# 序章简介。

Shell作为linux上沟通与隔绝user和kernel的工具，是服务器端工程师使用最多的几种语言之一，无论从事哪一个方向，无论上层用java还是python，只要需要对linux系统的机器进行快速更改或操作，shell都是必不可少的。Shell适合执行简单的重复操作，当然，循环和test是基本功能，如需使用复杂的逻辑结构，可能需要搭配python等一同使用。

## 1.1 使用shell写程序

No programming language is perfect. There is not even a single best language; there are only languages well suited or perhaps poorly suited for particular purposes. （没有编程语言是完美的。甚至没有一个最好的语言;只有适合的语言或可能不适合特定用途的语言。）

从学习C Language开始这句名言就是座右铭一般。

一个机器所搭载的linux系统一般是公版的，如果我需要对不同业务的机器做不同的定制，对shell的理解是必须的，在Linux机器启动时，它会在/etc/rc.d中执行shell脚本来恢复系统配置和设置服务，所以去阅读、分析并修改这些启动脚本是Ops的基本功。

由于linux的灵活性，无论是超算级的计算机还是类似于树莓派或更小的嵌入式计算机都可以搭载定制的linux系统，所以从某方面来说，了解shell（bash）编程就获得了玩转绝大多数专业计算机的机会。

正如文中所说的，shell的思想是高级的—类似于分而治之的思想，将复杂、缠绕的大型问题分割成细小的、便于调试与审阅的简单问题，再连接在一起。与每隔几年便兴起的某些所谓用更简单的规则、语法来提高效率的“高级语言”对比，这种思想可能保质期更长—至少从我国历史来看秦朝或更早就有了分而治之的思想。

shell不适用的情况总的来说可以归为一下几点：

1. 不能开源的代码。
2. 不需要系统级操作或要保证系统安全的。
3. 对高级数据结构有要求的（多维数组…）。
4. 需要对大图像、多文件、复杂计算操作的。
5. 对网络有多需求的（socket、api…）
6. 需要多平台通用的。

## 1.2 从#！开始shell编程

这里介绍了如何执行、调用一个脚本、如何修改权限以及在一个脚本中#！的意义和用法。#！作为一个magic number，标记了此脚本所需要的解释器的位置。

在这一小节中，列出了一些shell脚本常用技巧，比如：脚本异常提示的写法，一个#！与多个#！的关系（除第一个被理解为解释器，其他#！都被理解为注释。）

这里有个不适用于现有linux环境的例子：

#!/bin/rm

# Self-deleting script.

# Nothing much seems to happen when you run this... except that the file disappears.

WHATEVER=85

echo "This line will never print (betcha!)."

exit $WHATEVER # Doesn't matter. The script will not exit here.

# Try an echo $? after script termination.

# You'll get a 0, not a 85.

此例子旨在让读者理解调用不同的解释器会起到不同作用，这里调用了rm脚本，但是在centos环境下，并不会rm自身，并且会默认使用bash解释器，在执行结束后echo $?结果是85.

# 二．基础

## 3.特殊字符

# 多用注释起始符，在复杂环境中亦可使 用（管道、转义字符后）

; 命令分隔符，一行中分隔多条命令

;; case选择的终止符(高版本上case选择的break符号可以用;;&或;&)

. dot符可用来组成文件名，作为文件名前缀时代表隐藏文件。也可代表目录名，.代表当前目录，..代表父目录。在正则中，dot表示匹配单个字符。

“ 部分引用，保留解释引用中的特殊字符。（如转义字符、强引用、回车符等）

‘ 强引用字符，引用中的所有字符保留输出。

, 逗号运算符连接所有的算数操作，也可作为连接字符串。

\ 反斜杠常用语转义字符，如：\x ()

/ 斜杠常用语连接目录，如：/home/cache/tmp。也可用于除法运算符、

$$ 进程ID变量. 这个$$ 变量 保存了它所在脚本的进程 ID

() 可用于初始化数组

{} cat {file1,file2,file3} > combined\_file把file1, file2, file3连接在一起, 并且重定向到combined\_file中。

[ ] 条件测试.条件测试表达式放在[ ]一个array结构的上下文中, 中括号用来引用数组中每个元素的编号.字符范围.用作正则表达式的一部分, 方括号描述一个匹配的字符范围。

> &> >& >> < <> 重定向符号。

\<, \> 正则表达式中的单词边界 。

| 管道. 分析前边命令的输出, 并将输出作为后边命令的输入. 这是一种产生命令链的好方法。

>| 强制重定向(即使设置了noclobber选项 -- 就是-C选项). 这将强制的覆盖一个现存文件。

|| 或-逻辑操作. 在一个条件测试结构中, 如果条件测试结构两边中的任意一边结果为true的话,||操作就会返回0(代表执行成功)。

& 后台运行命令. 一个命令后边跟一个& 表示在后台运行。

&& 与-逻辑操作. 在一个条件测试结构中, 只有在条件测试结构的两边结果都为true的时候, &&操作才会返回0(代表sucess)。

~ home目录[波浪号, 即~]. 相当于$HOME内部变量. ~bozo是bozo的home目录, 并且ls ~bozo将列出其中的内容. ~/就是当前用户的home目录, 并且ls ~/将列出其中的内容。

