**shell day 02**

================================================

**shell中的运算**

**1,方法一 expr**

**2,方法二用$[ ]结构，配合echo输出，同样可以实**

**现加 减 乘 除 取余,另外$(( ))也可以实现相同效果**

echo $[1+1] 或者 echo $((1+1))

echo $[2-1]

echo $[2\*2]

echo $[4/2]

echo $[5%3]

**3,使用let命令,不输出计算结果，专用于创建变量，或者对变量进行自增减**

功能一: let a=1+1 创建变量a

功能二: 对变量进行自增减

常规写法 主流写法

let a=a+1 let a++ 变量的自增减，将变量a的值加1

let a=a-1 let a-- 变量a减1

let a=a+2 let a+=2 变量a加2

let a=a-2 let a-=2 变量a减2

let a=a\*10 let a\*=10 变量a乘以10

**4，使用bc 计算器，可以进行小数运算**

echo "1.1+1" | bc //非交互的方式使用bc工具

echo "10/3" | bc

echo "scale=3;10/3" | bc //scale可以定义小数点后面的长度

**条件测试,可以为脚本提供智能判断的能力**

**使用方式: test 表达式 或者 [ 表达式 ]**

**1,对字符串进行判断**

可以使用的方式有： == 两边是否相等 != 两边是否不等

-z 判断变量是否为空 ! -z 判断变量是否非空

[ a == a ] 判断字符串a是否等于a，如果相等，echo $?的返回值就是0

a=abc

b=xyz

[ $a == $b ] //判断变量a是否等于变量b,如果相等,则正确

[ $a != $b ] //判断变量a是否不等于变量b,如果不相等,则正确

[ -z $a ] 判断变量a是否为空，如果是，返回值是0

[ ! -z $a ] 判断变量a是否非空，如果是，返回值是0

**2,逻辑符号**

&& 之前任务成功就执行之后任务

|| 之前任务失败就执行之后任务

编写脚本,仅可以让管理员运行

#!/bin/bash

[ $USER == root ] || exit //如果当前用户不是管理员，则退出

yum -y install vsftpd

systemctl start vsftpd

**编写脚本可以创建用户，如果用户没有输入用户名直接敲回车则结**

**束脚本程序**

#!/bin/bash

read -p "请输入用户名：" u

[ -z $u ] && exit //如果没有输入用户名就退出脚本，写法一

[ ! -z $u ] || exit //如果没有输入用户名就退出脚本，写法二

useradd $u

**逻辑符号的组合应用**

**当2个逻辑符号连用时，第二个逻辑符号后面的任务是否执行不能**

**只看前面一个任务，要看前面所有任务的组合，比如下列情况：**

touch a b c //创建3个测试文件

ls a && ls b && ls c //结果是都显示，第一个ls执行成功，然后导致第二个ls也执

行，并且可以成功，这样的话第一个和第二个任务都成功了，那么这个组合就算成

功，然后会导致第二个逻辑符号&&后面的ls c 也执行并且成功。

ls a && ls b || ls c //结果是显示a和b，第一个ls执行成功，然后导致第二个ls也

执行，并且可以成功，这样的话第一个和第二个任务都成功了，那么这个组合就算

成功，但是由于第二个逻辑符号是||，就不会执行最后的ls c任务了

ls a || ls b && ls c //结果是显示a和c，第一个ls执行成功，第二个ls就不会执行

了，这两个任务中间是|| ，那么只要有一个成功就算这个组合执行成功，所以就会

导致第二个逻辑符号&&后面的ls c执行

ls a || ls b || ls c //结果是显示a，第一个ls执行成功，第二个ls就不会执行了，这

两个任务中间是|| ，那么只要有一个成功就算这个组合执行成功，由于这个组合算

成功，所以就不会执行第二个||后面的任务了

**编写脚本,创建用户时如果没给用户名则给出提示信息并退出**

#!/bin/bash

read -p "请输入用户名：" u

[ -z $u ] && echo "你到是给个名字阿！" && exit

useradd $u

**3,数字的条件测试**

-eq等于 -ne不等于 -gt大于 -ge大于等于 -lt小于 -le小于等于

[ 1 -eq 1 ] 判断1是否等于1，判断成功的话返回值是0，判断失败返回值是非0

a=10

b=20

[ $a -eq $b ] //判断两个变量是否相等

[ $UID -eq 0 ] //判断当前用户是否为管理员

**编写脚本,每隔2分钟检查登录服务器的账户数量,如果超过3人,就发**

**邮件给管理员**

#!/bin/bash

x=$(who | wc -l) //统计登录服务器的账户数量，存在变量x里

[ $x -gt 3 ] && echo "有人入侵服务器！牛老师来了！～～～" | mail -s test root

**脚本写完后，执行下列指令：**

rm -rf /var/spool/mail/root //清空邮箱(非必须)

[root@svr7 opt]#chmod +x test03.sh //给脚本添加执行权限

[root@svr7 opt]# crontab -e //编写计划任务

\*/2 \* \* \* \* /opt/test03.sh //每2分钟执行脚本

然后故意多登陆几个账户，每2分钟就可以收到新邮件

**4,文件的条件测试**

-e 判断是否存在,不关心文件类型

-f 判断是否存在且是普通文件

-d 判断是否存在且是目录

-r 判断当前用户对文件是否可读,对root无效

-w判断当前用户对文件是否可写,对root无效

-x判断当前用户对文件是否可执行(目录是否可以进入)

