Shell脚本：可以执行的一个文件，可以实现某种功能

中文：创建用户dc Shell：useradd dc

案例1：书写hello.sh脚本

[root@server0 ~]# vim /root/hello.sh

echo hello world

[root@server0 ~]# /root/hello.sh

-bash: /root/hello.sh: 权限不够

[root@server0 ~]# chmod +x /root/hello.sh #赋予执行权限

[root@server0 ~]# ls -l /root/hello.sh

-rwxr-xr-x. 1 root root 17 8月 8 08:57 /root/hello.sh

[root@server0 ~]# /root/hello.sh

规范Shell脚本的一般组成

• #! 环境声明，下面所有代码由那个程序进行翻译

• # 注释文本

• 可执行代码

案例2：书写hello.sh脚本

1)输出当前红帽系统的版本信息

2)输出当前使用的内核版本

3)输出当前系统的主机名

[root@server0 ~]# vim /root/hello.sh

#!/bin/bash

echo hello world

cat /etc/redhat-release

uname -r

hostname

ifconfig | head -2

[root@server0 ~]# /root/hello.sh

####################################################

重定向输出

>: 只收集前面命令的正确输出

2>: 只收集前面命令的错误输出

&>: 收集前面命令的正确与错误输出

>&2:将当前输出变成错误输出

]# echo 123 > /opt/1.txt

]# cat /opt/1.txt

]# cat /opt/1.txt /a

]# cat /opt/1.txt /a > /opt/nsd.txt #只收集正确信息

]# cat /opt/nsd.txt

]# cat /opt/1.txt /a 2> /opt/nsd.txt #只收集错误信息

]# cat /opt/nsd.txt

]# cat /opt/1.txt /a &> /opt/nsd.txt #收集正确与错误信息

]# cat /opt/nsd.txt

单引号 ''：取消所有字符的特殊意义，原样输出

[root@server0 ~]# echo '\*'

[root@server0 ~]# echo '\* | >? ^& $%'

案例3：书写创建用户的脚本，并且设置密码

/dev/null:黑洞设备，专用于收集不要的输出信息

[root@server0 ~]# vim /root/user.sh

#!/bin/bash

useradd nsd04 &> /dev/null

echo 用户nsd04创建成功

echo 123 | passwd --stdin nsd04 &> /dev/null

echo 用户nsd04密码设置成功

[root@server0 ~]# chmod +x /root/user.sh

[root@server0 ~]# /root/user.sh

为了提高脚本适应多变的环境与需求，所以引入了变量

变量类似于容器：以不变的名称,存储可以变化的值,方便重复运用某一个值

变量名=存储的值

[root@server0 ~]# vim /root/user.sh

#!/bin/bash

a=nsd08

useradd $a &> /dev/null

echo 用户$a创建成功

echo 123 | passwd --stdin $a &> /dev/null

echo 用户$a密码设置成功

为了降低脚本的使用难度，以及用户的体验，所以产生交互以及提示信息

read -p '屏幕的提示信息'

1.产生交互

2.记录用户在键盘上的输入

3.将记录的所有内容，最终赋值给一个变量储存

[root@server0 ~]# vim /root/user.sh

#!/bin/bash

read -p '请输入您要创建的用户名：' a

useradd $a &> /dev/null

echo 用户$a创建成功

echo 123 | passwd --stdin $a &> /dev/null

echo 用户$a密码设置成功

####################################################

定义/赋值变量

• 设置变量时的注意事项

– 等号两边不要有空格

– 变量名只能由字母/数字/下划线组成,区分大小写

– 变量名不能以数字开头,不要使用关键字和特殊字符

– 若指定的变量名已存在,相当于为此变量重新赋值

查看/引用变量

• 基本格式

– 引用变量值:$变量名

– 查看变量值:echo $变量名、echo ${变量名}

[root@server0 ~]# a=rhel

[root@server0 ~]# echo $a

rhel

[root@server0 ~]# echo $a7

[root@server0 ~]# echo ${a}7

[root@server0 ~]# b=8

[root@server0 ~]# echo ${a}${b}

##################################################

$( )或 反撇号` `: 将命令的输出，直接作为参数，参与运行

[root@server0 ~]# date

[root@server0 ~]# date +%F #显示 年-月-日

[root@server0 ~]# cd /opt/

[root@server0 opt]# mkdir nsd-$(date +%F)

