# Shell

## shell 翻译官

/bin/bash

cat /etc/shells

yum -y install ksh

bash优点

tab键, 历史记录,快捷键,别名,管道,重定向

bash-->sh-->kshawk

## 一个规范的脚本格式

1,声明解释器 , 作者信息

#!/bin/bash

#作者:

2,编写注释,解释脚本功能,步骤,变量含义等等...

#XXXX

3,编写代码

echo 123

#!/bin/bash

#这是一个测试脚本

echo "hello world"

chmod u+x test1.sh

## 执行脚本的方式

1,添加x权限

2,调用新解释器执行脚本

bash test1.sh //开启子进程

3,使用当前解释器

source test1.sh //不开启子进程,可以简写为 .

. test1.sh //效果同上

编写脚本,在/opt中创建目录abc,然后进入目录abc

#!/bin/bash

mkdir /opt/abc

cd /opt/abc

#!/bin/bash

echo 123

exit

/etc/yum.repos.d/XXX.repo

编写一键部署yum仓库的脚本:

#!/bin/bash

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

echo "[abc]

name=test

baseurl=http://172.25.254.254/content/rhel7.0/x86\_64/dvd/

enabled=1

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

脚本编写完后,可以用bash test.sh执行

然后使用yum repolist检查结果

编写,一键部署ftp服务的脚本:

#!/bin/bash

yum -y install vsftpd &> /dev/null

systemctl restart vsftpd

systemctl enable vsftpd

systemctl stop firewalld //为了测试脚本,可以临时关防火墙

## 变量

常量 不会变化 脚本功能单一

变量 灵活多变 增加脚本功能, 增加灵活,多用变量可使脚本更强大!

### 变量的种类:

#### 1, 自定义变量

变量名称=变量的值,变量名可以使用大小写字母,数字,下划线,禁止使用特殊符号,不能以数字开头.

a=10 //定义变量

echo $a //查看变量

unset a //取消变量

123a=10 //错误命名

echo ${a}RMB

#### 2, 环境变量

UID USER HOME HOSTNAME SHELL PWD PS1 一级提示符 PS2

PATH 存放命令的路径

#### 3, 位置变量

$1 $2 $3 ....

#### 4,预定义变量

$$ $\* $# $? $0

#!/bin/bash

echo $1 执行脚本后的第1个参数

echo $2 执行脚本后的第2个参数

echo $3 执行脚本后的第3个参数

echo $0 执行的脚本名

echo $$ 执行脚本时的进程号

echo $\* 所有位置变量

echo $# 所有位置变量的个数

echo $? 判断上一条指令是否成功,0是成功,非0是失败

编写脚本,创建用户tom,配置密码789

#!/bin/bash

useradd tom

echo 789 | passwd --stdin tom

升级版本

#!/bin/bash

useradd $1

echo $2 | passwd --stdin $1

bash test.sh abcd 1234 //使用脚本创建用户abcd并配置密码1234

env 查看所有环境变量

set 查看所有变量

"" 双引号 界定范围

touch "a b" //创建1个文件,不加引号则创建2个

a=10

echo "$a" //效果不变

'' 单引号 界定范围 ,且具有屏蔽特殊符号的作用

touch 'x y'

echo '$a' //$符号调用变量的值失效

`` 反撇号 或 $( ) 获取命令输出的结果

a=ls

echo $a

a=`ls` 或 a=$(ls)

echo $a

#!/bin/bash

read -p "请输入用户名" u

useradd $u

stty -echo

read -p "请输入密码" p

stty echo

echo $p | passwd --stdin $u

stty -echo 屏蔽回显

stty echo 恢复回显

## export 发布全局变量,可以使子进程使用父进程定义的变量

export a=10 //定义+发布全局变量

export a //如果变量存在,则直接发布全局变量

bash //进入子进程

echo $a //可以使用父进程定义的变量,以为已经发布为全局变量

vim /etc/profile

unset 变量名 //取消变量

export -n 变量名 // 取消全局效果,恢复局部

\ 转义符号

求模 取余数

## shell中的运算

### 方法一:

expr 1 + 1 加

expr 2 - 1 减

expr 2 \\* 2 乘

expr 4 / 2 除

expr 10 % 3 取余

a=10

expr $a + $a //也支持变量

### 方法二:

echo $[1+1]

