脚本格式:

1. 声明解释器类型

#!/bin/bash

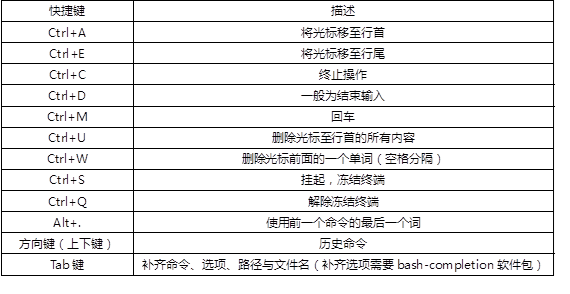
1. 注释,说明该脚本的作用,变量的含义
2. 编写代码

写完脚本后,赋予脚本执行权限shmod +x test1.sh

./test1.sh 执行脚本

bash test2.sh 调用另外一个bash执行脚本,开启子进程

source test2.sh 或者. test.sh 使用当前解释器执行脚本,不开启子进程



常量 固定不变

变量 变化的量,灵活多变,增加功能

1. 自定义变量
   1. 变量名称=变量的值 名称必须是字母数字下划线,不能以数字开头,不能使用特殊符号

a=10 定义变量a

echo $a 查看变量的值

echo ${a}RMB ${}运行大括号里面的变量, 输出变量+常量

unset a 取消变量

1. 环境变量,系统自带

$USER 显示当前用户

$UID 显示当前用户的UID

$HOME 显示当前用户的home目录

$SHELL 显示当前的编译器

$PWD 显示当前用户的当前所在目录

$HOSTNAME 显示当前用户的主机名

$PATH 显示当前环境变量的路径

]# echo $PS1 一级提示符

[\u@\h \W]\$

]# echo $PS2 二级提示符

>

which ls 查找ls命令的路径

1. 位置变量与预定义变量

$0 脚本的名称

$$ 进程号

$# 位置参数的个数

$\* 所有位置变量

$? 判断上一条命令的成功与失败,非0即失败

$1 $2 $3 ... 位置变量

env 参看所有系统变量

set 查看所有的变量

变量的扩展应用

“ ” 界定范围  
’ ’ 界定范围,屏蔽特殊符号

` `和$( ) 可以获取命令的执行结果

stty -echo 屏蔽回显

stty echo 恢复回显

a=100 局部变量 某个解释器中生效

export a 发布全局变量

export -n a 取消变量a的全局状态,恢复为局部变量

export b=100 发布全局变量b

export -n b 取消变量b的全局状态,恢复为局部变量

shell中运算

方法一:

expr 1 + 1 加

expr 1 - 1 减

expr 2 \\* 2 乘

expr 4 / 2 除

expr 4 % 2 求模(取余数)

\ 转意符号,可以屏蔽一个符号

方法二:

a=10

b=20

echo $[a+b]

echo $[a+10]

echo $[1+1]

echo $[1\*1]

echo $[1/1]

方法三: let 运算结果不显示,通常用于变量创建或变量自增减

a=0

let a++ 相当于a=a+1

let a-- 相当于a=a-1

let a\*=2 相当于a=a\*2

let a/=2 相当于a=a/2

方法四:bc 计算器,可以小数运算

echo "scale=3;10/3" | bc

read -p “XXX” n 输出想要的信息

1. 条件测试,可以让脚本更智能的工具

==等于(针对字符串) !=不等于(针对字符串)

]# [ a == b ]

]# echo $?

1

]# [ a == a ]

]# echo $?

0

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

[root@server0 ~]# echo $?

0

\*\*

[ -n “$a” ] 测试变量是否非空(不为空)

[ -z “$a” ] 测试变量是否为空

案例:

[root@server0 ~]# a=10

[root@server0 ~]# b=

[root@server0 ~]# [ -n "$a" ]

[root@server0 ~]# echo $?

0

[root@server0 ~]# [ -n "$b" ]

[root@server0 ~]# echo $?

1

[root@server0 ~]# [ -z "$a" ]

[root@server0 ~]# echo $?

1

[root@server0 ~]# [ -z "$b" ]

[root@server0 ~]# echo $?

