Linux & unix shell 筆記 (正則表達式，grep, awk, sed)

第七章 正則表達式

* 表任意字符，但要使用時前面要加一個字母，表示該字母可以爲任意數，任意多個;它還有任意結果任意多次的意思，注意并不單指前面的字符任意多次

如：grep a* test.txt 或 grep a.* test

查找任意字母開頭的行 grep [a-zA-Z] test.txt grep ^ [a-zA-Z] test.txt

非字母開頭的 [^a-zA-Z]

非數字開頭的[^0-9]

空白行 grep ^$ test.txt

顯示非空白行 grep [^$] test.txt grep –v ^$ test.txt 中括號內加^意指否定，不匹配括號內的內容

本章设计的基本元字符使用在grep和sed命令中，同时结合{\\}（以字符出现情况进行匹配的元字符）使用在awk语言中。

表7-1 基本元字符集及其含义

^	只匹配行首
$	只匹配行尾
*	一个单字符后紧跟*，匹配0个或多个此单字符
[ ]	匹配[]内字符。可以是一个单字符，也可以是字符序列。可以使用 - 表示[]内字符序列范围，如用[1-5]代替[12345]
\	用来屏蔽一个元字符的特殊含义。因为有时在 shell中一些元字符有特殊含义。\可以使其失去应有意义
.	匹配任意单字符
pattern\{n\}	用来匹配前面pattern出现次数。n为次数
pattern\{n，\}m	含义同上，但次数最少为n
pattern\{n，m\}	含义同上，但pattern出现次数在n与m之间

7.4 使用*匹配字符串中的单字符或其重复序列

使用此特殊字符匹配任意字符或字符串的重复多次表达式。例如：

```
compu*t
```

将匹配字符u一次或多次：

computer
computing
compuuuuute

7.5 使用\屏蔽一个特殊字符的含义

有时需要查找一些字符或字符串，而它们包含了系统指定为特殊字符的一个字符。什么是特殊字符？一般意义上讲，下列字符可以认为是特殊字符：

$. ' " * [ ] ^ | ( ) \ + ?

假定要匹配包含字符“.”的各行而“,”代表匹配任意单字符的特殊字符，因此需要屏蔽其含义。操作如下：

```
\.
```

上述模式不认为反斜杠后面的字符是特殊字符，而是一个普通字符，即句点。

假定要匹配包含^的各行，将反斜杠放在它前面就可以屏蔽其特殊含义。如下：

```
\^
```

如果要在正则表达式中匹配以*.pas结尾的所有文件，可做如下操作：

```
\*\.pas
```

即可屏蔽字符*的特定含义。

如果熟知一个字符串匹配操作，应经常使用 []模式。

假定要匹配任意一个数字，可以使用：

```
[0123456789]
```

然而，通过使用“-”符号可以简化操作：

```
[0-9]
```

或任意小写字母

```
[a-z]
```

要匹配任意字母，则使用：

```
[A-Za-z]
```

表明从A-Z、a-z的字母范围。

如要匹配任意字母或数字，模式如下：

```
[A-Za-z0-9]
```

在字符序列结合使用中，可以用 []指出字符范围。假定要匹配一单词，以 s开头，中间有任意字母，以t结尾，那么操作如下：

```
s[a-z A-Z]t
```

7.7 使用\{\}匹配模式结果出现的次数

使用*可匹配所有匹配结果任意次，但如果只要指定次数，就应使用 \{\}，此模式有三种形式，即：

pattern\{n\} 匹配模式出现n次。

pattern\{n,\} 匹配模式出现最少n次。

pattern\{n,m} 匹配模式出现n到m次之间，n,m为0-255中任意整数。

请看第一个例子，匹配字母A出现两次，并以B结尾，操作如下：

```
A\{2\}B
```

匹配值为AAB

匹配A至少4次，使用：

```
A\{4,\}B
```

可以得结果AAAAB或AAAAAAAB，但不能为AAAB。

如给出出现次数范围，例如A出现2次到4次之间：

```
A\{2,4\}B
```

则结果为AAB、AAAB、AAAAB，而不是AB或AAAAAB等。

假定从下述列表中抽取代码：

1234XC9088
4523XX9001
0011XA9912

```
[0-9]\{4\}XX[0-9]\{4\}
```

具体含义如下：

1) 匹配数字出现4次。

2) 后跟代码xx。

3) 最后是数字出现4次。

结果为：

1234XC9088 – no match
4523XX9001 – match
0011XA9912 – no match
9931XC3445 – no match

模式 A 爲單個字符時可用，爲一組合字符時如：abc;查找多次時查找會出現問題

表7-2　经常使用的正则表达式举例

^	行首
$	行尾
^[the]	以the开头行
[Ss]igna[lL]	匹配单词signal、signaL、Signal、SignaL
[Ss]igna[lL]\.	同上，但加一句点
[mayMAY]	包含may大写或小写字母的行
^USER$	只包含USER的行
[tty]$	以tty结尾的行
\.	带句点的行
^d..x..x..x	对用户、用户组及其他用户组成员有可执行权限的目录
^[^l]	排除关联目录的目录列表
[.*0]	0之前或之后加任意字符
[000*]	000或更多个
[iI]	大写或小写I
[iI][nN]	大写或小写i或n
[^$]	空行
[^.*$]	匹配行中任意字符串
^.....$	包括6个字符的行
[a- zA-Z]	任意单字符
[a-z][a-z]*	至少一个小写字母
[^0-9\$]	非数字或美元标识
[^0-0A-Za-z]	非数字或字母
[123]	1到3中一个数字
[Dd]evice	单词device或Device
De..ce	前两个字母为De，后跟两个任意字符，最后为ce

\^q	以^q开始行
^.$	仅有一个字符的行
^\.[0-9][0-9]	以一个句点和两个数字开始的行
'"Device"'	单词device
De[Vv]ice\.	单词Device或device
[0-9]\{2\}-[0-9]\{2\}-[0-9]\{4\}	日期格式dd-mm-yyyy
[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}	IP地址格式nnn. nnn.nnn.nnn
[^.*$]	匹配任意行

grep 的使用

8.1.2 grep选项

常用的grep选项有：

-c 只输出匹配行的计数。

-i 不区分大小写（只适用于单字符）。

-h 查询多文件时不显示文件名。

-l 查询多文件时只输出包含匹配字符的文件名。

-n 显示匹配行及行号。

-s 不显示不存在或无匹配文本的错误信息。

-v 显示不包含匹配文本的所有行。

-o only match 只打印匹配的文本

-P perl regexp 使用 perl 表達式

8.1.3 查询多个文件

如果要在当前目录下所有.doc文件中查找字符串“sort”，方法如下：

```
$ grep "sort"*.doc
```

或在所有文件中查询单词“sort it”

```
$ grep "sort it" *
```

现在讲述在文本文件中grep选项的用法。

8.1.7 精确匹配

可能大家已注意到，在上一例中，抽取字符串“ 48”，返回结果包含诸如484和483等包含“48”的其他字符串，实际上应精确抽取只包含 48的各行。注意在每个匹配模式中抽取字符串后有一个<Tab>键，所以应操作如下：

```
$ grep "48<tab>" data.f
48      Dec     3BC1997  LPSX    68.00   LVX2A   138
```

<Tab>表示点击tab键。

使用grep抽取精确匹配的一种更有效方式是在抽取字符串后加 \>。假定现在精确抽取 48，方法如下：

```
$ grep '48\>' data.f
48      Dec     3BC1997  LPSX    68.00   LVX2A   138
```

8.2 grep和正则表达式

使用正则表达式使模式匹配加入一些规则，因此可以在抽取信息中加入更多选择。使用正则表达式时最好用单引号括起来，这样可以防止 grep中使用的专有模式与一些 shell命令的特殊方式相混淆。