echo $[1-1]

echo $[a+a] //调用变量不用多次添加$符号

echo $[1\*1] //乘法无需转义

### 方法三:

let 不输出结果,可以方便的修改变量的值

创建新变量 :

let c=1+1 //通过运算新创建变量c

let c=a+1 //常量变量均可使用

修改现有变量 :

let a=a+1 //常规思路,将a本身加1

let a++ //主流写法,将a本身加1

let a=a-1 //常规思路,将a本身减1

let a-- //主流写法,将a本身减1

let a=a+2 //常规思路,将a本身加2

let a+=2 //主流写法

let a\*=100 //修改变量本身,将a乘以100

回顾：

规范的脚本格式

1,声明解释器

2,注释

3,代码

变量的种类

1,自定义

2,环境 USER UID HOME HOSTNAME SHELL PWD PATH PS1 PS2

3,位置 $1 $2 $3....

4,预定义 $$ $\* $# $? $0

"" '' `` $( )

read -p "XXXX" 变量名

stty -echo

stty echo

局部 --> 全局 export

expr 1 + 1

expr 1 \\* 1

echo $[1+1]

echo $[1\*1]

echo $((1+1))

let c=1+1

let c=x+y

a=10

let a++

let a--

let a+=2

## bc //计算器,可以计算小数

1.1+1

10/3

scale=3 //定义小数点后位数

10/3

quit //退出

非交互式计算

echo "1.1+1" | bc

echo "10/3" | bc

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

useradd

## 条件测试

条件测试 [ ] test //能够使脚本更智能的工具

### 1,字符串

== 判断是否相等 != 判断是否不相等

-z 判断是否为空 ! -z

[ a == b ] 或者 test a == b //判断a是否等于b

echo $? //0是判断成功,非0是失败

[ a == a ]

echo $?

[ a != a ] //判断a是否不等于a

echo $?

[ $USER == root ] //判断当前用户是否为root

echo $?

a=10

[ -z $a ] //判断变量a是否为空

echo $? //如果不是空,返回值是非0

a= //a等于空

[ -z $a ]

echo $? //判断结果为0

### 2,数字

-eq 等于

-ne 不等于

-gt 大于

-ge 大于等于

-lt 小于

-le 小于等于

X=20 //定义一个测试变量

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

相等

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

不等于

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

否

[ $X -ge 10 ] && echo "大于或等于" || echo "否"

大于或等于

[ $X -lt 30 ] && echo "小于" || echo "否"

小于

[ $X -le 20 ] && echo "小于或等于" || echo "否"

小于或等于

### 3,文件 ,

-e 判断文件是否存在,不关心文件类型

-f 判断文件是否存在,且类型必须是普通文件

-d 判断文件是否存在,且类型必须是目录

### 判断当前用户是否拥有相关权限:

-r 是否有读 对管理员判断无效

-w 是否有写 对管理员判断无效

-x 是否能执行

### 4,逻辑判断

&& 并且 之前任务成功,才执行之后任务

|| 或者 之前任务失败,才执行之后任务

; 前后无逻辑关系, 执行完前面任务,继续执行后面的

[ a == a ] && echo ok || echo no

[ a == b ] && echo ok || echo no

[ a == b ] || echo ok && echo no

[ a == b ] || echo ok || echo no

[ a == b ] && echo ok && echo no

[ -z $1 ] && echo no && exit

编写脚本实现以下需求:

每隔2分钟检查登录服务器的账户,如果超过3人,则发邮件给管理员报警

## 非交互式发邮件的两种方法

echo 123 |mail -s test root

mail -s test root < a.txt //输入重定向,需要先准备好a.txt

#!/bin/bash

n=`who |wc -l`

[ $n -gt 3 ] && echo "有$n个人入侵服务器啦!隔壁老王来啦" | mail -s test root

chmod u+x test1.sh

crontab -e

\*/2 \* \* \* \* /opt/test1.sh

rm -rf /var/spool/mail/root //测试前可以先删除所有邮件

== != -z ! -z

&& ||

挂载本地yum源文件 && 编写yum仓库文件 && 安装ftp服务 && 开启ftp服务 &&

## if 单分支

if 条件测试 ;then

命令序列

fi

## if 双分支

if 条件测试 ;then

命令序列1

else

命令序列2

fi

## if 多分支

#!/bin/bash

x=$[RANDOM%10]

read -p "请输入一个数字(0-9)" n

if [ $x -eq $n ] ;then

echo "恭喜猜对了"

elif [ $n -gt $x ] ;then

echo "猜大了"

else

echo "猜小了"

fi

ping -c 定义ping次数 -i 定义ping间隔时间,单位秒 -W 1 加快反馈时间,单位秒

$?

