从程序员的角度来看， Shell本身是一种用C语言编写的程序，从用户的角度来看，Shell是用户与Linux操作系统沟通的桥梁。用户既可以输入命令执行，又可以利用 Shell脚本编程，完成更加复杂的操作。在Linux GUI日益完善的今天，在系统管理等领域，Shell编程仍然起着不可忽视的作用。深入地了解和熟练地掌握Shell编程，是每一个Linux用户的必修 功课之一。

Linux的Shell种类众多，常见的有：Bourne Shell（/usr/bin/sh或/bin/sh）、Bourne Again Shell（/bin/bash）、C Shell（/usr/bin/csh）、K Shell（/usr/bin/ksh）、Shell for Root（/sbin/sh），等等。不同的Shell语言的语法有所不同，所以不能交换使用。每种Shell都有其特色之处，基本上，掌握其中任何一种 就足够了。在本文中，我们关注的重点是Bash，也就是Bourne Again Shell，由于易用和免费，Bash在日常工作中被广泛使用；同时，Bash也是大多数Linux系统默认的Shell。在一般情况下，人们并不区分 Bourne Shell和Bourne Again Shell，所以，在下面的文字中，我们可以看到#!/bin/sh，它同样也可以改为#!/bin/bash。

利用vi等文本编辑器编写Shell脚本的格式是固定的，如下：

#!/bin/sh

#comments

Your commands go here

首行中的符号#!告诉系统其后路径所指定的程序即是解释此脚本文件的Shell程 序。如果首行没有这句话，在执行脚本文件的时候，将会出现错误。后续的部分就是主程序，Shell脚本像高级语言一样，也有变量赋值，也有控制语句。除第 一行外，以#开头的行就是注释行，直到此行的结束。如果一行未完成，可以在行尾加上"，这个符号表明下一行与此行会合并为同一行。

编辑完毕，将脚本存盘为filename.sh，文件名后缀sh表明这是一个Bash脚本文件。执行脚本的时候，要先将脚本文件的属性改为可执行的：

chmod +x filename.sh

执行脚本的方法是：

./filename.sh

下面我们从经典的“hello world”入手，看一看最简单的Shell脚本的模样。

#!/bin/sh

#print hello world in the console window

a = "hello world"

echo $a

Shell Script是一种弱类型语言，使用变量的时候无需首先声明其类型。新的变量会在本地数据区分配内存进行存储，这个变量归当前的Shell所有，任何子进 程都不能访问本地变量。这些变量与环境变量不同，环境变量被存储在另一内存区，叫做用户环境区，这块内存中的变量可以被子进程访问。变量赋值的方式是：

variable\_name = variable\_value

如果对一个已经有值的变量赋值，新值将取代旧值。取值的时候要在变量名前加$，$variable\_name可以在引号中使用，这一点和其他高级语言是明显不同的。如果出现混淆的情况，可以使用花括号来区分，例如：

echo "Hi, $as"

就不会输出“Hi, hello worlds”，而是输出“Hi，”。这是因为Shell把$as当成一个变量，而$as未被赋值，其值为空。正确的方法是：

echo "Hi, ${a}s"

单引号中的变量不会进行变量替换操作。

关于变量，还需要知道几个与其相关的Linux命令。

env用于显示用户环境区中的变量及其取值；set用于显示本地数据区和用户环境区中的变量及其取值；unset用于删除指定变量当前的取值，该值将被指定为NULL；export命令用于将本地数据区中的变量转移到用户环境区。

下面我们来看一个更复杂的例子，结合这个例子，我们来讲述Shell Script的语法。

1 #!/bin/bash  
 2 # we have less than 3 arguments. Print the help text:  
 3 if [ $# -lt 3 ]; then  
 4 cat<<HELP  
 5      ren -- renames a number of files using sed regular expressions  
 6   
 7      USAGE: ren 'regexp' 'replacement' fileshttp://www.cnblogs.com/Images/dot.gif  
 8      EXAMPLE: rename all \*.HTM files in \*.html:  
 9      ren 'HTM$' 'html' \*.HTM  
10   
11 HELP  
12      exit 0  
13 fi  
14 OLD="$1"  
15 NEW="$2"  
16 # The shift command removes one argument from the list of  
17 # command line arguments.  
18 shift  
19 shift  
20 # $\* contains now all the files:  
21 for file in $\*; do  
22 if [ -f "$file" ]; then  
23     newfile=`echo "$file" | sed  "s/${OLD}/${NEW}/g"`  
24         if [ -f "$newfile" ]; then  
25             echo "ERROR: $newfile exists already"  
26         else  
27             echo "renaming $file to $newfile http://www.cnblogs.com/Images/dot.gif"  
28             mv "$file" "$newfile"  
29         fi  
30 fi  
31 done