8.2.1 模式范围

假定要抽取代码为484和483的城市位置，上一章中讲到可以使用 []来指定字符串范围，这里用48开始，以3或4结尾，这样抽出484或483。

```
$ grep '48[34]' data.f
483     Sept    5AP1996  USP     65.00   LVX2C   189
484     nov     7PL1996  CAD     49.00   PLV2C   234
483     may     5PA1998  USP     37.00   KVM9D   644
```

8.2.2 不匹配行首

如果要抽出记录，使其行首不是48，可以在方括号中使用^记号，表明查询在行首开始。

```
$ grep '^[^48]' data.f
219     dec     2CC1999 CAD     23.00   PLV2C    68
216     sept    3ZL1998 USP     86.00   KVM9E    234
```

8.2.3 设置大小写

使用-i开关可以屏蔽月份Sept的大小写敏感，也可以用另一种方式。这里使用 []模式抽取各行包含Sept和sept的所有信息。

```
$ grep '[Ss]ept' data.f
483     Sept    5AP1996 USP      65.00   LVX2C    189
216     sept    3ZL1998 USP      86.00   KVM9E    234
```

如果要抽取包含Sept的所有月份，不管其大小写，并且此行包含字符串 483，可以使用管道命令，即符号“|”左边命令的输出作为“|”右边命令的输入。举例如下：

```
$ grep '[Ss]ept' data.f | grep 483
483     Sept    5AP1996  USP     65.00   LVX2C    189
```

不必将文件名放在第二个grep命令中，因为其输入信息来自于第一个 grep命令的输出。

8.2.4 匹配任意字符

如果抽取以L开头，以D结尾的所有代码，可使用下述方法，因为已知代码长度为 5个字符：

```
$ grep 'K...D' data.f
47      Oct     3ZL1998  LPSX    43.00   KVM9D    512
483     may     5PA1998  USP     37.00   KVM9D    644
```

将上述代码做轻微改变，头两个是大写字母，中间两个任意，并以 C结尾：

```
$ grep '[A-Z][A-Z]..C' data.f
483     Sept    5AP1996  USP     65.00   LVX2C    189
```

查询包含1998的所有记录的另外一种方法是使用表达式 [0-9]\{3\}[8]，含义是任意数字重复3次，后跟数字8，虽然这个方法不像上一个方法那么精确，但也有一定作用。

```
$ grep '[0-9]\ {3\}[8]' data.f
47      Oct     3ZL1998  LPSX    43.00   KVM9D    512
483     may     5PA1998  USP     37.00   KVM9D    644
216     sept    3ZL1998  USP     86.00   KVM9E    234
```

8.2.7 模式出现机率

抽取包含数字4至少重复出现两次的所有行，方法如下：

```
$ grep '4\{2,\}' data.f
483     may     5PA1998  USP     37.00   KVM9D    644
```

上述语法指明数字4至少重复出现两次。

同样，抽取记录使之包含数字999（三个9），方法如下：

```
$ grep '9\{3,\}' data.f
219     dec     2CC1999  CAD     23.00   PLV2C     68
```

如果要查询重复出现次数一定的所有行，语法如下，数字 9重复出现两次：

```
$ grep '9\{2\}' data.f
```

有时要查询重复出现次数在一定范围内，比如数字或字母重复出现 2到6次，下例匹配数字8重复出现2到6次，并以3结尾：

```
$ grep '6\{2,6}3'  myfile
```

```
83              – no match
888883          – match
8884            – no match
```

8.2.8 使用grep匹配“与”或者“或”模式

grep命令加-E参数，这一扩展允许使用扩展模式匹配。例如，要抽取城市代码为 219或216，方法如下：

```
$ grep -E '219|216' data.f
219     dec     2CC1999  CAD     23.00   PLV2C    68
216     sept    3ZL1998  USP     86.00   KVM9E   234
```

grep –E 與 –e 選項的區別，在書寫上有差別，意思是一致的

grep –E ‘219|216’ data.f

grep –e ‘219 –e 216 data.f

egrep ‘219|216’ data.f

8.2.9 空行

结合使用^和$可查询空行。使用-n参数显示实际行数：

```
$ grep '^$' myfile
```

8.2.10 匹配特殊字符

查询有特殊含义的字符，诸如$. ' " * [] ^ | \ + ? .必须在特定字符前加\。假设要查询包含“.”的所有行，脚本如下：

```
$ grep '\.' myfile
```

或者是一个双引号：

```
$ grep '\"' myfile
```

以同样的方式，如要查询文件名conftroll.conf（这是一个配置文件），脚本如下：

```
$ grep 'conftroll\.conf' myfile
```

8.2.11 查询格式化文件名

使用正则表达式可匹配任意文件名。系统中对文本文件有其标准的命名格式。一般最多六个小写字符，后跟句点，接着是两个大写字符。例如，要在一个包含各类文件名的文件filename.deposit中定位这类文件名，方法如下：

```
$ grep '[^a-z]\{1,6/}\.[^A-Z]\{1,2/}' filename.deposit
yrend.AS        – match
mothdf          – nomatch
soa.PP          – match
qp.RR           – match
```

8.3 类名

grep允许使用国际字符模式匹配或匹配模式的类名形式。

表8-1 类名及其等价的正则表达式

类	等价的正则表达式	类	等价的正则表达式
[[:upper:]]	[A-Z]	[[:alnum:]]	[0-9a-zA-Z]
[[:lower:]]	[a-z]	[[:space:]]	空格或tab键
[[:digit:]]	[0-9]	[[:alpha:]]	[a-zA-Z]

现举例说明其使用方式。要抽取产品代码，该代码以 5开头，后跟至少两个大写字母。使用的脚本如下：

```
$ grep '5[[:upper:]][[:upper:]]' data.f
483      Sept    5AP1996  USP     65.00    LVX2C  189
483      may     5PA1998  USP     37.00    KVM9D  644
```

如果要抽取以P或D结尾的所有产品代码，方法如下：

```
$ grep '[[:upper:]][[:upper:]] [P,D]' data.f
483      Sept    5AP1996  USP     65.00    LVX2C  189
219      dec     2CC1999  CAD     23.00    PLV2C  68
484      nov     7PL1996  CAD     49.00    PLV2C  234
483      may     5PA1998  USP     37.00    KVM9D  644
216      sept    3ZL1998  USP     86.00    KVM9E  234
```

使用通配符*的匹配模式

使用通配符*的匹配模式

现在讲述grep中通配符*的使用。现有文件如下：

```
$ pg testfile
looks
likes
looker
long
```

下述grep模式结果显示如下：

```
$ grep 'l.*s' testfile
looks
likes

$ grep 'l.*k.' testfile
looks
likes

$ grep 'ooo*' testfile
looks
```

如在行尾查询某一单词，试如下模式：

```
$ grep 'device$' *
```

这将在所有文件中查询行尾包含单词 device的所有行。

8.4.1 目录

如果要查询目录列表中的目录，方法如下：

```
$ ls -l | grep '^d'
```

如果在一个目录中查询不包含目录的所有文件，方法如下：

```
$ ls -l | grep '^[^d]'
```

要查询其他用户和其他用户组成员有可执行权限的目录集合，方法如下：

```
$ ls -l | grep '^d.....x..x'
```

8.4.2 passwd文件

```
$ grep "louise" /etc/passwd
louise:lxAL6GW9G.ZyY:501:501:Accounts Sect 1C:/home/accts/louise:
/bin/sh
```

上述脚本查询/etc/passwd文件是否包含louise字符串，如果误输入以下脚本：

```
$ grep "louise" /etc/password
```

将返回grep命令错误代码'No such file or directory'。

上述结果表明输入文件名不存在，使用 grep命令-s开关，可屏蔽错误信息。

```
$ grep -s "louise" /etc/password
$
```

