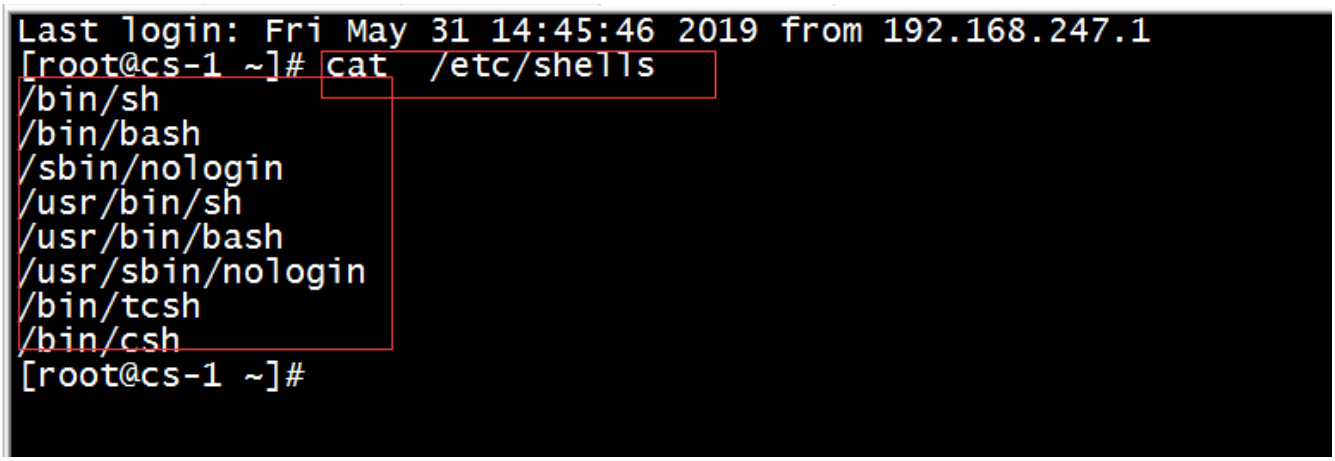
shell编程

一、什么是shell编程

- shell使用C语言编写的程序，可以用来访问操作系统的内核。shell是一个命令语言，又是一个程序设计语言(变量，逻辑判断，函数)。
- shell编程指的是shell脚本编程，不是开发shell自身
- shell script 是为shell编写的脚本程序，shell和Java、php编程一样，只需要一个编写代码的编辑器和一个解释脚本执行的解释器就可以了。
- Linux系统中shell种类不唯一，可以通过cat /etc/shells 查看系统中安装的shell。



- Bash因为易用和免费被广泛使用，大多数Linux系统默认使用Bash

二、基本格式

开始编程：vi编辑器新建一个shell脚本文件如hello.sh，扩展名不影响脚本的运行，只不过可读性较好。如Java写shell脚本就是hello.java

#!/bin/bash

echo “hello world!”

- !是一个标记，告诉系统这个脚本使用哪一个解释器执行，即哪一个shell；
- echo是向窗口输出脚本；
- 新建的shell脚本文件没有执行权限 chmod 777 hello.sh ;chmod u+x hello.sh

```
Linux系统中的每个文件和目录都有访问许可权限，
用它来确定谁可以通过何种方式对文件和目录进行访问和操作。
文件或目录的访问权 限分为只读，只写和可执行三种。

这里显示的权限是依次排列的，分别为：[用户][同组][其他]
用户权限，就是你自己的权限。英文：user，简写：u（覆盖标号123）
用户组权限，就是和你同组的人的权限。英文：group，简写：g（覆盖标号456）
其他权限，就是不和你同组的人的权限。英文：others，简写：o（覆盖标号789）
所有人的权限，英文：all，简写：a

r，即Read，读，权限值为4
w，即Write，写，权限值为2
x，即eXecute，执行，权限值为1
-，在标号0位置，表示普通的文件
-，其他位置，表示对应权限未开启，不具备权限
d，即directory，表示目录文件

无任何权限：数字0表示
开所有权限：数字7表示，即7=4+2+1

chmod 命令是用于改变文件或目录的访问权限。

+ 表示增加权限，如u+x，u+r，u+w，g+w，g+r，o+r， a+r等
- 表示取消权限，如u-x，u-r，u-w，g-w，g-r，o-r， a-r等
= 表示赋予给定权限，并取消其他所有权限（如果有的话，如原来u是rwx，设置u=r，u就剩r）
```

