[一.Shell脚本基础 1](#_Toc397780686)

[脚本的定义 1](#_Toc286297064)

[解释器 2](#_Toc603322543)

[别名(alias,unalias) 2](#_Toc156777872)

[通配符 3](#_Toc1182122872)

[快捷键 3](#_Toc1736105300)

[重定向命令输出 3](#_Toc1072901384)

[管道操作 4](#_Toc596892561)

[历史命令 4](#_Toc233435386)

[数值计算 5](#_Toc1451987287)

[二. 变量 6](#_Toc956572305)

[自定义变量的定义/查看/撤销 6](#_Toc1123242611)

[环境变量 7](#_Toc1577159484)

[系统变量 7](#_Toc974040597)

[位置变量与预定义变量 7](#_Toc885571378)

[三. 条件测试 7](#_Toc315019159)

[四. if选择结构 8](#_Toc945330739)

[循环处理 9](#_Toc2049888656)

[1. for循环 9](#_Toc1885384074)

[2. while循环 9](#_Toc1685156269)

[case语句 10](#_Toc1207441992)

[Shell函数 10](#_Toc1327729511)

[字符串处理 11](#_Toc1816945221)

[expect预期交互 11](#_Toc1856409412)

[正则表达式 12](#_Toc727789019)

[sed 13](#_Toc1415877288)

[awk 14](#_Toc1965576745)

[基本操作方法 14](#_Toc160335024)

[awk处理的时机 14](#_Toc986107602)

[awk处理条件 15](#_Toc747538702)

[awk流程控制 15](#_Toc1925004563)

# 一.Shell脚本基础

### 脚本的定义

可以运行文本文件，可以实现某种功能

• 提前设计可执行语句,用来完成特定任务的文件将命令都写入到文本文件，让后赋予文本文件执行权限(命令的堆积)

### 解释器

Shell: 解释器

– 命令的执行依赖于解释器(例如:/bin/bash)

默认的解释器程序：/bin/bash

用户-------> 解释器 --------> 内核 --------> 硬件

提示符： [**用户名@计算机的主机名 当前所在目录** ]

**管理员**的表示符号： **#**

**普通用户**的表示符号： **$**

Source \*.sh 不起子进程

Bash \*。Sh 起子进程

./\*.sh 起子进程

Shell -x 调试模式

### 别名(alias,unalias)

命令的别名：简化复杂命令的执行

**查看已设置的别名**

- alias [别名名称]

**定义新的别名**

- alias 别名名称=‘实际执行的命令行’

**取消已设置的别名**

- unalias [别名名称]

[root@A ~]# alias hn='hostname' #定义别名

[root@A ~]# hn

[root@A ~]# alias #显示当前所有生效的别名

[root@A ~]# unalias hn #删除别名hn

[root@A ~]# hn #命令没有找到

别名设置存放位置:

1. /etc/bashrc //对所有用户有效
2. ~/.bashrc //对当前用户有效

用户初始化文件(别名放在里面)

1. 用户个性化配置文件 **用户家目录/.bashrc**

• 影响**指定用户**的 bash 解释环境

– **用户家目录/.bashrc**,**每次开启 bash 终端时生效**

2.全局环境配置 **/etc/bashrc**

• 影响**所有用户**的 bash 解释环境

– **/etc/bashrc**,**每次开启 bash 终端时生效**

**定义完再开一个新的终端才能执行新的别名**

**临时取消别名** 命令前面加上\

### 通配符

• 针对不确定的文档名称,以特殊字符表示

**– \*:任意多个任意字符**

**– ?:单个字符**

• 针对不确定的文档名称,以特殊字符表示

**– [a-z]:多个字符或连续范围中的一个,若无则忽略**

**– {a,min,xy}:多组不同的字符串,全匹配**

[root@A ~]# ls /dev/tty\* #匹配/dev目录下tty开头的后面有任意多个

[root@A ~]# ls /etc/\*.conf #匹配/etc目录下.conf结尾

[root@A ~]# ls /etc/\*.tab #匹配/etc目录下.tab作为结尾的

[root@A ~]# ls /dev/tty? #匹配/dev目录下tty开头后面有一个字符的

[root@A ~]# ls /dev/tty?? #匹配/dev目录下tty开头后面有两个字符的

[root@A ~]# ls /dev/tty[1-8]

