

# 正则表达式 RE

=====

重要的文本处理工具：vim、sed、awk、grep

mysql、oracle、php、python ...

## 一、什么是正则表达式？

正则表达式（regular expression, RE）是一种字符模式，用于在查找过程中匹配指定的字符。在大多数程序里，正则表达式都被置于两个正斜杠之间；例如 `/l[oo]ve/` 就是由正斜杠界定的正则表达式，

它将匹配被查找的行中任何位置出现的相同模式。在正则表达式中，元字符是最重要的概念。

匹配数字: `^[0-9]+$` 123 456 5y7

匹配 Mail: `[a-z0-9_]+@[a-z0-9]+\.[a-z]+` yangsheng131420@126.com

匹配 IP: `[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}`

或

`[:digit:]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}`

```
[root@tianyun scripts]# egrep '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}'  
/etc/sysconfig/network-scripts/ifcfg-eth0
```

IPADDR=172.16.100.1

NETMASK=255.255.255.0

GATEWAY=172.16.100.254

```
[root@tianyun scripts]# egrep '[:digit:]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}'  
/etc/sysconfig/network-scripts/ifcfg-eth0
```

IPADDR=172.16.100.1

NETMASK=255.255.255.0

GATEWAY=172.16.100.254

## 二、元字符

定义：元字符是这样一类字符，它们表达的是不同于字面本身的含义

shell 元字符(也称为通配符) 由 shell 来解析，如 `rm -rf *.pdf`，元字符 `*` Shell 将其解析为任意多个字符

正则表达式元字符 由各种执行模式匹配操作的程序来解析，比如 vi、grep、sed、awk、python

```
[root@tianyun ~]# rm -rf *.pdf
```

```
[root@tianyun ~]# grep 'abc*' /etc/passwd
```

abrt:x:173:173::/etc/abrt:/sbin/nologin

vim 示例:

:1,\$ s/tom/David/g //如 tom、anatomy、tomatoes 及 tomorrow 中的“tom”被替换了, 而 Tom 确没被替换

:1,\$ s/\<[Tt]om\>/David/g

## 1. 正则表达式元字符:

===基本正则表达式元字符

元字符 功能 示例

=====

^ 行首定位符 ^love

\$ 行尾定位符 love\$

. 匹配单个字符 l.e

\* 匹配前导符 0 到多次 ab\*love

. \* 任意多个字符

[] 匹配指定范围内的一个字符 [IL]ove

[-] 匹配指定范围内的一个字符 [a-z0-9]ove

[^] 匹配不在指定组内的字符 [^a-z0-9]ove

\ 用来转义元字符 love\.

\< 词首定位符 \<love

\> 词尾定位符 love\>

\(.\) 匹配稍后使用的字符的标签 :% s/172.16.130.1/172.16.130.5/

:% s/\(172.16.130.\)1/\15/

:% s/\(172.\)\(16.\)\(130.\)1/\1\2\35/

:3,9 s/\(.\*\)/#\1/

x\{m\} 字符 x 重复出现 m 次 o\{5\}

x\{m,\} 字符 x 重复出现 m 次以上 o\{5,\}

x\{m,n\} 字符 x 重复出现 m 到 n 次 o\{5,10\}

===扩展正则表达式元字符

+ 匹配一个或多个前导字符 [a-z]+ove

? 匹配零个或一个前导字符 lo?ve

a|b 匹配 a 或 b love|hate

() 组字符 loveable|rs love(able|rs) ov+ (ov)+

(..)(..)\1\2 标签匹配字符 (love)able\1er

x{m} 字符 x 重复 m 次 o{5}

x{m,} 字符 x 重复至少 m 次 o{5,}

x{m,n} 字符 x 重复 m 到 n 次 o{5,10}

## 2. POSIX 字符类:

表达式 功能 示例

[[:alnum:]] 字母与数字字符 [[:alnum:]]+  
[[:alpha:]] 字母字符(包括大小写字母) [[:alpha:]]{4}  
[[:blank:]] 空格与制表符 [[:blank:]]\*  
[[:digit:]] 数字字母 [[:digit:]]?  
[[:lower:]] 小写字母 [[:lower:]]{5,}  
[[:upper:]] 大写字母 [[:upper:]]+  
[[:punct:]] 标点符号 [[:punct:]]  
[[:space:]] 包括换行符, 回车等在内的所有空白[[:space:]]+

### 三、正则匹配示例: vim

```
/love/  
/^love/  
/love$/  
/l.ve/  
/lo*ve/  
/[Ll]ove/  
/love[a-z]/  
/love[^a-zA-Z0-9]/  
  
/.*/  
/^$/  
/^[A-Z]..$/  
/^[A-Z][a-z ]*3[0-5]/  
/[a-z]*\./  
/^[|][A-Z][a-z][a-z]$/  
/^[A-Za-z]*[^\,][A-Za-z]*$/  
/\<fourth\>/  
/\<f.*th\>/  
/5{2}2{3}\./
```