返回命令提示符，而没有文件不存在的错误提示。

如果grep命令不支持-s开关，可替代使用以下命令：

```
$ grep "louise" /etc/password >/dev/null 2>&1
```

脚本含义是匹配命令输出或错误（ 2>$1），并将结果输出到系统池。大多数系统管理员称/dev/null为比特池，没关系，可以将之看成一个无底洞，有进没有出，永远也不会填满。

上述两个例子并不算好，因为这里的目的只想知道查询是否成功。本书后面部分将讨论grep命令的exit用法，它允许查询并不成功返回。

如要保存grep命令的查询结果，可将命令输出重定向到一个文件。

```
$ grep "louise" /etc/passwd >/tmp/passwd.out
```

8.5 egrep

egrep代表expression或extended grep，适情况而定。egrep接受所有的正则表达式，egrep的一个显著特性是可以以一个文件作为保存的字符串，然后将之传给 egrep作为参数，为此使用-f开关。如果创建一个名为grepstrings的文件，并输入484和47：

```
$ pg grepstrings
484
47
$ egrep -f grepstrings data.f
```

上述脚本匹配data.f中包含484或47的所有记录。当匹配大量模式时，-f开关很有用，而在一个命令行中敲入这些模式显然极为繁琐。

如果要查询存储代码32L或2CC，可以使用（|）符号，意即“|”符号两边之一或全部。

```
$ egrep '(3ZL|2CC)' data.f
47      Oct     3ZL1998  LPSX    43.00   KVM9D   512
219     dec     2CC1999  CAD     23.00   PLV2C   68
216     sept    3ZL1998  USP     86.00   KVM9E   234
```

可以使用任意多竖线符“|”，例如要查看在系统中是否有帐号 louise、matty或pauline ，使用who命令并管道输出至egrep。

```
$ who | egrep (louise|matty|pauline)
 louise pty8
 matty tty02
pauline pty2
```

还可以使用^符号排除字符串。如果要查看系统上的用户，但不包括 matty和pauline，方法如下：

```
$ who | egrep -v '^(matty|pauline)'
```

如果要查询一个文件列表，包括 shutdown、shutdowns、reboot和reboots，使用egrep可容易地实现。

```
$ egrep '(shutdown | reboot) (s)?'  *
```

9.AWK 介紹

9.1 调用awk

有三种方式调用awk，第一种是命令行方式，如：

awk [-F field-separator] 'commands' input-file(s)

这里，commands是真正的awk命令。本章将经常使用这种方法。

上面例子中，[-F域分隔符]是可选的，因为awk使用空格作为缺省的域分隔符，因此如果要浏览域间有空格的文本，不必指定这个选项，但如果要浏览诸如 passwd文件，此文件各域以冒号作为分隔符，则必须指明 -F选项，如：

awk -F: 'commands' input-file

第二种方法是将所有 awk命令插入一个文件，并使 awk程序可执行，然后用 awk命令解释器作为脚本的首行，以便通过键入脚本名称来调用它。

第三种方式是将所有的awk命令插入一个单独文件，然后调用：

awk -f awk-script-file input-files(s)

-f选项指明在文件 awk_script_file中的awk脚本，input_file(s)是使用awk进行浏览的文件名。

9.2 awk脚本

在命令中调用awk时，awk脚本由各种操作和模式组成。

如果设置了-F选项，则awk每次读一条记录或一行，并使用指定的分隔符分隔指定域，但如果未设置-F选项，awk假定空格为域分隔符，并保持这个设置直到发现一新行。当新行出现时，awk命令获悉已读完整条记录，然后在下一个记录启动读命令，这个读进程将持续到文件尾或文件不再存在。

参照表9-1，awk每次在文件中读一行，找到域分隔符（这里是符号 #），设置其为域n，直至一新行（这里是缺省记录分隔符），然后，划分这一行作为一条记录，接着 awk再次启动下一行读进程。

表9-1 awk读文件记录的方式

域1	分隔符	域2	分隔符	域3	分隔符	域4及换行
P.Bunny(记录1)	#	02/99	#	48	#	Yellow \n
J.Troll(记录2)	#	07/99	#	4842	#	Brown-3 \n

9.2.1 模式和动作

任何awk语句都由模式和动作组成。在一个 awk脚本中可能有许多语句。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件开始执行。 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标志。如果不特别指明模式， awk总是匹配或打印行数。

实际动作在大括号{}内指明。动作大多数用来打印，但是还有些更长的代码诸如 if和循环(looping) 语句及循环退出结构。如果不指明采取动作， awk将打印出所有浏览出来的记录。

下面将深入讲解这些模式和动作。

9.2.2 域和记录

awk执行时，其浏览域标记为$1，$2...$n。这种方法称为域标识。使用这些域标识将更容易对域进行进一步处理。

使用$1,$3表示参照第1和第3域，注意这里用逗号做域分隔。如果希望打印一个有 5个域的记录的所有域，不必指明 $1,$2,$3,$4,$5，可使用$0，意即所有域。Awk浏览时，到达一新行，即假定到达包含域的记录末尾，然后执行新记录下一行的读动作，并重新设置域分隔。注意执行时不要混淆符号$和shell提示符$，它们是不同的。

为打印一个域或所有域，使用print命令。这是一个awk动作（动作语法用圆括号括起来）。

1. 抽取域

真正执行前看几个例子，现有一文本文件 grade.txt，记录了一个称为柔道数据库的行信息。

```
$ pg grade.txt
M.Tansley    05/99     48311  Green     8    40    44
J.Lulu       06/99     48317  green     9    24    26
P.Bunny      02/99     48     Yellow    12   35    28
J.Troll      07/99     4842   Brown-3   12   26    26
L.Tansley    05/99     4712   Brown-2   12   30    28
```

此文本文件有7个域，即（1）名字、（2）升段日期、（3）学生序号、（4）腰带级别、（5）年龄、（6）目前比赛积分、（7）比赛最高分。

因为域间使用空格作为域分隔符，故不必用 -F选项划分域，现浏览文件并导出一些数据。

在例子中为了利于显示，将空格加宽使各域看得更清晰。

2. 保存awk输出

有两种方式保存 shell提示符下awk脚本的输出。最简单的方式是使用输出重定向符号 >文件名，下面的例子重定向输出到文件 wow。

```
$ awk '{print $0}' grade.txt >wow
```

使用这种方法要注意，显示屏上不会显示输出结果。因为它直接输出到文件。只有在保证输出结果正确时才会使用这种方法。它也会重写硬盘上同名数据。

第二种方法是使用tee命令，在输出到文件的同时输出到屏幕。在测试输出结果正确与否时多使用这种方法。例如输出重定向到文件 delete_me_and_die，同时输出到屏幕。使用这种方法，在awk命令结尾写入 | tee delete_me_and_die。

```
$ awk '{print $0}' grade.txt | tee delete_me_and_die
```

3. 使用标准输入

在深入讲解这一章之前，先对 awk脚本的输入方法简要介绍一下。实际上任何脚本都是从标准输入中接受输入的。为运行本章脚本，使用 awk脚本输入文件格式，例如：

```
$ belts.awk grade_student.txt
```

也可替代使用下述格式：

使用重定向方法：

```
$ belts.awk < grade2.txt
```

或管道方法：

```
$ grade2.txt|belts.awk
```

4. 打印所有记录

```
$ awk '{print $0}' grade.txt
```

awk读每一条记录。因为没有模式部分，只有动作部分 {print $0}(打印所有记录)，这个动作必须用花括号括起来。上述命令打印整个文件。

```
M.Tansley    05/99     48311  Green     8    40    44
J.Lulu       06/99     48317  green     9    24    26
```

