Shell脚本编程入门

- 前言：

我们平时用的命令行，其实就是Shell。计算机真正工作的是各种各样的硬件，但是人不可能直接和硬件对话，所以需要走一个流程来进行沟通。那就是发送指令到shell命令行，命令行把指令翻译给Linux内核，内核翻译给底层的硬件，硬件开始工作。所以说，shell是一个桥梁，像学各大语言一样，身为运维我们也需要学会这个桥梁的用法（各种语法），以此来和内核沟通。

- 正文：

1、初步认识shell

看完前言，有的人就会有疑惑了，我们用的命令行不是叫做bash吗。没错，shell只是一个统称，它旗下有着bash、ksh、zsh、csh等多种类型命令行，Linux发行版使用bash作为默认shell，bash也是使用率最高的shell。前面也说过，Linux不同版本是内核不同，底层实现有区别，连通它们的shell只是一个沟通渠道，没区别，所以Linux指的是内核，shell只是外壳、工具，但是这个工具掌握好了能使我们更高效地命令Linux。

2、几行命令做出第一个脚本

shell除了执行命令外，还可程序编程，说白了shell命令行本身就是一个编程平台，可以在上面直接写程序。话不多说，直接实践：

```
[root@localhost shell]# ls
myfirst_shell.sh
[root@localhost shell]# cat myfirst_shell.sh
#!/bin/bash
echo "this is my first shell script"
hostname
whoami
id
echo "that is all.Very easy~"
[root@localhost shell]# chmod +x myfirst_shell.sh
[root@localhost shell]# ls -l myfirst_shell.sh
-rwxr-xr-x. 1 root root 98 Sep 24 10:53 myfirst_shell.sh
[root@localhost shell]# bash myfirst_shell.sh
this is my first shell script
localhost.localdomain
root
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c102
that is all.Very easy~
[root@localhost shell]#
```

如图，我在root目录里创建了一个shell目录，然后touch一个.sh文件，并且vim打开它，输入图示内容，最后加上 x 权限，使得它能被bash执行。运行一个脚本的方法就是bash+脚本文件名。是不是很有意思？另外提一嘴，以前我们进程能简单到echo，其实它的作用就是在命令行上输出一行指定的文字（字符串或数字）。

整个编程的代码什么意思呢？其实就是逐条执行命令，和之前学crontab很像。由于这种是按照用户的意愿逐步执行某些具体的操作，比较简单，不像高级语言有着复杂的结构，shell就是你叫它干啥马上干啥，不用经过编译，比较接近自然语言，所以我们管shell叫脚本语言。

```
[root@localhost shell]# echo "this is my first shell script"
this is my first shell script
[root@localhost shell]# hostname
localhost.localdomain
[root@localhost shell]# whoami
root
[root@localhost shell]# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c
[root@localhost shell]# echo "that is all.Very easy~"
that is all.Very easy~
[root@localhost shell]# _
```

我也可以直接命令行要求输出，和上面在.sh里编程一样的结果。但是如果命令多了，而且要重复使用，肯定是写个脚本舒服一点。我们学shell脚本编程，其实变相学命令。

3、先从变量学起

```
[root@localhost shell]# ls
myfirst_shell.sh  shell2.sh
[root@localhost shell]# cat shell2.sh
        a="hello"
        b=200
        c=250
        echo $a
        echo $b
        echo $c
        echo a
        echo b
        echo c

        a=300
        let c=a+b
        echo $c

        c=a+b
        echo $c
[root@localhost shell]# bash shell2.sh
hello
200
250
a
b
c
500
a+b
[root@localhost shell]# _
```

和别的语言一样，=号直接赋值。但是这个等号有点厉害，shell不用定义什么类型的变量，直接写变量名，然后等号后面的内容直接传给它（感觉会自己识别的文字和单纯的数字），不管是字符串、字符、数字。echo打印变量内容时，要加上＄美元符号，表示对变量的内容引用（可以理解为一种替代，如＄b被200替代，echo 200就打印出200），没加echo会把后面的东西当成文字内容，直接打印出文字内容。使用let命令就可以计算数值，加减乘除都可，如果不用let，就只是c变量等于a+b这个字符串而已，成了一种赋值。另外，我的shell2.sh没加x执行权限，所以没有绿，但本身有x权限，所以不影响执行。