测试一个ip地址,如果成功则返回"ok",如果失败返回"no"

## for 循环 ,关心2个点

1, 循环次数

2, 循环过程中调用的变量

for 变量名 in 值1 值2 值3......

do

循环任务

done

for i in a b c

do

echo $i

done

#!/bin/bash

for i in {1..5} //花括号里不能放变量再交给for循环

do

echo $i

done

#!/bin/bash

a=5

for i in `seq $a` //seq可以使用变量

do

echo $i

done

编写脚本,测试172.25.0.1~172.25.0.15是否能ping通

#!/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"

回顾：

条件测试

test 测试内容 [ 测试内容 ]

1,字符串

== != -z ! -z

2,数字

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

3,文件

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

4,逻辑

&& || ;

单分支

if 条件测试;then

命令序列

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

循环

for 变量名 in 值1 值2 值3...

do

执行指令

done

while 条件测试

do

执行指令

done

## case分支,功能类似if,代码比if要精简,但功能没有if强大,是简化版本的if

case 变量 in

模式1)

命令序列1 ;;

模式2)

命令序列2 ;;

.. ..

\*)

默认命令序列

esac

在真机操作

cd /linux-soft/02

scp lnmp\_soft.tar.gz root@172.25.0.10:/

回到虚拟机

cd /

tar xf /lnmp\_soft.tar.gz

cd /lnmp\_soft

scp nginx-1.10.3.tar.gz /opt

编写脚本,实现一键源代码编译安装nginx服务

#!/bin/bash

tar xf nginx-1.10.3.tar.gz

cd nginx-1.10.3

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

./configure

make

make install

systemctl stop firewalld //关闭防火墙

cd /usr/local/nginx //最后进到这个目录测试软件是否已经安装

/usr/local/nginx/sbin/nginx //开启服务

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

netstat -ntulp | grep nginx //查询nginx服务状态

netstat -ntulp | grep :80 //查询80端口被哪个服务占用

## shell函数,相当于增强版别名,可以利用一个函数名称,存储多个命令

### 方法一

abc(){

> echo 123

> ls

> }

调用函数,直接输abc即可

### 方法二

function xyz {

> echo xyz

> ls

> }

#!/bin/bash

cecho(){

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

}

cecho 31 ABCDEF

cecho 32 ABCDEF

cecho 33 ABCDEF

cecho 34 ABCDEF

cecho 35 ABCDEF

cecho 36 ABCDEF

while :

do

exit

done

echo XXXXX

## 对循环的控制

1, exit 直接退出脚本

2, break 终止循环,继续循环之后的任务

3, continue 终止当前循环,继续下一次循环

编写脚本,使用户输入的数字求和,用户输入0时,结束计算并输出之前所有数字之和

#!/bin/bash

x=0

while :

do

read -p "请输入一个数字" n

[ $n -eq 0 ] && break

let x+=n

done

echo $x

从数字1~20中查找6的倍数,找到之后输出到屏幕

#!/bin/bash

for i in {1..20}

do

x=$[i%6]

[ $x -ne 0 ] && continue

echo $i

done

## 字符串

### 1,字串截取

a=abcdef //定义变量

echo ${a:1:2} //从第2位开始截取2位

echo ${a:4:2} //从第5位开始截取2位

echo ${a:2:4}

echo ${a:0:3} //从第1位开始截取3位

echo ${a::3} //效果同上

编写脚本,从所有的字母大写,小写,数字中找一个随机字符显示在屏幕

编写脚本, 获取8位随机字符的密码

#!/bin/bash

x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

pass=

for i in {1..8}

do

n=$[RANDOM%62]