shell脚本执行：

- ./hello.sh；告诉系统就在当前目录找
- /bin/bash hello.sh
- /root/hello.sh

```
[root@cs-1 ~]# /bin/bash hello.sh
“hello world!”
[root@cs-1 ~]# ./hello.sh
“hello world!”
[root@cs-1 ~]# /root/hello.sh
“hello world!”
[root@cs-1 ~]#
```

三、shell变量

变量语法格式：变量=值 ，如name=“zxing”

- 变量名和等号之间不能有空格
- 变量名首个字符必须为字母（a-z A-Z）
- 变量名字符之间不能有空格可以有下划线（_）

- 变量名字符之间不能标点符号
- 变量名字符之间不能有bash里面的关键字（help命令查看）

变量的使用：

在 变量名前面加上$符号即可以使用定义过的变量

```
name=“zxing”

echo $name
#花括号是可选的，用来定义变量的边界；变量值可以被修改；
echo ${name}
#使用readOnly命令可以把变量定义为只读变量，只读变量的值不可以被修改；
readonly name
#使用unset命令可以删除变量，不能删除只读变量
unset name
```

变量类型：

- 局部变量：在脚本和命令中定义，仅在当前shell实例中有效，其他shell启动的程序无法访问。
- 环境变量：所有的程序包括shell程序都能访问，保证其程序的正常运行，可以用set命令查看环境变量。
- shell变量：是shell程序设置的特殊变量，其中一部分是环境变量，一部分是局部变量，保证了shell的正常运行。

三、参数传递

参数传递：

在shell脚本执行的时候可以传递参数，脚本内获取参数的格式是：$n，n是数字，1表示第一个参数，2是表示第二个参数，$0是当前脚本的名称

```
[root@cs-1 ~]# vi 1.sh
#!/bin/bash
echo "hello bash"
echo $0
echo $1
echo $3
~
```

```
~
[root@cs-1 ~]# ./1.sh 1 2 3 4
hello bash
./1.sh
1
3
[root@cs-1 ~]#
```

特殊符号：

- $#传递给脚本参数的个数
- $*用一个字符串显示所有向脚本传递的参数
- $$脚本运行的进程ID号

- $@ 与$*相同，使用时加上引号“”，并在引号中返回每一个参数
- $!后台运行的最后一个进程号
- $?显示最后命令的退出状态，0表示没有错误，其他的值都是有错误