没啥好说的，有点C基础都是卵杀。

4、变量的规范化定义和用户输入

```
[root@localhost shell]# cat shell3.sh
num1=$1
num2=$2
echo -n "you have input two numbers"
echo -n "$num1 and"
echo "$num2"
let ret_add=num1+num2
let ret_sub=num1-num2
let ret_mul=num1*num2
let ret_div=num1/num2
echo ret of add : $ret_add
echo ret of sub : $ret_sub
echo ret of mul : $ret_mul
echo ret of div : $ret_div
[root@localhost shell]# bash shell3.sh  204
you have input two numbers204 and
shell3.sh: line 9: let: ret_div=num1/num2: division by 0 (error token is "num2")
ret of add : 204
ret of sub : 204
ret of mul : 0
ret of div :
[root@localhost shell]# bash shell3.sh  20 4
you have input two numbers20 and4
ret of add : 24
ret of sub : 16
ret of mul : 80
ret of div : 5
[root@localhost shell]# _
```

小手敲敲键盘，模仿书上打了这段代码。最开始num1=$1，我们都没有给1这个变量初始化过值，那么num1直接用，岂不是有语法问题？其实，1这个变量的值就是等待我们用户输入。如图，只有两个变量都输入，才能出现正确结果。可是我只输入一个，也能得出答案呢，这是因为shell脚本里运行空参数的存在，不会像高级语言那样在编译阶段报错，而是在命令执行不了的时候报错。这就是Linux一个简单的计算器应用了。

5、编写偏向实际应用的脚本

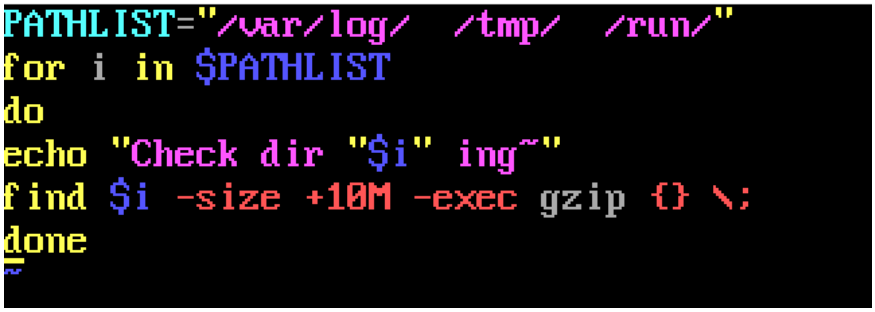
```
i=0
sum=0
while [ $i -le 100 ]
do
    let sum+=$i
    let i++
done
echo $sum
```

这是从1加到100的累加结果的小程序，我们通过它快速地学到while的用法。 -le 这个参数呢，其实是一个判断参数，后面说if时候再说。

```
i=0;
sum=0;
while [ $i -le 100 ]
do
 let sum+=$i;
 let i++;
done
echo $sum;
~
```

