shell开发

在前面，超哥已经带着大家，学习了linux系统命令操作，为了提升运维效率，我们得学习shell脚本开发。

##使用多个命令

shell脚本的关键是在于处理多个命令，且处理每个命令的结果，或者将不同的命令结果进行传递，再次加工。

shell支持多个命令连接起来执行。

```
[root@web01 ~]# clear
[root@web01 ~]# date;whoami
Sun Sep 27 17:57:45 CST 2020
root
```

恭喜你，铁子，你已经会了第一个shell脚本。这个脚本简单到只有2个命令，这种形式，命令少还好说，如果命令较长，较多，那就很难在命令行敲打了。

我们可以把命令组合成一个文本文件，然后执行这个文件即可。

shell脚本文件

shell脚本文件，第一行必须指定想要用的shell。

```
#!/bin/bash
```

shell脚本里，`#号`是注释行，shell不会处理脚本中的注释行，然而第一行例外，该写法会通知shell用哪个解释器来运行脚本。

shell脚本中可以书写你想要执行的命令，且可以添加注释行。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# This is test script
date
whoami
echo "It's Over."
```

执行脚本

用解释器读取脚本

```
[root@web01 ~]# sh test.sh
Mon Sep 28 09:34:28 CST 2020
root
It's Over.


[root@web01 ~]# bash test.sh
Mon Sep 28 09:34:36 CST 2020
root
It's Over.
```

直接运行文件，需要赋予权限

```
[root@web01 ~]# ./test.sh
-bash: ./test.sh: Permission denied

[root@web01 ~]# ll test.sh
-rw-r--r-- 1 root root 64 Sep 28 09:34 test.sh
[root@web01 ~]# chmod +x test.sh
[root@web01 ~]#
[root@web01 ~]# ll test.sh
-rwxr-xr-x 1 root root 64 Sep 28 09:34 test.sh

# 执行
[root@web01 ~]# ./test.sh
Mon Sep 28 09:35:35 CST 2020
root
It's Over.
```


当以`./test.sh`该形式执行脚本，则以文件第一行的`shebang`指定的解释器执行文件。

常用echo

在shell脚本里，shell命令会有自己的输出，若是你想要自定义内容打印，以告知用户，当前脚本在做什么事。

可以通过echo命令来实现，注意`单引号`、`双引号`。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# This script displays the date and who's logged on
echo "The time and date are:"
date
echo ''
echo "Let's see who's logged into the system:"
who


[root@web01 ~]# ./test.sh
The time and date are:
Mon Sep 28 09:45:20 CST 2020
```

```
Let's see who's logged into the system:
root     tty1         2020-09-27 11:01
root     pts/0        2020-09-27 17:57 (10.0.1.1)
```

使用变量与echo

脚本里，可以在环境变量名称前加上 `$` 来使用环境变量。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# display user information from the system
echo "User info for userid: $USER"
echo UID: $UID
echo HOME: $HOME


[root@web01 ~]# ./test.sh
User info for userid: root
UID: 0
HOME: /root
```

转义符

当一些情况，我们不希望shell解析一些符号，可以使用 `单引号` 和 `转义符` 对符号不做解析。

echo识别变量的情况

```
[root@web01 ~]# echo "The cost of the item is $15"
The cost of the item is 5
```

这显然不是超哥想要的，改成如下写法

```
[root@web01 ~]# echo ' The cost of the item is $15'
 The cost of the item is $15
```

方法二

```
[root@web01 ~]# echo "The cost of the item is \$15"
The cost of the item is $15
```

现在允许shell解读 `$` 只是一个普通的美元符了。

变量应用

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# test variables
days=10
guest="超哥"
echo "$guest checked in $days days ago"


[root@web01 ~]# ./test.sh
超哥 checked in 10 days ago
```

变量被引用时会输出赋予给它的值，变量会在shell脚本退出后消失。

变量再赋值

变量每次被引用的时候，都会输出赋予的值，需要加上`$`符

```
[root@web01 ~]#
[root@web01 ~]# cat test.sh
#!/bin/bash
# assigning a variable value to another variable
value1=10
value2=$value1
echo "The resulting value is $value2"

[root@web01 ~]# ./test.sh
The resulting value is 10
[root@web01 ~]#
```


