符串；
2. 字符串长度：使用**${#string}**方式可以获取字符串长度；
3. 字符串提取：使用**${string:n:m}**从位置n开始截取m个字符；
4. 字符位置查找：**`**expr index “$string” c**`**，输出字符’c’在string中首次出现的位置；

## Shell数组

### Shell数组定义

array\_name={value0 vlalue1 value2 …}

单个赋值：array\_name[0]=value0

获取元素：单个获取${array\_name[index]}

全部获取：${array\_name[@]}

数组长度：${#array\_name[@]} / ${#array\_name[\*]}

元素长度：${#array\_name[idx]}

# Shell参数传递

## 向脚本传参

格式：./test.sh parm1 parm2 …

脚本参数的引用$n：

|  |  |
| --- | --- |
| 参数引用 | Description |
| $0 | 脚本本身运行的路径名 |
| $1-9 | 运行脚本时依次传入的参数 |
| $# | 向脚本传递的参数个数，不含$0 |
| $\* | 把脚本输入的所有参数以一个字符串输出以空格分隔 |
| **“**$@**”** | 把脚本输入的参数分别以字符串的形式输出 |
| $$ | 脚本运行的当前ID号 |
| $! | 后台运行的最后一个进程的ID号 |
| $- | 显示shell使用的当前选项，与set命令相同 |
| $? | 输出命令执行后的退出状态，0表示没有错误其他值为错误 |

注意：如果向脚本传递的参数有空格，需要将该参数用引号括起来

# Shell基本运算符

原生的bash是不支持简单的数学运算符，需要通过其他命令一起完成运算，常见的如awk，expr表达计算工具求值操作，完整的表达式格式：`expr val1 + val2`

## 算术运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| + | 加法 | `expr $a + $b` |
| - | 减法 | `expr $a - $b` |
| **\\*** | 乘法 | `expr $a \\* $b` |
| / | 除法 | `expr $b / $a` |
| % | 取余 | `expr $b % $a` |
| = | 赋值 | a=$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true / false。 |
| != | 不相等。用于比较两个数字，不相同则返回 true。 | [ $a != $b ] 返回 true / false。 |

注意：乘号需转义使用因为shell中\*表示通配符；条件表达式需要用方括号括起来，并且括号内两侧须有空格。

## 关系运算符

关系运算符只支持数字间的关系运算，不支持字符串，除非字符串的值是数字

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| -eq | 检测两个数是否相等，返回 true / false。 | [ $a -eq $b ] |
| -ne | 检测两个数是否不相等，返回 true / false。 | [ $a -ne $b ] |
| -gt | 检测左边的数是否大于右边的，返回true / false。 | [ $a -gt $b ] |
| -lt | 检测左边的数是否小于右边的，返回true / false。 | [ $a -lt $b ] |
| -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 ] |
| -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，返回 true / false。 | [ ! false ] |
| -o | 或运算，返回 true / false。 | [ $a -lt 20 -o $b -gt 100 ] |
| -a | 与运算，返回 true / false。 | [ $a -lt 20 -a $b -gt 100 ] |

## 逻辑运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| && | 逻辑AND，返回true/false | [[ $a -lt 100 && $b -gt 100 ]] |
| || | 逻辑OR, 返回true/false | [[ $a -lt 100 || $b -gt 100 ]] |

## 字符串运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **说明** | **举例** |
| = | 检测两个字符串是否相等，返回 true / false | [ $a = $b ] 返回 false。 |
| != | 检测两个字符串是否相等，返回 true / false。 | [ $a != $b ] 返回 true。 |
| -z | 检测字符串长度是否为0，返回 true / false。 | [ -z $a ] 返回 false。 |
| -n | 检测字符串长度是否不为 0，返回 true / false。 | [ -n "$a" ] 返回 true。 |
| $ | 检测字符串是否为空，返回 true / false。 | [ $a ] 返回 true。 |

## 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 | [ -b $file ] |
| -c file | 检测文件是否是字符设备文件 | [ -c $file ] |
| -d file | 检测文件是否是目录 | [ -d $file ] |
| -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） | [ -f $file ] |
| -g file | 检测文件是否设置了 SGID 位 | [ -g $file ] |
| -k file | 检测文件是否设置了粘着位(Sticky Bit) | [ -k $file ] |
| -p file | 检测文件是否是有名管道 | [ -p $file ] |
| -u file | 检测文件是否设置了 SUID 位 | [ -u $file ] |
| -r file | 检测文件是否可读 | [ -r $file ] |
| -w file | 检测文件是否可写 | [ -w $file ] |
| -x file | 检测文件是否可执行 | [ -x $file ] |
| -s file | 检测文件是否为空（文件大小是否大于0） | [ -s $file ] |
| -e file | 检测文件（包括目录）是否存在 | [ -e $file ] |
| -S | 判断文件是否是socket |  |
| -L | 检测文件是否存在并且是一个符号链接 |  |

# Shell流程控制

Shell中的流程控制体不能为空，控制流程中可以使用break和continue，用法同C语言类似

## 条件判断

If condition ; then command… ; fi

If condition; then comand1; else comand2; fi

If condition; then comand1; elif condition; then comand2; else comand3; fi

## For循环

for value in item1 item2 … item3; do command; done

## Wile循环

Wile condition; do command; done

## 无限循环

While : ; do command; done

While true; do command; done

For (( ; ; ));do command; done

## Until循环

Until循环直到条件为true时终止循环

Until condition; do command; done

## Case选择语句

Case vale in

Select1)

Command

**;;**

Select2)

Command

**;;**

Ecas

# Shell函数

标准定义模式

[function] fun\_name ([])

{

Action;

[return val;]

}

简单函数定义可以直接fun\_nam(){}，可以没有函数声明关键字、参数和返回值；

函数的参数在调用的时候传递，函数内部使用参数同执行脚本时参数的传递与使用方式一致；

注意：函数参数是从$1开始，并且$n当n>=10需要加{}进行引用（${n}）

# Shell输入输出重定向

|  |  |
| --- | --- |
| **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。 |

# Shell相关

## 文件包含

shell的文件包含使用source filename形式；

## printf的使用

printf的使用同C语言基本一致

printf format-string [arguments …]