Linux Shell简明教程（二）

一篇迟到的总结

本来说好在总结完第一篇《Linux Shell简明教程（一）》，就接着总结这篇的，但是中间出了很多事情，一直没有进行总结，这篇文章的大纲都写好了，就是一直没有去完善那些细节部分。今天终于有时间了，就开始陆续的完成这篇文章吧。

这篇文章主要还是对Linux Shell中的一些常用命令进行总结，这里将要总结的命令在你日后的工作中，或者阅读别人的代码中，都会或多或少的使用。不管用不用吧，至少能学点东西，废话少说，来吧。

break命令

就像其它语言中的一样，break的意思同它们都一样，就是跳出while或者for等循环，例如以下代码：

```
#!/bin/bash

while：
do
    echo "Please input your choose:"
    read input

    case "$input" in
        1 ) echo "Your input number is 1.";;
        2 ) echo "Your input number is 2.";;
        3 ) echo "Your input number is 3.";;
        4 ) echo "Your input number is 4.";;
        5 ) echo "It's over."
            break #这里跳出while循环
           ;;
        * ) echo "Your input number is too larger.";;
    esac
done
exit 0
```

请注意，在case语句中，执行完每个分支以后，就会结束，并不会像C++那样，还会依次执行它下面的分支语句。

: 命令

Shell命令总是有一些奇怪的命令符，比如之前看到的“[”；这里又来了一个。你没有看错，“:”这货也是有特殊含义的。

看上面break命令的示例代码，看到while后面跟着的“:”了么？它就是这么用的，它是true的别名，就是表示真，再例如以下代码：

```
#!/bin/bash

if : # 有木有看到
then
    echo ": is true"
fi

if true
then
    echo "true is true"
fi
```

就是这样的了，由于“:”是内置命令，所以它的运行速度比true快。

continue命令

break和continue命令，和其它语言中的意思是一样的。break是结束整个循环动作；而continue是结束这一个单次循环，而整个循环动作不会终止。这里就不举例说明了。

eval命令

eval命令允许对参数进行求值，它的格式如下：

```
eval [arg ...]
```

把参数当做一个Shell命令来执行。这是对eval最简单的解释了。eval命令会将参数连接为一个整的字符串，然后将这个字符串作为一个新的Shell输入进行执行。

当我们将一些字符串作为eval的参数时，Shell在执行整个eval命令时，它会对eval命令的参数进行两次扫描。例如以下示例代码：

```
#!/bin/bash

a=10
x=a

eval y='$'$x
echo $y

exit 0
```

在eval执行的过程中，先对`y='$'$x`进行第一次扫描，得到`y=$a`；然后再进行第二次扫描，也就是执行`y=$a`，得到了`y=10`；当执行`echo $y`的时候，就输出结果10。

exit n命令

exit命令使脚本以退出码n结束运行。如果允许自己的脚本程序在退出时不指定一个退出状态，那么该脚本中最后一条被执行命令的状态将被用作为返回值。在脚本程序中提供一个退出码总是一个良好的习惯。

在Shell脚本编程中，退出码0表示成功，退出码1~125是脚本程序可以使用的错误代码。其余数字具有保留的含义，具体请Google。

expr命令

`expr`命令是一个手工命令行计数器，用于在UNIX/LINUX下求表达式变量的值，一般用于整数值，也可用于字符串。以下通过实际的简单代码来看看`expr`的使用。

```
#!/bin/bash
# 获取字符串的长度
expr length "http://www.jellythink.com"

#截取子串
expr substr "http://www.jellythink.com" 12 21

#整数运算

var=`expr 10 / 2`
echo $var

var=$(expr 10 \* 2)
echo $var

exit 0
```

在上面的脚本中，使用了反引号(“)，使用了该引号，就可以使expr的执行结果赋值给var变量，当然了，我们在脚本中也看到了$()来替换反引号的用法，这都是可以的。

`expr`命令的功能比较简单、单一，但是很强大。但是需要注意的是，在进行乘法运算时，如果写成：

```
expr 10 * 2
```

这样就会报错，这是由于`*`号在Shell有特殊意义，我们需要加上转义字符`\`。

printf命令

`printf`命令就是带有format功能的echo。它可以对输出的内容进行格式化，就好比C语言中的printf是一样的。比如以下示例代码：

```
#!/bin/bash

printf "%s\n" http://www.jellythink.com
printf "%c + %s = %s\n" 1 10 11
```

字符转换限定符	说明
d	输出一个十进制数字
c	输出一个字符
s	输出一个字符串
%	输出一个%字符

return命令

return命令的作用是使函数返回。return命令有一个数值参数，这个参数在调用该函数的脚本程序中被看做是该函数的返回值；如果没有指定参数，return命令默认返

回最后一条命令的退出码。

set命令

set命令作用主要是显示系统中已经存在的shell变量，以及为已经存在的shell变量设定新的值。使用set更改shell特性时，符号”+”和”-“的作用分别是打开和关闭指定的模式。set命令不能够定义新的shell变量。如果要定义新的变量，可以使用declare命令以`变量名=值`的格式进行定义即可。