0

1. 逻辑测试 (有判断需求的)

&&　　符号之前的任务成功后,才执行之后的任务

|| 符号之前的任务失败后,才执行之后的任务

案例(可以用到脚本中):

#!/bin/bash

[ -z $1 ] && exit 如果$1为空,就会执行exit退出脚本

yum -y install $1 &> /dev/null

systemctl restart $1

systemctl enable $1

案例:

[ root == $USER ] && echo "我是管理员" || echo "我不是管理员"

脚本案例:(两种不同的用法,可以实现相同的要求)

#!/bin/bash

read -p "请输入用户名:" n

[ -z “$n” ] && echo "给个名字啊" && exit

[ -n “$n” ] || echo "给个名字啊" && exit

useradd $n

1. 数字的测试

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

1. 文件的测试

-e 是否存在

-f 是否存在且为普通文件

-d 是否存在且为目录

-r -w -x 判断当前用户对某文件是否有相关权限

总结:

== != -z -n

-eq -ne -gt -ge -lt -le

-e -f -d -r -w -x

&& || ;

if判断

种类1

单分支

if 条件测试 ;then

执行指令

fi

双分支

if 条件测试 ;then

执行指令1

else

执行指令2

fi

多分支

if 条件测试1 ;then

执行指令1

elif 添加测试2 ;then

执行指令2

else

执行指令3

fi

案例:

#!/bin/bash

if [ ! -d /opt/xyz ];then

mkdir /opt/xyz

fi

判断xyz目录是否存在,没有就创建xyz目录

案例:

#!/bin/bash

if [ -f a ];then

rm -rf a

mkdir a

else

[ -d a ] || mkdir a

fi

判断a是否为文件,如果是文件的话就删除然后创建一个a的文件夹,判断a是否为文件夹,如果有就退出,如果没有就创建一个a文件夹

\*\*

ping -c 检测次数 -i 间隔时间(秒) -W 网络无法通讯时的反馈时间

ping -c 3 -i 0.2 -W 1 172.25.0.101

案例(把ping命令检测添加到脚本里):

#!/bin/bash

ping -c 3 -i 0.2 -W 1 172.25.0.10 &> /dev/null

if [ $? -eq 0 ];then

echo "通了!"

else

echo "没有通啊..."

fi

改良版:

#!/bin/bash

read -p "请输入ip地址:" n

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

if [ $? -eq 0 ];then

echo "通了!"

else

echo "没有通啊..."

fi

案例:

#!/bin/bash

read -p "请输入期末考试成绩:" x

if [ $x -ge 90 ];then

echo "优秀!"

elif [ $x -ge 70 ];then

echo "良好!"

elif [ $x -ge 60 ];then

echo "及格"

else

echo "回炉!"

fi

案例:

#!/bin/bash

read -p "请输入数字(1~9):" n

x=$[ RANDOM%10 ]

echo $x

if [ $x -eq $n ];then

echo "猜中了!"

elif [ $n -lt $x ];then

echo "小了老弟"

else

echo "大了.."

fi

循环

for循环 有次数限制 可以关注循环次数或者变量

for 变量名 in 值1值2值3...

do

执行指令

done

案例:

#!/bin/bash

x=0

y=0

for i in {1..15}

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++

else

echo "172.25.0.$i 不通."

let y++

fi

done

echo "$x台通了,$y台不通"

whlie循环,可以无限次循环执行任务

while 条件测试

do

执行指令

done

终极案例:

#!/bin/bash

x=$[RANDOM%101]

y=0

while :

do

read -p "请输入一个数字(1~100):" n

let y++

if [ $x -eq $n ];then

echo "猜对了"

echo "输入了$y 次成功了"

exit

elif [ $n -lt $x ];then

echo "猜小了"

else

echo "猜大了"

fi

done

case分支,功能类似if,不如if强大,代码比if精简

case 变量名称 in

模式1)

执行指令1;;

模式2)

执行指令2;;

\*)

执行指令n

esac

案例:

#!/bin/bash

case $1 in

t|T) 多模式可以用|分隔开

touch $2

ls;; 如果多执行指令的话;;放在模式里的最后一个执行指令上

m|M)

mkdir $2

ls ;;

r|R)

rm -rf $2

ls;;