空行

```
/^$/  
/^[|]\t]*$/
```

注释行

```
/^#/  
/^[| \t]*#/
```

```
:1,$ s/\([Oo]ccur\)ence/\1rence/  
:1,$ s/\(square\) and \(fair\) /\2 and \1/
```

=====

=====

```

graph LR
    A[文本文件  
this is a line] --> B(sed 的模式空间(缓冲区)  
this is a line)
    B --> C(sed 的输出  
this is a line)
  
```

中，称为“**模式空间**”（pattern space），接着用 sed 命令处理缓冲区中的内容，处理完成后，把缓冲区的内容

Sed 主要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程序等。

## 二、命令格式

注：

只有当命令存在语法错误时，sed 的退出状态才是非 0

与 `grep` 一样，`sed` 在文件中查找模式时也可以使用正则表达式(RE)和各种元字符。正则表达式是

使用基本元字符集 `^, $, ., *, [], [^], \< \>, \(), \{\}`

使用扩展元字符的方式:

```
\+
sed -r
```

#### 四、sed 基本用法

```
# sed -r '' /etc/passwd
# sed -r 'p' /etc/passwd
# sed -r -n 'p' /etc/passwd
# sed -r -n '/root/p' /etc/passwd
# sed -r 's/root/alice/' /etc/passwd
# sed -r 's/root/alice/g' /etc/passwd
# sed -r 's/root/alice/gi' /etc/passwd

# sed -r '/root/d' /etc/passwd
# sed -r '\crootcd' /etc/passwd
=====

[root@tianyun ~] vim a.txt
/etc/abc/456
etc

[root@tianyun ~]# sed -r '//etc/abc/456/d' a.txt
sed: -e 表达式 #1, 字符 0: no previous regular expression

[root@tianyun ~]# sed -r '/^etc\abc\456/d' a.txt
/etc/abc/456

[root@tianyun ~]# sed -r '\#/etc/abc/456#d' a.txt
/etc/abc/456

[root@tianyun ~]# sed -r 's#/etc/abc/456#/dev/sda1#'
=====
```

#### 五、sed 扩展

==地址（定址）

地址用于决定对哪些行进行编辑。地址形式可以是数字、正则表达式或二者的结合。如果没有指定

地址，sed 将处理输入文件中的所有行。

```
# sed -r 'd' /etc/passwd
# sed -r '3d' /etc/passwd
# sed -r '1,3d' /etc/passwd
# sed -r '/root/d' /etc/passwd
# sed -r '/root/,5d' /etc/passwd
# sed -r 's/root/alice/g' /etc/passwd

# sed -r '/^adm/,+20d' /etc/passwd
# sed -r '2,5d' /etc/passwd
```

```
# sed -r '/root/d' /etc/passwd
# sed -r '/root/!d' /etc/passwd
```

```
# sed -r '1~2d' /etc/passwd //删除所有奇数行 odd-numbered
# sed -r '0~2d' /etc/passwd //删除所有偶数行 even-numbered
```

## ==sed 命令

sed 命令告诉 sed 对指定行进行何种操作，包括打印、删除、修改等。

### 命令 功能

- a** 在当前行后添加一行或多行
- c** 用新文本修改（替换）当前行中的文本
- d** 删除行
- i** 在当前行之前插入文本
- l** 列出非打印字符
- p** 打印行
- n** 读入下一输入行，并从下一条命令而不是第一条命令开始对其的处理
- q** 结束或退出 sed
- !** 对所选行以外的所有行应用命令

## s 用一个字符串替换另一个

### s 替换标志

- g** 在行内进行全局替换
- i** 忽略大小写
- r** 从文件中读
- w** 将行写入文件
- y** 将字符转换为另一字符（不支持正则表达式）
- h** 把模式空间里的内容复制到暂存缓冲区(覆盖)
- H** 把模式空间里的内容追加到暂存缓冲区
- g** 取出暂存缓冲区的内容，将其复制到模式空间，覆盖该处原有内容
- G** 取出暂存缓冲区的内容，将其复制到模式空间，追加在原有内容后面
- x** 交换暂存缓冲区与模式空间的内容

## ==选项

### 选项 功能

- e** 允许多项编辑
- f** 指定 sed 脚本文件名
- n** 取消默认的输出
- i** inplace，就地编辑

**-r** 支持扩展元字符

## 六、sed 命令示例

打印命令: **p**

```
# sed -r '/north/p' datafile  
# sed -r -n '/north/p' datafile
```

