day01

200行

cat /etc/shells //查看本机可用解释器

准备环境

rht-vmctl reset classroom

rht-vmctl reset desktop

rht-vmctl reset server

进入desktop或者server中:

yum -y install ksh //安装新解释器

> >> 收集正确信息

2> 2>> 收集错误信息

&> &>> 收集所有信息

< 输入重定向

bash优点:

历史记录,tab键,快捷键,管道, 重定向

一个标准的shell脚本包含:

1, 声明解释器

2, 注释,脚本功能,变量的含义等

3,执行指令(脚本内容)

#!/bin/bash

#这是一个测试脚本

echo "你好"

执行脚本的方式

1,添加完x权限后,使用完整路径

chmod +x 脚本.sh

/opt/脚本.sh

2,bash 脚本.sh

3,source 脚本.sh

bash 执行脚本时开启子进程

source 执行脚本不开启子进程

pstree可以看到进程树

测试用以下脚本:

#!/bin/bash

mkdir abcd

cd abcd

bash看不到进入目录的效果,新开启的bash进程进入目录后的效果随着脚本结束而消失

source可以看到,因为不开启子进程,只使用当前解释器的进程

编写自动配置yum仓库的脚本:

#!/bin/bash

rm -rf /etc/yum.repos.d/\*.repo

echo "[test]

name=abcdefg

baseurl=http://172.25.0.254/content/rhel7.0/x86\_64/dvd/

enabled=1

gpgcheck=0" > /etc/yum.repos.d/test.repo

执行脚本后,使用yum repolist检查

编写自动安装ftp并设置开机自启的脚本:

#!/bin/bash

yum -y install vsftpd &> /dev/null //安装过程不显示

systemctl start vsftpd

systemctl enable vsftpd

systemctl stop firewalld //运行脚本后关闭防火墙用真机链接虚拟机的ftp测试

vim /etc/vsftpd/vsftpd.conf //ftp的配置文件可以根据需要开启匿名上传,修改第29行, 另外/var/ftp/pub需要给写权限

anon\_upload\_enable=YES //创建文件

anon\_mkdir\_write\_enable=YES //创建目录

如果要测试写入权限,需要修改配置后重启服务,并关掉selinux

systemctl restart vsftpd

setenforce 0

变量,使用变量名代替未知量,大大增加脚本灵活

变量名的定义,使用字母,数字,下划线,不能以数字开头,不能用特殊符号

User=10 //等号两边不要出现空格

User=20 //再次给变量赋值会覆盖之前的值

unset User //删除变量

变量种类:

1,自定义变量

2,系统变量,包含

环境变量 PWD,HOME,UID,HOSTNAME,PATH,USER,SHELL,PS1,PS2

位置变量 $1 $2 $3 ....

预定义变量 $0 $? $$ $\* $#

$0 显示脚本名

$$ 显示脚本进程号

$? 显示上一条指令是否成功 ,0是成功,非0是失败

$# 统计位置变量的个数

$\* 显示所有位置变量的内容

#!/bin/bash

echo $1

echo $2

echo $3

echo $0 显示脚本名

echo $\* 所有参数

echo $# 参数的个数

echo $? 上一任务是否成功,0成功,非0失败

echo $$ 显示进程号(PID)

编写脚本实现 创建用户abc,并配置密码123

#!/bin/bash

useradd abc

echo 123 | passwd --stdin abc

改良版

#!/bin/bash

useradd $1

echo $2 | passwd --stdin $1

env 显示所有环境变量

set 显示所有变量

'' 界定范围,屏蔽特殊符号的效果

"" 界定范围

touch a b // 创建2个文件

touch "a b" //创建1个文件

touch 'a b' //创建1个文件

a=10

echo $a //显示变量内容

echo "$a" //显示变量内容

echo '$a' //显示$a ,$的特殊含义被单引号屏蔽

`` 获取命令结果

mv a `date +%Y%m%d` //将a文件改名为当天日期

tar -czf log-`date +%Y%m%d`.tar.gz /var/log //备份日志改名为log-当天日期.tar.gz

编写创建用户配置密码的脚本:

#!/bin/bash

read -p "请输入用户名" User

useradd $User

stty -echo

read -p "请输入密码" Pass

stty echo

echo $Pass | passwd --stdin $User

stty -echo //关闭回显,屏蔽用户输入的内容,不影响系统弹出的信息

stty echo //开启回显

export 发布全局变量,使子进程可以使用父进程定义的变量

通常没有发布全局变量时,定义的变量为局部效果

测试:

a=10 //定义局部变量

export a //修改为全局变量

export b=20 //新定义+发布全局变量

bash //进入子进程

echo $a //检测全局变量是否可用

echo $b

exit //退出子进程

export -n a //取消变量a的全局效果,恢复局部变量效果

bash //进入子进程

echo $a //检测效果,发现无内容

shell中的运算

方法一:

expr + - \\* / %求模(取余数)