[root@A ~]# ls /dev/tty[3-6]

[root@A ~]# ls /dev/tty{1,3,7,9,18}

[root@A ~]# ls /dev/tty{S0,S1}

### 快捷键

– Ctrl + c :结束正在运行的命令

– Esc+. :粘贴上一个命令最有一个参数

– Ctrl + l :清空整个屏幕,等同于clear命令

– Ctrl+Shift + t :开启一个新的终端的快捷键

– Ctrl + u:清空至行首

– Ctrl + w:往回删除一个单词(以空格界定)

• 虚拟控制台切换 ( Ctrl + Alt + Fn 组合键)

– tty1:图形桌面

– tty2~tty6:字符控制台

### 重定向命令输出

标准输入（stdin），描述号为0；标准输出（stdout），描述号为1；标准错误（stderr），描述号为2。

操作符号 > :覆盖重定向 会覆盖目标文件（先清空、再写入）

>> :追加重定向

**>**  : 只收集前面命令的**正确输出**，将其正确输出写入文本文件

**2>** : 只收集前面命令的**错误输出**，将其错误输出写入文本文件

**&>** : 收集前面命令的**正确与错误输出**，将其正确与错误输出写入文本文件

**>&2**: 输出的**性质为错误输出，但内容不变**

**/dev/null : 黑洞设备**，专门收集无用的输出信息

[root@server0 ~]# head -2 /etc/passwd > /opt/1.txt

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

ls /nofile /etc/fstab >suc.txt 2>fail.txt

**重定向标准输入。**

[root@svr5 ~]# mail –s Error root < /etc/passwd //把passwd文件的内容作为邮件的内容

[root@svr5 ~]# mail -s test root << EOF

test mail

hell world

EOF //把EOF之间的内容作为mail的内容

Exec 0<&- //关闭标准输入

Exec 0>&- //关闭标准输出

### 管道操作

将前面的命令输出结果，交由后面命令，当作后面命令的参数在处理一遍

操作符号：|

[root@server0 ~]# ifconfig | head -2

[root@server0 ~]# head -12 /etc/passwd | tail -5

[root@server0 ~]# cat -n /etc/passwd | head -12 | tail -5

### 历史命令

/etc/profile History 配置文件

**管理/调用曾经执行过的命令**

# history 查看历史

# history -c 清空历史命令

# 上下键盘 调用历史

# !命令字串 调用最近的历史命令

# !sys 执行最近一次以sys开头的历史命令

# !n 通过历史命令的编号调用历史命令

**调整历史命令的数量**

#vim **+46** **/etc/profile**

直接到/etc/profile文件的第46行

**HISTSIZE**=1000

默认历史记录是最多1000条

**清理历史命令**

清理 history -c

#> .bash\_t=history //（还有个文件.Bash\_history还存有）

### 数值计算

**整数运算工具**

1）使用**expr**命令

乘法操作应采用 \\* 转义，避免被作为Shell通配符；参与运算的整数值与运算操作符之间需要以空格分开，引用变量时必须加$符号。

# X=1234                             //定义变量X

# expr $X + 78              //加法 1312

# expr $X - 78             //减法1156

# expr $X \\* 78          //乘法，操作符应添加\转义96252

# expr $X / 78            //除法，仅保留整除结果15

# expr $X % 78             //求模64

**2\*\*3 2的三次方 2\*\*4 2的4次方**

2. **$[]或$(())表达式**

乘法操作\*无需转义，运算符两侧可以无空格；引用变量可省略 $ 符号；计算结果替换表达式本身，可结合echo命令输出。

[root@svr5 ~]# X=1234

