**字符串的截取：**

**方法一，使用 ${}表达式 格式：${变量名:起始位置:长度} 使用${}方式起始位置是从0开始的。**

[root@server0 opt]# a=13973169942

[root@server0 opt]# echo ${a::1} #输出：1 截取第一个,起始位置省略,从第一个开始

[root@server0 opt]# echo ${a:0:1} #输出：1截取第一个,起始位置为零,从第一个开始

[root@server0 opt]# echo ${a:0:} #输出： 长度不写,无法截取

[root@server0 opt]# echo ${a:0:0} #输出： 长度为零,无法截取

[root@server0 opt]# echo ${a:0} #输出：13973169942从第一个开始,后全部截取

[root@server0 opt]# echo ${a:3} #输出：73169942从第四个开始,后面的全部截取

**方法二，使用 expr substr 格式：expr substr "$变量名" 起始位置 长度**

[root@server0 opt]# a=13973169942

[root@server0 opt]# expr substr "$a" 1 3 #输出139

[root@server0 opt]# expr substr "$a" 7 4 #输出6994

[root@server0 opt]# expr substr "$a" 8 4 #输出9942

[root@server0 opt]# expr substr "$a" 8 #无法省略长度expr: 语法错误

**方式三，使用cut分割工具 格式：echo $变量名 | cut –b 起始位置 -结束位置**

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。

]# a=123456789

]# echo $a |cut -b 1 2 #输出：cut: 2: 没有那个文件或目录 起始位置与结束位置之间少-

]# echo $a |cut -b 1-3 #输出123 从第一个到第三个

]# echo $a |cut -b 2-3 #输出23 从第二个到第三个

]# echo $a |cut -b 1 #输出1 只截取第一个

]# echo $a |cut -b 2 #输出2 只截取第二个

]# echo $a |cut -b 2- #输出23456789 从第二个开始到最后,省略了结束位置

]# echo $a |cut -b -3 #输出：123 从第一个开始到第三个,省略了起始位置

]# echo $a |cut -b 1,3 #输出：13 截取第一个和第三个

]# echo $a |cut -b 1,5 #输出：15 截取第一个和第五个

]# echo $a |cut -b 1-3,9 #输出1239 截取第一个到第三个和第九个

#! /bin/bash #批量创建用户以及为用户设置随机密码

pass(){ #定义名为pass的函数

p=

b=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

for i in `seq 8`

do

a=$[RANDOM%62]

c=${b:a:1}

p=$p$c

done

echo $p

}

d=$(cat /opt/4day/user)