删除命令: **d**

```
# sed -r '3d' datafile  
# sed -r '3{d;}' datafile  
# sed -r '3{d}' datafile  
  
# sed -r '3,$d' datafile  
# sed -r '$d' datafile  
# sed -r '/north/d' datafile  
# sed -r '/sout/d' datafile
```

替换命令: **s**

```
# sed -r 's/west/north/g' datafile  
# sed -r 's/^west/north/' datafile  
# sed -r 's/[0-9][0-9]$/&.5/' datafile //&代表在查找串中匹配到的内容  
# sed -r 's/Hemenway/Jones/g' datafile  
# sed -r 's/(Mar)got/\1ianne/g' datafile  
# sed -r 's#3#88#g' datafile
```

读文件命令: **r**

```
# sed -r '/Suan/r /etc/newfile' datafile  
# sed -r '2r /etc/hosts' a.txt  
# sed -r '/2/r /etc/hosts' a.txt
```

写文件命令: **w**

```
# sed -r '/north/w newfile' datafile  
# sed -r '3,$w /new1.txt' datafile
```

追加命令: **a**

```
# sed -r '2a\111111111111' /etc/hosts  
# sed -r '2a\111111111111\  
> 222222222222\  
> 333333333333' /etc/hosts
```

插入命令: **i**

```
# sed -r '2i\111111111111' /etc/hosts  
# sed -r '2i1111111111\  
> 222222222222'
```

```
> 2222222222\
> 3333333333' /etc/hosts
```

修改命令: c

```
# sed -r '2c\111111111111' /etc/hosts
# sed -r '2c\11111111111\
> 2222222222\
> 3333333333' /etc/hosts
```

获取下一行命令: n

```
# sed -r '/eastern/{ n; d }' datafile
# sed -r '/eastern/{ n; s/AM/Archile/ }' datafile
```

暂存和取用命令: h H g G

```
# sed -r '1h;$G' /etc/hosts
# sed -r '1{h;d};$G' /etc/hosts
# sed -r '1h; 2,$g' /etc/hosts
# sed -r '1h; 2,3H;$G' /etc/hosts
```

暂存空间和模式空间互换命令: x

```
# sed -r '4h; 5x; 6G' /etc/hosts
```

反向选择: !

```
# sed -r '3d' /etc/hosts
# sed -r '3!d' /etc/hosts
```

多重编辑选项: e

```
# sed -r -e '1,3d' -e 's/Hemenway/Jones/' datafile
# sed -r '1,3d; s/Hemenway/Jones/' datafile

# sed -r '2s/WE/UPLOOKING/g; 2s/Gray/YYY/g' datafile
# sed -r '2{s/WE/UPLOOKING/g; s/Gray/YYY/g}' datafile
```

## 七、sed 常见操作

删除配置文件中#号注释行

```
sed -ri '/^#/d' file.conf
```

```
sed -ri '/^[ \t]*#/d' file.conf
```

删除配置文件中//号注释行



```
sed -ri '\Y^[ \t]*//Yd' file.conf
```

删除无内容空行

```
sed -ri '/^[ \t]*$/d' file.conf
```

删除注释行及空行：

```
# sed -ri '/^[ \t]*#/d; /^[ \t]*$/d' /etc/vsftpd/vsftpd.conf
```

```
# sed -ri '/^[ \t]*#|^[ \t]*$/d' /etc/vsftpd/vsftpd.conf
```

```
# sed -ri '/^[ \t]*($|#)/d' /etc/vsftpd/vsftpd.conf
```

修改文件：

```
# sed -ri '$a\chroot_local_user=YES' /etc/vsftpd/vsftpd.conf
```

```
# sed -ri '/^SELINUX=cSELINUX=disabled' /etc/selinux/config
```

```
# sed -ri '/UseDNS/cUseDNS no' /etc/ssh/sshd_config
```

```
# sed -ri '/GSSAPIAuthentication/cGSSAPIAuthentication no' /etc/ssh/sshd_config
```

给文件行添加注释：

```
# sed -r '2,6s/^/#/' a.txt
```

```
# sed -r '2,6s/(.*)/#\1/' a.txt
```

```
# sed -r '2,6s/.*/#&/' a.txt &匹配前面查找的内容
```

```
# sed -r '3,$ s/^##*/#/' a.txt 将行首零个或多个#换成一个#
```

```
# sed -r '30,50s/^[ \t]*##*/#' /etc/nginx.conf
```

```
# sed -r '2,8s/^[ \t#]*#/' /etc/nginx.conf
```

sed 中使用外部变量

```
# var1=11111
```

```
# sed -ri "3a$var1" /etc/hosts
```

```
# sed -ri "$a$var1" /etc/hosts
```

```
# sed -ri '$a\'"$var1" /etc/hosts
```

```
# sed -ri 3a$var1 /etc/hosts
```

```
# sed -ri "\$a$var1" /etc/hosts
```