注意,运算符号两边必须加空格,乘号需要使用\(转义符号)或者''(单引号)取消原本含义

方法二:使用$[]或$(())表达式

echo $[1+1] 或 echo $((1+1))

echo $[2-1]

echo $[2\*2]

echo $[4/2]

echo $[10%3]

方法三:使用let命令,let在运算时不显示结果,可以方便的对变量进行自增减

let a=a+10 相当于 a+=10,后面是主流写法

a=a-10 a-=10

a=a+1 a++

a=a-1 a--

a=a\*10 a\*=10

a=a/10 a/=10

a=a%10 a%=10

验证可以用echo $a

方法四: bc,可以进行小数运算,scale可以确定小数点后面位数

[root@desktop0 opt]# echo "1+1" | bc

2

[root@desktop0 opt]# echo "11.1+1" | bc

12.1

[root@desktop0 opt]# echo "10/3" | bc

3

[root@desktop0 opt]# echo "scale=3;10/3" | bc

3.333

[root@desktop0 opt]# echo "scale=3;10/3;1+1" | bc

3.333

2

day02

条件测试:

一.字符串匹配

语法格式

使用“test 表达式”或者[ 表达式 ]都可以，表达式两边至少要留一个空格。

1）== 比较两个字符串是否相同

[zengye@svr5 ~]$ [ $USER == "root" ]

[zengye@svr5 ~]$ echo $? //查看结果0为对，非0为错

2）!= 比较两个字符串是否不相同

[zengye@svr5 ~]$ [ $USER != "root" ]

3）一行执行多条命令的情况

# A || B //仅当A命令执行失败，才执行B命令 或

# A && B //仅当A命令执行成功，才执行B命令 且

# A ; B //执行A命令后执行B命令，两者没有逻辑关系

# A && B || C //A且B,或C

4) -z 检查变量的值是否未设置（空值）

! -z 与 - n 一样

[ -z $a ] //测试变量a是否为空,是空返回0

多个逻辑符号连用时:

|| 或,前后任务有一个成功就算成功

&& 且,前后任务都成功,才算成功

[ ! -z $a ] 或 [ -n $a ] //测试变量a,不是空返回0

[ ! -z $a ] && echo "a不是空的" || echo "a是空的"

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

二.比较数值大小

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

脚本:每两分钟检查登录用户数量,超过三人报警

[root@server0 opt]# vim ceshi.sh

#!/bin/bash

a=`who|wc -l`

[ $a -gt 3 ] && echo "已有$a人登录" | mail -s test root

[root@server0 opt]# crontab -e -u root

\*/2 \* \* \* \* /bin/bash /opt/ceshi.sh

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

3.识别文件/目录的状态

-r -w -x

-e存在 -f存在且是文件 -d存在且是目录

[ -r txt ] //测试当前用户对文件txt有没有读权限(对root无效)

[ -w txt ] //测试当前用户对文件txt有没有写权限(对root无效)

[ -x txt ] //测试当前用户对文件txt有没有执行权限

[ -e txt ] //txt是否存在,不在乎文件类型

[ -f txt ] //txt是否存在,必须是文件

[ -d txt ] //txt是否存在,必须是目录

4.多个条件/操作的逻辑组合

&& ||

== != -z !-z(-n) -eq -ne -gt -ge -lt -le -r -w -x -e -f -d

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

if判断

单分支

if 条件测试;then

执行指令

fi

双分支

if 条件测试;then

命令序列1

else

命令序列2

fi

多分支

if 条件测试;then

命令序列1

elif 条件测试;then

命令序列2

else

命令序列3

fi

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

for循环

for 变量名 in 参数1 参数2 参数3 .....

do

任务指令

done

1.有变量有关,而且关心循环次数

2.只关心循环次数

大括号造数

#!/bin/bash

for i in {0..8}

do

echo nb

done

seq造数工具:

#!/bin/bash

for i in `seq 8`

do

echo nb

done

seq造数工具:

#!/bin/bash

x=5

for i in `seq $x`

do

echo nb

done

批量创建用户:

#!/bin/bash

for i in `cat /opt/user`

do

useradd $i &> /dev/null

done

修改vim tab间隔几个字符

vim /etc/vimrc

第一行添加 set ts=4 //修改为间隔四个字符

编辑脚本实现一次性检测12台主机,显示结果及总数

#!/bin/bash

a=0

b=0

for i in {1..12}

do

ping -c 3 -i 0.2 -W 1 172.25.0.$i &> /dev/null

if [ $? -eq 0 ];then

echo "172.25.0.$i 通了"

let a++

else

echo "172.25.0.$i 不通"

let b++

fi

done

echo "$a台通了,$b 台没通"

while 循环

while 条件测试

do

任务序列

done

day02 2

bash 快捷键,tab,管道,历史记录,重定向

1,声明解释器 2注释脚本功能,重要变量的含义 3脚本代码

1,自定义变量

2,系统变量,包含环境变量 PWD HOSTNAME USER HOME UID PATH PS1 PS2 SHELL ,位置变量 $1 $2 $3.... ,预定义变量 $0 $\* $# $? $$