for d in $d

do

useradd $d

pass #调用函数

echo $p | passwd --stdin $d

echo "$d 的密码是:$p" >> /opt/4day/userpasswd

done

**字符串的替换**

方式一：只替换第1个子串 格式：${变量名/old/new}

方式二：替换全部子串 格式：${变量名//old/new}

[root@server0 4day]# a=aabbccddeeffgg

[root@server0 4day]# echo ${a/g/h} #输出：aabbccddeeffhg 将第一个g替换成h

[root@server0 4day]# echo ${a//g/h} #输出：aabbccddeeffhh 将所有的g替换成h

**字符串的匹配删除**

[root@server0 4day]# a=`head -1 /etc/passwd` #为以下实验定义一个变量

[root@server0 4day]# echo $a #查看实验的变量

root:x:0:0:root:/root:/bin/bash

**1）从左向右，最短匹配删除**  **格式：${变量名#\*关键词}**

删除从左侧第1个字符到最近的关键词“：”的部分，\* 作通配符理解：

[root@server0 4day]# echo ${a#\*:} #包括:都被删除

x:0:0:root:/root:/bin/bash

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

**2）从左向右，最长匹配删除** **格式：${变量名##\*关键词}**

删除从左侧第1个字符到最远的关键词“:”的部分：

[root@server0 4day]# echo ${a##\*:}

/bin/bash

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

**3）从右向左，最短匹配删除** **格式：${变量名%关键词\*}**

删除从右侧最后1个字符到往左最近的关键词“:”的部分，\* 做通配符理解：

[root@server0 4day]# echo ${a%:\*}

root:x:0:0:root:/root

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

**4）从右向左，最长匹配删除 格式：${变量名%%关键词\*}**

删除从右侧最后1个字符到往左最远的关键词“:”的部分：

[root@server0 4day]# echo ${a%%:\*}

root

root:x:0:0:root:/root:/bin/bash ---红色为被删除部分

----------批量修改文件名--------------------

#! /bin/bash

a=`ls /opt/4day/\*.txt`

for i in $a

do

mv $i ${i%.\*}.doc

done

[root@server0 4day]# touch a{1..10}.txt

[root@server0 4day]# ls

1.sh 3.sh a1.txt a3.txt a5.txt a7.txt a9.txt userpasswd

2.sh a10.txt a2.txt a4.txt a6.txt a8.txt user

[root@server0 4day]# vim 3.sh

[root@server0 4day]# bash 3.sh

[root@server0 4day]# ls

1.sh 3.sh a1.doc a3.doc a5.doc a7.doc a9.doc userpasswd

2.sh a10.doc a2.doc a4.doc a6.doc a8.doc user

-----------------------------------------------------------------

**正则表达式**

**非打印字符：**

\f 匹配一个换页符。等价于 \x0c 和 \cL。

\n 匹配一个换行符。等价于 \x0a 和 \cJ。

\r 匹配一个回车符。等价于 \x0d 和 \cM。

\d 匹配数字

\t 匹配一个制表符。等价于 \x09 和 \cI。

\v 匹配一个垂直制表符。等价于 \x0b 和 \cK。

\S 匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。

\s 匹配任何空白字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。注意 Unicode 正则表达式会匹配全角空格符。

\cx 匹配由x指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。

**特殊字符**：

$ 匹配输入字符串的结尾位置。要匹配 $ 字符本身，请使用 \$。

( ) 标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \( 和 \)。

. 匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 \. 。

[ 标记一个中括号表达式的开始。要匹配 [，请使用 \[。

\* 匹配前面的子表达式零次或多次。要匹配 \* 字符，请使用 \\*。

+ 匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。

? 匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。要匹配 ? 字符，请使用 \?。

^ 匹配输入字符串的开始位置，如在方括号中使用，它表示不接受该字符集合。要匹配^字符本身，请使用 \^。

{ 标记限定符表达式的开始。要匹配 {，请使用 \{。

| 指明两项之间的一个选择。要匹配 |，请使用 \|。

\ 将下一个字符标记为或特殊字符、或原义字符、或向后引用、或八进制转义符。例如， 'n' 匹配字符 'n'。'\n' 匹配换行符。序列 '\\' 匹配 "\"，而 '\(' 则匹配 "("。

**限定符：**

限定符用来指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。

\* 匹配前面的子表达式零次或多次。例如，zo\* 能匹配 "z" 以及 "zoo"。\* 等价于{0,}。

+ 匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。

? 匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 、 "does" 中的 "does" 、 "doxy" 中的 "do" 。? 等价于 {0,1}。