[root@svr5 ~]# echo $[X+78] //1312

[root@svr5 ~]# echo $[X-78] //1156

[root@svr5 ~]# echo $[X\*78] //96252

[root@svr5 ~]# echo $[X/78] //15

[root@svr5 ~]# echo $[X%78] //64

**3. 使用let命令**

expr或$[]、$(())方式只进行运算，并不会改变变量的值；而let命令可以直接对变量值做运算再保存新的值。因此变量X=1234，在执行let运算后的值会变更；另外，let运算操作并不显示结果，但是可以结合echo命令来查看：

[root@svr5 ~]# let X++; echo $X                    # X++(X=X+1)

[root@svr5 ~]# let X--; echo $X                    # X--(X=X-1)

[root@svr5 ~]# let X+=78 ; echo $X                    # X+=78(X=X+78)

[root@svr5 ~]# let X-=78 ; echo $X                 # X-=78(X=X-78)

[root@svr5 ~]# let X\*=78 ; echo $X                 # X\*=78(X=X\*78)

[root@svr5 ~]# let X/=78 ; echo $X                 # X/=78(X=X/78)

[root@svr5 ~]# let X%=78 ; echo $X                 # X%=78(X=X%79)

**小数运算工具**

bc非交互式运算

将需要运算的表达式通过管道操作交给bc运算。注意，小数位的长度可采用scale=N限制，除此以外也受参与运算的数值的小数位影响。以计算小数12.34与5.678的四则运算为例，相关操作如下：

[root@svr5 ~]# echo 'scale=4;12.34+5.678' | bc //18.018

[root@svr5 ~]# echo 'scale=4;12.34-5.678' | bc //6.662

[root@svr5 ~]# echo 'scale=4;12.34\*5.678' | bc //70.0665

[root@svr5 ~]# echo 'scale=4;12.34/5.678' | bc //2.1733

# 变量

**变量的种类**

- 位置变量 变量名一般都**大写**，用来设置用户/系统环境

- 环境变量 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

## 自定义变量的定义/查看/撤销

1）新建/赋值变量

新建变量test，赋值“hello world”，通过set命令可以检查变量设置：

[root@svr5 ~]# test=11

2）通过echo $变量名 可输出变量值：查看变量时，若变量名称与后面要输出的字符串连在一起，则应该以{}将变量名括起来以便区分：

[root@svr5 ~]# echo $testRMB      //无法识别变量名test

3. 若要撤销已有的变量，可使用unset命令：

[root@svr5 ~]# unset test       //撤销变量test

[root@svr5 ~]# echo ${test}RMB     //区分后可以识别

**变量的注意事项**

– 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

– 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

**双引号**: 使用双引号可以界定一个完整字符串。

**单引号**: 界定一个完整的字符串，并且可以实现屏蔽特殊符号的功能。

**反撇号或$()**:使用反撇号或$()时，可以将命令执行的标准输出作为字符串存储，因此称为命令替换。

**stty终端显示控制**

将回显功能关闭（stty -echo），

将回显功能恢复（stty echo）。

**全局变量:** 使用export定义.默认情况下，自定义的变量为局部变量，只在当前Shell环境中有效，而在子Shell环境中无法直接使用。若希望定义的变量能被子进程使用，可以使用export命令将其发布为全局变量。使用export发布时，只需指定变量名（可以有多个）即可，也可以通过export命令直接设置新的全局变量