注意：在没有“”的情况下$*和$@是相同的都是输出所有的参数

在有“”的时候$*是把所有的参数以一个整体的字符串输出，$@则是以一个参数列表输出。

四、运算符

Shell 和其他编程语言一样，支持多种运算符，包括：

算术运算符

原生bash不支持简单的数学运算，但是可以通过其他命令来实现，例如 awk 和 expr，expr 最常用。

expr 是一款表达式计算工具，使用它能完成表达式的求值操作。

- 表达式和运算符之间要有空格，例如 2+2 是不对的，必须写成 2 + 2，这与我们熟悉的大多数编程语言不一样。
- 完整的表达式要被 ` 包含，注意这个字符不是常用的单引号，在 Esc 键下边。

```
+ 加法 `expr $a + $b` 结果为 30。
- 减法 `expr $a - $b` 结果为 -10。
* 乘法 `expr $a * $b` 结果为  200。
/ 除法 `expr $b / $a` 结果为 2。
% 取余 `expr $b % $a` 结果为 0。
= 赋值 a=$b 将把变量 b 的值赋给 a。
== 相等。用于比较两个数字，相同则返回 true。 [ $a == $b ] 返回 false。
!= 不相等。用于比较两个数字，不相同则返回 true。 [ $a != $b ] 返回 true。
注意：条件表达式要放在方括号之间，并且要有空格，例如: [$a==$b] 是错误的，必须写成 [ $a == $b ]。
```

```
#!/bin/bash
# author:自兴教程

a=10
b=20

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"

val=`expr $b % $a`
echo "b % a : $val"

if [ $a == $b ]
then
   echo "a 等于 b"
fi
if [ $a != $b ]
then
   echo "a 不等于 b"
fi
```

关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。下表列出了常用的关系运算符，假定变量 a 为 10，变量 b 为 20：

```
-eq 检测两个数是否相等，相等返回 true。 [ $a -eq $b ] 返回 false。
-ne 检测两个数是否不相等，不相等返回 true。 [ $a -ne $b ] 返回 true。
-gt 检测左边的数是否大于右边的，如果是，则返回 true。 [ $a -gt $b ] 返回 false。
-lt 检测左边的数是否小于右边的，如果是，则返回 true。 [ $a -lt $b ] 返回 true。
-ge 检测左边的数是否大于等于右边的，如果是，则返回 true。 [ $a -ge $b ] 返回 false。
-le 检测左边的数是否小于等于右边的，如果是，则返回 true。 [ $a -le $b ] 返回 true。

#!/bin/bash
# author:自兴教程
# url:www.zxing.com

a=10
b=20

if [ $a -eq $b ]
```

```
then
   echo "$a -eq $b : a 等于 b"
else
   echo "$a -eq $b: a 不等于 b"
fi
if [ $a -ne $b ]
then
   echo "$a -ne $b: a 不等于 b"
else
   echo "$a -ne $b : a 等于 b"
fi
if [ $a -gt $b ]
then
   echo "$a -gt $b: a 大于 b"
else
   echo "$a -gt $b: a 不大于 b"
fi
if [ $a -lt $b ]
then
   echo "$a -lt $b: a 小于 b"
else
   echo "$a -lt $b: a 不小于 b"
fi
if [ $a -ge $b ]
then
   echo "$a -ge $b: a 大于或等于 b"
else
   echo "$a -ge $b: a 小于 b"
fi
if [ $a -le $b ]
then
   echo "$a -le $b: a 小于或等于 b"
else
   echo "$a -le $b: a 大于 b"
fi
```

布尔运算符

下表列出了常用的布尔运算符，假定变量 a 为 10，变量 b 为 20：

```
! 非运算，表达式为 true 则返回 false，否则返回 true。 [ ! false ] 返回 true。
-o 或运算，有一个表达式为 true 则返回 true。 [ $a -lt 20 -o $b -gt 100 ] 返回 true。
-a 与运算，两个表达式都为 true 才返回 true。 [ $a -lt 20 -a $b -gt 100 ] 返回 false。
#!/bin/bash
# author:自兴教程
# url:www.zxing.com

a=10
b=20
```

```
if [ $a != $b ]
then
   echo "$a != $b : a 不等于 b"
else
   echo "$a != $b: a 等于 b"
fi
if [ $a -lt 100 -a $b -gt 15 ]
then
   echo "$a 小于 100 且 $b 大于 15 : 返回 true"
else
   echo "$a 小于 100 且 $b 大于 15 : 返回 false"
fi
if [ $a -lt 100 -o $b -gt 100 ]
then
   echo "$a 小于 100 或 $b 大于 100 : 返回 true"
else
   echo "$a 小于 100 或 $b 大于 100 : 返回 false"
fi
if [ $a -lt 5 -o $b -gt 100 ]
then
   echo "$a 小于 5 或 $b 大于 100 : 返回 true"
else
   echo "$a 小于 5 或 $b 大于 100 : 返回 false"
fi
```