{n} n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。

{n,} n 是一个非负整数。至少匹配n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o\*'。

{n,m} m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。

**\*、+限定符都是贪婪的，因为它们会尽可能多的匹配文字，只有在它们的后面加上一个?就可以实现非贪婪或最小匹配。**

**定位符：**

定位符使您能够将正则表达式固定到行首或行尾。它们还使您能够创建这样的正则表达式，这些正则表达式出现在一个单词内、在一个单词的开头或者一个单词的结尾。

^ 匹配输入字符串开始的位置。

$ 匹配输入字符串结尾的位置。

\b 匹配一个单词边界，即字与空格间的位置。

\B 非单词边界匹配。

**grep:**

**grep [OPTION选项] ... PATTERN模式 [FILE]...**

-E，--extended-regexp 模式是扩展正则表达式（ERE）

-P，--perl-regexp 模式是 Perl 正则表达式

-f，--file=FILE 从文件每一行获取匹配模式

-e，--regexp=PATTERN 使用模式匹配，可指定多个模式匹

-i，--ignore-case 忽略大小写

-w，--word-regexp 模式匹配整个单词

-x，--line-regexp 模式匹配整行

-v，--invert-match 打印不匹配的行

-q，--quiet 不输出正常信息

-o，--only-matching 只打印匹配的内容

-c，--count 只打印每个文件匹配的行数

--include=FILE\_PATTERN   只检索匹配的文件

--exclude=FILE\_PATTERN   跳过匹配的文件

--exclude-from=FILE       跳过匹配的文件，来自文件模式

--exclude-dir=PATTERN     跳过匹配的目录

---------------------

# grep -f a b #输出 b 文件中在 a 文件相同的行

# grep -v -f a b #输出 b 文件中在 a 文件不同的行

# echo "a bc de" |xargs -n1 |grep -e 'a' -e 'bc #匹配多个模式

# grep -E -v "^$|^#" /etc/httpd/conf/httpd.conf #去除空格 http.conf 文件空行或开头#号的行

# echo "A a b c" |xargs -n1 |grep -i a # -i匹配不分大小写的单词

# echo "this is a test" |grep -o 'is' #-o只显示匹配的字符串

# seq 1 20 |grep -m 5 -E '[0-9]{2}' #输出匹配的前五个结果，-E 支持扩展正则

# seq 1 20 |grep -c -E '[0-9]{2}' # 统计匹配多少行-c 只打印每个文件匹配的行数

11

# echo "a bc de" |xargs -n1 |grep '^b' #匹配 b 字符开头的行

bc

# echo "a ab abc abcd abcde" |xargs -n1 |grep -n 'de$' #匹配 de 字符结尾的行并输出匹配的行

5:abcde

# grep -r '192.167.1.1' /etc --include \*.conf 递归搜索/etc 目录下包含 ip 的 conf 后缀文件

# grep -r '192.167.1.1' /opt --exclude \*.bak 排除搜索 bak 后缀的文件

# grep -r '192.167.1.1' /opt --exclude-from file 排除来自 file 中的文件

# seq 41 45 |grep -E '4[12]' 匹配 41 或 42 的数字

# seq 13 |grep -E '[0-9]{2}' 匹配至少 2 个字符

# echo "a ab abc abcd abcde" |xargs -n1 |grep -E -w -o '[a-z]{2,3}' #匹配至少2个字符的单词，最多 3 个字符的单词

# ifconfig |grep -E -o "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" #匹配所有 IP

# seq 1 10 |grep 5 -A 3 打印匹配结果及后 3 行

# seq 1 10 |grep 5 -B 3 打印匹配结果及前 3 行

# seq 1 10 |grep 5 -C 3 打印匹配结果及前后 3 行

# grep -s 'a' abc -s不显示错误输出：

# grep -q 'a' a.txt -q不显示正常输出：

grep 支持基础和扩展正则表达式字符

xargs 是一条 Unix 和类 Unix 操作系统的常用命令；它的作用是将参数列表转换成小块分段传递给其他命令，以避免参数列表过长的问题

seq 是Linux 中一个预设的外部命令，一般用作一堆数字的简化写法。

**sed：行编辑器或者流编辑器，过滤和替换文本。**

工作原理：一次处理一行sed 命令将当前处理的行读入“模式空间”进行处理，处理完把结果输出，并清空模式空间。然后再将下一行读入模式空间进行处理输出，以此类推，直到最后一行。还有一个空间叫保持空间，又称暂存空间，可以暂时存放一些处理的数据，但不能直接输出，只能放到模式空间输出。