## 环境变量

存放位置: 全局文件为/etc/profile，对所有用户有效；用户文件为~/.bash\_profile，仅对指定的用户有效。

USER 记录了用户名

LOGNAME 记录了登录名

HOME 记录了宿主目录

SHELL 记录了登录Shell

HOSTNAME 记录主机名

TERM 记录终端类型：

PS1 表示Shell环境的一级提示符，即命令行提示符（\u 用户名、\h 主机名、\W 工作目录、\$ 权限标识）：

PS2 表示二级提示符，出现在强制换行、at任务编辑等场合：

**PATH ： 储存命令的搜寻路径**

**即执行命令时，都会在区域PATH变量值中去寻找该命令的执行程序，如果能找到就执行，不能报告命令没有找到**

## 系统变量

使用env可查看所有环境变量

使用set可查看所有变量（包括env能看到的环境变量）

## 位置变量与预定义变量

echo $0                                        //脚本的名称

echo $1                                        //第一个参数

echo $2                                        //第二个参数

echo $\*                                        //所有参数 $@一样的功能

echo $#                                        //所有的综合

echo $$                                        //当前进程的进程号

echo $?                                        //上一个程序的返回状态码

$!: 最后一个进程的进程号.比如放在后台的进程号

# 条件测试

**几种可用的测试方式**

**- test -选项 参数...**

**- [ 测试表达式 ] (注意空格号）**

**命令行/程序的退出状态 $?**

- echo $? 值为0，表示执行成功

- echo $? 值不为0，表示执行异常或失败

**检查文件状态**

-**e**：文档**存在**为真

-**d**：**存在**并且为**目录**，才为真

-**f**：**存在**并且为**文本文件**，才为真

-**r**：**存在**并且有**读取权限**，才为真

-**w**：**存在**并且有**写入权限**，才为真

-**x**：**存在**并且有**执行权限**，才为真

**比较整数大小**

(带e字母的都有 等于 二字)

-**gt** ： 大于

-**ge** ： 大于等于

-**eq** ： 等于

-**ne** ： 不等于

-**lt** ： 小于

-**le** ： 小于等于

**字符串比对**

注意比较以下两种的区别 ,一般用前者

== : 相等为真 (前者较规范）

!= : 不相等为真

-z 检查变量的值是否未设置（空值） -n字符串的值不为空

**执行顺序**

A && B 仅当A成功，猜执行B

A || B 仅当A失败。才执行B (A或者B成功一个就行)

A ；B 执行A执行B

A&&B||C A成功就执行B，B成功语句结束。A成功执行B，B失败执行C，语句结束。A失败执行C，语句结束。 能判定表达式值的时候就结束执行

# if选择结构

格式:

if 条件测试1; then

命令序列1

elif 条件测试2; then

命令序列2

else

命令序列n

fi

**案例:**

大于等于90 神功绝世

大于等于80，小于90 登峰造极

大于等于70，小于80 炉火纯青

大于等于60，小于70 略有小成

大于60 初学乍练

#!/bin/bash

read -p "请输入积分（0-100）：" JF

if [ $JF –ge 90 ] ; then

echo "$JF 分，神功绝世"

elif [ $JF –ge 80 ] ; then

echo "$JF 分，登峰造极"

elif [ $JF –ge 70 ] ; then

echo "$JF 分，炉火纯青"

elif [ $JF –ge 60 ] ; then

echo "$JF 分，略有小成"

else

echo "$JF 分，初学乍练"

fi

# 循环处理

## for循环

for i in {1..10}

for i in 1 2 3 4 5 6 7 8 9 10

for i in `seq(10)`

for((i=1;i<=10;i++))

do

命令序列

done

## 2. while循环

while循环的语法结构如下所示：

while 条件测试

do

命令序列

Done

While : //死循环

Do

done

**中断及退出**

break可以结束整个循环

continue结束本次循环，进入下一次循环

exit结束整个脚本

# case语句

case分支的语法结构如下所示：

case 变量值 in

模式1)

命令序列1 ;;

模式2)

命令序列2 ;;

.. ..

\*)

默认命令序列

esac

# Shell函数

**定义方法**

function 函数名() {

命令序列

.. ..