若是忘记用美元符，则结果就错误了。

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# assigning a variable value to another variable
value1=10
value2=value1
echo "The resulting value is $value2"
[root@web01 ~]#
[root@web01 ~]#
[root@web01 ~]# ./test.sh
The resulting value is value1
```

命令替换

shell一大特性就是可以从命令中提取结果，再赋予给变量，这样处理脚本数据特别方便。

有两个方式将命令输出赋予给变量

> 反引号 ``

> $() 格式

来看超哥的用法

```
[root@web01 ~]# cat test.sh
#!/bin/bash
testing=$(date)
echo "The date and time are:" $testing

# 结果
[root@web01 ~]# ./test.sh
The date and time are: Mon Sep 28 10:31:39 CST 2020
```

再来看案例

```
[root@web01 ~]# cat test.sh
#!/bin/bash
# copy the /usr/bin directory listing to a log file
today=$(date +%y%m%d)
ls /usr/bin -al > log.$today

today变量是取出命令的值，也就是提取了日期，然后在结合到文件名里使用
[root@web01 ~]# ./test.sh
[root@web01 ~]# cat log.200928 |wc -l
1240
```


###反引号

```
反引号和$()的作用相同，用于命令替换（command substitution），即完成引用的命令的执行，将其结果替换出来，与变量替换差不多。比如：

[root@web01 ~]# echo `date '--date=1 hour ago' +%Y-%m-%d-%H`
2020-09-28-10

等同于
[root@web01 ~]# echo $(date '--date=1 hour ago' +%Y-%m-%d-%H)
2020-09-28-10
```

#数值计算

编程语言都支持数字计算，但是对于shell来说，数值计算稍微麻烦点。

> 算数运算符

表 5-1　Shell 中常见的算术运算符号

算术运算符	意义（* 表示常用）
+、-	加法（或正号）、减法（或负号）*
*、/、%	乘法、除法、取余（取模）*
**	幂运算 *
++、--	增加及减少，可前置也可放在变量结尾 *
!、&&、||	逻辑非（取反）、逻辑与（and）、逻辑或 (or)*
<、<=、>、>=	比较符号（小于、小于等于、大于、大于等于）
==、!=、=	比较符号（相等、不相等，对于字符串“=”也可以表示相当于）*
<<、>>	向左移位、向右移位
~、|、&、^	按位取反、按位异或、按位与、按位或
=、+=、-=、*=、/=、%=	赋值运算符，例如 a+=1 相当于 a=a+1，a-=1 相当于 a=a-1*

运算符命令

表 5-2　Shell 中常见的算术运算命令

运算操作符与运算命令	意义
(())	用于整数运算的常用运算符，效率很高
let	用于整数运算，类似于“(())”
expr	可用于整数运算，但还有很多其他的额外功能
bc	Linux 下的一个计算器程序（适合整数及小数运算）
$[]	用于整数运算
awk	awk 既可以用于整数运算，也可以用于小数运算
declare	定义变量值和属性，-i 参数可以用于定义整形变量，做运算

Shell if参数

shell 编程中使用到得if语句内判断参数
-b 当file存在并且是块文件时返回true
-c 当file存在并且是字符文件时返回true
-d 当pathname存在并且是一个目录时返回true
-e 当pathname指定的文件或目录存在时返回true
-f 当file存在并且是文件时返回true
-g 当由pathname指定的文件或目录存在并且设置了SGID位时返回为true
-h 当file存在并且是符号链接文件时返回true，该选项在一些老系统上无效
-k 当由pathname指定的文件或目录存在并且设置了“粘滞”位时返回true
-p 当file存在并且是命令管道时返回为true
-r 当由pathname指定的文件或目录存在并且可读时返回为true
-s 当file存在文件大小大于0时返回true
-u 当由pathname指定的文件或目录存在并且设置了SUID位时返回true
-w 当由pathname指定的文件或目录存在并且可执行时返回true。一个目录为了它的内容被访问必然是可执行的。
-o 当由pathname指定的文件或目录存在并且被子当前进程的有效用户ID所指定的用户拥有时返回true。
UNIX Shell 里面比较字符写法:
-eq 等于
-ne 不等于
-gt 大于
-lt 小于
-le 小于等于
-ge 大于等于
-z 空串