作业：

```
[root@tianyun ~]# vim 12345.txt
```

1  
2  
3  
4  
5

```
[root@tianyun ~]# sed -r '1!G; $!h; $!d' 12345.txt
```

5  
4  
3  
2  
1

# 文本处理 awk

=====

## 一、awk 简介

awk 是一种编程语言,用于在 linux/unix 下对文本和数据进行处理。数据可以来自标准输入、一个或多个文件,或其它命令的输出。它支持用户自定义函数和动态正则表达式等先进功能,是 linux/unix 下的一个强大编程工具。它在命令行中使用,但更多是作为脚本来使用。

awk 的处理文本和数据的方式是这样的,它逐行扫描文件,从第一行到最后一行,寻找匹配的特定模式的行,并在这些行上进行你想要的操作。如果没有指定处理动作,则把匹配的行显示到标准输出(屏幕),如果没有指定模式,则所有被操作所指定的行都被处理。awk 分别代表其作者姓氏的第一个字母。因为它的作者是三个人,分别是 Alfred Aho、Brian Kernighan、Peter Weinberger。gawk 是 awk 的 GNU 版本,它提供了 Bell 实验室和 GNU 的一些扩展。

## 二、awk 的两种形式语法格式

```
awk [options] 'commands' filenames
awk [options] -f awk-script-file filenames
```

==options:

-F 定义输入字段分隔符,默认的分隔符是空格或制表符(tab)

==command:

```
BEGIN{} {} END{}
```

行处理前 行处理 行处理后

```
# awk 'BEGIN{print 1/2} {print "ok"} END{print "-----"}' /etc/hosts
```

0.5

ok

ok

ok

-----

BEGIN{} 通常用于定义一些变量,例如 BEGIN{FS=":";OFS="----"}

==awk 命令格式:

awk 'pattern' filename 示例: awk -F '/' /etc/passwd

awk '{action}' filename 示例: awk -F: '{print \$1}' /etc/passwd  
awk 'pattern {action}' filename 示例: awk -F: '/root/{print \$1,\$3}' /etc/passwd  
示例: awk 'BEGIN{FS=":"}' /root/{print \$1,\$3}' /etc/passwd  
command |awk 'pattern{action}' 示例: df -P | grep '/' |awk '\$4 > 25000 {print \$4}'

=====

## awk 工作原理

# awk -F: '{print \$1,\$3}' /etc/passwd

(1)awk 使用一行作为输入，并将这一行赋给内部变量\$0，每一行也可称为一个记录，以换行符结束

(2)然后，行被:（默认为空格或制表符）分解成字段（或域），每个字段存储在已编号的变量中，从\$1 开始，  
最长达 100 个字段

(3)awk 如何知道用空格来分隔字段的呢？ 因为有一个内部变量 FS 来确定字段分隔符。初始时，FS 赋为空格

(4)awk 打印字段时，将以设置的方法使用 print 函数打印，awk 在打印的字段间加上空格，  
因为\$1,\$3 之间  
有一个逗号。逗号比较特殊，它映射为另一个内部变量，称为输出字段分隔符 OFS，OFS 默认为空格

(5)awk 输出之后，将从文件中获取另一行，并将其存储在\$0 中，覆盖原来的内容，然后将新的字符串分隔  
成字段并进行处理。该过程将持续到所有行处理完毕

=====

==记录与字段相关内部变量: man awk

\$0: awk 变量\$0 保存当前记录的内容 # awk -F: '{print \$0}' /etc/passwd

NR: The total number of input records seen so far. # awk -F: '{print NR, \$0}' /etc/passwd /etc/hosts

FNR: The input record number in the current input file # awk -F: '{print FNR, \$0}' /etc/passwd /etc/hosts

NF: 保存记录的字段数，\$1,\$2...\$100 # awk -F: '{print \$0,NF}' /etc/passwd

FS: 输入字段分隔符，默认空格 # awk -F: '/alice/{print \$1, \$3}' /etc/passwd

# awk -F: '\t' '{print \$1,\$2,\$3}' /etc/passwd

# awk 'BEGIN{FS=":"}' {print \$1,\$3}' /etc/passwd

OFS: 输出字段分隔符 # awk -F: '/alice/{print \$1,\$2,\$3,\$4}' /etc/passwd

# awk 'BEGIN{FS=":"; OFS="+++"}' /root/{print \$1,\$2,\$3,\$4}' /etc/passwd

RS The input record separator, by default a newline. # awk -F: 'BEGIN{RS=" "}' {print \$0}' a.txt