}

**函数的调用**

直接使用“函数名”的形式调用，如果该函数能够处理位置参数，则可以使用“函数名 参数1 参数2 .. ..”的形式调用

颜色输出的命令:echo -e "\033[32mOK\033[0m"。

3X为字体颜色，4X为背景颜色。

#!/bin/bash

cecho() {

echo –e "\033[$1m$2\033[0m"

}

cecho 32 OK

cecho 33 OK

cecho 34 OK

cecho 35 OK

# 字符串处理

**字符串相加**

string=$string1$string2

**子串截取**：

${变量名:起始位置:长度} 位置从0开始,起始位置默认为0,可以省略

expr substr "$变量名" 起始位置 长度 位置从1开始算

echo $变量名 | cut -b 起始位置-结束位置 位置从1 开始算.选项 -b 表示按字节截取字符，其中起始位置、结束位置都可以省略。当省略起始位置时，视为从第1个字符开始（编号也是从1开始，与expr类似），当省略结束位置时，视为截取到最后。

**子串替换**：

只替换第一个匹配结果：${变量名/old/new}

替换全部匹配结果：${变量名//old/new}

**字符串掐头去尾**：

从左向右，最短匹配删除：${变量名#\*关键词}

从左向右，最长匹配删除：${变量名##\*关键词}

从右向左，最短匹配删除：${变量名%关键词\*}

从右向左，最长匹配删除：${变量名%%关键词\*}

**字符串初值的处理**

${var:-word}判断变量var是否存在，存在返回var,如果不存在就返回word

# expect预期交互

1）安装expect工具

[root@svr5 ~]# yum -y install expect                 //安装expect

[root@svr5 ~]# which expect                             //确认expect路径

/usr/bin/expect

**编写expect\_ssh脚本，实现免交互登录**

#!/bin/bash

for i in 10 11

do

expect << EOF

spawn ssh 172.25.0.$i                          #//创建交互式进程,必须有这句话

expect "password:" { send "123456\r" }             #//自动发送密码

expect "# { send "pwd > /tmp/$user.txt \r" } #//发送命令

expect "#" { send "exit\r" }

EOF

done

**注意事项：**

expect脚本的最后一行默认不执行

expect 要求顶行写

# 正则表达式

表－1 基本正则列表

正则符号 描述

^ 匹配行首

$ 匹配行尾

[ ] 集合，匹配集合中的任意单个字符

[^] 对集合取反

. 匹配任意单个字符

\* 匹配前一个字符任意次数（不能单独使用）

.\* 任何字符任何次数

\{n,m\} 匹配前一个字符n到m次

\{n\} 匹配前一个字符n次

\{n，\} 匹配前一个字符n次以上

\{,m\} 最多匹配m次

扩展正则表达式

+ 最少匹配一次

? 最多匹配一次

{n,m} 匹配n到m次

() 组合为整体,保留

\nn 匹配以保存的子组

| 或者

\b 单词边界

\d 数字

\D 非数字

\w 与[0-9a-zA-Z\_] 同义

\W 取反,与[^0-9a-zA-Z]

\s 匹配空白字符,与[\r\n\f\t\n]同义

\S 匹配非空字符

literal 匹配字符串的值,就是你写什么,就匹配什么

reg1|reg2 匹配reg1或者reg2

蓝色:匹配单个字符/模式

黄色:匹配多个字符/模式

**案例:**

**提取出httpd.conf文件的有效配置行**

# egrep -v '#|^$' /etc/httpd/conf/httpd.conf > httpd.conf.min

**匹配MAC地址** mac地址实例:00:0C:29:82:09:E9

ifconfig eth0 | egrep '[0-9a-fA-F]{2}(:[0-9a-fA-F]{2}){5}'

**匹配邮箱地址**

egrep '[0-9a-zA-Z\_.]{3,}@[0-9a-zA-Z.-]{2,}(\.[0-9a-zA-Z-]{2,})+' mailadd.txt

t[a-z]m 能匹配tam,不能匹配tabm

[] 匹配里面的特殊字符: - 不放在字母中间就行,比如开头

^ 不放在开头就行

t\dm

# sed

sed工作原理. sed顺序读取文件每一行,去匹配条件,如果匹配就执行相应的操作.

sed文本处理工具的用法：

用法1：前置命令 | sed [选项] '条件指令'

用法2：sed [选项] '条件指令' 文件.. ..