然后我发现一个很好玩的事情，因为我有C编程的习惯，所以会不自觉加个分号。我以为这样会报错，但其实不会，如图照样可以出5050。

好吧，这个一点都不实用，确实，接下来给一个真正会用到。



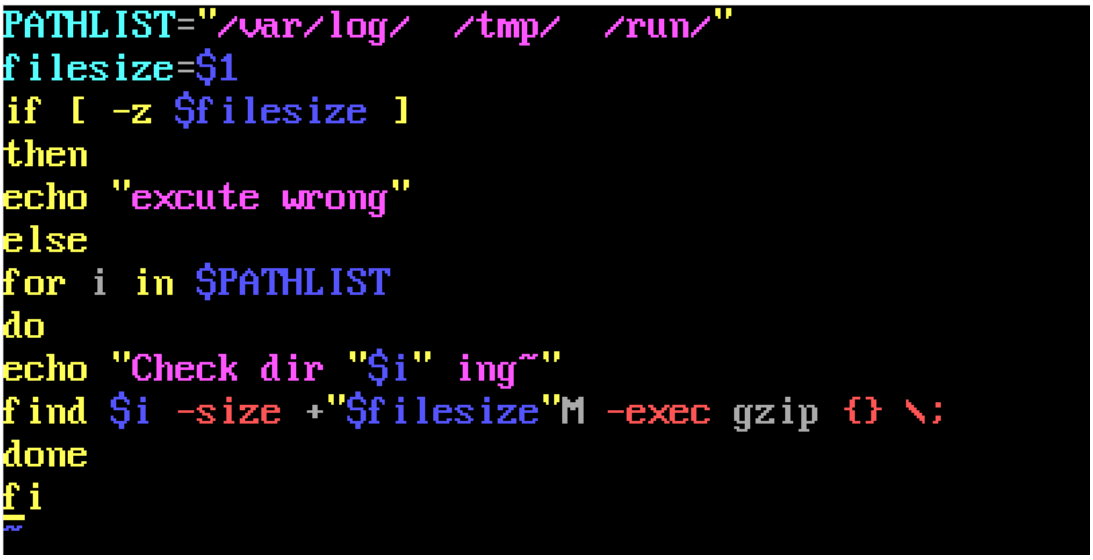
有点太花了，看一下md软件的代码块怎么显示这段代码的吧：

```
#vim fileCheck.sh  用于将目标目录下大于10M的日志文件压缩一波

PATHLIST="/var/log/  /tmp/  /run/"
for i in $PATHLIST
do
echo "Check dir "$i" ing~"
find $i -size +10M -exec gzip {} \;
done
```

现在，这个脚本还不够灵活，我们加入参数输入进去，让用户可以自己选择文件大小。这很简单，悄悄在PATHLIST后加入一个filesize=$1的变量就好了。但是，如果像之前一样用户不输入参数，岂不是又会出错？能不能想办法处理掉这个错误，加强脚本实用性？

所以接下来，我们学习if语法。



就是这么简单， if 中括号接条件，如果成立就 then 后加入条件成立要做的事情，不成立就 else 加上不成立要做的事，最后用 fi 标志 if 代码块已经结束了。下面是改良版，和上面实现了一样的功能，只是 if 提早退出了。另外 -z 表示判断后面的参数是不是空。

```
#vim fileCheck.sh  用于将目标目录下大于10M的日志文件压缩一波，无敌加强版

PATHLIST="/var/log/  /tmp/  /run/"
filesize=$1
if [ -z $filesize ]
then
echo "excute wrong"
exit
fi
```

```
for i in $PATHLIST
do
echo "Check dir "$i" ing~"
find $i -size +"$filesize"M -exec gzip {} \;
done
```

6、比较运算符的简单罗列

算术比较运算符：

①num1 -eq num2 ： 如果num1等于num2。

②num1 -ne num2 ： 如果num1不等于num2。

③num1 -lt num2 ： 如果num1小于num2。

④num1 -le num2 ： 如果num1小于等于num2。

⑤num1 -gt num2 ： 如果num1大于num2。

⑥num1 -ge num2 ： 如果num1大于等于num2。

文件比较运算符：

① -e filename ： 如果filename存在，则为真[-e /var/log/]。

② -d filename ： 如果filename为目录，则为真[-d /tmp/]。

③ -f filename ： 如果filename为常规文件，则为真[-f /usr/bin/ls]。

④ -L filename ： 如果filename为符号链接，则为真[-L /etc/systemd/system/multi-user.target.wants/rpcbind.service]。