当使用set命令不带任何参数时，它会输出系统中已经存在的所有Shell变量；我们可以使用set设置我们本地的Shell变量，当退出Shell时，这些本地的Shell变量就会失效，如果需要永久的保存这些Shell变量，则需要写入配置文件中。

shift命令

shift命令把所有参数变量左移一个位置，使2变成1，3变成2，以此类推；而2占领1的位置之后，原来$1的值就会被丢弃。

在扫描处理脚本程序的时候，经常要用到shift命令。如果你的脚本命令程序需要10个或10个以上的参数，我们就需要使用shift命令来访问第十个及其后面的参数。

trap命令

trap命令用于指定在接收到信号后将要采取的行动。trap命令的一种常见用途是在脚本程序中断时完成清理工作。在Linux系统中定义了许多信号，它们被定在头文件signal.h中，在使用信号名时需要省略SIG前缀。我们在命令行下可以使用`trap -l`查看所有的信号量定义，及其对应的数字。

trap有两个参数，第一个参数是接收到指定信号时将要采取的行动，第二个参数是要处理的信号名。格式如下：

```
trap command signal
```

下面就通过一个简单的实例来学习一下trap命令的使用，代码如下：

```
# !/bin/bash

# 当按下ctrl+c触发中断信号以后，就删除临时文件
trap 'rm -f /tmp/my_tmp_file_$$' INT
    echo creating file /tmp/my_tmp_file_$$
date > /tmp/my_tmp_file_$$

echo "Press interrupt (ctrl + c) to interrupt..."
while [ -f /tmp/my_tmp_file_$$ ]
do
    echo file exists
    sleep 1
done

echo file is no longer exists

#取消INT信号对应的处理
trap INT
echo creating file /tmp/my_tmp_file_$$
    date > /tmp/my_tmp_file_$$

echo "Press interrupt (ctrl + c) to interrupt..."
while [ -f /tmp/my_tmp_file_$$ ]
do
    echo file exists
    sleep 1
done

echo I never get here
exit 0
```

当按下`ctrl + c`组合键时，就会触发INT中断，然后就会删除创建的临时文件。

unset命令

unset命令的作用是从环境中删除变量或函数，这个命令不能删除shell本身定义的只读变量。

命令的执行

我们在编写脚本的时候，经常需要捕获一条命令的执行结果，然后将这个结果保存在另一个变量中。在总结`expr`命令时，说到可以使用“符号。是的，不错，使用反引号确实可以完成，但是我们更建议使用$(command)来完成这项工作。

`$(command)`的结果就是其中命令的输出。注意，这不是该命令的退出状态，而是它的字符串形式的输出结果。比如以下脚本代码：

```
x=$(date)
```

```
echo $x
```

我们将date命令的输出结果保存在x变量中，然后就可以输出对应的结果。

这种把命令的执行结果存储到变量中的功能是巨大的，它使得在脚本程序中使用现有命令并捕获其输出变得很容易；这样我们可以配合xargs命令，将标准输出作为另一个程序的参数。

算术扩展

在上面总结了`expr`命令可以完成简单的算术运算。但是这个命令执行起来是非常的慢得，因为它需要调用一个新的shell来处理expr命令。

一种更好的方式是使用`$((...))`扩展，把我们准备求值的表达式放在`$((...))`中就能够更有效地完成简单的算术运算。例如以下代码：

```
# !/bin/bash

x=0
while [ "$x" -ne 10 ]
do
   echo $x
   x=$(($x+1))
done

exit 0
```

在使用的时候一定要记住了，是双括号。

参数扩展

首先看一段简单的代码：

```
# !/bin/bash

for ((i=0;i<=10;++i))
do
   echo $i_JellyThink
done
exit 0
```

我们期望得到的是0_JellyThink、1_JellyThink…等值，但是实际上是什么也不会得到。这是为什么呢？

问题出在于Shell试图替换变量`$i_JellyThink`中`$i`的值的时候，理解为替换`$i_JellyThink`的值，而实际上不存在`$i_JellyThink`这个变量，所以就被替换成空值了。在实际写代码时，为了保护变量名中类似于`$i`部分的扩展，我们需要把i放在花括号中，如下：

```
echo ${i}_JellyThink
```

这样就不会出现问题了，对于参数扩展，还有很多内容，这里是总结不全面的，如果需要，请各位自行Google吧。

调试脚本程序

写程序，肯定是离不开调试的，就是你再NB，也很少把复杂的东西一次搞定吧。所以，调试就是每个程序员必备的一项技能。

对于Shell的调试，内容比较多，这里推荐一篇文章，好好阅读这篇文章就足以了，搞定Shell的调试就不再是问题。看这里

总结

这篇文章总结的比较零碎，也是花费业余时间来总结的，很多地方可能总结的不全面，后期查阅的时候，再补吧。

2014年12月21日 于深圳。