pass1=${x:n:1}

pass=$pass$pass1

done

echo $pass

### 2,字串替换

a=11223344

echo ${a/1/X} //将第一个1替换成X

echo ${a//1/X} //将所有1替换成X

echo ${a/2/} //将第一个2替换成空,等于删除第一个2

echo ${a//2/} //将所有2替换成空,删除所有2

a=222333

如何得到替换结果 XX2333

### 3,字串删除

# 从左至右删除 掐头

a=`head -1 /etc/passwd`

echo ${a#root} //从左至右删除第一个root

echo ${a#root:x:} //从左至右删除到root:x:

echo ${a#root:x:0:0:root:/root} //从左至右删除到root:x:0:0:root:/root

echo ${a##\*root} //上述目的可以简写, 删除到最后一个root,以及root左边所有内容

从右往左删除 去尾

echo ${a%/bin/bash} //从右往左删除到/bin/bash

echo ${a%root\*} //从右往左删除到第一个root,以及root右边所有内容

echo ${a%%root\*} //从右往左删除到最后一个root,以及root右边所有内容,相当于删除整行

通过字串删除功能,编写脚本,实现批量修改文件扩展名

touch abc{1..8}.txt //先制作素材

#!/bin/bash

for i in `ls \*.txt`

do

x=${i%.\*} //通过去尾功能,获取没有扩展名的文件名

mv $i $x.doc //将原来的文件,修改为文件名.doc

done

升级版

#!/bin/bash

for i in `ls \*.$1` //使用位置变量

do

x=${i%.\*}

mv $i $x.$2

done

## 设置变量初值(备用值)

${a:-XXX} //当变量a有值时,使用自身的值,是空时使用XXX

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

函数

函数名(){

命令序列

}

function 函数名 {

命令序列

}

echo -e "\033[31mABCD\033[0m"

循环的中断

1,exit

2,break

3,continue

字串处理

1,字串截取

${a:}

2,字串替换

${a/}

3,字串删除

掐头 ${a#}

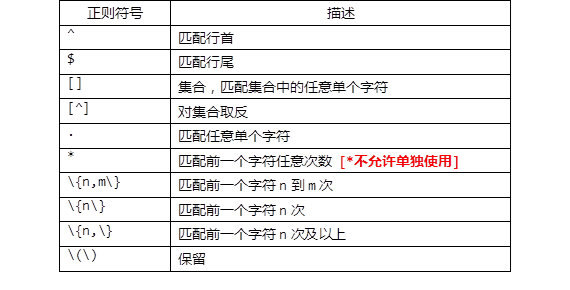
去尾 ${a%}

设置初值

${a:-}

## 正则表达式,可以对文本匹配查找,配合很多工具实现丰富的功能

基本正则:



grep ^$ user //搜索空行

grep "^ " user //搜以空格开头的行

grep " $" user //搜以空格结尾的行

grep "[bin]" user //搜索字母b或者i或者n

grep "roo[tdg]" user //搜索root或者rood或者roog

grep "[a-z]" user //找所有小写字母

grep "[0-9]" user //找所有数字

grep "[A-Z]" user //找所有大写字母

grep "[rot]" user //找r或o或t

grep "[^rot]" user //找除了r或o或t之外的字符

grep "[^0-9]" user //不找数字

grep "[^a-z]" user //不找小写字母

grep "[^a-zA-Z]" user //不找字母

grep "." user //找任意字符

grep "roo." user //找roo后面追加1个任意字符

grep "ro.." user //找ro后面追加2个任意字符

grep "^." user //找以任意字符开头的行

grep ".$" user //找以任意字符结尾的行

grep ".\*" user //找任意

grep "\*" user //不能单独使用

grep "a\*" user //找有a的行,a可以出现任意次,包括0次

grep "o\{1,2\}" user //搜索o出现了1到2次

grep "o\{2\}" user //搜索o出现了2次

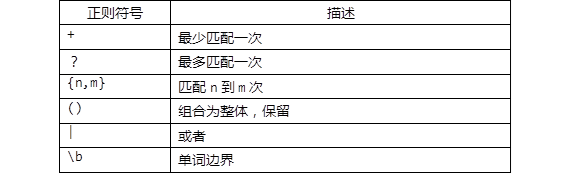
grep "o\{1,\}" user //搜索o出现了1次以及1次以上

grep "o\{2,\}" user //搜索o出现了2次以及2次以上

grep "o\{3,\}" user //搜索o出现了3次以及3次以上

grep "\(0:\)\{2\}" user

扩展正则



\{\} 可以 省略为 { }

\(\) 可以 省略为 ()