```
= 两个字符相等
!= 两个字符不等
-n 非空串
```

用于数值计算的命令

双括号(())

效率极高，常用方法

表 5-3　双小括号“(())”的操作方法

运算操作符与运算命令	意义
((i=i+1))	此种书写方法为运算后赋值法，即将 i+1 的运算结果赋值给变量 i。注意，不能用“echo ((i=i+1))”的形式输出表达式的值，但可以用 echo $((i=i+1)) 输出其值
i=$((i+1))	可以在“(())”前加 $ 符，表示将表达式运算后赋值给 i
((8>7&&5==5))	可以进行比较操作，还可以加入逻辑与和逻辑或，用于条件判断
echo $((2+1))	需要直接输出运算表达式的运算结果时，可以在“(())”前加 $ 符

> 案例，注意必须是整数，双括号不支持浮点数，浮点数需要其他命令计算

```
# 加减乘除
[root@chaogelinux shell_program]# echo $((3+4))
7
[root@chaogelinux shell_program]# echo $((5-3))
2
[root@chaogelinux shell_program]# echo $((5*3))
15
[root@chaogelinux shell_program]# echo $((5/3)) # 取商，除法
1
[root@chaogelinux shell_program]# echo $((5**2))
25
[root@chaogelinux shell_program]# echo $((7%4)) # 取余数，取模计算
3

# 变量计算
[root@chaogelinux shell_program]# ((i=5))
[root@chaogelinux shell_program]# ((i=i*i))
[root@chaogelinux shell_program]# echo $i
25

# 复杂数学运算
[root@chaogelinux shell_program]# ((a=2+2**3-4%3))
[root@chaogelinux shell_program]#
[root@chaogelinux shell_program]#
[root@chaogelinux shell_program]# echo $a
9
```

```
# 变量定义在括号外，写法2
[root@chaogelinux shell_program]# b=$((2+2**3-4%3))
[root@chaogelinux shell_program]# echo $b
9

# 不用变量赋值，直接看结果，一定记得用$符号
[root@chaogelinux shell_program]# echo $((2+2**3-4%3))
9

#结果判断，真假值，1为真，0位假
[root@chaogelinux shell_program]# echo $((3<4))
1
[root@chaogelinux shell_program]# echo $((3>4))
0
[root@chaogelinux shell_program]# echo $((3==4))
0
```

特殊运算符

++ 加一

-- 减一

注意这里有坑

++a 是先计算加一，然后赋值a

a++ 是先对变量a操作，再加一

注意这个先后顺序


```
[root@chaogelinux shell_program]# a=5
[root@chaogelinux shell_program]# echo $((++a)) # 这是先对a变量，加一，然后再赋值
6

[root@chaogelinux shell_program]# echo $a
6

# 先打印，再赋值
[root@chaogelinux shell_program]# echo $((a++))
6
[root@chaogelinux shell_program]# echo $a
7
```

案例，对用户输入判断是否是整数的脚本

利用read命令，获取用户输入

```
# [-n string] or [string] “string”的长度为非零non-zero则为真
# -p 后面跟提示信息，即在输入前打印提示信息。
# 状态码不为0 就是表示有错误



[root@chaogelinux shell_program]# cat is_int.sh
#!/bin/bash
print_usage(){
    printf "Please enter an integer\n"
    exit 1
}

# 接受用户输入
read -p "Please input first number: " firstnum


# 如果用户输入，去掉了整数部分，为空，说明是一个纯数字
# if的-n参数，如果 string 长度非零，则为真
# 如果该字符串不为空，那么if逻辑成立，说明用户输入的不是一个纯数字

# 逻辑判断，限制用户必须输入一个纯数字
if [ -n "`echo $firstnum|sed 's/[0-9]//g'`" ];then
    print_usage
fi

read -p "Please input the operators: " operators

# 判断运算符
if [ "${operators}" != "+" ] && [ "${operators}" != "-" ] && [ "${operators}" != "*" ] && [
"${operators}" != "/" ];then
    echo  "Please use {+|-|*|/}"
    exit 2
fi

# 输入第二个数字
read -p "Please input second number: " secondnum
if [ -n "`echo $secondnum|sed 's/[0-9]//g'`" ];then
    print_usage
fi

# 最后数值计算
echo "${firstnum}${operators}${secondnum}结果是：$((${firstnum}${operators}${secondnum}))"
```

执行

```
[root@chaogelinux shell_program]# bash is_int.sh
Please input first number: 12
Please input the operators: +
Please input second number: 4
12+4结果是：16
```

> 一个简单的玩法，直接对用户输入计算