5. 打印单独记录

假定只打印学生名字和腰带级别，通过查看域所在列，可知为 field-1和field-4，因此可以使用$1和$4，但不要忘了加逗号以分隔域。

```
$ awk  '{print $1,$4}' grade.txt
M.Tansley   Green
J.Lulu      green
P.Bunny     Yellow
J.Troll     Brown-3
L.Tansley   Brown-2
```

6. 打印报告头

上述命令输出在名字和腰带级别之间用一些空格使之更容易划分，也可以在域间使用 tab 键加以划分。为加入tab键，使用tab键速记引用符 \t，后面将对速记引用加以详细讨论。也可以为输出文本加入信息头。本例中加入 name和belt及下划线。下划线使用 \n，强迫启动新行，并在\n下一行启动打印文本操作。打印信息头放置在 BEGIN模式部分，因为打印信息头被界定为一个动作，必须用大括号括起来。在 awk查看第一条记录前，信息头被打印。

```
$ awk  'BEGIN {print "Name    Belt\n-------------------------------"}
{print $1"\t"$4}' grade.txt

Name            Belt
--------------------------------
M.Tansley       Green
J.Lulu          green
P.Bunny         Yellow
J.Troll         Brown-3
```

7. 打印信息尾

如果在末行加入end of report信息，可使用END语句。END语句在所有文本处理动作执行完之后才被执行。 END语句在脚本中的位置放置在主要动作之后。下面简单打印头信息并告之查询动作完成。

```
$ awk 'BEGIN {print "Name\n-------"} {print $1} END {"end-of-report"}'
grade.txt
Name
-------
M.Tansley
J.Lulu
P.Bunny
J.Troll
L.Tansley
end-of-report
```

awk 'BEGIN {print "Name\n----------"}{print $1} END {print "end-of-report"}' grade.txt

awk ‘模式{動作}模式{動作}模式{動作}’

8. awk错误信息提示

几乎可以肯定，在使用awk时，将会在命令中碰到一些错误。 awk将试图打印错误行，但由于大部分命令都只在一行，因此帮助不大。

系统给出的显示错误信息提示可读性不好。使用上述例子，如果丢了一个双引号， awk将返回：

```
$ awk 'BEGIN {print "Name\n-------} {print $1} END {"end-of-report"}''
grade.txt

awk: cmd. line:1: BEGIN {print "Name\n-------} {print $1} END
```

```
{"end-of -report"}
awk: cmd. line:1: ^ unterminated string
```

当第一次使用awk时，可能被错误信息搅得不知所措，但通过长时间和不断的学习，可总结出以下规则。在碰到awk错误时，可相应查找：

- 确保整个awk命令用单引号括起来。
- 确保命令内所有引号成对出现。
- 确保用花括号括起动作语句，用圆括号括起条件语句。
- 可能忘记使用花括号，也许你认为没有必要，但 awk不这样认为，将按之解释语法。

如果查询文件不存在，将得到下述错误信息：

```
$ awk 'END {print NR}' grades.txt

awk: cmd. line:2: fatal: cannot open file 'grades.txt' for
reading (No such file or directory)
```

如果查询文件不存在，将得到下述错误信息：

```
$ awk 'END {print NR}' grades.txt

awk: cmd. line:2: fatal: cannot open file 'grades.txt' for
reading (No such file or directory)
```

9. awk键盘输入

如果在命令行并没有输入文件 grade.txt，将会怎样？

```
$ awk 'BEGIN {print "Name  Belt\n-------------------------------"}
      {print $1"\t"$4}'

Name              Belt
--------------------------------
>
```

BEGIN部分打印了文件头，但awk最终停止操作并等待，并没有返回 shell提示符。这是因为awk期望获得键盘输入。因为没有给出输入文件， awk假定下面将会给出。如果愿意，顺序输入相关文本，并在输入完成后敲 <Ctrl-D>键。如果敲入了正确的域分隔符， awk会像第一个例子一样正常处理文本。这种处理并不常用，因为它大多应用于大量的打印稿。

==========

9.2.3 awk中正则表达式及其操作

在grep一章中，有许多例子用到正则表达式，这里将不使用同样的例子，但可以使用条件操作讲述awk中正则表达式的用法。

这里正则表达式用斜线括起来。例如，在文本文件中查询字符串 Green，使用/Green/可以查出单词Green的出现情况。

9.2.4 元字符

这里是awk中正则表达式匹配操作中经常用到的字符，详细情况请参阅本书第 7章正则表达式概述。

```
\ ^ $ . [] | () * + ?
```

这里有两个字符第7章没有讲到，因为它们只适用于awk而不适用于grep或sed。它们是：

+ 使用+匹配一个或多个字符。

? 匹配模式出现频率。例如使用/XY?Z/匹配XYZ或YZ。

9.2.5 条件操作符

表9-2给出awk条件操作符，后面将给出其用法。

表9-2 awk条件操作符

操 作 符	描 述	操 作 符	描 述
<	小于	>=	大于等于
<=	小于等于	~	匹配正则表达式
==	等于	!~	不匹配正则表达式
!=	不等于		

1. 匹配

为使一域号匹配正则表达式，使用符号‘～’后紧跟正则表达式，也可以用 if语句。awk中if后面的条件用（）括起来。

观察文件grade.txt，如果只要打印brown腰带级别可知其所在域为field-4，这样可以写出表达式{if($4~/brown/) print }意即如果field-4包含brown，打印它。如果条件满足，则打印匹配记录行。可以编写下面脚本，因为这是一个动作，必须用花括号 {}括起来。

```
$ awk  {if($4~/Brown/) print $0}' grade.txt
J.Troll     07/99   4842    Brown-3   12    26      26
L.Tansley   05/99   4712    Brown-2   12    30      28
```

匹配记录找到时，如果不特别声明， awk缺省打印整条记录。使用if语句开始有点难，但不要着急，因为有许多方法可以跳过它，并仍保持同样结果。下面例子意即如果记录包含模式brown，就打印它：

```
$ awk  '$0 ~ /Brown/' grade.txt
J.Troll     07/99     4842    Brown-3   12    26      26
L.Tansley   05/99     4712    Brown-2   12    30      28
```

awk ‘{if($4~/Brown/) print $0}’ grade.txt

/Brown/ 爲下則表達式在 awk 中的使用方法

awk ‘$0~/Brown/’ grade.txt 匹配模式后，默認會打印輸出符合的所有記錄；

awk 命令可以只有模式，或只有動作，很靈活

匹配记录找到时，如果不特别声明， awk缺省打印整条记录。使用if语句开始有点难，但不要着急，因为有许多方法可以跳过它，并仍保持同样结果。下面例子意即如果记录包含模式brown，就打印它：

```
$ awk  '$0 ~ /Brown/' grade.txt
J.Troll     07/99     4842   Brown-3  12    26      26
L.Tansley   05/99     4712   Brown-2  12    30      28
```

2. 精确匹配

假定要使字符串精确匹配，比如说查看学生序号 48，文件中有许多学生序号包含 48，如果在field-3中查询序号48，awk将返回所有序号带48的记录：

```
$ awk  '{if($3~/48/) print $0}' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
P.Bunny     02/99     48     Yellow   12    35    28
J.Troll     07/99     4842   Brown-3  12    26    26
```

为精确匹配48，使用等号==，并用单引号括起条件。例如 $3==“48”，这样确保只有 48 序号得以匹配，其余则不行。

```
$ awk '$3=="48" {print $0}' grade.txt
P.Bunny     02/99     48     Yellow   12    35    28
```

awk ‘$3==”48” {print $0}’ grade.txt

awk ‘模式 {動作}’ 文件(輸入)　　awk 命令的標准使用方法

精確匹配用==，模糊的，使用與正則表達式匹配用~,/內容/

3. 不匹配

有时要浏览信息并抽取不匹配操作的记录，与 ~相反的符号是!~，意即不匹配。像原来使用查询brown腰带级别的匹配操作一样，现在看看不匹配情况。表达式 $0 !~/brown/，意即查询不包含模式brown腰带级别的记录并打印它。