egrep "(0:){2}" user //搜索2个0:连在一起的行

grep -E "(0:){2}" user //效果同上

+ 相当于 \{1,\} //匹配前一个字符1次以及1次以上

? 相当于 \{0,1\} //匹配前一个字符0次或1次

172.40.50.115

/var/ftp/pub

vim 交互式

## sed 流式编辑器, 非交互式, 增删改查, 逐行处理

### 使用方式

1, 前置命令 | sed 选项 定址符 指令

2, sed 选项 定址符 指令 操作的文件

### 选项:

1, -n 屏蔽默认输出

2, -r 支持扩展正则表达式

3, -i 写入文件

### 指令

#### p 输出文档内容

sed -n '1p' user //输出第1行

sed -n '2,4p' user //输出2~4行

sed -n '3p;5p' user //输出3行和5行

sed -n '2,+2p' user //输出第2行,以及后面2行

sed -n '1~2p' user //查看奇数行

sed -n '2~2p' user //查看偶数行

sed的定址符可以使用正则表达式, 在 / / 中间填写

sed -n '/root/p' user //查看有root的行

sed -n '/^root/p' user //查看以root开头的行

sed -n '=' user //看所有行的行号

sed -n '$=' user //看最后一行的行号

#### d 删除行,使用方式和指令p基本一致,注意无需加 -n选项

sed '1d' user //删除第1行

#### s 替换

s/old/new/

s///

sed 's/2017/XXXX/' test //替换文档中所有行的第1个2017

sed '2s/2017/XXXX/' test //替换第2行的第1个2017

sed '2s/2017/XXXX/2' test //替换第2行的第2个2017

sed '3s/2017/XXXX/3;3s/2017/XXXX/2' test //替换第3行的第三个2017, 再替换第3行的第2个2017

sed 's/2017/XXXX/g' test //替换所有2017

sed 's/2017//' test //将所有行的第1个2017替换为空,等于删除

sed -n 's/root//p' /etc/passwd //替换passwd文件中所有行的第一个root为空,并显示替换的行

尝试使用sed替换功能将 /bin/bash 替换为 /sbin/sh

sed 's//bin/bash//sbin/sh/' user //常规方式更换,报错

sed 's/\/bin\/bash/\/sbin\/sh/' user //使用转义符号屏蔽冲突的斜杠, 可以成功,但麻烦

sed 's!/bin/bash!/sbin/sh!' user //更改替换符号为!

sed 's(/bin/bash(/sbin/sh(' user //或用(

删除文件中每行的第二个、最后一个字符

sed 's/.//2;s/.$//' test

将文件中每行的第1个、第2个字符互换

abc

sed -r 's/(.)(.)(.\*)/\2\1\3/' abc

sed -r 's/(.)(.)(.\*)/\2\1\3/' test

将文件中每行的第1个、最后1个字符互换

sed -r 's/^(.)(.\*)(.)$/\3\2\1/' test

删除文件中所有的数字

sed 's/[0-9]//g' test //找到所有数字,替换成空

为文件中每个大写字母添加括号

sed -r 's/([A-Z])/(\1)/' test //找到所有大写字母,并复制,然后在后面粘贴时,添加括号

编写脚本,一键部署ftp服务,并实现匿名上传普通文件功能

#!/bin/bash

yum -y install vsftpd &> /dev/null

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

systemctl restart vsftpd

systemctl enable vsftpd

systemctl stop firewalld //临时关闭防火墙

chmod 777 /var/ftp/pub //给pub目录权限

setenforce 0 //关闭selinux

脚本运行后,用真机访问虚拟机的ftp,可以拖拽普通文件到pub目录中

回顾

正则表达式

基本正则

^ $ [ ] [^] . \*

\{n,m\} \{n,\} \(ro\)\{2\} \(\)

扩展正则

egrep

grep -E

+ ? {n,m} | ()

