第15章 shell脚本

学习Linux请加QQ群： 群1(163262181) 群2(148412746) 群3(246401509) 群4(173884211)

跟阿铭学Linux邀请函 (http://www.aminglinux.com)，猿课已上线，请加微信aminglinux84索要配套视频教程。

终于到shell 脚本这章了，在以前阿铭卖了好多关子说shell脚本怎么怎么重要，确实shell脚本在linux系统管理员的运维工作中非常非常重要。下面阿铭就带你正式进入shell脚本的世界吧。

到现在为止，你明白什么是shell脚本吗？如果明白最好了，不明白也没有关系，相信随着学习的深入你就会越来越了解到底什么是shell脚本。首先它是一个脚本，并不能作为正式的编程语言。因为是跑在linux的shell中，所以叫shell脚本。说白了，shell脚本就是一些命令的集合。举个例子，我想实现这样的操作：

1）进入到/tmp/目录；

2）列出当前目录中所有的文件名；

3）把所有当前的文件拷贝到/root/目录下；

4）删除当前目录下所有的文件。

简单的4步在shell窗口中需要你敲4次命令，按4次回车。这样是不是很麻烦？当然这4步操作非常简单，如果是更加复杂的命令设置需要几十次操作呢？那样的话一次一次敲键盘会很麻烦。所以不妨把所有的操作都记录到一个文档中，然后去调用文档中的命令，这样一步操作就可以完成。其实这个文档呢就是shell脚本了，只是这个shell脚本有它特殊的格式。

Shell脚本能帮助我们很方便的去管理服务器，因为我们可以指定一个任务计划定时去执行某一个shell脚本实现我们想要需求。这对于linux系统管理员来说是一件非常值得自豪的事情。现在的139邮箱很好用，发邮件的同时还可以发一条邮件通知的短信给用户，利用这点，我们就可以在我们的linux服务器上部署监控的shell脚本，比如网卡流量有异常了或者服务器web服务器停止了就可以发一封邮件给管理员，同时发送给管理员一个报警短信这样可以让我们及时的知道服务器出问题了。

在正式写shell脚本之前阿铭先给你一条建议：凡是自定义的脚本建议放到/usr/local/sbin/目录下，这样做的目的是，一来可以更好的管理文档；二来以后接管你的管理员都知道自定义脚本放在哪里，方便维护。

shell脚本的基本结构以及如何执行

下面请跟着阿铭写第一个你的shell脚本吧：

```
[root@localhost ~]# cd /usr/local/sbin/
[root@localhost sbin]# vim first.sh
#! /bin/bash

## This is my first shell script.
## Writen by Aming 2013-05-24.

date
echo "Hello world!"
```

Shell脚本通常都是以.sh 为后缀名的，这个并不是说不带.sh这个脚本就不能执行，只是大家的一个习惯而已。所以，以后你发现了.sh为后缀的文件那么它可能是一个shell脚本了。test.sh中第一行要以 “#!/bin/bash” 开头，它代表的意思是，该文件使用的是bash语法。如果不设置该行，虽然你的shell脚本也可以执行，但是这不符合规范。 # 表示注释，在前面讲过的。后面跟一些该脚本的相关注释内容以及作者和创建日期或者版本等等。当然这些注释并非必须的，如果你懒的很，可以省略掉，但是阿铭不建议省略。因为随着工作时间的逐渐过渡，你写的shell脚本也会越来越多，如果有一天你回头查看自己写过的某个脚本时，很有可能忘记该脚本是用来干什么的以及什么时候写的。所以写上注释是有必要的。另外系统管理员并非只有你一个，如果是其他管理员查看你的脚本，他看不懂岂不是很郁闷。下面该运行一下这个脚本了：

```
[root@localhost sbin]# sh first.sh
2013年 05月 24日 星期五 18:58:02 CST
Hello world!
```