逻辑运算符

以下介绍 Shell 的逻辑运算符，假定变量 a 为 10，变量 b 为 20:

```
&&  逻辑的 AND  [[ $a -lt 100 && $b -gt 100 ]] 返回 false
|| 逻辑的 OR [[ $a -lt 100 || $b -gt 100 ]] 返回 true

#!/bin/bash
# author:自兴教程
# url:www.zxing.com

a=10
b=20

if [[ $a -lt 100 && $b -gt 100 ]]
then
   echo "返回 true"
else
   echo "返回 false"
fi

if [[ $a -lt 100 || $b -gt 100 ]]
then
   echo "返回 true"
else
   echo "返回 false"
fi
```

字符串运算符:

下表列出了常用的字符串运算符，假定变量 a 为 "abc"，变量 b 为 "efg"：

```
= 检测两个字符串是否相等，相等返回 true。 [ $a = $b ] 返回 false。
!= 检测两个字符串是否相等，不相等返回 true。 [ $a != $b ] 返回 true。
-z 检测字符串长度是否为0，为0返回 true。 [ -z $a ] 返回 false。
-n 检测字符串长度是否为0，不为0返回 true。 [ -n "$a" ] 返回 true。
str 检测字符串是否为空，不为空返回 true。 [ $a ] 返回 true。

#!/bin/bash
# author:自兴教程
# url:

a="abc"
b="efg"

if [ $a = $b ]
then
   echo "$a = $b : a 等于 b"
else
   echo "$a = $b: a 不等于 b"
fi
if [ $a != $b ]
then
   echo "$a != $b : a 不等于 b"
else
   echo "$a != $b: a 等于 b"
fi
if [ -z $a ]
then
   echo "-z $a : 字符串长度为 0"
else
   echo "-z $a : 字符串长度不为 0"
fi
if [ -n "$a" ]
then
   echo "-n $a : 字符串长度不为 0"
else
   echo "-n $a : 字符串长度为 0"
fi
if [ $a ]
then
   echo "$a : 字符串不为空"
else
   echo "$a : 字符串为空"
fi
```

五、流程控制

if

```
#末尾的fi就是if倒过来拼写
if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi
```

if else

if else-if else

```
#!/bin/bash
a=10
b=20
if [ $a == $b ]
then
   echo "a 等于 b"
elif [ $a -gt $b ]
then
   echo "a 大于 b"
elif [ $a -lt $b ]
then
   echo "a 小于 b"
else
   echo "没有符合的条件"
fi
```

for 循环

```
#!/bin/bash
for loop in 1 2 3 4 5
do
    echo "The value is: $loop"
done
输出结果：
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5
```

```
for str in 'This is a string'
do
    echo $str
done
```

输出结果：

```
This is a string
```

while 语句

```
#!/bin/bash
int=1
while(( $int<=5 ))
do
    echo $int
    let "int++"
done
```

运行脚本，输出：

```
1
2
3
4
5
```

case

```
#!/bin/bash
echo '输入 1 到 4 之间的数字:'
echo '你输入的数字为:'
read aNum
case $aNum in
    1)  echo '你选择了 1'
    ;;
    2)  echo '你选择了 2'
    ;;
    3)  echo '你选择了 3'
    ;;
    4)  echo '你选择了 4'
    ;;
    *)  echo '你没有输入 1 到 4 之间的数字'
    ;;
esac
```