\*)

echo "请输入t|m|r"

esac

nginx是个源码包,需要安装编译源代码

编写自动化安装nginx脚本

#!/bin/bash

yum -y install gcc openssl-devel pcre-devel &> /dev/null

tar -xf nginx-1.10.3.tar.gz

cd nginx-1.10.3

./configure

make

make install

启用nginx服务

/usr/local/nginx/sbin/nginx

关闭nginx服务

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

查看nginx软件信息

/usr/local/nginx/sbin/nginx -V

查看开启端口服务的黄金组合命令

netstat -ntulp

nginx开启/关闭/重启/状态服务的脚本

#!/bin/bash

case $1 in

kai)

netstat -nutlp | grep -q nginx

[ $? -eq 0 ] && echo "服务已经开启了,喵" && exit

/usr/local/nginx/sbin/nginx;;

guan)

netstat -nutlp | grep -q nginx

[ $? -ne 0 ] && echo "服务已经关闭了..." && exit

/usr/local/nginx/sbin/nginx -s stop;;

cq)

/usr/local/nginx/sbin/nginx -s stop &> /dev/null

/usr/local/nginx/sbin/nginx;;

zt)

netstat -ntulp | grep -q nginx

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

v)

/usr/local/nginx/sbin/nginx -V;;

\*)

echo "请输入:kai | guan | cq | zt | v"

esac

修改输出的字符串的颜色:

echo -e "\033[32mABCD\033[0m"

函数,能够将公共的语句块存储在一个变量中,达到精简脚本的目的

函数名称(){

指令

}

案例:

#!/bin/bash

cecho () {

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

}

cecho $1 $2

炸弹(无限死循环)

#!/bin/bash

.(){

.|.&

}

.

\*\*\*

中断和退出

exit 退出脚本

break 退出循环,继续执行循环之外的任务

continue 退出当前循环,继续下一次循环

## 字符串的处理

1.字符串的截取

${变量名:第几位:显示多少位} 字符串的起初位为0

#!/bin/bash

for i in {1..20}

do

x=$[i%6]

[ $x -eq 0 ] || continue

echo $[i\*i]

done

#!/bin/bash

x=0

while :

do

read -p "请输入一个数字:" n

[ -z $n ] && exit

[ $n -eq 0 ] && break

let x+=$n

done

echo "求和为:$x"

随机生成8位密码脚本:

#!/bin/bash

a=abcdefghijklmnopqrstuvwsyzABCDEFGHIJKLMNOPQRSTUVWHIZ0123456789

for i in {1..8}

do

x=$[RANDOM%62]

p=${a:x:1}

pa=$pa$p

done

echo $pa

2.字符串的替换

${变量名/old/new} 替换一个old为new

${变量名//old/new} 替换所有old为new

${变量名/old/} 删除一个old

${变量名//old/} 删除所有old

案例:

[root@server0 opt]# a=abcd

[root@server0 opt]# echo ${a/c/1} 替换一个c为1

ab1d

1. 字符串的删除

a)从左往右删除(掐头)

${变量名#被删除的内容}

${变量名##被删除的内容}

b)从右往左删除(去尾)

${变量名%被删的内容}

${变量名%%被删的内容}

编写脚本,可以批量修改扩展名的案例:

#!/bin/bash

for i in $(ls \*.$1)

do

x=${i%.\*}

mv $i $x.$2

done

1. 字符串初值处理

${变量名:-word} 或 ${变量名:=word}

总结:

字符串处理

${变量名:} 截取

${变量名/} 替换

${变量名#} 掐头

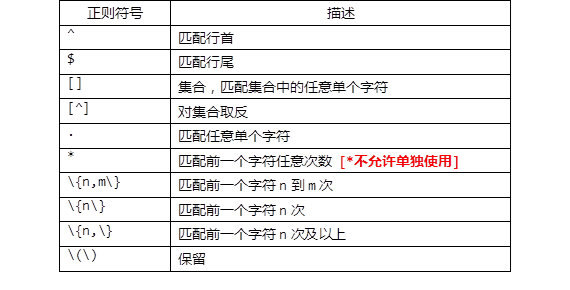
${变量名%} 去尾

${变量名:-} 定义初值

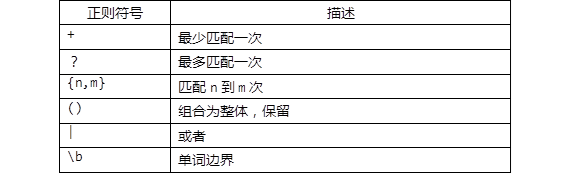
## \*\*正则表达式\*\*

用于搜索,过滤,筛选文本文档的工具,使用若干符号表示,需要配置某工具执行

基本正则



## 扩展正则



grep [参数]