其实shell脚本还有一种执行方法就是：

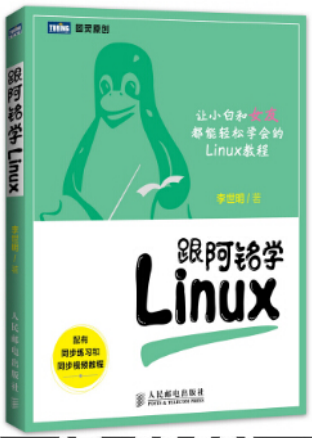
```
[root@localhost sbin]# ./first.sh
-bash: ./first.sh: 权限不够
[root@localhost sbin]# chmod +x first.sh
[root@localhost sbin]# ./first.sh
2013年 05月 24日 星期五 18:58:51 CST
Hello world!
```

要想使用该种方法运行shell脚本，前提是脚本本身有执行权限，所以需要给脚本加一个 ‘x’ 权限。另外使用sh命令去执行一个shell脚本的时候是可以加-x选项来查看这个脚本执行过程的，这样有利于我们调试这个脚本哪里出了问题：

目录列表

第1章 前言
第2章 关于Linux的历史
第3章 对Linux系统管理员的建议
第4章 安装Linux操作系统
第5章 初步认识Linux
第6章 Linux系统的远程登陆
第7章 Linux文件与目录管理
第8章 Linux系统用户及用户组管理
第9章 Linux磁盘管理
第10章 文本编辑工具vim
第11章 文档的压缩与打包
第12章 安装RPM包或者安装源码包
第13章 学习 shell脚本之前的基础知识
第14章 正则表达式
第15章 shell脚本
第16章 linux系统日常管理
第17章 LAMP环境搭建
第18章 LNMP环境搭建
第19章 学会使用简单的MySQL操作
第20章 NFS服务配置
第21章 配置FTP服务
第22章 配置Squid服务
第23章 配置Tomcat
第24章 配置Samba服务器
第25章 MySQL replication(主从)配置
结语

阿铭著作：



微信扫码获取最新版linux电子书和视频

```
[root@localhost sbin]# sh -x first.sh
+ date
2013年 05月 24日 星期五 20:00:11 CST
+ echo 'Hello world!'
Hello world!
```

本例中有一个命令 `date` 之前阿铭从未介绍过，这个命令在shell脚本中使用非常频繁，阿铭有必要向你介绍一下它的用法。

命令：date

阿铭举几个比较实用的例子来带你掌握它的用法：

```
[root@localhost sbin]# date +"%Y-%m-%d %H:%M:%S"
2013-05-24 19:41:01
```

date在脚本中最常用的几个用法：

`data +%Y` 以四位数字格式打印年份

`date +%y` 以两位数字格式打印年份

`date +%m` 月份

`date +%d` 日期

`date +%H` 小时

`date +%M` 分钟

`date +%S` 秒

`date +%w` 星期，如果结果显示0 则表示周日

有时在脚本中会用到一天前的日期：

```
[root@localhost sbin]# date -d "-1 day" +%d
23
```

或者一小时前：

```
[root@localhost sbin]# date -d "-1 hour" +%H
18
```

甚至1分钟前：

```
[root@localhost sbin]# date -d "-1 min" +%M
50
```

shell脚本中的变量

在shell脚本中使用变量显得我们的脚本更加专业更像是一门语言，变量的作用当然不是为了专业。如果你写了一个长达1000行的shell脚本，并且脚本中出现了某一个命令或者路径几百次。突然你觉得路径不对想换一下，那岂不是要更改几百次？你固然可以使用批量替换的命令，但也是很麻烦，并且脚本显得臃肿了很多。变量的作用就是用来解决这个问题的。

```
[root@localhost sbin]# cat variable.sh
#! /bin/bash

## In this script we will use variables.
## Writen by Aming 2013-05-24.

d=`date +%H:%M:%S`
echo "The script begin at $d."
echo "Now we'll sleep 2 seconds."
sleep 2
d1=`date +%H:%M:%S`
echo "The script end at $d1."
```