```
[root@chaogelinux shell_program]# cat jisuan.sh
#!/bin/bash
echo $(($1))
[root@chaogelinux shell_program]# bash jisuan.sh 3+4
7

[root@chaogelinux shell_program]# bash jisuan.sh "3*4"
12
```

let命令

> let命令等同于双括号计算，`"((赋值表达式))"`
>
> 但是双括号效率更高

```
[root@chaogelinux shell_program]# # 计算变量相加
[root@chaogelinux shell_program]# i=2
[root@chaogelinux shell_program]# i=i+8
[root@chaogelinux shell_program]# echo $i
i+8
[root@chaogelinux shell_program]# unset i
[root@chaogelinux shell_program]# i=2
[root@chaogelinux shell_program]# let i=i+8
[root@chaogelinux shell_program]# echo $i
10

# 上述等同于
[root@chaogelinux shell_program]# i=2
[root@chaogelinux shell_program]# echo $((i=i+8))
10
```

> 实际脚本开发，检测nginx服务状态

```
[root@chaogelinux shell_program]# cat check_nginx_status.sh
#!/bin/bash

# 定义功能函数
CheckUrl(){
timeout=5
fails=0
success=0
while true
    do
        wget --timeout=$timeout --tries=1 http://pythonav.cn/ -q -O /dev/null
        # 如果上述状态码不等于0，表示出错
```

```
        if [ $? -ne 0 ]
            then
                let fails=fails+1 # 失败错误次数+1
        else
                let success+=1
        fi

        if [ $success -ge 1 ]
            then
                   echo "It's success."
                   exit 0
        fi

        # 如果出错次数大于2，报警
        if [ $fails -ge 2 ];then
            echo "该网站一定是挂了，超哥快去检查下你的服务器！"
            exit 2
        fi
done
}
CheckUrl
[root@chaogelinux shell_program]#
```

##expr命令

expr命令允许在命令行处理数学表达式，简单的计算器命令

查看帮助

```
expr --help
```

> 注意参数之间必须有空格

```
[root@web01 ~]# expr 1 + 5
6

# expr并不是很好用，比较麻烦
# 很多符号在shell里有特殊含义因此必须得转义使用
[root@web01 ~]# expr 5 * 2
expr: syntax error
[root@web01 ~]# expr 5 \* 2
10

# 求长度
[root@web01 ~]# expr length 123132134524asdqweqweqewqfgdfgdfgdf
35

# 逻辑判断，必须加上引号
[root@web01 ~]# expr 33 '>'  3
1
```

```
[root@web01 ~]# expr 33 '>'  333
0

# 除法
[root@web01 ~]# expr 50 / 5
10
```

expr模式匹配

> expr支持模式匹配，也就是如同正则的作用
>
> 有2个模式符号
>
> : 冒号，计算字符串中的字符数
>
> .* 任意字符串重复0次或多次
>
> 语法
>
> 统计字符串的字符个数
>
> expr 字符串 ： ".*"

实践

```
# 统计yc.jpg字符个数
[root@chaogelinux ~]# expr yc.jpg ":"  ".*"
6

# 统计jpg后缀的文件，字符个数，找不到符合的就为0
# 无论jpg后面有多少个字符，只匹配到jpg这3个字符结尾
[root@chaogelinux ~]# expr yc.jpg   ":"  ".*\.jpg"
6
[root@chaogelinux ~]# expr yc.jpg123123   ":"  ".*\.jpg"
6
# 找不到的情况
[root@chaogelinux ~]# expr yc.jpwg123123   ":"  ".*\.jpg"
0
```

expr案例

> expr判断文件名是否符合要求

```
[root@chaogelinux shell_program]# cat expr1.sh
#!/bin/bash

if expr "$1" : ".*\.jpg" &> /dev/null
    then
        echo "This is jpg file"
```

```
else
    echo "这不是jpg文件"
fi

# 执行
[root@chaogelinux shell_program]# bash expr1.sh 蔡徐坤.jpg
This is jpg file

[root@chaogelinux shell_program]# bash expr1.sh t1.sh
这不是jpg文件
```

> 找出字符长度不大于6的单词

```
[root@chaogelinux shell_program]# cat word_length.sh
#!/bin/bash
# 注意后面这个变量，别加引号
for str in I am Yu Chao, I teach you to learn linux
do
    if [ `expr length $str` -le 6  ]
        then
            echo $str
    fi
done
```


bc命令用法

bc命令可以当作计算器来用，当作命令行计算器用。

> 交互式计算