ORS The output record separator, by default a newline. # awk -F: 'BEGIN{ORS=" "}' {print \$0}' passwd

lab1:

```
[root@yang ~]# awk 'BEGIN{ORS=" "}{print $0}' /etc/passwd #将文件每一行合并为一行
```

lab2:

```
[root@yang ~]# head -1 /etc/passwd > passwd1
```

```
[root@yang ~]# cat passwd1
```

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

```
[root@yang ~]#
```

```
[root@yang ~]# awk 'BEGIN{RS=":"}{print $0}' passwd1
```

```
root
```

```
x
```

```
0
```

```
0
```

```
root
```

```
/root
```

```
/bin/bash
```

```
[root@yang ~]# awk 'BEGIN{RS=":"}{print $0}' passwd1 |grep -v '^$' > passwd2
```

```
=====
```

== 格式化输出:

print 函数

```
# date |awk '{print "Month: " $2 "\nYear: " $NF}'
```

```
# awk -F: '{print "username is: " $1 "\t uid is: " $3}' /etc/passwd
```

```
# awk -F: '{print "\tusername and uid: " $1,$3 "!"}' /etc/passwd
```

printf 函数

```
# awk -F: '{printf "%-15s %-10s %-15s\n", $1,$2,$3}' /etc/passwd
```

```
# awk -F: '{printf "|%-15s| %-10s| %-15s|\n", $1,$2,$3}' /etc/passwd
```

%s 字符类型

%d 数值类型

占 15 字符

- 表示左对齐，默认是右对齐

printf 默认不会在行尾自动换行，加\n

```
=====
```

### 三、awk 模式和动作

任何 awk 语句都由**模式**和**动作**组成。模式部分决定动作语句何时触发及触发事件。处理即对数据进行的操作。

如果省略模式部分，动作将时刻保持执行状态。模式可以是任何**条件语句**或**复合语句**或**正则**

**表达式**。模式包括两

个特殊字段 **BEGIN** 和 **END**。使用 **BEGIN** 语句设置计数和打印头。**BEGIN** 语句使用在任何文本浏览动作之前，之

后文本浏览动作依据输入文本开始执行。**END** 语句用来在 **awk** 完成文本浏览动作后打印输出文本总数和结尾状态。

## 模式可以是：

**==正则表达式：**

**匹配记录（整行）：**

```
# awk '/^alice/' /etc/passwd
```

```
# awk '$0 ~ /^alice/' /etc/passwd
```

```
# awk '!/alice/' passwd
```

```
# awk '$0 !~ /^alice/' /etc/passwd
```

**匹配字段：匹配操作符（~ !~）**

```
# awk -F: '$1 ~ /^alice/' /etc/passwd
```

```
# awk -F: '$NF !~/bash$/ ' /etc/passwd
```

**==比较表达式：**

比较表达式采用对文本进行比较，只有当条件为真，才执行指定的动作。比较表达式使用关系运算符，

用于比较**数字**与**字符串**。

**关系运算符**

运算符 含义 示例

< 小于 x<y

<= 小于或等于 x<=y

== 等于 x==y

!= 不等于 x!=y

>= 大于等于 x>=y

> 大于 x>y

```
# awk -F: '$3 == 0' /etc/passwd
```

```
# awk -F: '$3 < 10' /etc/passwd
```

```
# awk -F: '$7 == "/bin/bash"' /etc/passwd
```

```
# awk -F: '$1 == "alice"' /etc/passwd
```

```
# awk -F: '$1 ~ /alice/' /etc/passwd
```

```
# awk -F: '$1 !~ /alice/' /etc/passwd
```

```
# df -P | grep '/' | awk '$4 > 25000'
```

==条件表达式:

```
# awk -F: '{ $3>300 {print $0} }' /etc/passwd
# awk -F: '{ if($3>300) print $0 }' /etc/passwd
# awk -F: '{ if($3>300) {print $0} }' /etc/passwd
# awk -F: '{ if($3>300) {print $3} else{print $1} }' /etc/passwd
```

==算术运算: + - \* / %(模) ^(幂 2^3)

可以在模式中执行计算, awk 都将按浮点数方式执行算术运算

```
# awk -F: '{ $3 * 10 > 500 }' /etc/passwd
# awk -F: '{ if($3*10>500){print $0} }' /etc/passwd
```

==逻辑操作符和复合模式

&& 逻辑与 a&&b

|| 逻辑或 a||b

! 逻辑非 !a

```
# awk -F: '{ $1~/root/ && $3<=15 }' /etc/passwd
# awk -F: '{ $1~/root/ || $3<=15 }' /etc/passwd
# awk -F: '{ !($1~/root/ || $3<=15) }' /etc/passwd
```