我们从头来看，前面两行上一个例子中已经解释过了，从第三行开始，有新的内容。if语句和其他编程语言相似，都是流程控制语句。它的语法是：

if …; then

…

elif …; then

…

else

…

fi

与其他语言不同，Shell Script中if语句的条件部分要以分号来分隔。第三行中的[]表示条件测试，常用的条件测试有下面几种：

[ -f "$file" ] 判断$file是否是一个文件

[ $a -lt 3 ] 判断$a的值是否小于3，同样-gt和-le分别表示大于或小于等于

[ -x "$file" ] 判断$file是否存在且有可执行权限，同样-r测试文件可读性

[ -n "$a" ] 判断变量$a是否有值，测试空串用-z

[ "$a" = "$b" ] 判断$a和$b的取值是否相等

[ cond1 -a cond2 ] 判断cond1和cond2是否同时成立，-o表示cond1和cond2有一成立

要注意条件测试部分中的空格。在方括号的两侧都有空格，在-f、-lt、=等符号两侧同样也有空格。如果没有这些空格，Shell解释脚本的时候就会出错。

$#表示包括$0在内的命令行参数的个数。在Shell中，脚本名称本身是$0，剩下的依次是$0、$1、$2…、${10}、${11}，等等。$\*表示整个参数列表，不包括$0，也就是说不包括文件名的参数列表。

现在我们明白第三行的含义是如果脚本文件的参数少于三个，则执行if和fi语句之间 的内容。然后，从第四行到第十一行之间的内容在Shell Script编程中被称为Here文档，Here文档用于将多行文本传递给某一命令。Here文档的格式是以<<开始，后跟一个字符串，在 Here文档结束的时候，这个字符串同样也要出现，表示文档结束。在本例中，Here文档被输出给cat命令，也即将文档内容打印在屏幕上，起到显示帮助 信息的作用。

第十二行的exit是Linux的命令，表示退出当前进程。在Shell脚本中可以使用所有的Linux命令，利用上面的cat和exit，从一方面来说，熟练使用Linux命令也可以大大减少Shell脚本的长度。

十四、十五两句是赋值语句，分别将第一和第二参数赋值给变量OLD和NEW。紧接下来的两句是注释，注释下面的两条shift的作用是将参数列表中的第一个和第二个参数删除，后面的参数依次变为新的第一和第二参数，注意参数列表原本也不包括$0。

然后，自二十一行到三十一行是一个循环语句。Shell Script中的循环有下面几种格式：

while [ cond1 ] && { || } [ cond2 ] …; do

…

done

for var in …; do

…

done

for (( cond1; cond2; cond3 )) do

…

done

until [ cond1 ] && { || } [ cond2 ] …; do

…

done

在上面这些循环中，也可以使用类似C语言中的break和continue语句中断 当前的循环操作。第二十一行的循环是将参数列表中的参数一个一个地放入变量file中。然后进入循环，判断file是否为一个文件，如果是文件的话，则用 sed命令搜索和生成新的文件名。sed基本上可以看成一个查找替换程序，从标准输入，例如管道读入文本，并将结果输出到标准输出，sed使用正则表达式 进行搜索。在第二十三行中，backtick（`）的作用是取出两个backtick之间的命令输出结果，在这里，也就是将结果取出赋给变量 newfile。此后，判断newfile是否已经存在，否则就把file改成newfile。这样我们就明白这个脚本的作用了，Shell Script编写的其他脚本与此相似，只不过是语法和用法稍有不同而已。

通过这个例子我们明白了Shell Script的编写规则，但还有几件事情需要讲述一下。

第一个，除了if语句之外，Shell Script中也有类似C语言中多分支结构的case语句，它的语法是：

case var in

pattern 1 )

… ;;

pattern 2 )

… ;;

\*)

… ;;

esac

我们再就下面一个例子，看看case语句的用法。

while getopts vc: OPTION

do

case $OPTION in

c) COPIES=$OPTARG

     ehco "$COPIES";;

v) echo "suyang";;

\?) exit 1;;

esac

done

上面的getopts类似于C语言提供的函数getopts，在Shell Script中，getopts经常和while语句联合起来使用。getopts的语法如下：

getopts option\_string variable

option\_string中包含一串单字符选项，若getopts在命令行参数中 发现了连字符，那么它会将连字符之后的字符与option\_string进行比较，若匹配成功，则把变量variable的值设为该选项，若无匹配，则把 变量的值设为？。有时候，选项还会带一个值，例如-c5等，这时要在option\_string中该选项字母后面加上一个冒号，getopts发现冒号 后，会读取该值，然后将该值放入特殊变量OPTARG中。这个命令比较复杂，如有需要，读者可以详细参阅Shell编写的相关资料。

上面这个循环的作用就是依次取出脚本名称后面的选项，进行处理，如果输入了非法选项，则进入"?指定的部分，退出脚本程序。

第二个，Bash提供了一种用于交互式应用的扩展select，用户可以从一组不同的值中进行选择。其语法如下：

select var in …; do

break;

done

例如，下面这段程序的输出是：

#!/bin/bash

echo "Your choice?"

select var in "a" "b" "c"; do

break

done

echo $var

----------------------------

Your choice?

1) a

2) b

3) c

第三，Shell Script中也可以使用自定义的函数，其语法形式如下：

functionname()

{

…

}

例如我们可以把上面第二个例子中第四到第十二行放入一个名为help函数体内，以后每次调用的时候直接写help即可。函数中处理函数调用参数的方法是，直接用上面讲过的$1、$2来分别表示第一、第二个参数，用$\*表示参数列表。

第四，我们也可以在Shell下调试Shell Script脚本，当然最简单的方法就是用echo输出查看变量取值了。Bash也提供了真正的调试方法，就是执行脚本的时候用-x参数。

sh ?x filename.sh

这会执行脚本并显示脚本中所有变量的取值，也可以使用参数-n，它并不执行脚本，只是返回所有的语法错误。