可执行文件开头第一行一般我们会指定用什么解释器来执行该文件比如shell角本的文件开头一般会加#! /bin/sh

如果是python文件（后缀名为.py）在第一行增加 #! /usr/bin/python

## shell 语法

### shell定义变量以及调用变量

运行shell 时会遇到三种变量

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。必要的时候shell脚本也可以定义环境变量。
3. 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

定义变量时，变量名开始必须以[a-zA-Z]开始，中间不可以有空格或标点符号（可以用“\_”)，变量名不可以使用bash的关键字。

调用变量，只需要在变量名前加”$”便可以了，考虑到解释器识别边界的问题，一般我们会在变量名外加大括号来确定变量名

删除变量可以用 unset 来取消变量的定义 .

现在我们便创建一个test.sh文件并且给它执行权限在里面输入以下内容

#! /bin/sh

#变量定义举例：

myName="sandow"

myUrl="http://magdre.github.io"

myAge=34

#调用变量

echo "hell everyone my name is $myName, my blog site is $myUrl,"

echo "Today, I'm ${myAge}years old"

#删除变量

unset myName

大括号（花括号）是为了让解释器识别变量名的边界，如果不加的话变量名就成了 $myAgeyears 这个变量名为空，输出来的便只有 **Todey, I’m old** 这样与期望值并不一样。

#### 在shell中的特殊变量名

|  |  |
| --- | --- |
| 变量 | 含义 |
| $0 | 当前脚本的文件名 |
| $n | 传递给脚本或函数的参数。 |
| $# | 传递给脚本或函数的参数个数。 |
| $\* | 传递给脚本或函数的所有参数。 |
| $@ | 传递给脚本或函数的所有参数。被双引号(” “)包含时，与 $\* 稍有不同 |
| $? | 上个命令的退出状态，或函数的返回值。 |
| $$ | 当前Shell进程ID。对于 Shell 脚本，就是这些脚本所在的进程ID。 |

**$n** n 是一个数字，表示第几个参数,例 $1 。 如果超过10便需要写成 ${10}

现在我们接着在test.sh里输下面内容

echo "File Name: $0"

echo "First Parameter : $1"

echo "Second Parameter : $2"

echo "Quoted Values: $@"

echo "Quoted Values: $\*"

echo "Total Number of Parameters : $#"

我们在命令行输入

$ ./test.sh hello world

便可以得到下面结果

./tesh.sh

hello

world

hello world

hello world

2

#### 变量赋值与转换

|  |  |
| --- | --- |
| 形式 | 说明 |
| ${var} | 变量本来的值 |
| ${var:-word} | 如果变量 var 为空或已被删除(unset)，那么返回 word，但不改变 var 的值。 |
| ${var:=word} | 如果变量 var 为空或已被删除(unset)，那么返回 word，并将 var 的值设置为 word。 |
| ${var:?message} | 如果变量 var 为空或已被删除(unset)，那么将消息 message 送到标准错误输出，可以用来检测变量 var 是否可以被正常赋值。 若此替换出现在Shell脚本中，那么脚本将停止运行。 |
| ${var:+word} | 如果变量 var 被定义，那么返回 word，但不改变 var 的值。 |

#### shell里的运算

在原生bash中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。

在使用expr时的格式为 expr 1 + 2

a=10

b=20

**算术运算符**

|  |  |  |
| --- | --- | --- |
| 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。 [ b ] 返回 false。 |
| != | 不相等 | 用于比较两个数字，不相同则返回 true。 [ b ] 返回 true。 |

**关系运算**

|  |  |  |
| --- | --- | --- |
| 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 true |
| -ne | 检测两个数是否相等，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。 |

**逻辑运算**

|  |  |  |
| --- | --- | --- |
| 运算符 | 说明 | 举例 |
| ! | 非运算,表达式为 true 则返回 false | [ ! 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 |

**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 | [str $a ] 返回 true |

**文件测试运算符**

file="/var/www/tutorialspoint/unix/test.sh

|  |  |  |
| --- | --- | --- |
| 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 | [ -b $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**，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 | 检测文件是否是具名**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 里处理字符

先定义一个变量：

string="sandow is a gentleman"

在python中用 len(string) 来计算字符串的长度，在shell里我们可以这样计算 echo ${#string}

在python中可以用 string[1:4] 来切片，在shell中 我们可以这样来实在 ${string:1:4}

幸运的是这里index都是以0开头。所以切出来的值是相同的。

在python中用 string.find("sandow") 来查找字符，而在shell中可以这样写 expr index "$string" sandow

#### shell 中的数组

在shell中只可以建立一维数组，并且index从0开始，创建数组用小括号。

array\_name=(value0 value1 value2 value3)

读取数组中某个值时可以用 ${array\_name[index]} 读取所有值可以用 “\*” 或者 “@” 计算长度仅需要要array\_name前加 “#” 与之前一样

for a in `seq 0 ${#ip[@]}`

do

echo ${ip[$a]}

#### shell 中的if 判断语法

if [ expression 1 ]

then

Statement(s) to be executed if expression 1 is true

elif [ expression 2 ]

then

Statement(s) to be executed if expression 2 is true

elif [ expression 3 ]

then

Statement(s) to be executed if expression 3 is true

else

Statement(s) to be executed if no expression is true

fi

#### case 语法

case 与excel里的case类似，取值先匹配每一个模式，模式匹配后，刚执行匹配模式相应命令，而不会继续其他模式。如果无一匹配模式，使用星号“\*”

来捕获该值，再执行后面的命令。 case的值后面必须为 关键字 in，每一模式必须以左括号结束，取值可以为变量或常数。匹配发现聚会符合某一模式后，其间所有命令开始执行直至遇到;;，结束。

case var in

pattern1)

command1

command2

command3

;;

pattern2）

command1

command2

command3

;;

\*)

command1

command2

command3

;;

esac

#### for 语法

for 变量 in 列表

do

command1

command2

...

commandN

done

#### while 语法

while command

do

Statement(s) to be executed if command is true

done

#### until 语法

until command

do

Statement(s) to be executed until command is true

done

#### 函数语法

function 可有可无，不过做为一个合格的编程人员，有必要加上的。这才是规范。

function function\_name () {

list of commands

[ return value ]

}

向函数内传递文件和上面一样 function\_name p1 p2 p3 然后在函数内部用 $n 调用