输入不同的内容，会有不同的结果，例如：

```
输入 1 到 4 之间的数字:
你输入的数字为:
3
你选择了 3
```

跳出循环

```
#!/bin/bash
while :
do
    echo -n "输入 1 到 5 之间的数字:"
    read aNum
    case $aNum in
        1|2|3|4|5) echo "你输入的数字为 $aNum!"
        ;;
        *) echo "你输入的数字不是 1 到 5 之间的! 游戏结束"
            break
        ;;
    esac
done
```

```
#!/bin/bash
while :
do
    echo -n "输入 1 到 5 之间的数字: "
    read aNum
    case $aNum in
        1|2|3|4|5) echo "你输入的数字为 $aNum!"
        ;;
        *) echo "你输入的数字不是 1 到 5 之间的!"
            continue
            echo "游戏结束"
        ;;
    esac
done
```

case的语法和C family语言差别很大，它需要一个esac（就是case反过来）作为结束标记，每个case分支用右圆括号，用两个分号表示break。

六、函数

linux shell 可以用户定义函数，然后在shell脚本中可以随便调用。

函数定义：

shell中函数的定义格式如下：

```
[ function ] funname [()]
{
    action;
    [return int;]
}
说明：
1、可以带function fun() 定义，也可以直接fun() 定义,不带任何参数。
```

2、参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。 return后跟数值n(0-255)

```
#!/bin/bash

demoFun(){
    echo "这是我的第一个 shell 函数!"
}
echo "-----函数开始执行-----"
demoFun
echo "-----函数执行完毕-----"
```

输出结果：

```
-----函数开始执行-----
这是我的第一个 shell 函数!
-----函数执行完毕-----
```

```
#!/bin/bash

 funWithReturn(){
    echo "这个函数会对输入的两个数字进行相加运算..."
    echo "输入第一个数字: "
    read aNum
    echo "输入第二个数字: "
    read anotherNum
    echo "两个数字分别为 $aNum 和 $anotherNum !"
    return $(($aNum+$anotherNum))
}
funWithReturn
echo "输入的两个数字之和为 $? !"
```

输出类似下面：

```
这个函数会对输入的两个数字进行相加运算...
输入第一个数字:
1
输入第二个数字:
2
两个数字分别为 1 和 2 !
输入的两个数字之和为 3 !
#函数返回值在调用该函数后通过 $? 来获得。
```

函数参数：

```
#!/bin/bash

funWithParam(){
    echo "第一个参数为 $1 !"
    echo "第二个参数为 $2 !"
    echo "第十个参数为 $10 !"
    echo "第十个参数为 ${10} !"
    echo "第十一个参数为 ${11} !"
    echo "参数总数有 $# 个!"
    echo "作为一个字符串输出所有参数 $* !"
```

```
}
funWithParam 1 2 3 4 5 6 7 8 9 34 73

输出结果：
第一个参数为 1 ！
第二个参数为 2 ！
第十个参数为 10 ！
第十个参数为 34 ！
第十一个参数为 73 ！
参数总数有 11 个!
作为一个字符串输出所有参数 1 2 3 4 5 6 7 8 9 34 73 ！

注意，$10 不能获取第十个参数，获取第十个参数需要${10}。当n>=10时，需要使用${n}来获取参数。
```

七、项目脚本

1.切割脚本

```
#!/bin/bash

#get the date of yesterday
YESTERDAY=`date -d '-1 day' "+%Y%m%d"`

#mv source target
#define the source file
SOURCE_FILE=/export/data/nginx/user_log/access.log

#define the target file
LOG_FILE=/export/data/nginx/logs/$YESTERDAY/access_$YESTERDAY.log
mkdir -p /export/data/nginx/logs/$YESTERDAY

#move
mv $SOURCE_FILE $LOG_FILE

#recreate the nginx log file
kill -USR1 `cat /var/run/nginx.pid`
```

2.上传脚本

```
#!/bin/bash

#get the date of yesterday
YESTERDAY=`date -d '-1 day' "+%Y%m%d"`

#set the source file path
DAY_LOG=/export/data/nginx/logs/$YESTERDAY/access_$YESTERDAY.log

#set the hdfs file path
HDFS_PATH=/nginx/$YESTERDAY/access_$YESTERDAY.log
```

```
#export the user of Hadoop
#export HADOOP_USER_NAME=laowang

#create the hdfs path
hadoop fs -mkdir -p /nginx/$YESTERDAY

#load log to hdfs
hadoop fs -put $DAY_LOG $HDFS_PATH
```