这两个空间其实就是在内存中初始化的一个内存区域，存放正在处理的数据和临时存放的数据。处理完之后文件内容并没有改变，除非你使用重定向存储输出。

**用法1：**前置命令 | sed [选项] '条件与指令' #管道

**用法2：**sed [选项] '条件与指令' 文件.. ..

**选项有：** -n 不打印模式空间 -i 修改原文件 -r 使用扩展正则表达式

-e 执行脚本、表达式来处理 -f 执行动作从文件读取执行

**指令有：**p：打印 d：删除 s：替换 i：在指定行之前插入文本 a:在指定行之后追加文本 c:替换指定的行

]# sed -n '1p' 1.txt

]# sed -n '2,4p' /opt/abcdef #显示第2到4行,加了-n

]# sed -n '2p;4p' /opt/abcdef #显示第2和第4行

]# sed -n '2,+2p' /opt/abcdef #从第2行开始,再打印后面2行,

]# sed -n '1~2p' /opt/abcdef #从第一行开始,每跨2行打印,奇数

]# sed -n '$=' /opt/abcdef #显示最后一行的行数

]# sed -n '$p' /opt/4day/xyz #显示最后一行的内容

]# sed -n '=' /opt/4day/xyz #显示行号

]# sed -n ‘/^the/p’ xyz #显示含”the”开头的行

]# sed '1d' 1.txt #删除1.txt文件中第一行

]# sed '1~3d' /opt/abcdef #删除第一行,每隔三行删除一个

]# sed '1,2d' nssw.txt #删除1到2行

]# sed '/the/d' xyz #删除含”the”的行

]# sed ‘/^the/d’ xyz #删除含”the”开头的行

]# sed ‘/^the/!d’ xyz #含”the”开头的行不删除,别的全部删除

]# sed ‘/^$/d ; /^#/d’ xyz #删除空行 与 #开头的(注释)

]# sed 's/the/xyz/' /opt/4day/xyz #替换每行第一个the为xyz

]# sed 's/the/xyz/2' /opt/4day/xyz #替换每行第二个the为xyz

]# sed 's/the/xyz/g' /opt/4day/xyz #替换每行所有的the为xyz

]# sed '1s/the/xyz/' /opt/4day/xyz #替换第1行的第一个the为xyz

]# sed '2s/a/x/' /opt/4day/xyz #第2行第一个a替换成x

]# sed '2,5s/a/x/' /opt/4day/xyz #第二行到第五行第一个a替换成x

]# sed 's/^the/xyz/' /opt/4day/xyz #将以the开头的替换为xyz

]# sed 's/^the/xyz/' /opt/4day/xyz #将以the开头的替换为xyz

sed -n '1s/the/xyz/p' xyz #替换第1行的the为xyz,并且只显示修改了的行

sed '2,5s/a/x/' /opt/4day/xyz #第二行到第五行第一个a替换成x

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

sed 's#/bin/bash#/sbin/sh#' a.txt #将/bin/bash替换为/sbin/sh

sed 's/([A-Z])/(\1)/g' nssw.txt #将所有大写添加括号

sed 's/.$//;s/.//2' nssw.txt #每行的第二个和最后一个删除

sed -i 's/\(^[[:space:]]\)/#\1/g' /tmp/rc.local #将空格开头的前面添加一个#

sed -r 's/^(.)(.\*)(.)$/\3\2\1/' nssw.txt #第一个最后一个对调

sed ‘2a xxx’ #在第二行下面一行插入xxx，自己是第三行

sed ‘2i xxx’ #在第二行上面一行插入xxx，自己是第二行

sed ‘3c xxx’ #替换第三行

sed ‘/root/c xxx’ #替换包含root的行为xxx

**sed正则表达式匹配,各种括号的转义和不转义**

echo "[ ]"|sed 's/\[.\*\]/aaa/g' #[ ] 需要匹配的时候，需要转义(这个是叛徒)

echo "( )"|sed 's/( )/c/g' #( ) 需要匹配的时候，不要转义

echo "{ }"|sed 's/{ }/c/g' #{ } 需要匹配的时候，不要转义

当需要匹配数字，字母等使用中括号时候[]不要转义

echo "333"|sed 's/[0-9]\{3\}/ccc/g' #但使用{}大括号作为特殊字符时候，{}需要转义。

当需要适配符，需要使用\1来替换正则表达式的对应参数时：()需要转义

echo "{1234567}"|sed 's/{\([0-9]\*\)}/\1/g' #不能写(regrexxxx)，要写\(regrexxxx\)