脚本中使用到了反引号，你是否还记得它的作用？ `d` 和 `d1` 在脚本中作为变量出现，定义变量的格式为 `变量名=变量的值` 当在脚本中引用变量时需要加上 `$` 符号，这跟前面讲的在shell中自定义变量是一致的。下面看看脚本执行结果吧：

```
[root@localhost sbin]# sh variable.sh
The script begin at 20:16:57.
Now we'll sleep 2 seconds.
The script end at 20:16:59.
```

下面我们用shell计算两个数的和：

```
[root@localhost sbin]# cat sum.sh
#! /bin/bash
```

SEARCH

Go

Enter search terms or a module, class or function name.

```
## For get the sum of tow numbers.
## Aming 2013-05-24.

a=1
b=2
sum=$[$a+$b]

echo "$a+$b=$sum"
```

数学计算要用[ ]括起来并且外头要带一个‘$’脚本结果为:

```
[root@localhost sbin]# sh sum.sh
1+2=3
```

Shell脚本还可以和用户交互:

```
[root@localhost sbin]# cat read.sh
#! /bin/bash

## Using 'read' in shell script.
## Aming 2013-05-24.

read -p "Please input a number: " x
read -p "Please input another number: " y
sum=$[$x+$y]
echo "The sum of the two numbers is: $sum"
```

read 命令就是用在这样的地方，用于和用户交互，把用户输入的字符串作为变量值。脚本执行过程如下:

```
[root@localhost sbin]# sh read.sh
Please input a number: 2
Please input another number: 10
The sum of the two numbers is: 12
```

我们不妨加上 -x 选项再来看看这个执行过程:

```
[root@localhost sbin]# sh -x read.sh
+ read -p 'Please input a number: ' x
Please input a number: 22
+ read -p 'Please input another number: ' y
Please input another number: 13
+ sum=35
+ echo 'The sum of the two numbers is: 35'
The sum of the two numbers is: 35
```

有时候我们会用到这样的命令 `/etc/init.d/iptables restart` 前面的/etc/init.d/iptables文件其实就是一个shell脚本，为什么后面可以跟一个“restart”？这里就涉及到了shell脚本的预设变量。实际上，shell脚本在执行的时候后边是可以跟参数的，而且还可以跟多个。

```
[root@localhost sbin]# cat option.sh
#! /bin/bash

sum=$[$1+$2]
echo "sum=$sum"
```

执行结果为:

```
[root@localhost sbin]# sh -x option.sh 1 2
+ sum=3
+ echo sum=3
sum=3
```

在脚本中，你会不会奇怪，哪里来的$1和$2，这其实就是shell脚本的预设变量，其中$1的值就是在执行的时候输入的1，而$2的值就是执行的时候输入的$2，当然一个shell脚本的预设变量是没有限制的，这回你明白了吧。另外还有一个$0，不过它代表的是脚本本身的名字。不妨把脚本修改一下:

```
[root@localhost sbin]# cat option.sh
#! /bin/bash

echo "$1 $2 $0"
```

执行结果:

```
[root@localhost sbin]# sh option.sh  1 2
1 2 option.sh
```

shell脚本中的逻辑判断

如果你学过C或者其他语言，相信你不会对 if 陌生，在shell脚本中我们同样可以使用 if 逻辑判断。在shell中if判断的基本语法为：

1）不带else

```
if  判断语句; then
    command
fi
```

例如：

```
[root@localhost sbin]# cat if1.sh
#! /bin/bash

read -p "Please input your score: " a
if (($a<60)); then
    echo "You didn't pass the exam."
fi
```

