Shell脚本快速入门

`shell`脚本是每一个linux开发者所必备的技能，相比其它脚本语言，`shell`更加贴近linux本身，与bash高度整合，在不少应用中（尤其是涉及文本处理和IO的）有着更高的易用性和简洁性。

阅读本文需要一定的基础，笔者在写作时，期望读者：

- 有一定的linux使用经验
- 熟悉至少一门常规脚本语言/编程语言
- 知道管道**为何物**
- ~~不要读着读着就睡着了~~

  全文节奏较快，不会对细节作过多阐述（~~因为作者也是个半吊子~~），旨在让读者在短时间快速上手，如存在问题，欢迎指正交流。

变量与函数

变量定义与赋值的标准形式是`标识符=值`，不需要单独声明，等号附近**不能**有空格，示例如下：

```
#!/bin/bash

a=35
name="williams"
_o="o"
_ld='ld'
_old="$_o$_ld" # 通过 `$` 来引用变量 （此处只能使用双引号）

echo $name is $a years $_old # 输出

unset _o # 删除变量（置空）
```

用`sh`命令运行脚本，获得输出

```
williams is 35 years old
```

函数的定义方式如下

```
func1() {
    echo $1 # $1 代表函数的第一个参数
    echo $# # $# 代表函数参数的数目
}

func2() {
    return $(($1+$2)) # 返回第一个和第二个参数相加的值 （返回值必须为整型）
}

a=3
func1 "oops" 233 # 调用函数
func2 $a 2
```

```
echo $? # 输出上一个命令/函数的返回值
```

运行脚本，获得输出

```
oops
2
6
```

命令与IO

命令与IO是`shell`脚本中最重要的部分，在管道系统的帮助下，脚本可以很轻松地与系统命令和程序交互。

以下是一个简单的例子：

```
txts=`ls | egrep *.\.txt` # 执行ls列出当前目录文件，并用egrep筛选.txt文件
# 还可以写作 txts=$(ls | egrep *.\.txt)
echo $txts # 输出所有 txt 文件
```

此处的`符号与`Ruby`相近，用于执行命令并将`标准输出流`中的内容作为字符串值返回。

因此，我们得到的变量`txts`就是在bash下执行命令时控制台上输出的内容。

再比如，读取文件内容，或者获取当前用户，利用上述方法就能很容易地实现

```
text=`cat hello.txt` # => text 被赋值为 hello.txt 里的文本
username=`whoami` # => username 被赋值为当前用户名
echo $username # => 输出用户名，如 root
```

向文件输出同样也十分简单

```
text1="hello,"
text2="world!"
echo -n $text1 > hello.txt # 利用重定向输出（n 表示不换行）
echo $text2 >> hello.txt # 追加文本到hello.txt中
```

运行上述脚本，目录下就会出现一个`hello.txt`，内有一行文字：`hello,world!`

使用`read`命令，可以从控制台中读入一个变量（类似于`Haskell`中的read）

```
#!/bin/bash
# A + B Problem ~
read a
read b
c=$(($a+$b))
echo $a + $b = $c # => e.g. 1 + 3 = 4
```

值得注意的是，变量也可以参与`符号所包含的命令中，比如：

```
text="eureka!"
text=`echo $text | tr 'a-z' 'A-Z'` # 用tr替换所有小写字母为大写字母
echo $text
```

上述脚本输出为`EUREKA!`

运算与流程控制

运算与括号

提起`shell`脚本中的运算，就不得不介绍一下最常用的几种括号

双小括号`(( ))`用于算术（整型）运算，同时也可以用于一些特殊功能

```
a=2
b=$(( ($a + 1) / 2 )) # => b = (2 + 1) / 2 = 1
rd=$((RANDOM % 100)) # => rd 为0~99的随机数
c=$(( $a > 1 )) # => 判断 a > 1 , 结果为 1 (true)
```