（特殊符号太多，只总结了常用且不太好记忆的。）

## 4.变量和参数

#### 4.1 变量的值

定义一个变量a，a的值在shell中用$a 表示。

在这里有几点需要注意：

（1）$a是${a} 的简写，一般情况下没有错，如果编译出错，写全写。

（2）a在未被赋值的时候是一个“NULL”空值，并不是0。“NULL”值可以进行运算，但是可能会出错，建议定义变量时进行初始化。

（3）弱引用” ” 引用$a时，带入的是a的值，强引用’ ’时，带入的是$a这两个符号。

#### 4.2 变量赋值

定义一个变量a，a的值在shell中用$a 表示。

注意区别赋值时“=”“-eq”的区别，“=”要用let函数：let a=1+2

“-eq”：[ -eq 12 ]

赋值可以为字符串，可以为命令return的结果：a=`ls –a `

#### 4.3 bash中的变量都是无类型的

本质上，bash变量是字符串，但根据上下文，bash允许对变量进行算术运算和比较。

在使用时，无类型的特性会使代码更灵活简单，但是，一定小心不要出现类型转换的错误。

注：shell也可以声明变量，使用declare关键字配合参数：-i integer –r readonly –a array –f functions –x export –x var=$value

#### 4.4 特殊变量

（1）局部变量

其有效性只限定在某一个函数或一个代码块中。

（2）环境变量

形象当前shell和用户行为的变量，没启动一个shell就会套用一个环境，里面可以使在此次运行中所用到的共同的变量、参数等。

（3）位置参数

以“$”开头，后接数字的特殊变量：

$0 此shell脚本的文件名。

$1 传递给此脚本的第一个参数。

$2 传递给此脚本的第二个参数。

……

## 5 引用

指用引号括起来的字符串，通过引用灵活的让shell解释或不解释引用中的字符串（可能是命令、变量名、通配符等等）。

灵活使用引用可以解决某些时候的文本处理、匹配或echo的小问题如：

bash$ **echo $(ls -l)**

total 8 -rw-rw-r-- 1 bo bo 13 Aug 21 12:57 t.sh -rw-rw-r-- 1 bo bo 78 Aug 21 12:57 u.sh

bash$ **echo "$(ls -l)"**

total 8

-rw-rw-r-- 1 bo bo 13 Aug 21 12:57 t.sh

-rw-rw-r-- 1 bo bo 78 Aug 21 12:57 u.sh

（echo在某些时候不会讲换行符打印出来）

#### 5.1 引用变量

当引用一个变量时，通常建议把它的名字用双引号引起来。

这可以防止重新解释所引用的字符串中的所有特殊字符

（$ ` `和 \ 除外）

注：即使一个长短语中含有space分隔符，使用””依然可以将其解释为一个词。

单引号中引用单引号不会出现效果。

在使用echo –e时，如以下例子：

echo -e "x\ty" -》x y

echo -e x\ty -》xty

#### 5.2 转义字符

转义是一种引用单个字符的方法。字符前面的 \ 告诉shell从字面上解释该字符。

注：类似于””，\也可以防止分词，只需要在词中space前加上 \ 即可。

一条命令长度过长时，可以在适当的位置加上 \ 然后换行继续此条命令。

常用转义字符如下：

\n 换行

\r 返回

\t tab

\v 垂直tab

\b 退格

\a 警报

\0xx 转化为8进制

\” 直接打印”

\$ 直接打印$

\\ 直接打印\

## 6 退出和退出状态

在shell中，正确执行的脚本会在执行后返回0状态码，可以用命令 $? 查看上一个执行的脚本的exit状态码。此命令经常用于调试脚本。

在脚本中，我们可以手动定义退出状态码，范围为0-255。

在脚本中，未定义状态码时，退出状态码显示为最后执行的一条命令的退出状态。

还要注意的是，调用官方或他人写好的脚本或命令时，某些退出状态码是有特舒含义的，可以定位某些错误。

## 7 测试

这一张承接上一章，通过对某命令的退出状态码判断去做代码测试。

#### 7.1 结构测试

在这章，主要提到了几点对于整个代码结构做测试的方法语句。

在使用if/then判断语句时，起作用的其实还是状态码，if判断为真的条件是状态码为0。

一般来说，我们使用 [ ]来测试某句命令，这是个shell内建方法，用法如下：

if [ -f a.txt ]

then …

fi

类似于上一条的，还可以使用[[ ….. ]]来测试某句代码，bash会将其中的语句当做一条命令，成功执行则状态码为0。

如果要测试一条算术表达式，可以使用(( … ))，如：(( $a && $b)) ，结果为非零值时状态码为0，算术表达式结果为0时，状态码是1.。

#### 7.2 测试文件相关的参数

格式：if [ -e file1 ] else ls fi

-e 文件是否存在

-f 文件存在且是普通文件（非目录和设备）

-s 文件存在且不是空文件

-d 是个目录

-b 是个块设备

-c 是个字符设备

-p 是个管道

-h 是个符号链接

-S 是个套接字接口

-r/w/x 有读/写/执行权限

-O 当前用户是否为此文件拥有者

-G 组id是否和当前用户相同

File1 -nt file2 file1比file2新

File1 –ef file2 file1比file2老

#### 7.3其他比较运算符

注：字符或文件比较（非算数运算操作）一般使用 [ ]