在if1.sh中出现了 (($a<60)) 这样的形式，这是shell脚本中特有的格式，用一个小括号或者不用都会报错，请记住这个格式。执行结果为：

```
[root@localhost sbin]# sh if1.sh
Please input your score: 90
[root@localhost sbin]# sh if1.sh
Please input your score: 33
You didn't pass the exam.
```

2）带有else

```
if  判断语句  ; then
    command
else
    command
fi
```

例如：

```
[root@localhost sbin]# cat if2.sh
#! /bin/bash

read -p "Please input your score: " a
if (($a<60)); then
     echo "You didn't pass the exam."
else
     echo "Good! You passed the exam."
fi
```

执行结果：

```
[root@localhost sbin]# sh if2.sh
Please input your score: 80
Good! You passed the exam.
[root@localhost sbin]# sh if2.sh
Please input your score: 25
You didn't pass the exam.
```

和上一例唯一区别的地方是，如果输入大于等于60的数字会有所提示。

3）带有elif

```
if  判断语句一  ; then
    command
elif  判断语句二; then
    command
else
    command
fi
```

例如：

```
[root@localhost sbin]# cat if3.sh
#! /bin/bash

 read -p "Please input your score: " a
 if (($a<60)); then
```

```
        echo "You didn't pass the exam."
 elif (($a>=60)) && (($a<85)); then
        echo "Good! You pass the exam."
 else
        echo "very good! Your socre is very high!"
 fi
```

这里的 && 表示“并且”的意思，当然也可以使用 || 表示“或者”执行结果为:

```
[root@localhost sbin]# sh if3.sh
Please input your score: 90
very good! Your socre is very high!
[root@localhost sbin]# sh if3.sh
Please input your score: 60
Good! You pass the exam.
```

以上只是简单的介绍了if语句的结构。在判断数值大小除了可以用 (( )) 的形式外，还可以使用 [ ] 但是就不能使用>, < , = 这样的符号了，要使用 -lt （小于），-gt （大于），-le （小于等于），-ge （大于等于），-eq （等于），-ne （不等于）。下面阿铭就以命令行的形式简单比较一下，不再写shell脚本了。

```
[root@localhost sbin]# a=10; if [ $a -lt 5 ]; then echo ok; fi
[root@localhost sbin]# a=10; if [ $a -gt 5 ]; then echo ok; fi
ok
[root@localhost sbin]# a=10; if [ $a -ge 10 ]; then echo ok; fi
ok
[root@localhost sbin]# a=10; if [ $a -eq 10 ]; then echo ok; fi
ok
[root@localhost sbin]# a=10; if [ $a -ne 10 ]; then echo ok; fi
```

再看看if中使用 && 和 ||的情况:

```
[root@localhost sbin]# a=10; if [ $a -lt 1 ] || [ $a -gt 5 ]; then echo ok; fi
ok
[root@localhost sbin]# a=10; if [ $a -gt 1 ] || [ $a -lt 10 ]; then echo ok; fi
ok
```

shell 脚本中if还经常判断关于档案属性，比如判断是普通文件还是目录，判断文件是否有读写执行权限等。常用的也就几个选项:

-e ： 判断文件或目录是否存在

-d ： 判断是不是目录，并是否存在

-f ： 判断是否是普通文件，并存在

-r ： 判断文档是否有读权限

-w ： 判断是否有写权限

-x ： 判断是否可执行

使用if判断时，具体格式为:

```
if [ -e filename ] ; then
```

例子:

```
[root@localhost sbin]# if [ -d /home/ ]; then echo ok; fi
ok
[root@localhost sbin]# if [ -f /home/ ]; then echo ok; fi
```

因为 /home/ 为目录为非文件，所以并不会显示“ok”.

```
[root@localhost sbin]# if [ -f /root/test.txt ]; then echo ok; fi
ok
[root@localhost sbin]# if [ -r /root/test.txt ]; then echo ok; fi
ok
[root@localhost sbin]# if [ -w /root/test.txt ]; then echo ok; fi
ok
[root@localhost sbin]# if [ -x /root/test.txt ]; then echo ok; fi
[root@localhost sbin]# if [ -e /root/test1.txt ]; then echo ok; fi
```