中括号`[ ]`和双中括号`[[ ]]`用于比较运算，其中`[ ]`实际上是对linux中`test`命令的隐式调用（**务必注意**：在shell的定义中，`true`为0，而`false`为1）

```
a=2; b=1; c=3; # 单行可以用分号隔开多个表达式
str='ray-tracing'

[ $a -gt $b ] # 两边一定要空格， [$a -gt $b] 是不合法的
echo $? # 比较结果存储于 $? 中 => 输出为 0 (true)

[ $c -eq $c ] # 等价于 test $c -eq $c
echo $? # 输出为 0 (true)

[ $a -ne 1 -a $c -eq 3 ] # 相当于表达式 a != 1 && c == 3
[ $a -ne 1 -o $c -ne 3 ] # 相当于表达式 a != 1 || c != 3

[ $str != 'ray-tracing' ] # 结果为 1 (false)
[ $str = 'ray' ] # 结果为 1 (false)

[[ a + b = 3 && a != b ]] # 结果为 0 (true)
# 此语句只有
```

熟悉`test`命令的使用对写出正确的逻辑运算表达式很有帮助，读者可以参考维基百科来了解更多。

不难看出，双中括号[^1]`[[ ]]`相比中括号更加人性化，可以轻松胜任近似`C语言`的表达式。

大括号`{}`用于定义范围和通配，在此暂不详细阐述，有兴趣的读者可以参阅这篇博客。

分支语句

与`Python`不同，shell脚本并不强制要求任何形式的缩进，因此流程语句需要有块结束符号。

典型的if语句如下：

```
read a
echo -n $a is

b=$(( $a % 2 ))
if [ $b -eq 0 ]; then # 判断 a 是否为偶数
    echo " even"
else
    echo " odd"
fi

if [ `whoami` = 'root' ]; then
    echo "and you are root ?"
elif [ `whoami` = 'phosphorus15' ]; then # 相当于 C 中的 else if
    echo "seriously ?"
else
    echo "alright ~"
fi
```

注意：无论是循环还是分支控制，都不允许有空的块，即不能出现：

```
if [ `whoami` = 'root' ]; then # 不能为空!
else
    echo "oops !"
fi
```

循环语句

和许多高级语言一样，shell具有几种可用的循环控制结构，笔者在此会介绍自己常用的几种：

- while 语句 - *耳熟能详，不说就会~*

  ```
  a=1; b=0;
  while [ $a -le 100 ]; do    # 此处也可写作 while (( $a <= 100 )); do
      b=$(($a+$b))
      a=$(($a+1))
  done
  echo $b # 输出为 5050
  ```

  是不是觉得写起来怪难受？ 没关系~算术运算本来就不是shell脚本的特长，下面你要看到的，才是shell真正的精髓。

- until 语句 - *反其道而行*

  ```
  a=1; b=0;
  until [ $a -gt 100 ]; do
      b=$(($a+$b))
      a=$(($a+1))
  done
  echo $b # 输出为 5050
  ```

  其实就是把while反过来~

- for each 语句 - *来自C11*

```
for i in Apple Juice Cake; do
    echo do you like $i ?
done

for j in `seq 0 3`; do
    echo $j
done

for k in `seq 65 70`; do
    echo $k | awk '{printf("%c", $1)}' # 将ascii码转为字符输出
done
```

输出内容：

```
do you like Apple ?
do you like Juice ?
do you like Cake ?
0
1
2
3
ABCDEF
```

不难想到，我们先前的`while`循环可以写作

```
s=0
for i in `seq 1 100`; do # 创建一个 1 到 100 的序列
    s=$(($s+$i))
done
echo $s
```

假如我们想给目录下的每个文件都创建一个备份，可以这么写：

```
for file in $(ls); do # 列举所有文件
        # 用 file 判断是否是文件夹， 是就不进行备份
        if [ "$(file $file | grep -o directory)" != 'directory' ]
        then
                cp $file "$file.bak" # 用 cp 创建备份
                echo backup $file
        fi
done
```