⑤ -r filename ： 如果filename可读，则为真[-r /var/log/syslog]。

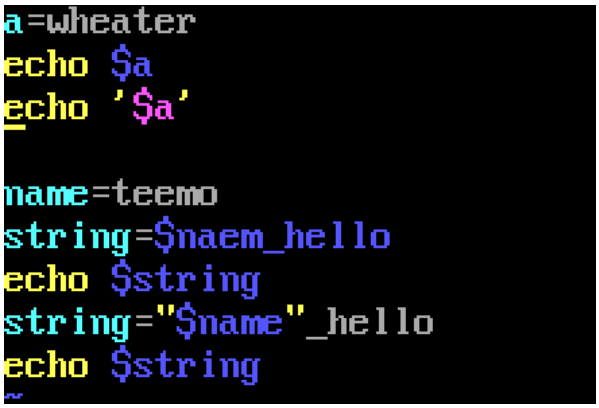
⑥ -w filename ： 如果filename可写，则为真[-w /var/mylog.txt]。

⑦ -x filename ： 如果filename可执行，则为真[-x /usr/bin/grep]。

7、三种引号作用

①单引号：纯纯的字符串，使得里面的内容保持原本的样子。

②双引号：保持里面各种特殊符号和变量、命令的作用，可以隔离字段。



至于会输出什么，其实从颜色里都能猜出个大概了：

```
[root@MiWiFi-R4CM-srv shell]# bash shell2.sh
wheater
$a

teemo_hello
[root@MiWiFi-R4CM-srv shell]# _
```

③反引号：先把里面的内容优先执行。键盘上就是在Esc键下面的那个，就是 ~ 波浪号这个键。

```
[root@MiWiFi-R4CM-srv shell]# cat shell5.sh
list=`ls -l /var/log/`
echo "$list"
[root@MiWiFi-R4CM-srv shell]# bash shell5.sh
total 5048
drwxr-xr-x. 2 root   root       204 Aug 29 16:12 anaconda
drwx------. 2 root   root        23 Aug 29 16:15 audit
-rw-------. 1 root   root    663559 Sep 24 15:28 boot.log
-rw-------. 1 root   utmp     15360 Sep 19 10:24 btmp
drwxr-xr-x. 2 chrony chrony       6 Aug  8  2019 chrony
-rw-------. 1 root   root     37273 Sep 24 16:01 cron
-rw-r--r--. 1 root   root     33102 Sep 24 15:28 dmesg
-rw-r--r--. 1 root   root     32389 Sep 24 10:39 dmesg.old
-rw-r-----. 1 root   root     11160 Sep 24 15:28 firewalld
-rw-r--r--. 1 root   root       193 Aug 29 16:06 grubby_prune_debug
drwx------. 2 root   root         6 Oct  2  2020 httpd
-rw-r--r--. 1 root   root    292584 Sep 24 15:28 lastlog
-rw-------. 1 root   root     18707 Sep 24 15:28 maillog
-rw-------. 1 root   root   4003027 Sep 24 16:01 messages
drwxr-xr-x. 2 root   root         6 Aug  8  2019 qemu-ga
drwxr-xr-x. 2 root   root         6 Aug 29 16:12 rhsm
drwxr-xr-x. 2 root   root       247 Sep 24 10:39 sa
-rw-------. 1 root   root    101115 Sep 24 15:28 secure
-rw-------. 1 root   root         0 Aug 29 16:07 spooler
-rw-------. 1 root   root     64064 Sep 15 07:41 tallylog
drwxr-xr-x. 2 root   root        23 Aug 29 16:15 tuned
-rw-rw-r--. 1 root   utmp    161280 Sep 24 15:28 wtmp
-rw-------. 1 root   root      7367 Sep 20 10:33 yum.log
[root@MiWiFi-R4CM-srv shell]# _
```

先执行 ls 命令，把执行结果按照原来的格式赋值给 list 变量，再用 echo 输出 list 的内容。

8、巧用Linux命令行返回值和运算符号“&&”、“||”