在shell 脚本中，除了用if来判断逻辑外，还有一种常用的方式，那就是case了。具体格式为:

```
case  变量  in
value1)
          command
          ;;
value2)
          command
          ;;
value3)
          command
```

```
        ;;
*)
        command
        ;;
esac
```

上面的结构中，不限制value的个数， `*` 则代表除了上面的value外的其他值。下面阿铭写一个判断输入数值是奇数或者偶数的脚本：

```
[root@localhost sbin]# cat case.sh
#! /bin/bash

read -p "Input a number: " n
a=$[$n%2]
case $a in

    1)
        echo "The number is odd."
        ;;
    0)
        echo "The number is even."
        ;;
    *)
        echo "It's not a number!"
        ;;
esac
```

$a 的值或为1或为0，执行结果为：

```
[root@localhost sbin]# sh case.sh
Input a number: 100
The number is even.
[root@localhost sbin]# sh case.sh
Input a number: 101
The number is odd.
```

case脚本常用于编写系统服务的启动脚本，例如/etc/init.d/iptables中就用到了，你不妨去查看一下。

shell脚本中的循环

Shell脚本中也算是一门简易的编程语言了，当然循环是不能缺少的。常用到的循环有for循环和while循环。下面就分别介绍一下两种循环的结构。

1. for循环

```
[root@localhost sbin]# cat for.sh
#!  /bin/bash

for i in `seq 1 5`; do
    echo $i
done
```

脚本中的 `seq 1 5` 表示从1到5的一个序列。你可以直接运行这个命令试下。脚本执行结果为：

```
[root@localhost sbin]# sh for.sh
1
2
3
4
5
```

通过这个脚本就可以看到for循环的基本结构：

```
for 变量名 in 循环的条件； do
     command
done
```

这里的 “循环的条件” 可以写成一组字符串或者数字(用1个或者多个空格隔开)， 也可以是一条命令的执行结果：

```
[root@localhost sbin]# for i in 1 2 3 a b; do echo $i; done
1
2
3
a
```

```
b
```

也可以写引用系统命令的执行结果，就像那个 `seq 1 5` 但是需要用反引号括起来：

```
[root@localhost sbin]# for file in `ls`; do echo $file; done
case.sh
first.sh
for.sh
if1.sh
if2.sh
if3.sh
option.sh
read.sh
sum.sh
variable.sh
```

2. **while循环**

```
[root@localhost sbin]# cat while.sh
#! /bin/bash

a=5
while [ $a -ge 1 ]; do
    echo $a
    a=$[$a-1]
done
```

while 循环格式也很简单：

```
while  条件; do

          command
done
```

上例脚本的执行结果为：

```
[root@localhost sbin]# sh while.sh
5
4
3
2
1
```

另外你可以把循环条件拿一个冒号替代，这样可以做到死循环，阿铭常常这样写监控脚本：

```
while :; do
    command
    sleep 3
done
```

shell脚本中的函数

如果你学过开发，肯定知道函数的作用。如果你是刚刚接触到这个概念的话，也没有关系，其实很好理解的。函数就是把一段代码整理到了一个小单元中，并给这个小单元起一个名字，当用到这段代码时直接调用这个小单元的名字即可。有时候脚本中的某段代总是重复使用，如果写成函数，每次用到时直接用函数名代替即可，这样就节省了时间还节省了空间。

下面阿铭写一个简单的带有函数功能的shell脚本：

```
[root@localhost sbin]# cat func.sh
#! /bin/bash

function sum()
{
    sum=$[$1+$2]
    echo $sum
}

sum $1 $2
```