```
# 简单的加减乘除
# 交互式操作
[root@chaogelinux ~]# bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2+2
4
5-3
2
6 * 3
18
6 / 4
1
6 /3
2
3.2-1.1
```

```
2.1
```

> 管道符计算

```
[root@chaogelinux ~]# echo 4*4|bc
16
[root@chaogelinux ~]# echo 4.5-1.1|bc
3.4
[root@chaogelinux ~]# num=5
[root@chaogelinux ~]# result=`echo $num+6|bc`
[root@chaogelinux ~]#
[root@chaogelinux ~]# echo $result
11
```

> 提示：
>
> 整数的计算，用双小括号，let，expr
>
> 带有小数的计算，用bc

bc案例

> 一条命令，算出1～10的总和。

```
# 思路，我们想要计算
1+2+3..+10

# 怎么先定义好这样的连续，有规律的字符串?
[root@chaogelinux ~]# seq -s + 10
1+2+3+4+5+6+7+8+9+10

# tr命令，从标准输入中替换、缩减和/或删除字符，并将结果写到标准输出。
[root@chaogelinux ~]# echo {1..10}|tr " " "+"
1+2+3+4+5+6+7+8+9+10

# 得到字符串了，交给bc计算命令即可
[root@chaogelinux ~]# echo {1..10}|tr " " "+" |bc
55

# 双括号，括号里直接写表达式
[root@chaogelinux ~]# echo $((`seq -s + 10`))
55

# expr命令，传参给expr，因此可以用xargs命令
# 注意expr需要传入多个参数执行计算，注意空格
[root@chaogelinux ~]# expr 1 + 2 + 3
6
# 因此上面的字符串就不适用了，要改成这样的
[root@chaogelinux ~]# seq -s " + " 10
```

```
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
# 最终交给expr
# xargs 又称管道命令，构造参数等。是给命令传递参数的一个过滤器,也是组合多个命令的一个工具 它把一个数据流分割为一些足够小的块,以方便过滤器和命令进行处理 。简单的说 就是把 其他命令的给它的数据 传递给它后面的命令作为参数
[root@chaogelinux ~]# seq -s " + " 10 | xargs expr
55


#
```

awk数值计算

```
[root@chaogelinux ~]# echo "3.2 2.4"|awk '{print ($1+$2)}'
5.6

[root@chaogelinux ~]# echo "4 8"|awk '{print ($1*$2)}'
32

[root@chaogelinux ~]# echo "4 8"|awk '{print ($1+4*$2)}'
36
```

这里大家要回忆起来awk玩法，且后面超哥会讲解awk进阶用法，逻辑条件，数组等。

中括号运算[]

> $[表达式] 语法如此
>
> 加减乘除

```
[root@chaogelinux ~]# num=5
[root@chaogelinux ~]# result=$[num+8]
[root@chaogelinux ~]# echo $result
13

[root@chaogelinux ~]# echo $[8**2]
64

[root@chaogelinux ~]# echo $[3+2]
5

[root@chaogelinux ~]# echo $[3%2]
1
[root@chaogelinux ~]# echo $[6%2]
0
```

读取用户输入

在前面的实战脚本中，超哥已经讲了read命令的基础用法。

shell变量除了直接赋值，或者脚本传参，还有就是read命令读取。

read也是内置命令。

> -p 设置提示信息
>
> -t 等待用户输入超时，timeout
>
> read -p "请输入: " vars

实践

```
[root@chaogelinux ~]# read -t 5 -p "我给你五秒钟时间输入密码: " my_pwd
我给你五秒钟时间输入密码: [root@chaogelinux ~]#

# 正确输入
[root@chaogelinux ~]# read -t 5 -p "我给你五秒钟时间输入密码: " my_pwd
我给你五秒钟时间输入密码: chaoge888
[root@chaogelinux ~]#
[root@chaogelinux ~]#
[root@chaogelinux ~]# echo $my_pwd
chaoge888

# 输入多个变量，注意用空格分割
[root@chaogelinux ~]# read -t 5 -p "请输入你的账户，密码: " my_user  my_pwd
请输入你的账户，密码: yu chaoge888
[root@chaogelinux ~]#
[root@chaogelinux ~]# echo $my_user $my_pwd
yu chaoge888
```


read的实践案例，之前已经用脚本演示过，这里大家再理解下。