==范围模式

```
# awk '/Tom/,/Suzanne/' filename
```

awk 示例:

```
# awk '/west/' datafile
# awk '/^north/' datafile
# awk '$3 ~ /^north/' datafile
# awk '/^(no|so)/' datafile
# awk '{print $3,$2}' datafile

# awk '{print $3 $2}' datafile
# awk '{print $0}' datafile
# awk '{print "Number of fields: "NF}' datafile
# awk '/northeast/{print $3,$2}' datafile
# awk '/E/' datafile

# awk '/^[ns]/{print $1}' datafile
# awk '$5 ~ /\.[7-9]+/' datafile
# awk '$2 !~ /E/{print $1,$2}' datafile
# awk '$3 ~ /^Joel/{print $3 " is a nice boy."}' datafile
```

```
# awk '$8 ~ /[0-9][0-9]$/{print $8}' datafile
```

```
# awk '$4 ~ /Chin$/{print "The price is $" $8 "."}' datafile
```

```
# awk 'Tj/{print $0}' datafile
```

```
# awk '{print $1}' /etc/passwd
```

```
# awk -F: '{print $1}' /etc/passwd
```

```
# awk '{print "Number of fields: "NF}' /etc/passwd
```

```
# awk -F: '{print "Number of fields: "NF}' /etc/passwd
```

```
# awk -F"[:]" '{print NF}' /etc/passwd
```

```
# awk '$7 == 5' datafile
```

```
# awk '$2 == "CT" {print $1, $2}' datafile
```

```
# awk '$7 != 5' datafile
```

lab3:

```
[root@yang ~]# cat b.txt
```

```
yang sheng:is a:good boy!
```

```
[root@yang ~]# awk '{print NF}' b.txt
```

```
4
```

```
[root@yang ~]# awk -F: '{print NF}' b.txt
```

```
3
```

```
[root@yang ~]# awk -F"[:]" '{print NF}' b.txt
```

```
6
```

```
# awk '$7 < 5 {print $4, $7}' datafile
```

```
# awk '$6 > .9 {print $1,$6}' datafile
```

```
# awk '$8 <= 17 {print $8}' datafile
```

```
# awk '$8 >= 17 {print $8}' datafile
```

```
# awk '$8 > 10 && $8 < 17' datafile
```

```
# awk '$2 == "NW" || $1 ~ /south/ {print $1, $2}' datafile
```

```
# awk '!( $8 == 13){print $8}' datafile
```

```
# awk '/southem/{print $5 + 10}' datafile
```

```
# awk '/southem/{print $8 + 10}' datafile
```

```
# awk '/southem/{print $5 + 10.56}' datafile
```

```
# awk '/southem/{print $8 - 10}' datafile
```

```
# awk '/southem/{print $8 / 2 }' datafile
```

```
# awk '/southem/{print $8 / 3 }' datafile
```

```
# awk '/southem/{print $8 * 2 }' datafile
```

```
# awk '/southem/{print $8 % 2 }' datafile
```

```
# awk '$3 ~ /^Suan/ {print "Percentage: "$6 + .2 " Volume: " $8}' datafile
```

```
# awk '/^western/,/^eastern/' datafile
```

```
# awk '{print ($7 > 4 ? "high "$7 : "low "$7)}' datafile //三目运算符 a?b:c 条件?结果 1:结果 2
```



```
# awk '$3 == "Chris" { $3 = "Christian"; print $0 }' datafile //赋值运算符
# awk '/Derek/ { $8 += 12; print $8 }' datafile // $8 += 12 等价于 $8 = $8 + 12
# awk '{ $7 % = 3; print $7 }' datafile // $7 % = 3 等价于 $7 = $7 % 3
```

## 四、awk 脚本编程

**== 条件判断**

**if 语句:**

格式

{if(表达式) { 语句;语句;... } }

awk -F: '{if(\$3==0) {print \$1 " is administrator."}}' /etc/passwd

awk -F: '{if(\$3>0 && \$3<1000){count++;} END{print count}}' /etc/passwd //统计系统用户数

**if...else 语句:**

格式

{if(表达式) { 语句;语句;... } else{语句;语句;...}}

awk -F: '{if(\$3==0){print \$1} else {print \$7}}' /etc/passwd

awk -F: '{if(\$3==0) {count++;} else{i++;} }' /etc/passwd

awk -F: '{if(\$3==0){count++;} else{i++;} END{print "管理员个数: "count ; print "系统用户数: "i}}' /etc/passwd

**if...else if...else 语句:**

格式

{if(表达式 1) { 语句;语句; ... } else if(表达式 2) { 语句;语句; ... } else if(表达式 3) { 语句;语句; ... } else { 语句;语句; ... } }

awk -F: '{if(\$3==0){i++;} else if(\$3>999){k++;} else{j++;} END{print i; print k; print j}}' /etc/passwd

awk -F: '{if(\$3==0){i++;} else if(\$3>999){k++;} else{j++;} END{print "管理员个数: "i; print "普通用户数: "k; print "系统用户: "j}}' /etc/passwd