'' "" `` read -p stty -echo stty echo

export a=10 export -n unset

运算,1 expr + - \(转义)\* / %

2 echo $[1+1]

3 let a\*=10 a=a\*10

a++ a--

条件测试: 使用“test 表达式”或者[ 表达式 ]都可以

1,字符串匹配

[ a == a ] //判断a是否等于a

echo $? //结果为0,正确

[ a != a ] //判断a是否不等于a

echo $? //结果为非0,错误

[ $USER == root ] //判断当前用户是不是管理员

&& 且,前置命令成功后才执行后面命令

|| 或,前置命令失败后才执行后面命令

[ a == a ] && echo ok

[ a == b ] && echo ok

[ a == a ] || echo "前置任务失败了"

[ a == b ] || echo "前置任务失败了"

[ -z $a ] //测试变量a是否为空,是空返回0

[ ! -z $a ] 或 [ -n $a ] //测试变量a,不是空的时候返回0

多个逻辑符号连用时:

&& 且, 前后任务都成功,才算成功

|| 或 ,前后任务有一个成功,就算成功

[ -z $a ] && echo "a是空的" || echo "a不是空的"

[ ! -z $a ] && echo "a不是空的" || echo "a是空的"

2,比较整数值的大小

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

X=20 //定义一个测试变量

[ $X -eq 20 ] && echo "相等" || echo "不相等"

[ $X -eq 30 ] && echo "相等" || echo "不相等"

[ $X -ne 20 ] && echo "不等于" || echo "等于"

[ $X -ne 30 ] && echo "不等于" || echo "等于"

[ $X -gt 10 ] && echo "大于" || echo "否"

[ $X -gt 20 ] && echo "大于" || echo "否"

[ $X -gt 30 ] && echo "大于" || echo "否"

写一个脚本,实现:

每2分钟检测服务器登录用户的数量,如果超过3人,则发邮件给管理员报警

素材:

crontab

who | wc -l

[ -gt 3 ] && echo "报警" | mail -s test root

#!/bin/bash

n=`who | wc -l` //定义变量n,为目前登录账户的数量

[ $n -gt 3 ] && echo "报警" | mail -s test root //用变量n与3对比,如果大于3,则发邮件给管理员,邮件标题是test,内容是"报警"

n=`who | wc -l` 与 [ $n -gt 3 ] 也可以写成[ `who | wc -l` -gt 3 ]

可以提前准备邮件文本,使用输入重定向,邮件内容就是文本内容

mail -s test root < txt

脚本写完后 添加x权限

并在计划任务中添加具体执行时间与脚本位置

crontab -e

\*/2 \* \* \* \* /opt/test1.sh //每2分钟执行一次脚本

/var/spool/mail/root //管理员所有邮件所在的文件,测试前可以先删除此文件

3,识别文件/目录的状态

-r读 -w写 -x执行

-e存在 -f存在且是文件 -d存在且是目录

[ -r txt ] //测试当前用户对文件txt有没有读权限(对root无效)

[ -w txt ] //测试当前用户对文件txt有没有写权限(对root无效)

[ -x txt ] //测试当前用户对文件txt有没有执行权限

[ -e txt ] //txt是否存在,不在乎文件类型

[ -f txt ] //txt是否存在,还必须是文件

[ -d txt ] //txt是否存在,还必须是目录

4,多个条件/操作的逻辑组合

&& ||

== != -z ! -z (-n) -eq -ne -gt -ge -lt -le -r -w -x -e -f -d

if判断

单分支

if 条件测试;then

执行指令

fi

if 条件测试

then

执行指令

fi

双分支

if 条件测试;then

命令序列1

else

命令序列2

fi

ping

使用if双分支编写脚本,测试与其他主机的链接情况:

ping命令可以使用选项增加工作效率

-c 定义ping的次数 -i 定义ping的间隔时间,单位秒 -W 定义反馈结果的时间,单位秒

#!/bin/bash

ping -c 3 -i 0.2 -W 1 $1 &> /dev/null

if [ $? -eq 0 ];then

echo "通了"

else

echo "不通"

fi

#!/bin/bash

[ -z $1 ] && echo "请在脚本名后输入ip" && exit

ping -c 3 -i 0.2 -W 1 $1 &> /dev/null

if [ $? -eq 0 ];then

echo "通了"

else

echo "不通"

fi

#!/bin/bash

read -p "请输入ip地址" ip

ping -c 3 -i 0.2 -W 1 $ip &> /dev/null

if [ $? -eq 0 ];then

echo "通了"

else

echo "不通"

fi

多分支

if 条件测试;then

命令序列1

elif 条件测试;then

命令序列2

else

命令序列3

fi

使用RANDOM获取随机数字,编写猜大小脚本

#!/bin/bash

x=$[RANDOM%11] //使用随机数除11,取余数范围在0-10之间,为变量x

read -p "请输入一个数字(0-10)" n //用户提供一个数字,为变量n

if [ $n -eq $x ];then //利用if多分支将x与n进行对比,如果相等

echo "猜对啦" //反馈信息

elif [ $n -gt $x ];then //如果用户给的数字大于随机数

echo "猜大了" //反馈信息

else

echo "猜小了" //最后一种情况

fi

for循环

for 变量名 in 参数1 参数2 参数3......