-n 输出的同时打印行号

-v 选中不匹配的行

-i 忽略大小写

基本正则:grep案例

grep "^bin" test 搜索bin单词开头的行

grep "n$" test 搜索n字母为结尾的行

grep "[bin]" test 找字母b或i或n的行

grep "[^bin]" test 找除了字母b或i或n的行

grep "[a-z]" test 找所有小写字母a到z

grep "[A-Z]" test 找所有大写字母a到z

grep "[a-Z]" test 找所有的字母

grep "[^a-Z]" test 找除了字母

grep "[0-9]" test 找所有数字

grep "roo." test 找roo后面任意1个字符的行

grep "ro.." test 找roo后面任意2个字符的行

grep "r..." test 找roo后面任意3个字符的行

grep "." test 找所有字符,包括空格,不包括空行

grep "ro\*t" test 找rot,其中o可以出现任意次(包括0次)

grep "\*" test 不能单独使用(这是错误示范)

grep "ro\{2,10\}t" test o出现了2~10次

grep "ro\{2,\}t" tset o出现了2次以及2次以上

grep "ro\{,10\}t" test o出现了10次以及10次一下

grep "ro\{4\}t" tset o出现了4次

扩展正则:grep -e 或egrep案例

egrep "ro+t" test 查找rot,其中o可以出现1次或1次以上

egrep "ro?t" test 查找rot,其中o可以出现0次或1次

egrep "ro{2,10}t" test 查找rot,其中o可以出现在2~10次

egrep "ro{2,}t" test 查找rot,其中o出现2次或2次以上

egrep "ro{,10}t" test 查找rot,其中o出现10次或10次以下

egrep "ro{2}t" test 查找rot,其中o出现2次

\*

grep "\(0:\)\{2\}" test 找0:0:的行

egrep "(0:){2}" test 效果同上

egrep "root|bin" test 找root或bin的行

egrep "^root|^bin" test 找root开头或bin开头的行

egrep "^(root|bin)" test 效果同上

\b 在匹配字符时,屏蔽字母,数字,下划线

grep "the" test1 找有the的行

grep "the\b" test1 找有the且右边不能是字母,数字,下划线

grep "\bthe" test1 找有the且左边不能是字母,数字,下划线

grep "\bthe\b" test1 找有the且两边不能是字母,数字,下划线

grep 过滤文本

vim 对文本增删改查

sed 流式编辑器,可以非交互式的增删改查,逐行处理

格式:

1. 前置命令 | sed 选项 (定址符)指令
2. sed 选项 ‘(定址符)指令’ 被处理文档(相对路径/绝对路径)

选项:

-n 屏蔽默认输出

-r 支持扩展正则

-i 写入文件(修改加保存)

指令:

p 输出

d 删除

s 替换

a 在行的下面添加一行

i 在行的上面添加一行

c 替换整行

案例:

## 指令p练习:

sed 'p' test 输出所有的行,包括默认输出,每行2遍

sed -n 'p' test 没有默认输出,输出所有行,只有1遍

sed -n '1p' test 只输出第1行

sed -n '2p' test 只输出第2行

sed -n '3p' test 只输出第3行

sed -n '1,4p' test 输出1~4行

sed -n '2p;4p' test 只输出第2行和第4行

sed -n '2,+1p' test 输出第2行以及后面1行的内容

sed -n '1~2p' test 只看奇数行

sed -n '2~2p' test 只看偶数行

sed -n '/root/p' test 使用正则表达式找到有root的行

sed -n '/^root/p' test 使用正则表达式找到以root为开头的行

sed -n '/bin/p' test 使用正则表达式找到有bin的行

sed -n '/^bin/p' test 使用正则表达式以bin开头的行

sed -n '/\.$/p' a 使用正则表达式找到以.为结尾的行

sed -n '/^[kK]/p' data.txt 显示以K或k开头的行

sed -n '/834/!p' data.txt 显示不包 834 的行

sed -n '=' test 显示行号

sed -n '$=' test 显示最后一行的行号

## 指令d练习: 都是删除一整行

sed '1d' test 删除第1行

sed '3d' test 删除第3行

sed '3,5d' test 删除第3行到第5行的内容

sed '3,+2d' test 删除第3行以及后面2行的内容

sed '/root/d' test 删除所有包含root的行

sed '/root/!d' test 删除所有不包含root的行,!表示取反

sed '/^install/d' test 删除以install开头的行

sed '$d' test 删除文件最后一行

sed '/^$/d' test 删除空行

## 指令s练习:

sed 's/2017/xxxx/' 1 将所有行的第1个2017替换为xxxx

sed 's/2017/xxxx/2' 1 将所有行的第2个2017替换为xxxx

sed 's/2017/xxxx/3' 1 将所有行的第3个2017替换为xxxx

sed '2s/2017/xxxx/2' 1 将第2行的第2个2017替换为xxxx

sed 's/2017/xxxx/g' 1 将所有行的每个2017替换为xxxx

/bin/bash换成 /sbin/sh 更改转移符(数字键1~0上的符号都可以用)

sed 's/\/bin\/bash/\/sbin\/sh/' test 这是正确的替换

sed 's#/bin/bash#/sbin/sh#' test 更改替换符号

sed 's(/bin/bash(/sbin/sh(' test 更改替换符号

sed 's^/bin/bash^/sbin/sh^' test 更改替换符号

复制粘贴:

echo abc > abc

s

sed -r 's/(a)(b)(c)/123/' abc 将每行的第1个abc替换为123

123

sed -r 's/(a)(b)(c)/\1\2\3/' abc 将每行的第1个abc替换为abc

abc

sed -r 's/(a)(b)(c)/\3\2\1/' abc 将每行的第1个abc替换为cba

cba

## 指令a练习:

sed '1a 666' 1 在第一行的下面添加一行666

sed 'a 666' 1 在每一行的下面添加一行666

## 指令i练习:

sed '1i 666' 1 在第一行的上面添加一行666

sed 'i 666' 1 在每一行的上面添加一行666

## 指令c练习:

sed '1c 666' 1 在第一行的替换为666

sed 'c 666' 1 在每一行的替换为666

案例:

找出以bash解释器的用户,按照 用户名 --> 密码的方式显示出来

#!/bin/bash

n=$(sed -n '/bash$/s/:.\*//p' /etc/passwd)

for i in $n

do

p=$(grep $i: /etc/shadow) 将名字与shadow的对比

p1=${p#\*:} 找到对应的行,掐头

p2=${p1%%:\*} 去尾,得到纯净的密码

echo "$i --> $p2" 输出结果

done

编写脚本,实现vsftpd服务的部署,还可以匿名上传文件

#!/bin/bash

yum -y install vsftpd &> /dev/null

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

systemctl restart vsftpd

systemctl enbale vsftpd

chmod o+w /var/ftp/pub

影响上传因素:

1. ftp的配置文件
2. pub文件没有权限
3. selinux
4. firewall

awk 精确搜索文档,不能修改文档

格式:

awk 选项 (条件)指令 被处理的文档

前置指令 | awk 选项 (条件)指令

awk的内置变量 $1 $2 $3... $0 所有列 NR行号 NF列号

正则表达式 / / ~包含 !~不包含

字符串和数字: == != > >= < <=

逻辑符号: && ||

运算x++

选项:

-F 自定义分隔符(默认是空格)

指令:

print 显示文档内容

案例:

awk '{print}' 1 显示所有

awk '{print $2}' 1 显示所有行的第2列

awk '/hello/{print $3}' 1 找到有hello的行,输出他的第3列

案例:

awk -F: '{print $7}' user -F 定义分隔符,定义1个分隔符

awk -F[:/] '{print $5}' user 定义多个分隔符的效果,使用[]来分隔里面的内容

awk -F: '{print $1"的编译器是"$7}' user 定义分隔符是: 输出特定列,常量+变量

获取网卡接收流量:

ifconfig eth0 | awk '/RX p/{print "本机eth0接收流量是"$5"字节"}'

获取根分区剩余空间

df -h | awk '/\/$/{print "当前主机根分区剩余间 是"$4}'

获取日志文件信息,截取输入错误密码的ip源

awk '/Failed/{print "这是入侵者的ip是"$11}' /var/log/secure

## awk的工作时机

BEGIN任务 执行1次

逐行任务 执行N次

END任务 执行1次

格式: awk ‘BEGIN{执行命令}{执行命令}END{执行命令}’ 文件

awk -F: 'BEGIN{print "User\tUID\tHome"}{print $1"\t"$3"\t"$6}END{print "Total",NR,"lines"}' user

输出结果如下:

User UID Home

root 0 /root

bin 1 /bin

daemon 2 /sbin

adm 3 /var/adm

lp 4 /var/spool/lpd

Total 5 lines

## awk处理条件

1. 使用正则作为条件

案例:

awk -F: '$1~/root/{print $1}' user 输出第一列包含root的行

awk -F: '$1!~/root/{print $1}' user 输出不包含第一列包含root的行

awk -F: 'NR==1{print}' user 输出第一的行

awk 'NR>=2 && NR<=4{print}' user 输出2~4的行

awk 'NR<=1 || NR>4{print}' user 输出第1行,或大于4的行

awk -F: '$3>=1000 && $3<=1005' /etc/passwd 输出第三列为1000~1005的行,当任务是print时可以省略{print}

案例:

所有普通用户的家目录与解释器

例如:张三的家目录是XXX,解释器是XXX

awk -F: '$3>=1000{print $1"的家目录是"$6",解释器是"$7}' /etc/passwd

awk 'BEGIN{x++;print x}' 把x+1 然后输出x的值

awk 'BEGIN{x=2;print x+8}' 定义变量x为2,然后+8输出结果

awk 'BEGIN{x=2;y=3;print x+y}' 定义变量x为2,y为3,然后输出结果

sed -n '/bash$/s/:.\*//p' /etc/passwd 显示以bash结尾的行把第1个:后面的都去掉

awk -F: '/bash$/{print $1}' /etc/passwd 效果同上

## 单分支判断 案例:

awk '{if(/bash$/){x++}}END{print x}' user

如果找到以bash结尾的行,就把x+1,最后输出x的值

awk -F: '{if($3>=1000){x++}}END{print x}' /etc/passwd

如果找到第3列(UID)大于1000,则把x+1,最后输出x的值

## 多分支判断 案例:

awk -F: '{if($3>=1000){x++}else{y++}}END{print x,y}' /etc/passwd

如果第3列大于等于1000,则把x+1,否则把y+1,最后输出x和y的值

## awk数组

数组:能够保存多个值的特殊变量

格式: 数组名[下标]=元素值

awk中使用if判断 案例:

awk 'BEGIN{a[1]=100;a[2]=200;print a[1]}'

awk 'BEGIN{a[1]=100;a[2]=200;print a[1],a[2]}'

awk 'BEGIN{a["x"]=100;a["y"]=200;print a["x"],a["y"]}'

awk 'BEGIN{a["x"]="abc";a["y"]="def";print a["x"],a["y"]}'

awk中使用for循环实现遍历数组 案例:

awk 'BEGIN{a[1]=100;a[2]=200;a[3]=300;for(i in a){print i}}'

awk 'BEGIN{a[1]=100;a[2]=200;a[3]=300;for(i in a){print i,a[i]}}'

awk '{a[$1]++}END{for(i in a){print a[i],i}}' 2

使用awk统计网站访问量:

/var/log/httpd/access\_log 是httpd服务的访问日志

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

如下:

3 172.25.0.10 最终显示访问的次数与对应的ip地址

8 172.25.0.11

5 ::1

4 172.25.0.250

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

注释:

sort -n:排序 -r:倒叙 -nr:倒叙排序

ab -c 1 -n 10000 http://172.25.0.11/ 网站压力测试10000次

监控主机脚本:

#!/bin/bash

while :

do

n=$(uptime | sed -n 's/.\*://p')

echo "主机cpu平均负载时$n"

ifconfig eth0 | awk '/RX p/{print "主机的eth0的网卡接收流量是"$5"字节"}'

ifconfig eth0 | awk '/TX p/{print "主机的eth0的网卡发送流量是"$5"字节"}'

free -h | awk '/^Mem/{print "主机剩余内存是"$4}'

df -h | awk '/\/$/{print "根分区剩余容量是"$4}'

awk 'END{print "当前主机用户总量是"NR}' /etc/passwd

w=$(who | wc -l)

echo "当前登录的账号数量是$w 个"

p=$(ps aux | wc -l)

echo "当前主机开启的进程数是$p 个"

r=$(rpm -qa | wc -l)

echo "当前主机安装软件包数量是$r 个"

sleep 3

clear

done

主机安全脚本:

awk '/Failed/{ip[$11]++}END{for(i in ip){print ip[i],i}}' /var/log/secure | awk '$1>5'

发现主机登录次数大于5,就发邮件给root.