执行结果如下：

```
[root@localhost sbin]# sh func.sh 1 2
3
```

func.sh中的 sum() 为自定义的函数，在shell脚本函数的格式为：

```
function 函数名() {

command

}
```

有一点阿铭要提醒你一下，在shell脚本中，函数一定要写在最前面，不能出现在中间或者最后，因为函数是要被调用的，如果还没有出现就被调用，肯定是会出错的。

shll脚本练习题

Shell脚本大体上就介绍这么多了，阿铭所举的例子都是最基础的，所以即使你把所有例子完全掌握也不代表你的shell脚本编写能力有多么好。所以剩下的日子里请你尽量要多练习，多写脚本，写的脚本越多，你的shell脚本能力就越强。希望你能够找专门介绍shell脚本的书籍深入的去研究一下它。随后阿铭将留几个shell脚本的练习题，你最好不要偷懒。

1. 编写shell脚本，计算1-100的和；
2. 编写shell脚本，要求输入一个数字，然后计算出从1到输入数字的和，要求，如果输入的数字小于1，则重新输入，直到输入正确的数字为止；
3. 编写shell脚本，把/root/目录下的所有目录（只需要一级）拷贝到/tmp/目录下；
4. 编写shell脚本，批量建立用户user_00, user_01, ... user_100并且所有用户同属于users组；
5. 编写shell脚本，截取文件test.log中包含关键词 'abc' 的行中的第一列（假设分隔符为 ":" ），然后把截取的数字排序（假设第一列为数字），然后打印出重复次数超过10次的列；
5. 编写shell脚本，判断输入的IP是否正确（IP的规则是，n1.n2.n3.n4，其中1<n1<255, 0<n2<255, 0<n3<255, 0<n4<255）。

习题答案:

```
1. #! /bin/bash

sum=0

for i in `seq 1 100`; do
        sum=$[$i+$sum]
done

echo $sum

2. #! /bin/bash

n=0
while [ $n -lt "1" ]; do
        read -p "Please input a number, it must greater than "1":" n
done

sum=0

for i in `seq 1 $n`; do
        sum=$[$i+$sum]
done

echo $sum


3. #! /bin/bash

cd /root
for f in `ls `; do
        if [ -d $f ] ; then
                cp -r $f /tmp/
        fi
done



4. #! /bin/bash

groupadd users

for i in `seq 0 9`; do
        useradd -g users user_0$i
done
```

```
for j in `seq 10 100`; do
        useradd -g users user_$j
done
```

5.
```
#! /bin/bash

awk -F':' '$0~/abc/ {print $1}' test.log >/tmp/n.txt
sort -n n.txt |uniq -c |sort -n >/tmp/n2.txt
awk '$1>10 {print $2}' /tmp/n2.txt
```

6.
```
#! /bin/bash

checkip() {

        if echo $1 |egrep -q '^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$' ; then
                a=`echo $1 | awk -F. '{print $1}'`
                b=`echo $1 | awk -F. '{print $2}'`
                c=`echo $1 | awk -F. '{print $3}'`
                d=`echo $1 | awk -F. '{print $4}'`

                for n in $a $b $c $d; do
                        if [ $n -ge 255 ] || [ $n -le 0 ]; then
                                echo "the number of the IP should less than 255 and greate than 0"
                                return 2
                        fi
                done
        else

                echo "The IP you input is something wrong, the format is like 192.168.100.1"
                return 1
        fi

}



rs=1
while [ $rs -gt 0 ]; do
    read -p  "Please input the ip:" ip
    checkip $ip
    rs=`echo $?`
done

echo "The IP is right!"
```

阿铭建议你最好再扩展学习一下: http://www.aminglinux.com/bbs/thread-5439-1-1.html

教程答疑: 请移步这里.

欢迎你加入 阿铭学院 和阿铭一起学习Linux，让阿铭成为你Linux生涯中永远的朋友吧！

© Copyright 2013, lishiming.net. Created using Sphinx 1.3b1网站统计 .