[root@server0 opt]# ls

[root@server0 opt]# mkdir $(hostname)-$(date +%F)

[root@server0 opt]# ls

[root@server0 opt]# mkdir abc-`hostname`

[root@server0 opt]# ls

##################################################

变量的种类

环境变量：变量名一般为大写字母，由系统定义并且赋值完成

USER=当前系统登陆的用户名

[root@server0 /]# echo $USER

root

[root@server0 /]# su - dc

[dc@server0 ~]$ echo $USER

dc

[dc@server0 ~]$ exit

logout

[root@server0 /]# su - dcc

[dcc@server0 ~]$ echo $USER

dcc

[dcc@server0 ~]$ exit

logout

[root@server0 /]#

###################################################

位置变量:由系统定义并且赋值完成

• 在执行脚本时提供的命令行参数

– 表示为 $n,n为序号

– $1、$2、$3、$4.. .. ${10}、${11}、.. ..

[root@server0 /]# vim /root/2.sh

#!/bin/bash

cat -n $1 | head -$2

[root@server0 /]# chmod +x /root/2.sh

[root@server0 /]# /root/2.sh /etc/passwd 3

[root@server0 /]# vim /root/3.sh

#!/bin/bash

echo $1

echo $2

echo $3

[root@server0 /]# chmod +x /root/3.sh

[root@server0 /]# /root/3.sh harry tom zhangsan

[root@server0 /]# vim /root/user.sh

#!/bin/bash

#read -p '请输入您要创建的用户名：' a #开头加上#注释

useradd $1 &> /dev/null

echo 用户$1创建成功

echo 123 | passwd --stdin $1 &> /dev/null

echo 用户$1密码设置成功

[root@server0 /]# /root/user.sh harry

用户harry创建成功

用户harry密码设置成功

####################################

预定义变量:由系统定义并且赋值完成

$# 已加载的位置变量的个数,专用于用户是否输入参数

[root@server0 /]# vim /root/3.sh

#!/bin/bash

echo $1

echo $2

echo $3

echo $#

[root@server0 /]# /root/3.sh aa bb cc dd

$? 程序退出后的状态值,0表示正常,其他值异常

##################################################

条件测试

– [ 测试表达式 ] #每一部分都要有空格

常用的测试选项

• 检查文件状态

-e:存在为真

-d:存在，且必须为目录才为真

-f:存在，且必须为文件才为真

[root@server0 /]# [ -e /etc ] #判断/etc目录是否存在

[root@server0 /]# echo $?

0 #表示正常

[root@server0 /]# [ -e /ete ] #判断/ete目录是否存在

[root@server0 /]# echo $?

1 #非0都表示异常

[root@server0 /]# [ -d /etc/passwd ]

[root@server0 /]# echo $?

1

[root@server0 /]# [ -f /etc/passwd ]

[root@server0 /]# echo $?

0

• 比较整数大小

-gt：大于

-ge：大于等于

-eq：等于

-ne：不等于

-lt：小于

-le：小于等于

[root@server0 /]# [ 1 -eq 2 ] #判断是否相等

[root@server0 /]# echo $?

1

[root@server0 /]# [ 10 -gt 11 ] #判断10是否大于11

[root@server0 /]# echo $?

1

[root@server0 /]# [ 10 -ge 10 ] #判断10是否大于等于10

[root@server0 /]# echo $?

0

[root@server0 /]# [ 10 -gt 10 ] #判断10是否大于10

[root@server0 /]# echo $?

1

• 字符串比对

==:一致为真

!=:不一致为真

[root@server0 /]# [ dc == tc ]

[root@server0 /]# echo $?

1

[root@server0 /]# [ $USER == student ]

[root@server0 /]# echo $?

1

[root@server0 /]# [ $USER == root ]

[root@server0 /]# echo $?

0

[root@server0 /]# [ dc != tc ]

[root@server0 /]# echo $?