与此对应的还有 +和\*，在做为特别字符时候+必须转义为\+才有效，而\*则不需要。

举个例子：

echo "ccc"| sed 's/c\*/aaa/g'#正确

echo "ccc"| sed 's/c\\*/aaa/g'#错误

echo "ccc"| sed 's/c+/aaa/g'#错误

echo "ccc"| sed 's/c\+/aaa/g'#正确

sed复制粘贴

**将文件中每行的第一个、倒数第1个字符互换**

每行文本拆分为“第1个字符”、“中间的所有字符”、“倒数第1个字符”三个部分，通过替换操作重排顺序为“3-2-1”：

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

^(.) 复制开头为\1 (.\*)复制身体为\2 (.)$复制结尾为\3 重新组合:\3\2\1

[root@server0 ~]# sed -r 's/([A-Z])/(\1)/' nssw.txt #将所有的大写添加()

(H)ello the world

ni hao a (B)eijing

dsfa

df(A)sd

sed (i a c)多行文本处理

sed [选项] '条件指令' 文件..

sed工具的多行文本处理操作指令：

i： 在指定的行之前插入文本

a：在指定的行之后追加文本

c：替换指定的行

[root@server0 ~]# sed 'i x' nssw.txt #不加行号,默认在所有行之前插入

x

Hello the wor44ld

x

ni hao a Bei333jing

x

dssdf12fa

[root@server0 ~]# sed '/^H/i x' nssw.txt #在H开头的行之前插入

x

Hello the wor44ld

ni hao a Bei333jing

[root@server0 ~]# sed '/^H/a x' nssw.txt #在H开头的行之后插入

Hello the wor44ld

x

ni hao a Bei333jing

[root@server0 ~]# sed '2c x' nssw.txt #将整个第2行替换为x

Hello the wor44ld

x

dssdf12fa

**sed读取(r)与保存文件(w)**

sed [选项] '条件指令' 文件..

指令: r 读取文件 w 保存到文件

]# sed '1r b.txt' a.txt #在a.txt文件的第1行插入b.txt里面的内容

]# sed '2w c.txt' a.txt #复制a.txt里面的第二行写入到c.txt

]# cat c.txt #查看c.txt(事先未创建)

bbb

]# sed '2w c.txt' b.txt #再复制b.txt里面的第二行写入到c.txt

]# cat c.txt #再查看c.txt 之前的内容被覆盖

222

sed复制剪切

基本动作:

H:模式空间--[追加]-->保持空间 追加复制

h:模式空间--[覆盖]-->保持空间 覆盖复制

G:保持空间--[追加]-->模式空间 追加粘贴

g:保持空间--[覆盖]-->模式空间 覆盖粘贴

]# sed '1H ; $G' a.txt #追加复制第一行到剪切板,在末尾追加粘贴

aaa

bbb

ccc

#追加粘贴到这末尾 空行:剪切板中默认第一行是空行

aaa #复制粘贴到这里

]# sed '1h ; $G' a.txt #覆盖复制第一行,在末尾追加粘贴

aaa

bbb

ccc

aaa #追加粘贴到这末尾 空行:剪切板中的空行被覆盖

Sed ‘1h;2g’ a.txt

Sed ‘1h;2G’ a.txt

**awk是行处理器:** 相比较屏幕处理的优点，在处理庞大文件时不会出现内存溢出或是处理缓慢的问题，通常用来格式化文本信息**，**awk处理过程: 依次对每一行进行处理，然后输出

**格式一: 前置命令 |　awk [选项] '[条件]{指令}' 文件 格式二：awk [选项] '[条件]{指令}' 文件**

**格式三: awk [-F|-f|-v] ‘BEGIN{} //{command1; command2} END{}’ file**

[-F|-f|-v] 大参数，-F指定分隔符(默认为空格)，-f调用脚本，-v定义变量 var=value

' ' 引用代码块

BEGIN 初始化代码块，进行处理之前，执行{}里的代码，初始化代码，主要是引用全局变量，设置FS分隔符

// 匹配代码块，可以是字符串或正则表达式

{} 命令代码块，包含一条或多条命令 多条命令使用分号；分隔

END 结尾代码块，处理之后再执行{}里面的代码，主要是进行最终计算或输出结尾摘要信息

特殊要点:

$0 表示整个当前行 $1 每行第一列字段 $2 每行第2列字段

NF 当前行列的总数，字段数量变量 NR 当前行总数，当前行号，每行的记录号，多文件记录递增

FNR 与NR类似，不过多文件记录不递增，每个文件都从1开始

\t 制表符 \n 换行符

FS BEGIN时定义分隔符

RS 输入的记录分隔符， 默认为换行符(即文本是按一行一行输入)

~ 匹配，与==相比不是精确比较 !~ 不匹配，不精确比较

== 等于，必须全部相等，精确比较 != 不等于，精确比较

&&　逻辑与 || 逻辑或

+ 匹配时表示1个或1个以上 \* 匹配时表示0个或0个以上

/[0-9][0-9]+/ 两个或两个以上数字

/[0-9][0-9]\*/ 一个或一个以上数字

FILENAME 文件名

OFS 输出字段分隔符， 默认也是空格，可以改为制表符等

ORS 输出的记录分隔符，默认为换行符,即处理结果也是一行一行输出到屏幕

-F'[:#/]' 定义三个分隔符

]# awk '{print}' test.txt 　　#未加-F选项以默认的空格为分隔符

]# awk '{print $1}' test.txt 　#$1表示第一列，不是位置变量

]# head -1 /etc/passwd | awk –F : '{print $1};{print $7}' # -F ： 指定：为分隔符

]# head -1 /etc/passwd | awk –F : '{print $1 888 $7}' #输出：root888/bin/bash

]# df -h

文件系统 容量 已用 可用 已用% 挂载点

/dev/vda1 10G 3.1G 7.0G 31% /

devtmpfs 906M 0 906M 0% /dev

[root@server0 ~]# df -h | awk '{print $1,$4}'

文件系统 可用

/dev/vda1 7.0G

devtmpfs 906M

~]# head -1 /etc/passwd | awk -F[:/] '{print NF}' #输出：10

~]# head -1 /etc/passwd | awk -F: '{print "NR"}' #输出：NR常量需要双引号

~]# df -h | awk '/vda1/{print $6,"分区的剩余容量是:"$4}' 查找 #输出：/ 分区的剩余容量是:7.0G

~]# ifconfig eth0 |grep "RX p"|awk -F[\(\)] '{print "当前主机接受 的网络数据是:",$2}'

#（ 和 ）都是分隔符 #输出：当前主机接受的网络数据是: 1016.2 KiB

~]# ifconfig eth0 |awk -F[\(\)] '/RX p/{print "当前主机接受的网络 数据是:",$2}'

#第二种方法与上同效果 #输出：当前主机接受的网络数据是: 765.0 KiB

~]# awk 'BEGIN{x=0}/bash$/{x++}END{print x}' /etc/passwd #输出17

解析:统计系统中使用bash作为登录Shell的用户总个数

a.预处理时赋值变量x=0

b.然后逐行读入/etc/passwd文件，如果发现登录Shell是/bin/bash则x加1

c.全部处理完毕后，输出x的值即可。  
~]# head -5 /etc/passwd |awk -F: 'BEGIN{print "User\tUID\tHome"}{print $1"\t"$6"\t"$7}END{print "Total",NR,"lines."}'

User UID Home

root /root /bin/bash

bin /bin /sbin/nologin

Total 5 lines.