do

任务指令

done

1,与变量有关,而且关心循环次数

2,只关心循环次数

#!/bin/bash

for i in {1..8}

do

echo nb

done

#!/bin/bash

for i in `seq 8`

do

echo nb

done

#!/bin/bash

x=5

for i in `seq $x`

do

echo nb

done

利用文本创建账户,账户数量由文本决定

#!/bin/bash

for i in `cat user` //查看文本user后,将内容依次交给i循环

do

useradd $i //第一次循环创建的是文本中的第一个用户,以此类推

done

vim /etc/vimrc

set ts=4 //ts是tabstop,vim编辑器中每次tab间隔4个空格,默认8个

编写脚本实现,一次检测12台主机172.25.0.1~172.25.0.12的网络状况,仅仅显示结果

#!/bin/bash

x=0

y=0

for i in {1..12}

do

ping -c 3 -i 0.2 -W 1 172.25.0.$i &> /dev/null

if [ $? -eq 0 ];then

echo "172.25.0.$i 通了"

let x++ //每通一台,x就+1

else

echo "172.25.0.$i 不通"

let y++ //每不通一台,y就+1

fi

done

echo "$x台通了,$y台不通"

while 条件测试

do

任务序列

done

使用while循环编写猜数脚本,猜对为止,并统计猜的次数

#!/bin/bash

x=$[RANDOM%101]

c=0

while :

do

read -p "请输入一个数字(0-100)" n

let c++

if [ $n -eq $x ];then

echo "猜对了,猜了$c次"

exit

elif [ $n -gt $x ];then

echo "猜大了"

else

echo "猜小了"

fi

done

day03

&& || if

case分支,简化版本if,功能不如if强大,代码比if简洁

case 变量 in

模式1)

命令序列1;;

模式2)

命令序列2;;

\*)

命令序列3

esac

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

方便使用nginx的脚本

#1/bin/bash

case $1 in

start)

/usr/local/nginx/sbin/nginx;; //开启服务

stop)

/usr/local/nginx/sbin/nginx -s stop;; //关闭服务

re)

/usr/local/nginx/sbin/nginx -s stop //先关闭

/usr/local/nginx/sbin/nginx;; //后开启,重启

status)

netstat -ntulp |grep -q nginx

[ $? -eq 0 ] && echo "服务已开启" || echo "服务未开启";; //查询是否开启

\*)

echo "请输入start|stop|re|status"

esac

break //终止循环,继续执行循环之后的任务

continue //终止本次循环,继续下一次循环

exit //直接退出脚本

day03 2

&& || if

case分支,简化版本ｉｆ，功能不如ｉｆ强大，代码比ｉｆ简洁

case 变量 in

模式1)

命令序列1;;

模式2)

命令序列2;;

模式3)

命令序列3;;

\*)

命令序列4

esac

httpd nginx

编写一键源代码编译安装nginx软件的脚本

源码安装Nginx需要提前安装依赖包软件gcc,openssl-devel,pcre-devel

#!/bin/bash

yum -y install gcc openssl-devel pcre-devel

tar -xf nginx-1.12.2.tar.gz

cd nginx-1.12.2

./configure

make

make install

运行脚本后执行:

/usr/local/nginx/sbin/nginx //开启nginx服务

systemctl stop firewalld //关闭防火墙后,用真机浏览器测试虚拟机的web服务是否正常 http://172.25.0.10

netstat -ntulp | grep nginx

-n以数字格式显示端口号

-t显示TCP连接的端口

-u显示UDP连接的端口

-l显示服务正在监听的端口信息，如httpd启动后，会一直监听80端口

-p显示监听端口的服务名称是什么（也就是程序名称）

编写方便使用nginx服务的工具脚本

#!/bin/bash

case $1 in

start)

netstat -ntulp | grep -q nginx

[ $? -eq 0 ] && echo "服务已经开启" && exit

/usr/local/nginx/sbin/nginx;; //开启服务

stop)

/usr/local/nginx/sbin/nginx -s stop;; //关闭服务

re)

/usr/local/nginx/sbin/nginx -s stop //先关闭再开启

/usr/local/nginx/sbin/nginx;; 可以实现重启效果

status)

netstat -ntulp | grep -q nginx //检查服务状态

[ $? -eq 0 ] && echo "服务开启" || echo "服务未开启";;

\*)

echo "start | stop | re | status"

esac

在Shell脚本中，将一些需重复使用的操作，定义为公共的语句块，即可称为函数。通过使用函数，可以使脚本代码更加简洁，增强易读性，提高Shell脚本的执行效率

格式1：

function 函数名 {

命令序列

.. ..

}

格式2：

函数名() {

命令序列

.. ..