\b

athey

\bthe\b

前置指令 | sed 选项 定址符 指令

sed 选项 定址符 指令 文件

逐行执行

-n -r -i

/ /p / /d / /s///

#### a 在行下面追加

#### i 在行上面添加

#### c替换整行

sed 'a 666' user //所有行下面追加666

sed '1a 666' user //第1行下面追加666

sed '/bash$/a 666' user //以bash结尾的行下面追加666

sed 'i 666' user //所有行上面添加666

sed '1i 666' user //第1行上面添加666

sed '2c 666' user //将第2行替换为666

sed 'c 666' user //将所有行替换为666

sed 修改文本

## awk 精确定位查找 逐行执行

前置指令 | awk 选项 条件 指令

awk 选项 条件 指令 文件

### 选项

-F 指定分隔符

### 指令

print 输出内容到屏幕

列 $1 第一列 $2 第二列

awk '{print}' hello //输出文档所有内容

awk '{print $2}' hello //输出文档每行第2列

awk '{print $2 , $3}' hello //输出文档每行第2列和第3列

awk '{print $3,$2}' hello

awk '{print $1,$2,$3}' hello

awk '{print $0}' hello //输出文档每行所有列

awk '{print $0,$3}' hello //输出文档每行所有列和第3列

awk -F: '{print $6}' user //修改分隔符为冒号,显示第6列

awk -F[:/] '{print $9}' user //修改分隔符为冒号或者斜杠,显示第9列

awk -Fo '{print $2}' hello //修改分隔符为字母o,显示第2列

### awk 常用内置变量

$0所有列 $1 $2 $3... NR 行号 NF 列号

awk '{print NR}' hello //查看行号

awk '{print NF}' hello //查看有几列

awk '{print NR,NF}' hello //查看行号和列

awk -Fbin/ '{print $2}' user //定义分隔符为bin/,显示第2列

df -h / | awk '{print $4}' //显示所有行第4列

df -h / | awk '/vda1/{print $4}' //显示有vda1的行的第4列

df -h / | awk '/vda1/{print "根分区剩余容量是:"$4}' //显示有vda1的行的第4列,并在前面追加常量, 常量要用双引号

使用awk过滤显示网卡流量信息:

ifconfig eth0 |awk '/RX p/{print $5}'

ifconfig eth0 |awk '/RX p/{print "eth0网卡接收的流量是" $5"字节"}'

ifconfig eth0 |awk '/TX p/{print "eth0网卡发送的流量是" $5"字节"}'

使用awk筛选安全日志中登录密码输错的主机

1,在desktop中输入tail -5 /var/log/secure 可以查看安全日志最新的5行记录

2,使用server远程登录desktop,故意输错密码,然后再次查看desktop日志发现最后一行多出记录了

3,使用真机远程登录desktop,也故意输入错误密码(需要提前在desktop中家目录的.ssh中把认证秘钥文件改名)

4,在desktop中使用命令过滤日志,查找密码输入失败的行的第11列

awk '/Failed/{print $11}' /var/log/secure

使用awk筛选输出用户相应信息

XXX的解释器路径是XXXX,家目录是XXXX

awk -F: '{print $1,"的解释器路径是"$7}' user

### 控制awk的工作流程

BEGIN{???} 执行1次

{print} 执行n次

END{???} 执行1次

制表符 \t tab键

使用awk按流程输出信息:

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

### awk处理条件

#### 1, 正则 ~ 包含 !~不包含

awk -F: '$1~/root/{print}' user //找第1列包含root的行

awk -F: '/bin/' user //找有bin的行

awk -F: '$6~/bin/' user //找第6列包含bin的行

awk -F: '$6~/\/bin/' user //找第6列包含/bin的行

awk -F: '$6!~/\/bin/' user //找第6列不包含/bin的行

#### 2,使用数值/字符串比较设置条件

比较符号：==(等于) !=（不等于） >（大于）

>=（大于等于） <（小于） <=（小于等于）

awk -F: '$1=="root"' user //找第1列等于root的行

awk -F: '$1=="roo"' user //找第1列等于roo的行

awk -F: '$1~/roo/' user //找第1列包含roo的行

awk -F: 'NR==3' user //找第3行

awk -F: '$3==0' /etc/passwd //找管理员

awk -F: '$3<10' /etc/passwd //找uid小于10的账户

awk -F: '$3<10{print $1}' /etc/passwd //找uid小于10的账户,并显示账户名

awk -F: '$3>=1000{print $1}' /etc/passwd //找普通账户,并显示账户名

#### 3,逻辑符号

&& 并且

|| 或者

awk 'NR>=3 && NR<=5' user //找3~5行

awk -F: '$3>=100 && $3<=1000' /etc/passwd //找udi大于等于100,并且小于等于1000的行

awk 'NR>3 && NR<5' user //找大于3且小于5的行

awk 'NR>3 || NR<5' user //找大于3或者小于5的行

awk 'NR>5 && NR<3' user //找大于5且小于3的行,逻辑错误

### awk中数学运算

awk 'BEGIN{x=10;y=20;print x\*y}' //定义变量,然后相乘并输出结果

awk 'BEGIN{x=10;y=20;print y%x}' //取余

awk 'BEGIN{x=10;y=20;print x+=y}' //x的自增, 支持主流写法

awk 'BEGIN{x=10;y=20;print x++}' //将x+1,但不输出

awk 'BEGIN{x=10;y=20;print x++;print x}' //将x+1,然后输出x

awk 'BEGIN{x=10;y=20;print x+=1}' //将x+1,同时输出

seq 20 | awk '$1%3==0' //找1~20以内谁是3的倍数

使用awk按照 用户名 --> 密码 的格式输出使用bash的账户信息

#!/bin/bash

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

for i in $u

do

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

done

回顾

vim 交互式 增删改查

grep 模糊查找

sed 非交互式 增删改查

awk 精确查找

前置指令 | awk 选项 条件 指令

awk 选项 条件 指令 文件

-F 修改分隔符

print

条件

1, 使用正则 ~ !~

2, 字符串与数字 == != > >= < <=

3,逻辑符号 && ||

BEGIN{} 执行1次

{ } 执行n次

END{} 执行1次

### awk的if判断

#### 单分支

{if(条件){执行指令}}

计算系统中普通账户的数量:

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

awk '{ }END{ }'

#### 双分支

{if(条件){执行指令1}else{执行指令2}}

计算系统中普通账户和系统账户的数量:

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

#### 多分支

{if(条件1){执行指令1}else if(条件2){执行指令2}else{执行指令3}}

### awk 数组

变量 可以存储1个值

数组 可以存储多个值

变量名=值

数组名[下标]=值

a[1]=10

a[2]=20

awk 'BEGIN{a[1]=10;a[2]=20;print a[2],a[1]}'

for(变量名称 in 值){执行指令}

#### awk数组+for循环基本用法:

awk 'BEGIN{a[0]=0;a[1]=11;a[2]=22; for(i in a){print i,a[i]}}' //定义一个数组a, 有3个下标,分别对应3个值, 然后交给for循环,循环显示 下标 和 值

#### awk数组+for循环扩展应用 :

准备文档包含3行内容

abc

abc

xyz

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

a[abc]++ a[abc]=2

a[xyz]++ a[xyz]=1

abc 2

xyz 1

ip[192.168.0.1]=2

ip[192.168.0.2]=1

ip[192.168.0.3]=3

使用awk统计网站访问量:

1,yum -y install httpd

2,systemctl restart httpd 开启服务

3,使用其他主机访问该网站,多访问几次

防火墙需要临时关闭 systemctl stop firewalld

4, 使用awk对日志分析, sort可以排序, -n对数字排序, -r降序

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

5, 可以使用ab工具测试网站后在交给awk分析(安装httpd之后自动安装ab工具)

ab -n 10000 -c 1 172.25.0.10/ 1个用户访问1万次网站

#!/bin/bash

while :

do

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

ifconfig eth0 | awk '/RX p/{print "网卡接收的数据量 是" $5"字节"}'

ifconfig eth0 | awk '/TX p/{print "网卡发送的数据量 是" $5"字节"}'

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

df -h | awk '/vda1/{print "根分区剩余容量是"$4}'

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

echo "服务器拥有账户数量是$user个"

u=`who | wc -l`

echo "当前服务器登录用户的数量是$u个"

p=`ps aux | wc -l`

echo "当前主机运行的进程数量是$p个"

ins=`rpm -qa |wc -l`

echo "当前主机安装软件包数量是$ins个"

sleep 3

clear

done

编写脚本,监控服务器,如果发现有人尝试登录失败5次则发邮件给管理员

#!/bin/bash

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

for i in $x

do

n=${i%,\*}

p=${i#\*,}

[ $n -ge 5 ] && echo "$p访问失败了$n次" | mail -s test root

done

测试前可以先删除邮件

rm -rf /var/spool/mail/root