算数运算一般使用 (( ))

if [ "$a" -eq "$b" ] 相等

if [ "$a" -ne "$b" ] 不相等

if [ "$a" -gt "$b" ] 大于

if [ "$a" -ge "$b" ] 大于或等于

if [ "$a" -lt "$b" ] 小于

if [ "$a" -le "$b" ] 小于或等于

(("$a" < "$b")) 小于（算数运算）

(("$a" <= "$b")) 小于等于

(("$a" > "$b")) 大于

(("$a" >= "$b")) 大于等于

if [ "$a" == "$b" ] 等于

if [ "$a" != "$b" ] 不等于

if [ -z "$String" ] 字符串为空

## 8 具体运算操作和相关贴士

本章定义了基本的运算操作符号和相关的小知识

四则远算符号：+ - \* /

次方和取余符号：\*\* %

连算符号：n += 5（n=n+5）

n -= 5（n=n-5）

n\*=5（n=n\*5）

n/=5（n=n/5）

n%=5（n=n%5）

位运算符：<< 按位左移

<<= 按位左移几位

>> 按位右移

>>= 按位右移几位

& 按位and运算

&= 按位and-equal

| 按位或运算

|= 按位或等运算

~ 按位非运算

^ 按位异或运算

^= 按位抑或等于运算

逻辑运算：! 非运算

&& 且运算

|| 或运算

注：shell脚本将一般数字解释为十进制数，除非该数字有前缀：0前缀表示8进制，0x前缀表示16进制 。

##在前面章节简单介绍了 (( )) 这里在说明一下，类似于let命令，（（...））结构允许算术扩展和评估。以最简单的形式，a = $（（5 + 3））会将a设置为5 + 3，或者8。这个双括号结构也是一种允许在bash中对c变量进行操作的机制。

## 9 shell进阶

#### 9.1 内部变量

内部变量是shell自带的已经被赋值的变量。用来方便用户拿到一些与当前shell环境相关的数据。

以下列出常用的一些：

$BASH 当前bash的路径

$BASH\_ENV 当前bash的环境变量文件路径

$BASHPID 当前bash的pid

$BASH\_VERSION 当前bash的版本

$EUID 当前有效的用户的ID，即使是通过su获得的

$GROUPS 当前用户所属的组

$HOME 当前用户的家目录

$HOSTNAME 当前主机名

$HOSTTYPE 当前主机所属架构（x86，i686，arm）

**$IFS 系统默认的解释字符串时的分隔符，系统默认是空格。**

$IGNOREEOF 本次结束之前，忽略掉文件结束符

$PPID 当前进程的父进程的pid

$PWD 目前所在的目录

$SECONDS 脚本运行的时间，单位是秒

$UID 当前用户的uid，无视是否使用su登录到另外的用户

$? 前面经常说道，退出时的状态

$$ 脚本本身进程的pid，一般情况下和$BASHPID相同

还有一些读取参数时常用的位置参数：

$0, $1, $2… 前面介绍过，0代表脚本名，1,2,3,4…代表第几个参数

$# 参数的数量

$\* 所有的位置参数，看做一个单词，必须引用起来

#### 9.2 shell中声明的用法（declare/typeset）

之前我们已经说过了shell中一般变量是不需要申明数据类型的，但是声明是可以有特殊用法的。

declare –r 只读

declare –i 整数

declare –a 数组

declare –f 函数

declare –x 从外部引入

declare的特殊用法：

声明命令可以帮助识别变量，环境或其他。这对数组尤其有用。

比如：

bash$ **declare | grep HOME**

HOME=/home/bozo

bash$ **zzy=68**

bash$ **declare | grep zzy**

zzy=68

bash$ **Colors=([0]="purple" [1]="reddish-orange" [2]="light green")**

bash$ **echo ${Colors[@]}**

purple reddish-orange light green

bash$ **declare | grep Colors**

Colors=([0]="purple" [1]="reddish-orange" [2]="light green")

#### 9.3 随机数内部函数：$RANDOM 的用法

$ random是一个内部的bash函数（不是一个常量），它返回一个范围为0 - 32767的伪随机数整数。由于他是个正整数，所以并不适用于产生秘钥等操作。

$ random可以在某些控制概率的模型中格外好用。

## 10 玩转变量

#### 10.1 玩转字符串

#### 10.2 参数的替换