**选项**

-n 不在屏幕上输出. 默认会把文件打印在屏幕一遍

-i 修改后写回文件. 默认是只处理内存副本,不写文件

-r 支持扩展正则

**条件指令**

条件可以是行号或者/正则/

没有条件时，默认为所有条件

p打印 d删除

s修改关键词 a追加行

i插入行 c修改行

r读取 w另存为

例子:

Sed -n ‘3,6p’ 打印3到6行

Sed -n ‘1p;4p’ 打印1行跟第4 行 多个指令可以使用分号隔离

Sed -n ‘2+10p’ 打印第三行以及后面的10行，共11行

Sed -n ‘1～2p’ 从第一行开始，每隔两行打印. 也就是打印奇数行

Sed -n ‘2~2p’ 打印偶数行

Sed ‘$=’ file 统计行号

Sed ‘$d’ file 删除最后一行

sed '2a XX' a.txt            //在第二行后面，追加XX

sed '2i XX' a.txt            //在第二行前面，插入XX

sed '2c XX' a.txt            //将第二行替换为XX

sed -n '/root/p' /etc/passwd 打印包含root的行 条件指令可以使用/正则/

sed -n ‘s/root/admin/p’ /etc/passwd 把root替换为admin,并打印该行

sed 's/^#an/an/' a.txt         //解除以#an开头的行的注释（去除行首的#号）

cat nssw.txt

Hello the world

ni hao ma beijing

删除文件中每行的第二个、最后一个字符:

[root@svr5 ~]# sed 's/.//2 ; s/.$//' nssw.txt

将文件中每行的第一个、倒数第1个字符互换

[root@svr5 ~]# sed -r 's/^(.)(.\*)(.)$/\3\2\1/' nssw.txt

以nssw2.txt文件为例，删除所有数字、行首空格的操作如下：

[root@svr5 ~]# sed -r 's/[0-9]//g;s/^( )+//' nssw2.txt

为文件中每个大写字母添加括号

[root@svr5 ~]# sed 's/([A-Z])/[\1]/g' nssw.txt

# awk

基本操作方法

格式： awk [选项] '[条件]{指令}' 文件

其中，print 是最常用的编辑指令；若有多条编辑指令，可用分号分隔。Awk过滤数据时支持仅打印某一列，如第2列、第5列等。处理文本时，若未指定分隔符，则默认将空格、制表符等作为分隔符。在awk中正则表达式跟命令都只能用‘’单引号包围.在awk中任何常量都得用“”

-F 可指定分隔符

awk -F: '{print $1,$7}' /etc/passwd

awk -F [:/] '{print $1,$10}' /etc/passwd 同时指定两个

awk常用内置变量：

$1 文本的第1列

$2 文件的第2列

$3 文件的第3列，依此类推

NR 文件当前行的行号

NF 文件当前行的列数（有几列）

ifconfig eth0 | awk '/RX p/{print $5}' //过滤接收数据的流量

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

awk '/Failed/{print $11}' /var/log/secure //根据/var/log/secure日志文件，过滤远程连接密码失败的IP地址

## awk处理的时机

awk会逐行处理文本，支持在处理第一行之前做一些准备工作，以及在处理完最后一行之后做一些总结性质的工作。在命令格式上分别体现如下：

**awk [选项] ' BEGIN{指令} {指令} END{指令}' 文件**

BEGIN{ } 行前处理，读取文件内容前执行，指令执行1次

{ } 逐行处理，读取文件过程中执行，指令执行n次

END{ } 行后处理，读取文件结束后执行，指令执行1次

只做预处理的时候，可以没有操作文件，比如：

awk 'BEGIN{A=24;print A\*2}'

awk 'BEGIN{print x+1}'

awk 'BEGIN{print 3.2+3.5}'

