# Shell的基本功能

/etc/shells可以查看当前系统支持的shell类型

## 输入输出重定向

标准输入 0

标准输出 1

错误输出 2

> 命令的正确执行，以覆盖的形式输出到文件

>>命令的正确执行，以追加的形式输出到文件

2> (2与>之间没有空格)，命令的错误执行。以覆盖的形式输出到文件

2>> 命令的错误执行。以追加的形式输出到文件

exa: echo "123456" >abc.txt

echo "hello" >>abc.txt

catc abc.txt 2> a.txt

&> 正确的输出和错误的输出保存到一个文件，以覆盖的形式

&>> 正确的输出和错误的输出保存到一个文件，以追加的形式

## 多命令顺序执行

; 命令之间没有逻辑关系

&&第一个命令执行正确，第二个才会执行

||第一个命令执行错误，第二个命令才会执行

## 管道符

查找的区别

grep 在文件中查找符合条件的行，并输出

find 在某一个目录下查找符合条件的文件或目录

\*0个或多个字符

？1个字符

[]范围内的字符

[root@localhost test]# echo '$(date)'

$(date)

[root@localhost test]# echo "$(date)"

2017年 09月 26日 星期二 15:06:33 CST

单引号中,$\` 三个字符没有特殊含义

双引号中,$\` 三个字符是有特殊含义

变量的调用 $变量名

如果变量的值含空格，一定要用单引号或者双引号括起来

Shell脚本

第一行 #!/bin/bash

赋予文件可执行的权限

chmod 755 t1.sh

./t1.sh

执行脚本文件

bash test1.sh

./test1.sh

source test1.sh

./test/sh/test1.sh（绝对路径的方式）

对用户重设密码

cat p.txt|passwd q12

注意：p.txt密码应该有两行重复的，否则会提示“passwd: 已经超出服务重试的最多次数”

1. Shell编写及运行

vi hello.sh

#!/bin/bash

echo “hello world”

执行前需要修改权限

chmod a+x hello.sh

./hello.sh

1. 变量

定义的同时进行赋值，注意：等号两边不能有空格

变量名由 字母，数字，下划线组成，不能以数字开头

[root@localhost sh]# a=1

[root@localhost sh]# b=2

[root@localhost sh]# echo $a

默认的变量类型是字符串类型，不能直接进行数值运算

用户自定义变量：用户可以增加，修改变量值

环境变量：与系统环境相关的变量，用户可以增加的

如：PS1

位置参数变量：向程序传递参数的。用户不能增加，变量作用是固定的

如：$0,$1,$2,……..

预定义变量：Bash预定义的特殊变量，变量名不能直接修改或增加，变量作用是固定的。如：$?

# 用户自定义变量

name=tom

变量的叠加

[root@localhost sh]# echo "$name"123

[root@localhost sh]# echo ${name}123

set 查看用户自定义变量和系统变量

unset 变量名 删除变量

# 环境变量

定义需要用export声明。也叫做全局变量

export NAME3=tom

env可以查看所有的环境变量

env|more

可以使用export 变量名 把用户自定义的变量提升为环境变量

环境变量在当前shelle以及子shell中都可以使用

用户自定义变量只能在当前shell中使用

# 位置参数

1）$n n是整数, 0代表程序名。1代表第一个输入数值

2）$\* 代表全部的参数，当做一个整体

3）$@代表全部的参数，每个加以区分

4）$# 代表输入参数的个数

#!/bin/bash

echo $0

num1=$1

num2=$2

num3=$3

sum=$(($num1+$num2+$num3))

echo "result=$sum"

echo $\*

echo $@

echo $#

for i in "$\*"

do

echo $i

done

for j in "$@"

do

echo $j

done

# 预定义参数

1）$? 最近一次的命令的运行结果

0表示正确执行

非0表示执行不正确

2）$$ 表示当前的进程号

3）$! 后台运行的最后一次的进程号

# 键盘输入

read [选项] [变量名]

-p 提示信息：

-t 等待时间：超时则不予等待

-s :不显示输入

-n 数字：限制输入字符的个数

# 数值运算

方法一使用 expr + - \\* / %（限于整数）

echo $(expr $n1 + $n2)

echo $(expr $n1 - $n2)

echo $(expr $n1 \\* $n2)

echo $(expr $n1 / $n2)

方法二：声明类型为整数（限于整数）

[root@localhost sh]# declare -i sum

[root@localhost sh]# sum=$n1+$n2

[root@localhost sh]# echo $sum

方法三使用$(()) 或者$[]

[root@localhost sh]# c=$(($n1+$n2))

[root@localhost sh]# echo $c

[root@localhost sh]# c=$[$n1+$n2]

[root@localhost sh]# echo $c

示例2:

* + 每隔3天对数据库目录做一次完整备份
    - 统计 /var/lib/mysql 目录占用的空间大小、查看当前的日期，并记录到临时文件 /tmp/dbinfo.txt 中
    - 将 /tmp/dbinfo.txt 文件、/var/lib/mysql 目录进行压缩归档，备份到/opt/dbbak/目录中
    - 备份后的包文件名中要包含当天的日期信息(20170927.tar.gz)
    - 最后删除临时文件/tmp/dbinfo.txt

cd /test

vi demo2.sh

#!/bin/bash

Size=`du –sh /var/lib/mysql`

Day=`date +%Y%d%m`

echo $Size >> /tmp/dbinfo.txt

echo $Day>>/tmp/dbinfo.txt

cd /opt/dbbak

tar –zcf mysql\_$Day.tar.gz /tmp/dbinfo.txt /var/lib/mysql

rm –rf /tmp/dbinfo.txt

创建定时任务 crontab –e

0 \* \*/3 \* \* /test/demo.sh

通配符用来匹配文件名的，常用于find ls cp rm

\*代表0个或多个字符

？代表1个字符

[1-9]代表满足条件的一个字符[ABCD]

