编写bash脚本常用功能总结

By simon0227

1. 基础常识

* 变量赋值
  + 赋值语句中”=”前后**不能**有空格
  + 把标准输出当做一个普通变量，从而把标准输出赋值给另一个变量。

如把echo “hello”的标准输出赋值给value变量，value=$(echo “hello”)

* 输入输出流

0：标准输入流(stdin)

1：标准输出流(stdout)

2：标准错误流(stderr)

几个常用的流重定向命令：

|  |  |
| --- | --- |
| **命令** | **解释** |
| > file | 将命令的执行结果送至指定的file中 |
| >! file | 将命令的执行结果送至指定的file中，若file已经存在，则覆盖 |
| >& file | 将命令执行时屏幕上所产生的任何信息写入指定的file中 |
| >> file | 将命令执行的结果附加到指定的file中 |
| >>& | 将命令执行时屏幕上所产生的任何信息附加到指定的file中 |

常用的还有，**把标准错误(stderr)重定向至标准输出(stdout)**。

**命令：**2>&1，即将标准错误也输出到标准输出当中。其中&是为了区别于普通文件，否则会在当前目录建个文件叫1，而不是输出到标准输出。

**高级示例：**ls xxx >out.txt 2>&1。这里为何2>&1要写在后面？  
     **解释：**command > file 2>&1。首先，command > file将标准输出重定向到file中；然后2>&1是标准错误拷贝了标准输出的行为，也就是同样被重定向到file中，最终结果就是标准输出和错误都被重定向到file中。  
     而command 2>&1 >file。2>&1 标准错误拷贝了标准输出的行为，但此时标准输出还是在终端，因此标准错误输出仍然在终端屏幕上。>file命令后输出才被重定向到file，但标准错误仍然保持在终端。

* Echo输出的颜色设置

常用的颜色码：0重置；

文本颜色(3)：30黑色，31红色，32绿色，33黄色，34蓝色，35洋红，36青色，37白色；

背景颜色(4)：40黑色，41红色，42绿色，43黄色，44蓝色，45洋红，46青色，47白色；

例：echo –e “\e[1;32mHello World\e[0m”

* 循环

两种形式for或者while

* 条件语句

注意if条件的书写格式。

if [ 条件 ]; then #注意”[“ ，“]”与条件间的空格不能省略

elif [ 条件 ]; then

else

fi

另外，还有多分支语句的case命令：

case $var in

\*) echo

;;

esac

1. 算术表达式

Bash中算术表达式有3种形式($[]方式不考虑)。

* let arg。arg是单独的算术表达式。这里的算术表达式使用C语言中表达式的语法、优先级和结合性。除++、--和逗号(，)之外，所有整型运算符都得到支持。此外，还提供了方幂运算符“\*\*”。命名的参数可以在算术表达式中直接利用名称访问，不要前面带有“$”符号。当访问命名参数时，就作为算术表达式计算它的值。算术表达式按**长整数**进行求值，并且不检查溢出。当然，用0作除数就产生错误。如let ″j=i\*6+2″。当表达式中有Shell的特殊字符时，必须用双引号将其括起来。
* ((expression))。是let命令的代替形式。如上面的例子可表达为((j=i\*6+2))。如果表达式的值是非0，那么返回的状态值是0；否则，返回的状态值是1。
* expr命令。通用求值表达式，能通过给定的操作(参数必须以空格分开)连接参数,并对参数求值.可以使算术操作, 比较操作， 字符串操作或者是逻辑操作.

1. 关联数组、索引数组

关联数组：可以用字符串索引值，类似Map或key-value对。bash 4.0上才开始支持。

索引数组：用整数下标进行索引，类似普通的数组(array)。

数组声明（可以省略，直接使用），可以声明4种类型的数组。

declare –A array #全局关联数组

local –A array #局部关联数组

declare –a array #全局索引数组

local –a array #局部索引数组

数组销毁：unset array[subscript] #销毁单个元素

unset array/unset array[\*]/unset[@] #销毁整个数组

与数组相关的常用符号：

数组大小：${#array[@]}，${#array[\*]}

数组的所有键值(key)：${!array[@]}，${!array[\*]}

数组的所有值(value)：${array[@]}，${array[\*]}

获取数组的某个元素：${array[subscript]}，取subscript关联的元素，根据数组类型subscript可以为整数或字符串。

1. 字符串处理

字符串截取：

#：左截取。从字符串左边开始匹配，截取最小匹配的字符串。如，str=”hello world”,

${str#\*l}的结果为”hel”

##：左截取。从字符串左边开始匹配，截取最大匹配的字符串。如str=”hello world”,

${str##\*l}的结果为”hell”

%：右截取。从字符串右边开始匹配，截取最小匹配的字符串。

%%：右截取。从字符串右边开始匹配，截取最大匹配的字符串。

替换:

查找:

匹配：expr表达式。如expr match

1. 全局更新与sub-shell

现有这样一个情景：

在一个bash脚本中间，需要读取某个文件的内容，并处理和设置某些变量，处理完以后，根据变量内容执行下一步操作。

如：

#!/bin/bash

flag=false

#read from file foo

cat foo | While read line

do

…

flag=true

done

….

if [ flag=”false ]; then

…

else

….

fi

这个脚本执行时，无论文件内容如何变，都只执行flag=false这个分支。因为在cat foo | while read line时，read是在一个sub-shell中进行的，此时flag成为sub-shell的局部变量，在sub-shell退出时，flag的值无法更新到原来的父shell中。即**因为sub-shell而出现了全局变量的一致性问题**。

此问题可以利用过程替换(process substitution)解决，即把脚本中文件处理部分代码改成如下方式：

#read from file foo

while read line

do

…

flag=true

done < <(cat foo)

其中<(cat foo)就是一个过程替换，详细内容可参考：[这里](http://www.linuxjournal.com/content/shell-process-redirection)。

另外，过程替换还能能解决一些重定向不能解决的问题，如让一个命令有多个输入的情况：

# comm -3 <(sort a | uniq) <(sort b | uniq)

1. 命令行参数解析

在Shell命令/函数中  shift常用来获取参数值(可以突破Linux 参数值只能传入$1-$9的限制)，而getopts可用来获取选项参数。

shift：位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1，原来的$5现在变成$2等等，原来的$1、$2、$3丢弃，$0不移动。不带参数的shift命令相当于shift 1。

getopts：例：getopts :ahfgc: OPTION 后面的冒号表示c这个选项为必选项，并且需要指定具体的参数值，同时需要获取到参数值后将参数值存放到变量OPTARG中，供以后读取用。

1. 常用工具

awk是一个不错的按行处理文件的工具，几个常用的参数。

-F sp：自定义分隔符

NF：是一行被解析后生成的域的个数。

-v var=value or --asign var=value，赋值一个用户定义变量。可用于变量通信。但方向的，如何把awk的内容传递到外面？print？

grep

bash 递归函数注意事项：

http://www.ibm.com /developerworks/cn/linux/l-cn-bashrecur/index.html

这段脚本问题的根源在于变量的作用域：在 shell 脚本中，不管是否在函数中定义，变量默认就是全局的，一旦定义之后，对于此后执行的命令全部可见。bash 也支持局部变量，不过需要使用 local 关键字进行显式地声明。local 是bash 中的一个内嵌命令，其作用是将变量的作用域设定为只有对本函数及其子进程可见。局部变量只能在变量声明的代码块中可见，这也就意味着在函数内声明的局部变量只能在函数代码块中才能被访问，它们并不会污染同名全局变量。因此为了解决上面这个程序的问题，我们应该使用 local 关键字将 i 声明为局部变量。