注意，缺省情况下，awk将打印所有匹配记录，因此这里不必加入动作部分。

```
$ awk  '$0 !~ /Brown/' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
P.Bunny     02/99     48     Yellow   12    35    28
```

可以只对field-4进行不匹配操作，方法如下：

```
$ awk '{if($4!~/Brown/) print $0}' grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
P.Bunny     02/99     48     Yellow   12    35    28
```

如果只使用命令awk$4 !="brown"{print $0} grade.txt，将返回错误结果，因为用引号括起了brown，将只匹配‘brown而不匹配brown-2和brown-3，当然，如果想要查询非brown-2的腰带级别，可做如下操作：

```
$ awk '$4 != "Brown-2" {print $0}' grade.txt
```

!~，!=，~，==　模糊與精確匹配操作符

4. 小于

看看哪些学生可以获得升段机会。测试这一点即判断目前级别分 field-6是否小于最高分field-7，在输出结果中，加入这一改动很容易。

```
$ awk '{if ($6 < $7) print $0 "$1 Try better at the next comp"}'
grade.txt
M.Tansley Try better at the next comp
J.Lulu Try better at the next comp
```

5. 小于等于

对比小于，小于等于只在操作符上做些小改动，满足此条件的记录也包括上面例子中的输出情况。

```
$ awk '{if ($6 <= $7) print $1}' grade.txt
M.Tansley
J.Lulu
J.Troll
```

6. 大于

大于符号大家都熟知，请看例子：

```
$ awk '{if ($6 > $7) print $1}' grade.txt
L.Tansley
P.Bunny
```

7. 设置大小写

为查询大小写信息，可使用[]符号。在测试正则表达式时提到可匹配[]内任意字符或单词，因此若查询文件中级别为green的所有记录，不论其大小写，表达式应为‘/[Gg]reen/:’

```
$ awk '/[Gg]reen/' grade.txt
M.Tansley   05/99     48311  Green    8    40    44
J.Lulu      06/99     48317  green    9    24    26
```

8. 任意字符

抽取名字，其记录第一域的第四个字符是 a，使用句点.。表达式/^...a/意为行首前三个字符任意，第四个是a，尖角符号代表行首。

```
$ awk '$1 ~/^ ... a/' grade.txt
M.Tansley   05/99     48311  Green    8     40   44
L.Tansley   05/99     4712   Brown-2  12    30   28
```

9. 或关系匹配

为抽取级别为yellow或brown的记录，使用竖线符 |。意为匹配｜两边模式之一。注意，使用竖线符时，语句必须用圆括号括起来。

```
$ awk '$0~/(Yellow|Brown)/' grade.txt
P.Bunny     02/99    48      Yellow   12   35    28
J.Troll     07/99    4842    Brown-3  12   26    26
L.Tansley   05/99    4712    Brown-2  12   30    28
```

上面例子输出所有级别为Yellow或Brown的记录。

使用这种方法在查询级别为Green或green时，可以得到与使用[]表达式相同的结果。

```
$ awk '$0~/(Green|green)/' grade.txt
M.Tansley   05/99    48311   Green    8     40   44
J.Lulu      06/99    48317   green    9     24   26
```

10. 行首

不必总是使用域号。如果查询文本文件行首包含 48的代码，可简单使用下面 ^符号：

```
$ awk '/^48/' input-file
```

这里讲述了在awk中怎样使用第7章中涉及的表达式。像第 7章的开头提到的，所有表达式(除字符重复出现外）在awk中都是合法的。

復合模式，使用邏輯語句

复合模式或复合操作符用于形成复杂的逻辑操作，复杂程度取决于编程者本人。有必要了解的是，复合表达式即为模式间通过使用下述各表达式互相结合起来的表达式：

&& AND：语句两边必须同时匹配为真。

|| OR：语句两边同时或其中一边匹配为真。

！ 非 求逆

11. AND

打印记录，使其名字为 ‘ P.Bunny且级别为 Yellow，使用表达式 ($1=="P.Bunny"&& $4=="Yellow")，意为&&两边匹配均为真。完整命令如下：

```
$ awk '{if ($1=="P.Bunny" && $4=="Yellow")print $0}' grade.txt
P.Bunny     02/99     48     Yellow   12    35   28
```

12. Or

如果查询级别为Yellow或Brown，使用或命令。意为“||”符号两边的匹配模式之一或全部为真。

```
$ awk '{if ($4=="Yellow" || $4~/Brown/) print $0}' grade.txt
P.Bunny     02/99     48       Yellow  12    35   28
J.Troll     07/99     4842     Brown-3 12    26   26
L.Tansley   05/99     4712     Brown-2 12    30   28
```

兩個條件間單獨考慮，“||”與“|”符合比較起來“||”可以使兩個條件分別精確或模糊匹配，“|”確只能用在同條件下。

---

awk 內置變量的使用，比較難記

9.2.6 awk内置变量

awk有许多内置变量用来设置环境信息。这些变量可以被改变。表 9-3显示了最常使用的一些变量，并给出其基本含义。

表9-3 awk内置变量

ARGC	命令行参数个数
ARGV	命令行参数排列
ENVIRON	支持队列中系统环境变量的使用
FILENAME	awk浏览的文件名
FNR	浏览文件的记录数
FS	设置输入域分隔符，等价于命令行 -F选项

NF	浏览记录的域个数
NR	已读的记录数
OFS	输出域分隔符
ORS	输出记录分隔符
RS	控制记录分隔符

ARGC支持命令行中传入awk脚本的参数个数。ARGV是ARGC的参数排列数组，其中每一元素表示为ARGV[n]，n为期望访问的命令行参数。

ENVIRON支持系统设置的环境变量，要访问单独变量，使用实际变量名，例如ENVIRON[“EDITOR”]=“Vi”。

FILENAME支持awk脚本实际操作的输入文件。因为awk可以同时处理许多文件，因此如果访问了这个变量，将告之系统目前正在浏览的实际文件。

FNR支持awk目前操作的记录数。其变量值小于等于 NR。如果脚本正在访问许多文件，每一新输入文件都将重新设置此变量。

FS用来在awk中设置域分隔符，与命令行中 -F选项功能相同。缺省情况下为空格。如果用逗号来作域分隔符，设置FS=",”。

NF支持记录域个数，在记录被读之后再设置。

OFS允许指定输出域分隔符，缺省为空格。如果想设置为 #，写入OFS="#"。

ORS为输出记录分隔符，缺省为新行（\n)。

RS是记录分隔符，缺省为新行(\n)。

9.2.7 NF、NR和FILENAME

下面看一看awk内置变量的例子。

要快速查看记录个数，应使用 NR。比如说导出一个数据库文件后，如果想快速浏览记录个数，以便对比于其初始状态，查出导出过程中出现的错误。使用 NR将打印输入文件的记录个数。print NR放在END语法中。

```
$ awk 'END {print NR}' grade.txt
```

以下例子中，所有学生记录被打印，并带有其记录号。使用 NF变量显示每一条读记录中有多少个域，并在END部分打印输入文件名。

```
$ awk '{print NF,NR,$0}END{print FILENAME}' grade.txt
7  1 M.Tansley  05/99   48311   Green    8       40      44
7  2 J.Lulu     06/99   48317   green    9       24      26
7  3 P.Bunny    02/99   48      Yellow   12      35      28
7  4 J.Troll    07/99   4842    Brown-3  12      26      26
7  5 L.Tansley  05/99   4712    Brown-2  12      30      28
grade.txt
```