正则表达式是用来匹配文件中满足条件的行

grep、awk、sed

grep -n 'a\*' abc 匹配所有行

grep -n 'aa\*' abc 匹配至少含有一个a的行

grep '^ab' abc 匹配行首是ab的行

grep 'm$' abc 匹配行尾是m的行

grep [^0-9] abc

grep 满足条件的行

截取满足条件的列cut

cut默认是制表符是分割符，如果不是需要用-d指出分割符（空格除外）

cat passwd |grep '/bin/bash'|cut -d : -f 1

cut 的局限性，对于空格不支持

df -h|grep 'sda1'|cut -d " " -f 5

df -h|grep 'sda1'|awk '{print $5}' 解决了空格的问题

BEGIN 从第一行开始就执行。

FS用来指定除空格和制表符外的分隔符

条件测试

test -d m1 &&echo 'yes'

[ -d m1 ] && echo 'yes'

[ -d m1 ] && echo 'yes'

注意[]与与字符中间要加空格

[ $a -gt $b ]

字符串的判断

grep -v ^[a-zA-Z] abc 列出不以字母开头的行

# if

yum –y install vim

yum –y install nmap

yum –y install httpd 安装apache

rpm –q nmap

nmap是一个网络扫描工具

单分支

if [ 判断条件]

then

命令序列

fi

双向分支

if [ 判断条件]

then

命令序列1

else

命令序列2

fi

多向分支

if [ 判断条件]

then

命令序列1

elif [ 判断条件]

then

命令序列1

………

else

命令序列2

fi

* 使用多分支判断用户的输入
* 如果输入为空，提示错误，并退出程序
* 如果不存在，提示不是文件，并退出程序
* 如果是目录文件，则提示是目录
* 如果是普通文件，提示普通文件
* -z是否为空 -e 是否存在-f -d

#!/bin/bash

read -p "please input filename:" fname

if [ -z $fname ]

then

echo "not null"

exit 1

elif [ ! -e $fname ]

then

echo "not exist"

exit 2

elif [ -d $fname ]

then

echo "is directory"

elif [ -f $fname ]

then

echo "is regular file"

else

echo " other file"

fi

# case

case 变量名 in

模式1）

命令列表1

;;

模式2）

命令列表2

;;

\*)

命令列表

esac

# for

语法1：（最常见）

for 变量名 in 取值列表

do

命令序列

done

语法2：

for((变量的初始值；循环控制；变量的变化))

do

命令序列

done

## 实例1

#!/bin/bash

for i in morning noon evening

do

echo $i

done

## 实例2

利用for实现1到100的整数和

#!/bin/bash

sum=0

for((i=1;i<=100;i++))

do

sum=$(($sum+$i))

done

echo $sum

## 实例3

对于使用“/bin/bash”作为登录Shell的系统用户，

cat /etc/passwd|grep "/bin/bash"|cut -d : -f1

检查他们在“/home”目录中拥有的子目录或文件数量，

find /home -user tom –type |wc -l

如果超过100个，则列出具体个数及对应的用户帐号

#!/bin/bash

dir=/home

limit=3

users=`grep "/bin/bash" /etc/passwd|cut -d : -f1`

for u in $users

do

num=`find $dir -user $u |wc -l `

if [ $num -gt $limit ]

then

echo $u $num

fi

done

## 实例4

批量解压缩

tar -zcf s1.tar.gz s1 压缩

tar -zxf s2.tar.gz 解压缩

#!/bin/bash

dir=/test/demo

cd $dir

#ls \*.tar.gz >> /test/new.txt

for i in `ls \*.tar.gz`

do

tar -zxf $i &> /dev/null

done

rm -rf /test/new.txt

# while

while [ 表达式/条件 ]

do

命令列表

done

可以使用let i++ 实现 i的自增

实例1

#!/bin/bash

i=1

read -p "please input username:" username

read -p "please input num:" num

while [ $i -le $num ]

do

useradd $username$i

echo $username$i

cat pass.txt|passwd $username$i &> /dev/null

let i++

done

# shift 实现位置参数的向左移动

#!/bin/bash

echo $#

# $# 输出参数的个数

while [ $# -gt 0 ]

do

echo $1

shift

done

# 函数

function 函数名 {

}

注意 函数名与{之间需要加空格}

$? 两个用途

1. 用于获取上一条命令的执行结果
2. 获取参数的返回值

# 面试准备

1.C，Java 面向对象的特征（编程实现） static jvm原理，冒泡，排序，查找算法的比较

找一些java的笔试题

2.数据结构 链表（单向，双向），队列，栈

3.操作系统 重点是线程，进程

4.网络原理 TCP/IP体系结构 重点：应用层，传输层:

描述打开百度页面的过程，http协议，响应的状态码，tcp三次握手

5.数据库：创建表，三表联合查询，索引，视图，存储过程，触发器

6.测试理论基础。黑盒，白盒测试用例的设计。测试流程，电梯的测试

7.linux基本命令，Shell脚本，定时任务crontab

8.自动化测试