**== 循环**

**while:**

[root@tianyun ~]# awk 'BEGIN{ i=1; while(i<=10){print i; i++} }'

[root@tianyun ~]# awk -F: '{i=1; while(i<=7){print \$i; i++}}' passwd

[root@tianyun ~]# awk -F: '{i=1; while(i<=NF){print \$i; i++}}' /etc/hosts

[root@tianyun ~]# awk -F: '{i=1; while(i<=10) {print \$0; i++}}' /etc/passwd //将每行打印 10 次

[root@tianyun ~]# cat b.txt

111 222

333 444 555

666 777 888 999

[root@tianyun ~]# awk '{i=1; while(i<=NF){print \$i; i++}}' b.txt

111

222  
333  
444  
555  
666  
777  
888  
999

**for:**

```
[root@tianyun ~]# awk 'BEGIN{for(i=1;i<=5;i++){print i}}' //C 风格 for
```

1  
2  
3  
4  
5

```
[root@tianyun ~]# awk -F: '{ for(i=1;i<=10;i++) {print $0} }' /etc/passwd //将每行打印 10 次
```

```
[root@tianyun ~]# awk -F: '{ for(i=1;i<=NF;i++) {print $i} }' passwd
```

root  
x  
0  
0  
root  
/root  
/bin/bash  
bin  
x  
1  
1  
bin  
/bin  
/sbin/nologin

**== 数组**

```
# awk -F: '{username[+i]=$1} END{print username[1]}' /etc/passwd
```

root

```
# awk -F: '{username[i++]=$1} END{print username[1]}' /etc/passwd
```

bin

```
# awk -F: '{username[i++]=$1} END{print username[0]}' /etc/passwd
```

root

**数组遍历:**

1. 按索引遍历

## 2. 按元数个数遍历

### ==按元数个数遍历==

```
# awk -F: '{username[x++]= $1} END{for(i=0;i<x;i++) print i,username[i]} ' /etc/passwd
# awk -F: '{username[++x]= $1} END{for(i=1;i<=x;i++) print i,username[i]} ' /etc/passwd
```

### ==按索引遍历==

```
# awk -F: '{username[x++]= $1} END{for(i in username) {print i,username[i]} } ' /etc/passwd
# awk -F: '{username[++x]= $1} END{for(i in username) {print i,username[i]} } ' /etc/passwd
```

### 练习:

#### 1. 统计/etc/passwd 中各种类型 shell 的数量

```
[root@tianyun ~]# awk -F: '{shells[$NF]++} END{ for(i in shells){print i,shells[i]} } ' /etc/passwd
```

#### 2. 网站访问状态统计 <当前时实状态 netstat>

```
[root@tianyun ~]# netstat -ant |grep :80 |awk '{access_stat[$NF]++} END{for(i in access_stat){print i,access_stat[i]}}
```

```
TIME_WAIT 1064
```

```
ESTABLISHED 1
```

```
LISTEN 1
```

```
[root@tianyun ~]# netstat -ant |grep :80 |awk '{access_stat[$NF]++} END{for(i in access_stat){print i,access_stat[i]}}' |sort -k2 -n |head
```

```
[root@tianyun ~]# ss -an |grep :80 |awk '{access_stat[$2]++} END{for(i in access_stat){print i,access_stat[i]}}
```

```
LISTEN 1
```

```
ESTAB 5
```

```
TIME-WAIT 97
```

```
[root@tianyun ~]# ss -an |grep :80 |awk '{access_stat[$2]++} END{for(i in access_stat){print i,access_stat[i]}}' |sort -k2 -rn
```

```
TIME-WAIT 18
```

```
ESTAB 8
```

```
LISTEN 1
```

#### 3. 统计当前访问的每个 IP 的数量 <当前时实状态 netstat,ss>

```
[root@tianyun ~]# netstat -ant |grep :80 |awk -F: '{ip_count[$8]++} END{for(i in ip_count){print i,ip_count[i]} }' |sort
```

```
172.16.130.16 289
```

```
172.16.130.33 254
```

```
172.16.130.44 158
```

```
172.16.130.99 4
```

```
[root@tianyun ~]# ss -an |grep :80 |awk -F":" '{!LISTEN/{ip_count[$NF-1]}++} END{for(i in ip_count){print i,ip_count[i]}}' |sort -k2 -rn |head
172.16.160.77 59
172.16.160.221 16
172.16.160.17 11
172.16.160.69 8
172.16.160.51 7
172.16.160.49 7
172.16.160.13 7
172.16.160.153 3
172.16.160.79 2
172.16.160.52 2
```