}

#!/bin/bash

a(){ //定义函数,名为a

echo abc //函数中的指令,可以是多条

echo xyz

}

a //执行函数

a //再次执行函数,多次执行可以体现函数优势

#!/bin/bash

cecho(){

echo -e "\033[$1m$2\033[0m"

}

cecho 31 ABCDEFG

cecho 32 ABCDEFG

cecho 33 ABCDEFG

cecho 34 ABCDEFG

cecho 35 ABCDEFG

cecho 36 ABCDEFG

break //终止循环,继续执行循环之后的任务

continue //终止本次循环,继续下一次循环

exit //直接退出脚本

编写为用户提供的数字求和脚本,0是结束

#!/bin/bash

x=0

while :

do

read -p "请输入一个数字(0是结束)" n

[ $n -eq 0 ] && break

let x+=n

done

echo "总和是 $x"

编写脚本,找出1~20中谁是6的倍数

#!/bin/bash

for i in {1..20}

do

x=$[i%6] //依次用数字除6取余数

[ $x -ne 0 ] && continue //余数不等于0的话,结束本次循环

echo $i 进入下一次循环

done

字串处理

1,字串截取

${变量名:起始位置:长度}

a=abcdef

echo ${a:0:2} //截取变量a中的字串,从第一位开始截取2位

echo ${a::2} //0可以省略

echo ${a:3:1} //截取第4位字符

获取1位随机字符

n=$[RANDOM%62]

echo ${x:n:1}

如何获取8位?

编写获取8位随机字符的脚本

#!/bin/bash

x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR

STUVWXYZ0123456789

for i in {1..8}

do

n=$[RANDOM%62] //求出随机数,范围是0~61之间

txt=${x:n:1} //截取出对应的字符

pass=$pass$txt //累加放在变量pass里面

done

echo $pass //循环结束后,得出结果

2,字串替换

${变量名/old/new}

a=111

echo ${a/1/a} //将1个1替换为a

echo ${a//1/a} //将所有1都替换为a

echo ${a/11/aa} //将11替换为aa

echo ${a//1/} //替换为空,等于删除

3,字串删除

掐头

${变量名#被删除内容} //从左往右删除

a=`head -1 /etc/passwd` //准备素材

echo ${a#root} //从左往右删除到root

echo ${a#x} //从左往右删除到x,无效,因为第一个字符不是x

echo ${a#root:x} //手工输入需要删除的内容

echo ${a#\*x} //使用通配符删除到x以及x左边所有内容,效果同上

echo ${a##\*root} //从左往右删除删除到最后一个root,以及root左边所有内容

echo ${a#\*:root} //从左往右删除到第一个:root,以及左边所有

去尾

echo ${变量名称%被删除的内容}

echo ${a%bash} //去尾,删除到第一个bash

echo ${a%root\*} //删除到第一个root,以及此root右边所有

echo ${a%%root\*} //删除到最后一个root,以及此root右边所有

echo ${a%%:\*} //删除到最后一个: ,以及右边所有

利用字串删除功能编写脚本,用来实现批量修改扩展名

将所有.txt 文件 修改为 .doc

#!/bin/bash

for i in `ls \*.txt` //找到所有.txt文件 依次交给for循环

do

n=${i%.\*} //把找到的每个文件去尾,得到无扩展名的文件名

mv $i $n.doc //修改文件,将名称与.doc组合

done

改良

#!/bin/bash

for i in `ls \*.$1`

do

n=${i%.\*}

mv $i $n.$2

done

定义变量初值(备用值)

aa= //aa等于空

echo ${aa:-123} //定义初值,如果aa是空,则a的内容就是123

非空的话,则显示具体内容

aa=10 //aa非空

echo ${aa:-123} //不会显示123

编写创建用户并配置密码的脚本,如果用户没有输入密码直接回车,则统一使用123,如果用户输入了密码则不使用123

#!/bin/bash

read -p "请输入用户名" n

useradd $n

read -p "请输入密码" p

echo ${p:-123} | passwd --stdin $n

休息

day04

case 变量 in

模式1)

命令序列1;;

模式2)

命令序列2;;

模式3)

命令序列3;;

\*)

命令序列4

esac

netstat -ntulp

函数名称(){

}

break 终止循环,继续执行循环之后的任务

continue 结束本次循环,继续下一次循环

exit 直接退出脚本

字串处理:

${变量名:} 截取

${变量名/} 替换

${变量名#} 掐头

${变量名%} 去尾

${变量名:-} 定义初值

正则表达式,用来查找过滤文档的工具符号

grep "root" user //在user文档中查找有单词root的行

grep "^root" user //在user文档中查找以单词root开头的行

grep "bash$" user //在user文档中查找以单词bash结尾的行

grep "^$" user //查找空行

grep "^ " user //查找以空格开头的行

grep "[rot]" user //查找字母r或者o或者t

grep "roo[tn]" user //查找root或者roon

grep "[^rot]" user //查找字母r或者o或者t之外的内容

grep "[a-z]" user //查找小写字母

grep "[A-Z]" user //查找大写字母

grep "[a-Z]" user //查找所有字母

grep "[0-9]" user //查找数字

grep "r..t" user //查找以r开头t结尾中间有2个任意字符的单词

grep "ro\*" user //查找以r开头后面有或没有字母o的单词,o可以出现任意次

grep "o\{1,2\}" user //查找o出现1次到2次的行

grep "o\{2,3\}" user //查找o出现2次到3次的行

grep "o\{2\}" user //查找o出现2次的行

grep "o\{1,\} " user //查找o出现1次以及1次以上

grep "\(:0\)\{2\}" user //查找:0(数字零)连续出现2次的行

扩展正则

egrep "o+" user //查找o出现1次以及1次以上

egrep "o?" user //查找o出现0次或1次

egrep "o{1,2}" user //查找o出现1次到2次的行

egrep "o{2,3}" user //查找o出现2次到3次的行

egrep "o{2}" user //查找o出现2次的行

egrep "o{1,}" user //查找o出现1次以及1次以上

egrep "(:0){2}" user //查找:0(数字零)连续出现2次的行

egrep "bash|nologin" user //查找bash或者nologin

egrep "\bthe\b" a //查找单词the,前后是空格或者特殊符号

vim 交互式文本编辑工具

sed 流式编辑器,能对文档执行增删改查等操作,逐行操作,非交互式

sed使用方式:

前置命令 | sed 选项 (定址符)指令

sed 选项 (定址符)指令 文档

选项

-n 屏蔽默认输出

-i 写入到文件

-r 使用扩展正则

指令

p 输出文档内容

d 删除

s 替换

sed -n 'p' user //使用sed输出user文档中所有行

sed -n '1p' user //使用sed输出user文档中第1行

sed -n '1,3p' user //使用sed输出user文档中第1行到第3行

sed -n '1,+3p' user //输出user文档中第1行以及后面的3行

sed -n '1p;4p' user //输出第1行和第4行

sed -n '1~2p' user //输出奇数行

sed -n '2~2p' user //输出偶数行

使用正则做定址符

sed -n '/root/p' /etc/passwd //指令前面添加2个斜杠,中间使用正则,查找有root单词的行

sed -n '/bash$/p' /etc/passwd

sed -rn '/bash|nologin/p' user //竖线"|" 是扩展正则,要加-r选项

sed -n '=' user //显示行号

sed -n '$=' user //显示最后一行的行号

sed 'd' user //使用sed删除user文档中所有行

sed '1d' user //使用sed删除user文档中第1行

sed '1,3d' user //使用sed删除user文档中第1行到第3行

sed '1,+3d' user //使用sed删除user文档中第1行以及后面3行

替换:

s/旧内容/新内容/ 或者 s$旧内容$新内容$ 等符号

sed 's/2017/AAAA/' test.txt //替换所有行的第1个2017为AAAA

sed 's/2017/AAAA/g' test.txt //替换所有行的所有2017为AAAA

sed 's/2017/AAAA/2' test.txt //替换所有行的第2个2017为AAAA

sed '1s/2017/AAAA/' test.txt //替换第1行的第1个2017为AAAA

sed '2s/2017/AAAA/2' test.txt //替换第2行的第2个2017为AAAA

sed '3s/2017/AAAA/3;3s/2017/AAAA/2;' test.txt //替换第3行的第3个, 然后在替换第3行的第2个

sed '3s/2017/AAAA/2;3s/2017/AAAA/2;' test.txt //效果同上

如何将 /bin/bash 替换为 /sbin/sh

sed 's#/bin/bash#/sbin/sh#' user

\(\)

()

保留=复制 \1 \2 \3

sed 's/.//2 ; s/.$//' nssw.txt //利用替换功能,删除文档中每行的第2个字符和最后一个字符

sed -r 's/^(.)(.\*)(.)$/\3\2\1/' nssw.txt //将每行的首个字符和最后一个字符互换,首先利用^(.)复制行首字符,定义为\1 , 然后复制行尾字符 (.)$ ,定义为\3 ,中间的字符无论多少都使用(.\*)涵盖,定义为\2, 然后根据需求粘贴.

sed -r 's/^(.)(.)(.)(.)$/\4\3\2\1/' aa //aa文档中有abcd,将abcd 换成 dcba

sed 's/[0-9]//g' nssw.txt //删除所有数字

sed -r 's/([A-Z])/[\1]/g' nssw.txt //将所有大写字母添加中括号

编写一键部署ftp服务的脚本,要求ftp具备匿名上传功能

vim /etc/vsftpd/vsftpd.conf

#anon\_upload\_enable=YES

#anon\_mkdir\_write\_enable=YES

sed 's/^#anon\_/anon\_/'