0

####################################################

if选择结构

if双分支处理

if [条件测试];then

命令序列xx

else

命令序列yy

fi

[root@server0 /]# vim /root/if01.sh

#!/bin/bash

if [ $1 -eq 1 ];then

echo I Love Dc

else

echo Hello World

fi

[root@server0 /]# chmod +x /root/if01.sh

[root@server0 /]# /root/if01.sh 1

[root@server0 /]# /root/if01.sh 2

案例：判断用户是否输入参数

[root@server0 /]# vim /root/if02.sh

#!/bin/bash

if [ $# -eq 0 ];then #判断用户输入参数的个数是否为0

echo 您没有输入任何参数

else

echo 您输入了参数，一共$#个参数

fi

[root@server0 /]# chmod +x /root/if02.sh

[root@server0 /]# /root/if02.sh

[root@server0 /]# /root/if02.sh harry

案例：利用 read进行获得用户，所输入的用户名

进行判断 如果当前系统有该用户，则输出用户已存在

如果当前系统没有该用户，则创建该用户

[root@server0 /]# vim /root/if03.sh

#!/bin/bash

read -p '请输入您要测试的用户名：' a

id $a &> /dev/null

if [ $? -eq 0 ];then

echo 用户$a已存在

else

useradd $a

echo 用户不存在，已经将$a创建成功

fi

[root@server0 /]# chmod +x /root/if03.sh

[root@server0 /]# /root/if03.sh

#############################################

if多分支处理

if [条件测试1];then

命令序列xx

elif [条件测试2];then

命令序列yy

elif [条件测试3];then

命令序列aa

else

命令序列zz

fi

案例：利用read获取用户的成绩

进行判断 如果 成绩大于等于90，则输出 优秀

如果 成绩大于等于80，则输出 良好

如果 成绩大于等于70，则输出 一般

如果 成绩大于等于60，则输出 合格

以上条件均不满足，再牛的肖邦也谈不出哥的悲伤

[root@server0 /]# vim /root/if04.sh

#!/bin/bash

read -p '请输入您的成绩：' num

if [ $num -ge 90 ];then #判断用户成绩是否大于等于90

echo 优秀

elif [ $num -ge 80 ];then #判断用户成绩是否大于等于80

echo 良好

elif [ $num -ge 70 ];then #判断用户成绩是否大于等于70

echo 一般

elif [ $num -ge 60 ];then #判断用户成绩是否大于等于60

echo 合格

else

echo 再牛的肖邦也弹不出哥的悲伤

fi

[root@server0 /]#

###################################################

for循环结构：重复性的操作

for循环处理

• 遍历/列表式循环

– 根据变量的不同取值,重复执行xx处理

for 变量名 in 值列表

do

重复性执行的代码

done

for 献血车 in zhangsan lisi wangwu

do

抽血

done

[root@server0 /]# vim /root/for01.sh

#!/bin/bash

for a in zhangsan lisi wangwu haha

do

useradd $a &> /dev/null

echo $a创建成功

done

[root@server0 /]# chmod +x /root/for01.sh

[root@server0 /]# /root/for01.sh

列表值可以不参与循环执行的代码

造数工具：制造连续范围的数字

{起始值..结束值}

[root@server0 /]# vim /root/for01.sh

#!/bin/bash

for a in {1..1000}

do

echo I Love Dc $a

done

[root@server0 /]#

#################################################

案例4:编写一个判断脚本

在 server0 上创建 /root/foo.sh 脚本

1)当运行/root/foo.sh redhat,输出为fedora

2)当运行/root/foo.sh fedora,输出为redhat

3)当没有任何参数或者参数不是 redhat 或者fedora时,

其错误输出产生以下信息: /root/foo.sh redhat|fedora

[root@server0 /]# vim /root/foo.sh

#!/bin/bash

if [ $# -eq 0 ];then #判断用户是否输入参数

echo '/root/foo.sh redhat|fedora'

elif [ $1 == redhat ];then #判断输入是否为redhat

echo fedora

elif [ $1 == fedora ];then #判断输入是否为fedora

echo redhat

else

echo '/root/foo.sh redhat|fedora' #单引取消特殊字符含义

fi

###################################################

案例5:编写一个批量添加用户脚本

在 server0 上创建 /root/batchusers 脚本

1)此脚本要求提供用户名列表文件作为参数

2)如果没有提供参数,此脚本应该给出提示

Usage: /root/batchusers,退出并返回相应值

3)如果提供一个不存在的文件,此脚本应该给出提

示 Input file not found,退出并返回相应值

4)新用户的登录Shell为 /bin/false,无需设置密码

5)用户列表测试文件:

http://classroom/pub/materials/userlist