**实例: 格式化输出passwd**

awk -F: 'BEGIN{print "User UID Home"}{print $1,$3,$6;x++}END{print NR" total user"}' /etc/passwd | column -t

User UID Home

root 0 /root

bin 1 /bin

daemon 2 /sbin

adm 3 /var/adm

lp 4 /var/spool/lpd

sync 5 /sbin

.. ..

Total 67 lines.

## awk处理条件

1. 使用正则设置条件

awk -F: '/bash$/{print}' /etc/passwd 输出其中以bash结尾的完整记录

awk -F: '/^(root|adm)/{print $1,$3}' /etc/passwd 输出root或adm账户的用户名和UID信息

awk -F: '$7!~/nologin$/{print $1,$7}' /etc/passwd 输出其中登录Shell不以nologin结尾（对第7个字段做!~反向匹配）的用户名、登录Shell信息

1. 使用数值/字符串比较设置条件

awk -F: 'NR==3{print}' /etc/passwd 输出第3行（行号NR等于3）的用户记录

awk -F: '$3<10{print $1,$3}' /etc/passwd 输出账户UID小于10的账户名称和UID信息

awk -F: '$1=="root"' /etc/passwd 输出用户名为“root”的行

awk -F: '$1~/root/' /etc/passwd 输出账户名称包含root的基本信息（第1列包含root）

1. 逻辑测试条件

awk -F: '$3>10 && $3<20' /etc/passwd 输出账户UID大于10并且小于20的账户信息

1. 数学运算

seq 200 | awk '$1%3==0{i++} END{print i}' //统计3的倍数的数量

## awk流程控制

**awk过滤中的if分支结构**

1）单分支

awk -F: '{if($3<=1000){i++}}END{print i}' /etc/passwd 统计/etc/passwd文件中UID小于或等于1000的用户个数

awk -F: '{if($7~/bash$/){i++}}END{print i}' /etc/passwd 统计/etc/passwd文件中登录Shell是“/bin/bash”的用户个数

2）双分支

awk -F: '{if($3<=500){i++}else{j++}}END{print i,j}' /etc/passwd 分别统计/etc/passwd文件中UID小于或等于1000、UID大于1000的用户个数

awk -F: '{if($7~/bash$/){i++}else{j++}} END{print i,j}' /etc/passwd 分别统计/etc/passwd文件中登录Shell是“/bin/bash”、 登录Shell不是“/bin/bash”的用户个数

**awk数组**

1）数组的语法格式

数组是一个可以存储多个值的变量，具体使用的格式如下：

定义数组的格式：数组名[下标]=元素值

调用数组的格式：数组名[下标]

遍历数组的用法：for(变量 in 数组名){print 数组名[变量]}。

awk 'BEGIN{a[0]=11;a[1]=88;print a[1],a[0]}'

88 11

awk 'BEGIN{a++;print a}'

1

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

1

awk 'BEGIN{a[0]=00;a[1]=11;a[2]=22; for(i in a){print i,a[i]}}'

0 0

1 11

2 22

注意，awk数组的下标除了可以使用数字，也可以使用字符串，字符串需要使用双引号：

awk 'BEGIN{a["hehe"]=11;print a["hehe"]}'

11

**案例:**

分析Web日志的访问量排名，要求获得客户机的地址、访问次数，并且按照访问次数排名

[root@svr5 ~]# **awk '{ip[$1]++} END{for(i in ip) {print i,ip[i]}}' /var/log/httpd/access\_log | sort -nr -k 2**

192.168.4.5 17

192.168.4.110 13

127.0.0.1 4

.. ..