在从文件中抽取信息时，最好首先检查文件中是否有记录。下面的例子只有在文件中至少有一个记录时才查询 Brown级别记录。使用AND复合语句实现这一功能。意即至少存在一个记录后，查询字符串Brown，最后打印结果。

```
$ awk '{if (NR >0 && $4~/Brown/)print $0}' grade.txt
J.Troll      07/99    4842   Brown-3  12   26      26
L.Tansley    05/99    4712   Brown-2  12   30      28
```

NF 的實際應用實例爲輸出文件名

NF的一个强大功能是将变量$PWD的返回值传入awk并显示其目录。这里需要指定域分隔符/。

```
$ pwd
/usr/local/etc
$ echo $PWD | awk -F/ '{print $NF}'
etc
```

另一个例子是显示文件名。

```
$ echo "/usr/local/etc/rc.sybase" | awk -F/ '{print $NF}'
rc.sybase
```

注意：輸出 NF 與 $NF 是不同的意思； NF 指域的數量，$NF 則爲每條記錄的最后一個域值

9.2.8 awk操作符

在awk中使用操作符，基本表达式可以划分为数字型、字符串型、变量型、域及数组元素，前面已经讲过一些。下面列出其完整列表。

在表达式中可以使用下述任何一种操作符。

= += *= / = %= ^ =	赋值操作符
?	条件表达操作符
|| && !	并、与、非（上一节已讲到）
~ !~	匹配操作符，包括匹配和不匹配
< <= == != >>	关系操作符
+ - * / % ^	算术操作符
+ + --	前缀和后缀

前面已经讲到了其中几种操作，下面继续讲述未涉及的部分。

1. 设置输入域到域变量名

在awk中，设置有意义的域名是一种好习惯，在进行模式匹配或关系操作时更容易理解。一般的变量名设置方式为name=$n，这里name为调用的域变量名，n为实际域号。例如设置学生域名为name，级别域名为belt，操作为name=$1;belts=$4。注意分号的使用，它分隔awk命令。下面例子中，重新赋值学生名域为 name，级别域为belts。查询级别为Yellow的记录，并最终打印名称和级别。

```
$ awk '{name=$1;belts=$4; if(belts ~/Yellow/)print name" is belt
"belts}' grade.txt
P.Bunny is belt Yellow
```

2. 域值比较操作

有两种方式测试一数值域是否小于另一数值域。

1) 在BEGIN中给变量名赋值。

2) 在关系操作中使用实际数值。

通常在BEGIN部分赋值是很有益的，可以在awk表达式进行改动时减少很多麻烦。使用关系操作必须用圆括号括起来。

下面的例子查询所有比赛中得分在27点以下的学生。

用引号将数字引用起来是可选的，“27”、27产生同样的结果。

```
$ awk '{if($6 < 27)print$0}' grade.txt
J.Lulu      06/99   48317   green   9     24      26
J.Troll     07/99   4842    Brown-3 12    26      26
```

關系操作，條件語句必須用圓括號括起來；關系操作中可用實際數值

把兩條語句放一起執行用花括號括起來，中間用分號分開(兩條命令)

逗號用于分開同不同的變量

第二个例子中给数字赋以变量名BASELINE和在BEGIN部分给变量赋值，两者意义相同。

```
$ awk 'BEGIN {BASELINE="27  "}{if($6 < BASELINE)print$0}'  grade.txt
J.Lulu      06/99   48317   green   9     24      26
J.Troll     07/99   4842    Brown-3 12    26      26
```

3. 修改数值域取值

当在awk中修改任何域时，重要的一点是要记住实际输入文件是不可修改的，修改的只是保存在缓存里的awk复本。awk会在变量NR或NF变量中反映出修改痕迹。

为修改数值域，简单的给域标识重赋新值，如： $1=$1+5，会将域1数值加5，但要确保赋值域其子集为数值型。

修改M.Tansley的目前级别分域，使其数值从 40减为39，使用赋值语句 $6=$6-1，当然在实施修改前首先要匹配域名。

```
$ awk '{if($1=="M.Tansley") $6=$6-1; print $1, $6, $7}' grade.txt
M.Tansley   39  44
J.Lulu      24  26
P.Bunny     35  28
J.Troll     26  26
L.Tansley   30  28
```

4. 修改文本域

修改文本域即对其重新赋值。需要做的就是赋给一个新的字符串。在 J.Troll中加入字母，使其成为J.L.Troll，表达式为$1="J.L.Troll"，记住字符串要使用双秒号（""），并用圆括号括起整个语法。

```
$ awk '{if($1=="J.Troll") ($1="J.L.Troll"); print $1}' grade.txt
M.Tansley
J.Lulu
P.Bunny
J.L.Troll
L.Tansley
```

5. 只显示修改记录

上述例子均是对一个小文件的域进行修改，因此打印出所有记录查看修改部分不成问题，但如果文件很大，记录甚至超过 100，打印所有记录只为查看修改部分显然不合情理。在模式后面使用花括号将只打印修改部分。取得模式，再根据模式结果实施操作，可能有些抽象，现举一例，只打印修改部分。注意花括号的位置。

```
$ awk '{if($1=="J.Troll") {$1="J.L.Troll";print $1}}' grade.txt
J.L.Troll
```

注意：1> awk ‘{if($1==”J.Troll”) ($1=”J.L.Troll”); print $1}’ grade.txt

2> awk ‘{if($1==”J.Troll”) {$1=”J.L.Troll” print $1}}’ grade.txt

兩個命令花括號的位置，及分號。兩個命令表達的含義是不同的

1> 中 if($1==”J.Troll”) ($1=”J.L.Troll”);爲一個模式，print $1 爲動作

2> 中 if($1==”J.Troll”)爲模式，{$1=”J.L.Troll” print $1}}爲動作

6. 创建新的输出域

在awk中处理数据时，基于各域进行计算时创建新域是一种好习惯。创建新域要通过其他域赋予新域标识符。如创建一个基于其他域的加法新域{$4=$2+$3}，这里假定记录包含3个域，则域4为新建域，保存域2和域3相加结果。

在文件grade.txt中创建新域8保存域目前级别分与域最高级别分的减法值。表达式为‘{$8=$7-$6}’，语法首先测试域目前级别分小于域最高级别分。新域因此只打印其值大于零的学生名称及其新域值。在BEGIN部分加入tab键以对齐报告头。

```
$ awk 'BEGIN{ print "Name\t Difference"}{if($6 <$7) {$8=$7--$6; print
$1,$8}}' grade.txt
Name        Difference
M.Tansley    4
J.Lulu       2
```

当然可以创建新域，并赋给其更有意义的变量名。例如:

```
$ awk 'BEGIN{ print "Name\t Difference"}{if($6 <$7) {diff=$7-$6; print
$1,diff}}' grade.txt
M.Tansley    4
J.Lulu       2
```

7. 增加列值

为增加列数或进行运行结果统计，使用符号 +=。增加的结果赋给符号左边变量值，增加到变量的域在符号右边。例如将 $1加入变量total，表达式为total+=$1。列值增加很有用。许多文件都要求统计总数，但输出其统计结果十分繁琐。在 awk中这很简单，请看下面的例子。

将所有学生的‘目前级别分’加在一起，方法是 tot+=$6，tot即为awk浏览的整个文件的域6结果总和。所有记录读完后，在 END部分加入一些提示信息及域 6总和。不必在awk中显示说明打印所有记录，每一个操作匹配时，这是缺省动作。

```
$ awk '(tot+=$6); END{print "Club student total points :" tot}'
grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
P.Bunny     02/99     48     Yellow   12    35    28
J.Troll     07/99     4842   Brown-3  12    26    26
L.Tansley   05/99     4712   Brown-2  12    30    28
Club student total points :155
```

