什么是脚本: 可以执行的一个文件,实现某种功能

创建用户tom = useradd tom

将我们所学习的命令,按照顺序写入文本文件,赋予文本文件执行权限

第一个Shell脚本:

1. 编写一个面世问候 /root/hello.sh 脚本

– 显示出一段话 Hello World

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

echo hello world

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

[root@server0 ~]# /root/hello.sh #绝对路径运行

规范Shell脚本的一般组成

• #! 环境声明,以下所有代码由那个程序进行翻译执行

• # 注释文本

• 可执行代码

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 ~]# chmod +x /root/hello.sh

[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

[root@server0 ~]# echo 123 > /opt/1.txt

[root@server0 ~]# cat /opt/1.txt

[root@server0 ~]# cat /a

[root@server0 ~]# cat /opt/1.txt /a

[root@server0 ~]# cat /opt/1.txt /a > /opt/2.txt

[root@server0 ~]# cat /opt/2.txt

[root@server0 ~]# cat /opt/1.txt /a 2> /opt/2.txt

[root@server0 ~]# cat /opt/2.txt

[root@server0 ~]# cat /opt/1.txt /a &> /opt/2.txt

[root@server0 ~]# cat /opt/2.txt

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

单引号 ' ' : 取消所有特殊字符意义

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

[root@server0 ~]# echo '\* ? {} [] |'

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

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

反撇号 ` ` 与 $( ) : 将命令的输出,当做参数直接参与下一个命令的运行

每一天在/opt/目录下创建一个以 年-月-日 结尾的目录

[root@server0 ~]# date +%F

[root@server0 ~]# cd /opt

[root@server0 opt]# mkdir nsd-`date +%F`

[root@server0 opt]# ls

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

[root@server0 opt]# ls

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

[root@server0 opt]# ls

[root@server0 ~]# date --help

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

案例:书写创建用户设置密码

黑洞设备:/dev/null

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

#!/bin/bash

a=harry

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

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

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

变量:为了适应多变的环境需求

变量:会变化的量 以不变的名称存储可以变化的值 类似容器

变量名=存储的值

使用变量储存的值: $变量名

为了用户的体验,降低脚本的使用难度,产生交互

向脚本中传递命令行参数

read -p '屏幕输出信息'

1.产生交互

2.记录用户在键盘输入

3.将记录的信息,赋值给一个变量储存

[root@server0 /]# cat /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

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

什么是变量

• 以不变的名称存放的可能会变化的值

– 变量名=变量值

– 方便以固定名称重复使用某个值

– 提高对任务需求、运行环境变化的适应能力

– 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

• 设置变量时的注意事项

– 若指定的变量名已存在,相当于为此变量重新赋值

– 等号两边不要有空格

• 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

[root@server0 /]# echo ${a}

[root@server0 /]# echo $a7

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

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

变量的种类

环境变量 变量名一般都大写,用来设置用户/系统环境

位置变量 bash内置,存储执行脚本时提供的命令行参数

预定义变量 bash内置,可直接调用的特殊值,不能直接修改

自定义变量 用户自主设置、修改及使用

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

环境变量:由系统定义并且赋值完成,一般变量名为大写

USER=当前登录的用户名

[root@server0 /]# echo $USER

root

[root@server0 /]# su - nsd01

[nsd01@server0 ~]$ echo $USER

nsd01

[nsd01@server0 ~]$ exit

logout

[root@server0 /]# su - tom

[tom@server0 ~]$ echo $USER

tom

[tom@server0 ~]$ exit

logout

[root@server0 /]#

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

位置变量: 由系统定义完成

• 在执行脚本时提供的命令行参数

– 表示为 $n,n为序号

– $1、$2、.. .. ${10}、${11}、.. ..

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

#!/bin/bash

#read -p '请输入您要创建的用户名:' a

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 /]# vim /root/1.sh

#!/bin/bash

echo $1

echo $2

echo $3

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

#!/bin/bash

cat -n $1 | head -$2

echo 显示文件$1内容的前$2行

[root@server0 /]# /root/2.sh /etc/passwd 4

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

预定义变量:

• 用来保存脚本程序的执行信息

– 直接使用这些变量

– 不能直接为这些变量赋值

$# 已加载的位置变量的个数

$? 程序退出后的状态值,0表示正常,其他值异常

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

#!/bin/bash

echo $1

echo $2

echo $3

echo $#

[root@server0 /]# /root/1.sh dc tc

[root@server0 /]# /root/1.sh a b c d

判断用户是否输入参数?

当$#的值为0的时候,说明用户没有输入参数

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

测试操作

[ 测试表达式 ]

• 检查文件状态

-e:文档存在才为真

-d:文档存在,且必须为目录才为真

-f:文档存在,且必须为文件才为真

[root@server0 /]# [ -e /etc/haha ]

[root@server0 /]# echo $?

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

[root@server0 /]# echo $?

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

[root@server0 /]# echo $?

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

[root@server0 /]# echo $?

• 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 /]# [ 10 -le 11 ]

[root@server0 /]# echo $?

• 字符串比对

==:一致为真

!=:不一致为真

[root@server0 /]# [ haha == xixi ]

[root@server0 /]# echo $?

1

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

[root@server0 /]# echo $?

0

[root@server0 /]# [ haha != xixi ]

[root@server0 /]# echo $?

0

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

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 hello

else

echo hi

fi

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

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

案例:用户是否输入参数的脚本

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

#!/bin/bash

if [ $# -eq 0 ];then

echo 您没有输入参数

else

echo 您输入了参数

fi

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

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

案例:

利用read获取用户输入的用户名

进行判断:

如果输入的用户存在,则输出 用户已存在

如果输入的用户不存在,则输出 用户不存在

[root@server0 /]# cat /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 $a存在

else

echo $a不存在

fi

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

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

else

命令序列zz

fi

案例:成绩查询脚本

利用read获取用户输入的成绩

进行判断:

如果大于等于90,则输出 优秀

如果大于等于80,则输出 良好

如果大于等于70,则输出 一般

如果大于等于60,则输出 合格

以上条件均不满足则输出 一首凉凉送给你

#!/bin/bash

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

read -p '请输入您的成绩:' num

if [ $num -ge 90 ];then

echo 优秀

elif [ $num -ge 80 ];then

echo 良好

elif [ $num -ge 70 ];then

echo 一般

elif [ $num -ge 60 ];then

echo 合格

else

echo '一首凉凉送给你!'

fi

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

列表式循环场景

• 给定一批对象,反复执行类似的操作

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

命令序列

done

for 献血车 in dc tc zhangsan dcc

do

抽血

done

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

#!/bin/bash

for a in zhangsan lisi dc tc nsd15

do

useradd $a &> /dev/null

echo $a创建成功

done

循环的列表值,可以不参与循环执行的重复操作

[root@server0 /]# vim /root/for02.sh

#!/bin/bash

for a in {1..5}

do

echo hello world

done

[root@server0 /]# /root/for02.sh

造数工具:{起始值..结束值}

{1..20}:制造1到20连续数字

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

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 /]# cat /root/foo.sh

#!/bin/bash

# [ $# -eq 0 ] 判断用户是否输入参数

# >&2 修改为错误输出

# exit 退出返回脚本状态值

if [ $# -eq 0 ];then

echo '/root/foo.sh redhat|fedora' >&2

exit 3

elif [ $1 == redhat ];then

echo fedora

elif [ $1 == fedora ];then

echo redhat

else

echo '/root/foo.sh redhat|fedora' >&2

exit 4

fi

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