在Linux上，每执行一个命令时，它最后都会存在一个返回数值。如果执行成功，就返回0；反之返回的不是0，可用“$？”获取它。

```
[root@MiWiFi-R4CM-srv shell]# whoami
root
[root@MiWiFi-R4CM-srv shell]# echo $?
0
[root@MiWiFi-R4CM-srv shell]# ls /test7/
ls: cannot access /test7/: No such file or directory
[root@MiWiFi-R4CM-srv shell]# echo $?
2
[root@MiWiFi-R4CM-srv shell]#
[root@MiWiFi-R4CM-srv shell]# cat shell6.sh
ls /root/test5 2>/dev/null
result="$?"
if [ $result == 0 ]
then
:
else
echo "don't exist"
fi
[root@MiWiFi-R4CM-srv shell]# ls
fileCheck.sh  myfirst_shell.sh  shell2.sh  shell3.sh  shell4.sh  shell5.sh  shell6.sh
[root@MiWiFi-R4CM-srv shell]# pwd
/root/shell
[root@MiWiFi-R4CM-srv shell]# ls /root/
anaconda-ks.cfg  rpmbuild  shell  sleep.log  test.txt  test.txt~
[root@MiWiFi-R4CM-srv shell]#
[root@MiWiFi-R4CM-srv shell]# bash shell6.sh
don't exist
[root@MiWiFi-R4CM-srv shell]#
```

看红色标记处即可。代码中有个冒号：，其实这个说明文件夹存在，就执行 ":" ，表示什么都不干，占个位而已。另外 2>/dev/null 这个流向黑洞的代码也很精髓，我们不像错误信息随之输出，就可以把错误信息输入到黑洞里。

后面的与和或两个运算符我不想解释太多，有点编程基础都知道，用&&时，如果前面为假，后面就不看了，如果前面为真，再看后面的；用||时，如果前面为真，后面就不看了，前面为假才看后面。这样可以化简合并一系列的操作，避开if。比如前面那个命令执行失败了，后面就别执行了一类的。