如果文件很大，你只想打印结果部分而不是所有记录，在语句的外面加上圆括号（）即可。

```
$ awk '{(tot+=$6)}; END{print "Club student total points :" tot}'
grade.txt
Club student total points :155
```

“+=”用于統計列值

8. 文件长度相加

在目录中查看文件时，如果想快速查看所有文件的长度及其总和，但要排除子目录，使用ls -l命令，然后管道输出到awk，awk首先剔除首字符为d（使用正则表达式）的记录，然后将文件长度列相加，并输出每一文件长度及在 END部分输出所有文件的长度。

本例中，首先用ls -l命令查看一下文件属性。注意第二个文件属性首字符为 d，说明它是一个目录，文件长度是第 5列，文件名是第 9列。如果系统不是这样排列文件名及其长度，应适时加以改变。

```
-rw-r--r--  1 root  root    80  Apr 11 18:56 acc.txt
drwx------  2 root  root  1024  Mar 26 20:53 nsmail
Columns     1       2 3     4     5   6  7  8     9
```

下面的正则表达式表明必须匹配行首，并排除字符 d，表达式为^[^d]。

使用此模式打印文件名及其长度，然后将各长度相加放入变量 tot中。

```
$ ls -l | awk ' /^[^d]/ {print $9"\t"$5} {tot+=$5} END
{print "total KB:"tot}'
dev_pkg.fail    345
failedlogin     12416
messages        4260
sulog           12810
```

9.2.9 内置的字符串函数

awk有许多强大的字符串函数，见表 9-4。

表9-4 awk内置字符串函数

gsub(r,s)	在整个$0中用s替代r
gsub(r,s,t)	在整个t中用s替代r
index(s,t)	返回s中字符串t的第一位置
length(s)	返回s长度
match(s,r)	测试s是否包含匹配r的字符串
split(s,a,fs)	在fs上将s分成序列a
sprint(fmt,exp)	返回经fmt格式化后的exp
sub(r,s)	用$0中最左边最长的子串代替s
substr(s,p)	返回字符串s中从p开始的后缀部分
substr(s,p,n)	返回字符串s中从p开始长度为n的后缀部分

gsub函数有点类似于 sed查找和替换。它允许替换一个字符串或字符为另一个字符串或字符，并以正则表达式的形式执行。第一个函数作用于记录 $0，第二个gsub函数允许指定目标，然而，如果未指定目标，缺省为 $0。

1. gsub

要在整个记录中替换一个字符串为另一个，使用正则表达式格式， /目标模式/，替换模式/。例如改变学生序号4842到4899：

```
$ awk 'gsub(/4842/,4899) {print $0}' grade.txt
J.Troll     07/99     4899   Brown-3  12    26    26
```

sub 替代

gsub(r,s,t) 指定在 t 中查找，用 s 替代 r

awk ‘gsub(/4832/,4899,$3) {print $0}’ grade.txt

gsub 查找所有的全部替换， sub 只查找替换匹配的第一個

index（s，t）函数返回目标字符串s中查询字符串t的首位置。length函数返回字符串s字符长度。match函数测试字符串s是否包含一个正则表达式r定义的匹配。split使用域分隔符fs将字符串s划分为指定序列a。sprint函数类似于printf函数（以后涉及），返回基本输出格式fmt的结果字符串exp。sub（r，s）函数将用s替代$0中最左边最长的子串，该子串被（r）匹配。sub（s，p）返回字符串s在位置p后的后缀。substr（s，p，n）同上，并指定子串长度为n。

2. index

查询字符串s中t出现的第一位置。必须用双引号将字符串括起来。例如返回目标字符串Bunny中ny出现的第一位置，即字符个数。

```
$ awk 'BEGIN {print index("Bunny","ny")}' grade.txt
4
```

3. length

返回所需字符串长度，例如检验字符串 J.Troll返回名字及其长度，即人名构成的字符个数。

```
$ awk '$1=="J.Troll" {print length($1)" "$1}' grade.txt
7 J.Troll
```

还有一种方法，这里字符串加双引号。

```
$ awk 'BEGIN {print length("A FEW GOOD MEN")}'
14
```

4. match

match测试目标字符串是否包含查找字符的一部分。可以对查找部分使用正则表达式，返回值为成功出现的字符排列数。如果未找到，返回 0，第一个例子在ANCD中查找d。因其不存在，所以返回0。第二个例子在ANCD中查找D。因其存在，所以返回ANCD中D出现的首位置字符数。第三个例子在学生J.Lulu中查找u。

```
$ awk 'BEGIN {print match("ANCD",/d/)}'
0
$ awk 'BEGIN {print match("ANCD",/C/)}'
3
$ awk '$1=="J.Lulu" {print match($1,"u")}' grade.txt
4
```

5. split

使用split返回字符串数组元素个数。工作方式如下：如果有一字符串，包含一指定分隔符 - ，例如AD2-KP9-JU2-LP-1，将之划分成一个数组。使用 split，指定分隔符及数组名。此例中，命令格式为("AD2-KP9-JU2-LP-1"，parts_array，"-")，split然后返回数组下标数，这里结果为4。

还有一个例子使用不同的分隔符。

```
$ awk 'BEGIN {print split("123#456#678", myarray, "#")}'
3
```

这个例子中，split返回数组myarray的下标数。数组myarray取值如下：

Myarray[1]="123"
Myarray[2]="456"
Myarray[3]="678"

6. sub

使用sub发现并替换模式的第一次出现位置。字符串 STR包含‘poped popo pill’，执行下列sub命令sub（/op/，"op"，STR）。模式op第一次出现时，进行替换操作，返回结果如下：‘pOPed pope pill’。

本章文本文件中，学生J.Troll的记录有两个值一样，“目前级别分”与“最高级别分”。只改变第一个为29，第二个仍为24不动，操作命令为sub（/26/，"29"，$0），只替换第一个出现24的位置。注意J.Troll记录需存在。

```
$ awk '$1=="J.Troll" sub(/26/,"29",$0)' grade.txt
M.Tansley   05/99     48311   Green     8     40    44
J.Lulu      06/99     48317   green     9     24    29
P.Bunny     02/99     48      Yellow    12    35    28
J.Troll     07/99     4842    Brown-3   12    29    26
L.Tansley   05/99     4712    Brown-2   12    30    28
```

awk '$1=="J.Troll" sub(/26/,"29",$0); {print $0}' grade.txt

awk '$1=="J.Troll" sub(/26/,"29"); {print $0}' grade.txt

sub (/op/,”OP”,STR)　　，sub(/op/,”OP”)模式　是一樣的輸出結果

7. substr

substr是一个很有用的函数。它按照起始位置及长度返回字符串的一部分。例子如下：

```
$ awk '$1=="L.Tansley" {print substr($1,1,5)}' grade.txt
L.Tan
```

上面例子中，指定在域1的第一个字符开始，返回其前面5个字符。

如果给定长度值远大于字符串长度， awk将从起始位置返回所有字符，要抽取 L Tansl-ey的姓，只需从第3个字符开始返回长度为7。可以输入长度99，awk返回结果相同。

```
$ awk '$1=="L.Tansley" {print substr($1,3,99)}' grade.txt
Tansley
```

substr的另一种形式是返回字符串后缀或指定位置后面字符。这里需要给出指定字符串及其返回字串的起始位置。例如，从文本文件中抽取姓氏，需操作域 1，并从第三个字符开始：

```
$ awk '{print substr($1,3)}' grade.txt
Tansley
Lulu
Bunny
Troll
Tansley
```