解析：格式化输出passwd文件，-F ：指定分隔符， 处理前先输出："User\tUID\tHome" \t 表示tab，

如\t在‘’中 要用双引号引起：’“\t”’tab才能生效， 处理行：输出第1、6、7行，

最后处理完之后再输出 "Total"（字符串） NR（总行号） "lines."（字符串）

**认识awk处理条件的设置**

格式二：awk [选项] '[条件]{指令}' 文件

1）使用正则设置条件

[root@server0 ~]# awk -F: '/root/' /etc/passwd #默认含有{print}

root:x:0:0:root:/root:/bin/bash

operator:x:11:0:operator:/root:/sbin/nologin

[root@server0 ~]# awk -F: '/root/{print}' /etc/passwd #{print}可省略

root:x:0:0:root:/root:/bin/bash

operator:x:11:0:operator:/root:/sbin/nologin

输出root或adm账户的用户名和UID信息：

[root@server0 ~]# awk -F: '/^root|^adm/' /etc/passwd #方法一

root:x:0:0:root:/root:/bin/bash

adm:x:3:4:adm:/var/adm:/sbin/nologin

[root@server0 ~]# awk -F: '/^(root|adm)/' /etc/passwd #方法二

root:x:0:0:root:/root:/bin/bash

adm:x:3:4:adm:/var/adm:/sbin/nologin

输出账户名称包含root的基本信息（第1列包含root）：

[root@server0 ~]# awk -F: '$1~/root/' /etc/passwd

root:x:0:0:root:/root:/bin/bash

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：

[root@server0 ~]# awk -F: '$7!~/nologin$/' /etc/passwd

[root@server0 ~]# cat a.txt

abc 123 xyz

xyz abc

123

456

aaa

[root@server0 ~]# awk '$1~/abc/' a.txt #查找第1列为abc的行 ~匹配

abc 123 xyz

[root@server0 ~]# awk '$2~/abc/' a.txt #查找第2列为abc的行

xyz abc

[root@server0 ~]# awk '$2!~/abc/' a.txt #查找除了第2列为abc其他的所有行

abc 123 xyz

123

456

aaa

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） -- ++

[root@server0 ~]# awk -F: 'NR==3{print}' /etc/passwd #输出行号等于3的

daemon:x:2:2:daemon:/sbin:/sbin/nologin

输出行中第三列大于等于1000的第一列与第三列

[root@server0 ~]# awk -F: '$3>=1000{print $1,$3}' /etc/passwd

[root@server0 ~]# awk -F: '$3<10{print $1,$3}' /etc/passwd #输出第三列小于10的

root 0

bin 1

[root@server0 ~]# awk -F: '$1=="root"' /etc/passwd #输出第一列等于root的行

root:x:0:0:root:/root:/bin/bash

第三列的值大于10和小于20的

[root@server0 ~]# awk -F: '$3>10&&$3<20' /etc/passwd operator:x:11:0:operator:/root:/sbin/nologin

games:x:12:100:games:/usr/games:/sbin/nologin

ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin

第三列值大于1000或者小于20的

[root@server0 ~]# awk -F: '$3>1000 && $3<20' /etc/passwd #错误

[root@server0 ~]# awk -F: '$3>1000 || $3<20' /etc/passwd

找到使用bash作登录Shell的本地用户,列出这些用户的shadow密码记录，

[root@server0 ~]# bash awk.sh

root ---> $6$UiGI4Tc2$htsXYn5cJnOqv3P1VLcUSgfjDu2pL5yiJBuua6foZAHdwqeuLHfYUfS/vBn27Wjvoel8EJgtdsMjyquqvKAmf1

student ---> $6$8oIjLCsc$/n1iQXYh1E6.uOEuJKgioqAtmqm2TQmkJGF2RwyteIr1tIfrPdiRYgWe6Sjen5/eMij2uHM/a1tue/QRlo3X80

----------脚本1-------------------------------------------------

#! /bin/bash

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

for i in $a

do

p=`awk -F: '$1=="'$i'"{print $2}' /etc/shadow`

echo "$i ---> $p"

Done

-----------脚本2---------------------------------------------

#! /bin/bash

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

for i in $a

do

p=`grep "^$i" /etc/shadow |awk -F: '{print $2}'`

echo "$i ---> $p"

done

------------------------------------------------------------------