#### 4. 统计 Apache/Nginx 日志中某一天的 PV 量 <统计日志>

```
[root@tianyun nginx_log]# grep '07/Aug/2012' access.log |wc -l
14519
```

#### 5. 统计 Apache/Nginx 日志中某一天不同 IP 的访问量 <统计日志>

```
[root@tianyun nginx_log]# grep '07/Aug/2012' access.log |awk '{ips[$1]} END{for(i in ips){print i,ips[i]}}' |sort -k2 -rn |head
222.130.129.42 5761
123.126.51.94 988
123.126.68.22 588
123.114.46.141 418
61.135.249.218 368
110.75.173.162 330
110.75.173.163 327
110.75.173.161 321
110.75.173.160 319
110.75.173.164 314
```

```
[root@tianyun nginx_log]# grep '07/Aug/2012' access.log |awk '{ips[$1]} END{for(i in ips){print i,ips[i]}}' |awk '$2>100' |sort -k2 -rn
222.130.129.42 5761
123.126.51.94 988
123.126.68.22 588
123.114.46.141 418
61.135.249.218 368
110.75.173.162 330
110.75.173.163 327
110.75.173.161 321
110.75.173.160 319
110.75.173.164 314
```

```
1.202.218.67 313
110.75.173.159 311
203.208.60.80 294
221.221.207.202 266
203.208.60.82 230
203.208.60.81 209
38.111.147.83 206
61.135.249.220 187
183.39.187.86 178
61.156.142.207 129
```

思路：将需要统计的内容（某一个字段）作为数组  
的索引 ++

awk 函数 统计用户名为 4 个字符的用户：

```
[root@tianyun ~]# awk -F: '$1~/^....$/{count++; print $1} END{print "count is: " count}'
/etc/passwd
root
sync
halt
mail
news
uucp
nscd
vcsa
pcap
sshd
dbus
jack
count is: 12
```

```
[root@tianyun ~]# awk -F: 'length($1)==4{count++; print $1} END{print "count is: "count}'
/etc/passwd
root
sync
halt
mail
news
uucp
nscd
```

vcsa  
pcap  
sshd  
dbus  
jack  
count is: 12

### awk 自定义变量:

```
[root@tianyun ~]# awk -v user=root -F: '$1 == user' /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
[root@tianyun apache_log]# awk -v user=root -F: '$1 == "user" {print $0}' /etc/passwd
```

### 作业:

1. 取得网卡 IP (除 ipv6 以外的所有 IP)
2. 获得内存使用情况
3. 获得磁盘使用情况
4. 清空本机的 ARP 缓存
5. 打印出/etc/hosts 文件的最后一个字段 (按空格分隔)
6. 打印指定目录下的目录名

### 方法一:

```
[root@tianyun apache_log]# arp -n | awk '/^[0-9]/{print "arp -d "$1}'  
arp -d 172.16.100.10  
arp -d 172.16.100.178  
arp -d 172.16.100.208  
arp -d 172.16.100.49  
arp -d 172.16.100.250  
arp -d 172.16.100.127  
arp -d 172.16.100.11  
arp -d 172.16.100.148  
arp -d 172.16.100.128  
arp -d 172.16.100.59  
arp -d 172.16.100.183  
[root@tianyun apache_log]# arp -n | awk '/^[0-9]/{print "arp -d "$1}' | bash
```

### 方法二:

```
[root@tianyun apache_log]# arp -n | awk '/^[0-9]/{print $1}' | xargs -l {} arp -d {}
```

```
[root@tianyun ~]# awk -F: '{print $7}' /etc/passwd  
[root@tianyun ~]# awk -F: '{print $NF}' /etc/passwd  
[root@tianyun ~]# awk -F: '{print $(NF-1)}' /etc/passwd
```

```
[root@tianyun ~]# ll |grep '^d'
drwxr-xr-x 104 root root 12288 09-22 05:37 192.168.0.48
drwxr-xr-x 2 root root 4096 10-30 15:47 apache_log
drwxr-xr-x 2 root root 4096 10-30 15:23 awk
drwxr-xr-x 2 root root 4096 10-24 09:09 Desktop
drwxr-xr-x 12 root root 4096 10-08 06:12 LEMP_Soft
drwxr-xr-x 2 root root 4096 10-24 07:38 scripts
drwxr-xr-x 6 root root 4096 2012-03-29 uplayer
drwxr-xr-x 7 root root 4096 10-23 04:53 vmware
```

```
[root@tianyun ~]# ll |grep '^d' |awk '{print $NF}'
192.168.0.48
apache_log
awk
Desktop
LEMP_Soft
scripts
uplayer
vmware
```