#!/bin/bash

yum -y install vsftpd &> /dev/null

sed -i 's/^#anon\_/anon\_/' /etc/vsftpd/vsftpd.conf

systemctl restart vsftpd

systemctl enable vsftpd

systemctl stop firewalld

setenforce 0

chmod o+w /var/ftp/pub

虚拟机执行脚本后,使用真机访问虚拟机的ftp,可以在pub目录中成功上传文件即可,关闭防火墙为了测试此脚本,生产环境中不可关闭

day05

前置指令 | sed 选项 (定址符)指令

sed 选项 (定址符)指令 文件

-n -r -i

/ /p 2d 3s

a行后追加 i行前插入 c替换整行

sed '1a AAAA' user //在第1行的后面追加AAAA

sed '/^bin/a AAAA' user //在以bin开头的行的后面追加AAAA

sed '1i AAAA' user //在第1行的前面追加AAAA

sed '/^bin/i AAAA' user //在以bin开头的行的前面追加AAAA

sed '1c AAAA' user //把第1行替换成AAAA

sed '/^bin/c AAAA' user //把bin开头的行替换成AAAA

sed '$a 1.1.1.1 www.baidu.com' /etc/hosts //在hosts文件的最后一行后面追加ip地址与域名

sed 修改,查找 文件

awk 查找输出文件的内容

awk使用方式:

1, 前置命令 | awk 选项 条件{指令}

2, awk 选项 条件{指令} 文件

指令 : print $1 $2 $3

选项 : -F 分隔符

awk '{print $1}' nssa //在nssa文件中查找所有行的第1列并输出

awk '{print}' nssa //输出文档所有内容

awk -F: '{print $6}' /etc/passwd //以冒号做分隔符,输出第6列,相当于显示所有用户的家目录

提取网卡流量:

ifconfig eth0 | sed -rn '/RX p|TX p/p' | awk -F[\(\)] '{print $2}' //首先使用前置命令查看eth0网卡所有信息, 然后使用sed过滤显示有RX p(接收的流量)和TX p(发送的流量)的行, 最后使用awk 定义小括号为分隔符 ,并输出第2列

ifconfig eth0 | awk -F[\(\)] '/RX p|TX p/{print $2}' //awk也支持正则, 效果同上

awk支持常量输出,放在双引号中间即可

df -h | awk '/vda1/{print "当前主机根分区剩余空间是"$4}' //配合常量输出

awk '/Failed/{print "登录本机失败的IP地址",$11}' /var/log/secure //在安全日志文件中利用awk筛选出访问本机失败的ip,可加逗号做空格

awk的内置变量:

$0显示所有 $1 $2 $3... NR显示行号 NF显示列号

awk处理的时机: 相当于逐行任务之外的任务

BEGIN{print } //在逐行任务之前添加任务,执行1次

{print} //逐行任务,执行n次

END{print } //在逐行任务之后添加任务,执行1次

awk 'BEGIN{print "早上好"}{print}END{print "再见"}' nssw

awk '{print}END{print NR}' nssw //显示所有内容,以及最后一行的行号

awk -F: 'BEGIN{print "User\tUid\tHome"}{print $1"\t"$3"\t"$6}END{print "总计" NR "行"}' user //使用awk处理时机,输出表头,内容,和结尾

awk 'BEGIN{x=0}{x++}END{print x}' user

awk处理条件:

1,使用正则

awk -F: '/root/{print}' /etc/passwd //输入有root的行

awk -F: '$1~/root/{print}' /etc/passwd //输入第一列包含root的行

awk -F: '$1!~/root/{print}' /etc/passwd //输入第一列不包含root的行

2,使用字符或数字做条件,可以配合符号 == != > < >= <=

awk -F: '$1=="root"{print}' user //找第1列是root的行

awk -F: 'NR==1{print}' user //找第一行

awk -F: '$3>=1000{print $1,$3}' /etc/passwd //找第3列大于等于1000的行,并输出第1列与第3列

awk -F: '$3<10{print $1,$3}' /etc/passwd //找第3列小于10的行,并输出第1列与第3列

awk -F: '$3==10{print $1,$3}' /etc/passwd //找第3列等于10的行,找不到则无任何显示

3,使用逻辑符号 && 且 || 或

awk -F: '$3>10 && $3<20{print}' /etc/passwd //找第3列大于10并且第3列小于20的行, {print}可以省略不写

awk -F: '$1=="root" ||$1=="bin"{print}' /etc/passwd //找第1列是root或是bin的行

awk -F: '$1=="rootabc" ||$1=="bin"{print}' /etc/passwd //找第1列是rootabc或是bin的行

awk -F: '$1=="root" && $1=="bin"{print}' /etc/passwd //找第1列既是root又是bin的行,逻辑错误,找不到

awk -F: '$3>100 && $3<10{print}' /etc/passwd //逻辑错误找不到

awk -F: '$3>10 || $3<1000print}' /etc/passwd //找所有行

4,运算

awk 'BEGIN{print 1+1}'

awk 'BEGIN{print 1.1+1.1}'

awk 'BEGIN{print 2\*2}'

awk 'BEGIN{print 2/2}'

awk 'BEGIN{print 2%2}'

awk 'BEGIN{x=2;y=5;print x+y}'