9、脚本编程的实践案例

```
#自动校准时间,配合crontab使用

ntpdate s1b.time.edu.cn || ntpdate ntp1.aliyun.com
```

```
[root@MiWiFi-R4CM-srv shell]# cat ntpdate.sh
ntpdate s1b.time.edu.cn || ntpdate ntp1.aliyun.com
[root@MiWiFi-R4CM-srv shell]# crontab -l
*/30 * * * * bash /root/shell/ntpdate.sh
[root@MiWiFi-R4CM-srv shell]# bash /root/shell/ntpdate.sh
24 Sep 16:53:41 ntpdate[1671]: no server suitable for synchronization found
24 Sep 16:53:47 ntpdate[1672]: adjust time server 120.25.115.20 offset -0.001352 sec
[root@MiWiFi-R4CM-srv shell]#
```

```
#同时检查CPU、内存和硬盘
#CPU的计算：uptime命令的第一位数值代表最近一段时间的负载量，如果这个数值小于机器CPU的核数，就认为不忙碌
#内存的计算：从free中取出最后一列的数据，用sed去除多余的空行。剩余量大于200，则提示OK
#硬盘的计算：取出根目录的已用量,大于90则提示FULL

#CHECK CPU
cpu_cores=`cat /proc/cpuinfo | grep processor | wc -l`
loads=`uptime | awk '{print $8}' | tr ',' ' ' | cut -d'.' -f1`
if [ $loads -lt $cpu_cores ]
then
echo "CPU WORKS OK"
```

```
else
echo "CPU TOO HIGH"
fi

#CHECK MEMORY
ram_left=`free -m | awk '{print $7}' | sed '/^$/d'`
if [ $ram_left -gt 200 ]
then
echo "RAM OK"
else
echo "LACKING OF RAM"
fi

#CHECK DISK
disk_used=`df -h | grep 'centos-root' | awk '{print $5}' | sed "s/%//g"`
if [ $disk_used -gt 90 ]
then
echo "DISK RUNNING FULL"
else
echo "DISK OK"
fi
```

```
[root@MiWiFi-R4CM-srv shell]# cat monitor.sh
#CHECK CPU
cpu_cores=`cat /proc/cpuinfo | grep processor | wc -l`
loads=`uptime | awk '{print $8}' | tr ',' ' ' | cut -d'.' -f1`
if [ $loads -lt $cpu_cores ]
then
echo "CPU WORKS OK"
else
echo "CPU TOO HIGH"
fi

#CHECK MEMORY
ram_left=`free -m | awk '{printf $7}' | sed '/^$/d'`
if [ $ram_left -gt 200 ]
then
echo "RAM OK"
else
echo "LACKING OF RAM"
fi

#CHECK DISK
disk_used=`df -h | grep 'centos-root' | awk '{print $5}' | sed "s/%//g"`
if [ $disk_used -gt 90 ]
then
echo "DISK RUNNING FULL"
else
echo "DISK OK"
fi
[root@MiWiFi-R4CM-srv shell]# bash monitor.sh
CPU WORKS OK
RAM OK
DISK OK
[root@MiWiFi-R4CM-srv shell]#
```

```
#分析文本文件内容之统计相关日志条数

journalctl > journalctl_log
pinglog_count=0
sshlog_count=0
networklog_count=0
while read line
do
```

```
echo $line | grep -i 'ping' && let pinglog_count++
echo $line | grep -i 'ssh' && let sshlog_count++
echo $line | grep -i 'network' && let networklog_count++
done < journalctl_log
echo $pinglog_count
echo $sshlog_count
echo $networklog_count
\rm journalctl_log
```

```
[root@MiWiFi-R4CM-srv shell]# cat census.sh
journalctl > journalctl_log
pinglog_count=0
sshlog_count=0
networklog_count=0
while read line
do
echo $line | grep -i 'ping' && let pinglog_count++
echo $line | grep -i 'ssh' && let sshlog_count++
echo $line | grep -i 'network' && let networklog_count++
done < journalctl_log
echo $pinglog_count
echo $sshlog_count
echo $networklog_count
\rm journalctl_log
[root@MiWiFi-R4CM-srv shell]# bash census.sh
Sep 24 18:58:26 localhost.localdomain kernel: Built 1 zonelists in Node order, mobility grouping on.
 Total pages: 257913
Sep 24 18:58:26 localhost.localdomain kernel: smpboot: CPU0: Intel(R) Core(TM) i7-10510U CPU @ 1.80G
Hz (fam: 06, model: 8e, stepping: 0c)
Sep 24 18:58:26 localhost.localdomain kernel: drop_monitor: Initializing network drop monitor servic
e
Sep 24 18:58:27 localhost.localdomain kernel: e1000: Intel(R) PRO/1000 Network Driver - version 7.3.
21-k8-NAPI
Sep 24 18:58:28 localhost.localdomain kernel: e1000 0000:00:03.0 eth0: Intel(R) PRO/1000 Network Con
nection
```

```
#远程登录多台服务器并执行任务，确保已进行过免密SSH登录与机器名解析

hostlist="server02 server03 server04"
for i in $hostlist
do
echo "updating script to host $i"
scp /root/shell/monitor.sh "$i":/tmp/ > /dev/null
if [ $? == 0 ]
then
echo "updated $i ok"
else
echo "updating $i failed"
fi
echo "checking on host $i"
ssh $i "chmod 777 /tmp/monitor.sh"
ssh $i "bash /tmp/monitor.sh"
done
```

这个我就不展示了，因为我根本就没那么多机子表演。另外，机器名解析就是说用名称来代替IP地址，之前我们学SSH时用的都是IP地址，太麻烦了，所以我们要 vim /etc/hosts 在文件最后加上：IP地址+完整机器名+简写机器名，来用名称代替IP地址，这样方便脚本的编写，有多少个机器要登录就写多少条。这个也是修改本地DNS解析的方法。

- 后记

Linux的入门，就到这里结束了，这是我学大米哥的Linux教程的笔记，大部分是抄书+实践+思考总结+变形尝试，哪里感觉写错了那就自己上机找正确答案，其实书上有一些地方大米哥打错了，编辑也没有校正，我都是上机尝试后得到的正确结果。

最后，Linux，都不学。再见。