[ -e a ] 当前位置如果有a文件，不关心类型，则测试成功

[ -f a ] 当前位置如果有叫a的普通文件，则测试成功

[ -d a ] 当前位置如果有叫a的目录，则测试成功

[ -r a ] 当前用户如果对a文件有读权限，则测试成功

[ -w a ] 当前用户如果对a文件有写权限，则测试成功

[ -x a ] 当前用户如果对a文件有执行(目录是进入)权限，则测试成功

-----------------------------------------------------------------------------------------------------

**虽然&&和||可以实现逻辑组合，但如果情况比较复杂就可能需要n多个&&或者||，会导致脚本难写，可读性差。此时就可以使用if分支**

**if分支**

**1,单分支**

语法结构:

if 条件测试;then //如果条件测试成功，那就执行下面的所有指令(指令可以有很多)

执行指令1

执行指令2

执行指令3

fi

#!/bin/bash

if [ $UID -eq 0 ];then //如果当前用户的id号是0,就执行下面任务

echo abc

fi

**2,双分支**

if 条件测试;then //如果条件测试成功，那就执行下面的指令1

执行指令1

else //如果上述条件测试失败，那就执行下面的指令2

执行指令2

fi

#!/bin/bash

if [ $UID -eq 0 ];then //判断如果当前用户的id号是0,就输出ok

echo ok

else //如果id号不是0,就输出no

echo no

fi

**ping -c(ping的次数)3 -i(ping的间隔时间) 0.2**

**-W(如果ping不通多久反馈结构) 1**

**编写脚本测试某ip是否能ping通，但不要显示过程，仅仅显示通了或者**

**不通即可**

ping -c 3 -i 0.1 -W 1 192.168.4.7 &> /dev/null //ping的具体过程扔黑洞不看

if [ $? -eq 0 ];then //使用$?可以判断上面的任务是否成功，如果等于0就

是成功

echo "通了！"

else

echo "不通！"

fi

**3,多分支,** **适合更复杂的情况，结果会有很多的情况**

if 条件测试1;then //如果条件测试成功，那就执行下面的指令1

执行指令1

elif 条件测试2;then //如果条件测试成功，那就执行下面的指令2

执行指令2

elif 条件测试3;then //如果条件测试成功，那就执行下面的指令3

执行指令3

else

执行指令n //如果上述条件测试都失败，那就执行指令n

fi

**练习：**

#!/bin/bash

read -p "请输入你的期末考试成绩：" n

if [ $n -ge 90 ];then

echo "优秀！"

elif [ $n -ge 80 ];then

echo "良好！"

elif [ $n -ge 60 ];then

echo "合格！"

else

echo "晚上没饭了！"

fi

**循环**

**有时我们需要某个或者某群任务反复在服务器中执行很多次，就可以使用循环命**

**令写成脚本，而无需手工一次次执行**

**for循环 可以定义循环次数，有限的循环**

**基本语法格式：**

for 变量名 in 值1 值2 值3 。。。。 //此处变量名可以自定义，通常习惯用i，值的多少决定了下面do与done之间的任务执行多少次，每个值之间有空格，这里是有3个值，所以就循环执行指令3次

do

指令

done

------------------------------------------------

#!/bin/bash

for i in a b c //给了abc三个值，下面的echo任务就循环三次

do

echo "abc"

done

-----------------------------------------------

#!/bin/bash

for i in {1..100} //循环100次时无需填写100个值，用此格式即可，代表1到100，一共100个值

do

echo "abc"

echo "$i" //执行任务时，还可以调用变量，变量的值是所有值的轮询，由于这里给了1到100的值，那第一次循环$i就是1，第二次循环$i就是2，依次类推到第100次循环$i就是100

done

**编写脚本，使用ping测试192.168.4.1~192.168.4.10，并显示通了或者不通，并且统计多少台通了，多少台不通**

#!/bin/bash

x=0 //定义通了的数量,没测试之前是0台

y=0 //定义没通的数量,没测试之前是0台

for i in {1..10}

do

ping -c 3 -i 0.1 -W 1 192.168.4.$i &> /dev/null

if [ $? -eq 0 ];then

echo "192.168.4.$i 通了！"

let x++ //每通了一次就把x+1

else

echo "192.168.4.$i 不通！"

let y++ //每不通一次就把y+1

fi

done

echo "$x台通了，$y台不通"

练习：

1，列出常见的整数值比较操作，并说明各自作用。

2，运用条件测试操作，检查当前的用户是否为root。

3，绘图描述if双分支结构的执行流程。

4，[ -z $abc ] && echo "yes" || echo "no" 可以实现什么测试效果。

参考答案

1，列出常见的整数值比较操作，并说明各自作用。

-eq 等于

-ne 不等于

-ge 大于等于

-le 小于等于

-gt 大于

-lt 小于

2，运用条件测试操作，检查当前的用户是否为root。

[ $USER == "root" ] && echo "yes" || echo no

3，绘图描述if双分支结构的执行流程。

if双分支结构判断一次条件，当条件成立时执行分支1、若不成立则执行分支2，如图-1所示。

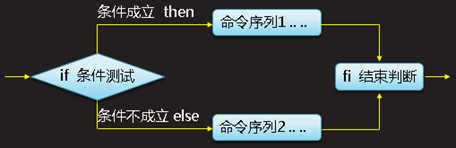


图-1

4，[ -z $abc ] && echo "yes" || echo "no" 可以实现什么测试效果。

如果变量abc是空则显示yes，否则显示no