列出UID间于1~1000的用户详细信息

输出/etc/hosts映射文件内以127或者192开头的记录

列出100以内整数中7的倍数或是含7的数

seq 100 | awk '$1%7==0 || $1~/7/'

方法一:

#/bin/bash

A=$(awk -F: '/bash$/{print $1}' /etc/passwd)

for i in $A

do

grep $i /etc/shadow | awk -F: '{print $1,"-->",$2}'

done

方法二:

#!/bin/bash

u=`awk -F: '/bash$/{print $1}' /etc/passwd`

for i in $u

do

awk -F: -v a=$i '$1==a{print $1,"-->" ,$2}' /etc/shadow

done

day06

awk 选项 条件 指令

-F -v 正则,字符和数字,逻辑符号&& ||,运算 {print $1 $2}

变量只可以存储一个值,再次赋值时,上一个值会被覆盖

a=10

a=20

数组可以利用多个下标,仅仅使用一个数组名存储多个值

数组名[下标]=值

a[0]=10

a[1]=20

awk 'BEGIN{a[0]=10;a[1]=20;print a[1],a[0]}' //首先定义1个数组,名称a,再定义多个下标对应的值, 然后输出

awk 'BEGIN{a[0]="a";a[1]="b";print a[1],a[0]}' //除了数字,还可以使用字符,注意要使用双引号

awk 'BEGIN{a["abc"]=10;a["xyz"]=20;print a["abc"],a["xyz"]}'

awk 'BEGIN{a[0]++;a[1]++;a[1]++;print a[0],a[1]}'

awk 'BEGIN{a[0]=0;a[1]=11;a[2]=22; for(i in a){print i,a[i]}}' //使用awk数组+for循环显示数组的所有内容

首先准备素材:

abc

abc

xyz

qqq

xyz

abc

awk '{cba[$1]++}END{for(i in cba){print cba[i],i}}' abc

//使用数组+for循环, 输出该文档中每行的第1列是什么内容,出现了几次. 首先定义数组名为cba,下标是每行的第1列, 然后执行++运算,逐行执行,都执行完后得到结果:

cba[abc]=3

cba[xyz]=2

cba[qqq]=1

然后这个结果再交给for循环处理,循环显示的变量i是下标.

ip[192.168.0.2]=3

ip[192.168.0.1]=2

ip[172.25.0.1]=1

ip[8.8.8.8]=1

awk '{ip[$1]++}END{for(i in ip){print i,ip[i]}}' abc | sort -rk 2 //将最后处理的结果交给sort排序, -k是定义按照哪列排序, -r是降序排列

netstat -ntulp | grep :80

使用awk统计网站访问者的ip与点击量

1,安装httpd与测试工具

yum -y install httpd elinks

2,开启httpd服务

3,使用测试工具访问本机网站

elinks -dump 127.0.0.1

4,查看httpd访问量日志

vim /var/log/httpd/access\_log

再使用其他主机访问此http服务(注意关闭防火墙),再次查看日志发现增加了新记录

5,使用awk查看最终结果

awk '{ip[$1]++}END{for(i in ip){print ip[i],i}}' /var/log/httpd/access\_log | sort -r

6,可以配合脚本增加访问次数再次检测

#!/bin/bash

for i in {1..5000}

do

elinks -dump 172.25.0.10 &> /dev/null

done

编写监控脚本,可以方便查看主机的若干参数

#!/bin/bash

while :

do

uptime |awk '{print "cpu平均负载:"$10,$11,$12}'

ifconfig eth0 |awk -F[\(\)] '/RX p/{print "网卡eth0接收的数据量是:"$2}'

ifconfig eth0 |awk -F[\(\)] '/TX p/{print "网卡eth0发送的数据量是:"$2}'

free -m | awk '/^Mem:/{print "主机剩余内存容量是:"$4"m"}'

df -h | awk '/vda1/{print "磁盘剩余空间是:"$4}'

u=`cat /etc/passwd | wc -l`

echo "计算机账户数量是:$u"

user=`who | wc -l`

echo "当前登录的用户数量是:$user"

p=`ps -aux | wc -l`

echo "目前开启的进程数量是:$p"

r=`rpm -qa | wc -l`

echo "目前安装的软件包数量是:$r"

sleep 2

clear

done

使用awk找出安全日志中对本机访问失败的ip记录与次数统计

awk '/Failed/{ip[$11]++}END{for(i in ip){print ip[i],i}}' /var/log/secure

#!/bin/bash

abc=`awk '/Failed/{ip[$11]++}END{for(i in ip){print ip[i]","i}}' /var/log/secure`

for i in $abc

do

aa=${i%,\*}

bb=${i#\*,}

[ $aa -ge 10 ] && echo "$bb猜错10次以上密码了,赶紧收拾他" | mail -s test root

done

\w 匹配数字字母下划线 a-z A-Z 0-9

\W 取反

grep "\w" aaaa //在aaaa文件中找所有的数字字母下划线

\s 各种空白,如空格,tab制表符等

\S 取反