还有一个例子，在 BEGIN部分定义字符串，在 END部分返回从第 t个字符开始抽取的子串。

```
$ awk 'BEGIN {STR="A FEW GOOD MEN"}END{print substr(STR,7)}' grade.txt
GOOD MEN
```

substr(r,s,t) 與 substr(r,s) 是一個意思，截取從某個字符開始的一串字符，t 表示最后一個是多少，沒有時爲全部

8. 从shell中向awk传入字符串

本章开始已经提到过，awk脚本大多只有一行，其中很少是字符串表示的。本书大多要求在一行内完成awk脚本，这一点通过将变量传入awk命令行会变得很容易。现就其基本原理讲述一些例子。

使用管道将字符串stand-by传入awk，返回其长度。

```
$ echo "Stand-by" |awk '{print length($0)}'
8
```

设置文件名为一变量，管道输出到awk，返回不带扩展名的文件名。

```
$ STR="mydoc.txt"
$ echo $STR| awk '{print substr($STR,1,5)}'
mydoc
```

设置文件名为一变量，管道输出到awk，只返回其扩展名。

```
$ STR="mydoc.txt"
$ echo $STR| awk '{print substr($STR,7)}'
txt
```

9.2.10 字符串屏蔽序列

使用字符串或正则表达式时，有时需要在输出中加入一新行或查询一元字符。

打印一新行时，（新行为字符\n），给出其屏蔽序列，以不失其特殊含义，用法为在字符串前加入反斜线。例如使用\n强迫打印一新行。

如果使用正则表达式，查询花括号（{}），在字符前加反斜线，如/\{/，将在awk中失掉其特殊含义。

表9-5列出awk识别的另外一些屏蔽序列

表9-5 awk中使用的屏蔽序列

\b	退格键	\t	tab键
\f	走纸换页	\ddd	八进制值
\n	新行	\c	任意其他特殊字符，例如\\为反斜线符号
\r	回车键		

使用上述符号，打印May Day，中间夹tab键，后跟两个新行，再打印May Day，但这次使用八进制数104、141、171、分别代表D、a、y。

```
$ awk 'BEGIN {print"\n\May\tDay\n\nMay \104\141\171"}'
May     Day

May     Day
```

注意，\104为D的八进制ASCII码，\141为a的八进制ASCII码，等等。

9.2.11 awk输出函数printf

目前为止，所有例子的输出都是直接到屏幕，除了tab键以外没有任何格式。awk提供函数printf，拥有几种不同的格式化输出功能。例如按列输出、左对齐或右对齐方式。

每一种printf函数（格式控制字符）都以一个%符号开始，以一个决定转换的字符结束。转换包含三种修饰符。

printf函数基本语法是printf（[格式控制符]，参数），格式控制字符通常在引号里。

printf 函數專用于格式化輸出，使整個輸出更美觀。

9.2.12 printf修饰符

表9-6 awk printf修饰符

-	左对齐
Width	域的步长，用0表示0步长
.prec	最大字符串长度，或小数点右边的位数

表9-7 awk printf格式

%c	ASCII字符
%d	整数
%e	浮点数，科学记数法
%f	浮点数，例如（123.44）
%g	awk决定使用哪种浮点数转换e或者f
%o	八进制数
%s	字符串
%x	十六进制数

1. 字符转换

观察ASCII码中65的等价值。管道输出65到awk。printf进行ASCII码字符转换。这里也加入换行，因为缺省情况下printf不做换行动作。

```
$ echo "65" | awk '{printf "%c\n",$0}'
A
```

当然也可以按同样方式使用awk得到同样结果。

```
$ awk 'BEGIN {printf "%c\n",65}'
A
```

所有的字符转换都是一样的，下面的例子表示进行浮点数转换后‘ 999’的输出结果。整数传入后被加了六个小数点。

```
$ awk 'BEGIN {printf "%f\n",999}
999.000000
```

2. 格式化输出

打印所有的学生名字和序列号，要求名字左对齐， 15个字符长度，后跟序列号。注意 \n 换行符放在最后一个指示符后面。输出将自动分成两列。

```
$ awk {printf "%-15s %s\n", $1,$3}' grade.txt
M.Tansley       48311
J.Lulu          48317
P.Bunny         48
J.Troll         4842
L.Tansley       4712
```

最好加入一些文本注释帮助理解报文含义。可在正文前嵌入头信息。注意这里使用 print 加入头信息。如果愿意，也可使用printf。

```
$ awk 'BEGIN {print "Name \t\tS.Number"}{printf "%-15s %s\n", $1,$3}'
grade.txt
```

awk 'BEGIN {print "Name \t\tS.number"} {printf "%-15s %s\n", $1,$3}' grade.txt

%-15s %s 表示輸出字符串，- 左對齊 15 個字符

這里輸出的是兩個域，第一個域被限制只輸出 15 個字符，$3 按第二個格式輸出，如果省略該格式意味著$3 不輸出。

```
[root@test ~]# awk '{printf "%15s ", $1,$3}' grade.txt
          M.Tansley               J.Lulu              P.Bunny              J.Troll
L.Tansley
```

3. 向一行awk命令传值

在查看awk脚本前，先来查看怎样在awk命令行中传递变量。

在awk执行前将值传入awk变量，需要将变量放在命令行中，格式如下：

```
awk  命令变量=输入文件值
```

(后面会讲到怎样传递变量到awk脚本中)。

下面的例子在命令行中设置变量 AGE等于10，然后传入awk中，查询年龄在10岁以下的所有学生。

```
$ awk '{if ($5 < AGE) print $0}' AGE=10 grade.txt
M.Tansley   05/99     48311  Green    8     40    44
J.Lulu      06/99     48317  green    9     24    26
```

要快速查看文件系统空间容量，观察其是否达到一定水平，可使用下面 awk一行脚本。因为要监视的已使用空间容量不断在变化，可以在命令行指定一个触发值。首先用管道命令将df -k传入awk，然后抽出第 4列，即剩余可利用空间容量。使用 $4~/^[0-9]/取得容量数值(1024块) 而不是df的文件头，然后对命令行与 ‘ if($4<TRIGGER)’ 上变量TRIGGER中指定的值进行查询测试。

```
$ df -k | awk '($4 ~/^[0-9]/) {if($4 < TRIGGER) print $6"\t"$4}'
TRIGGER=56000
/dos    55808
/apps   51022
```

当然可以使用管道将值传入 awk。本例使用who命令，who命令第一列包含注册用户名，这里打印注册用户，并加入一定信息。

```
$ who | awk '{print $1 " is logged on"}'
louisel is logged on
papam is logged on
```

awk也允许传入环境变量。下面的例子使用环境变量 LOGNAME支持当前用户名。可从who命令管道输出到awk中获得相应信息。

```
$ who | awk '{if ($1 == user) print $1" you are connected to
"$2}'user=$LOGNAME
```

awk 脚本文件

4. awk脚本文件

可以将awk脚本写入一个文件再执行它。命令不必很长（尽管这是写入一个脚本文件的主要原因），甚至可以接受一行命令。这样可以保存 awk命令，以使不必每次使用时都需要重新输入。使用文件的另一个好处是可以增加注释，以便于理解脚本的真正用途和功能。

使用前面的几个例子，将之转换成awk可执行文件。像原来做的一样，将学生目前级别分相加awk ‘(tot+=$6) END{print "club student total points: "tot}’ grade.txt。

创建新文件student_tot.awk，给所有awk程序加入awk扩展名是一种好习惯，这样通过查看文件名就知道这是一个awk程序。文本如下：

```
!/bin/awk -f
# all comment lines must start with a hash '#'
# name: student_tot.awk
# to call: student_tot.awk grade.txt
# prints total and average of club student points

#print a header first
BEGIN{
print "Student    Date  Member No.  Grade  Age   Points   Max"
print "Name       Joined                          Gained   Point Available"
print "=========================================================================="

}
# let's add the scores of points gained
(tot+=$6)

# finished processing now let's print the total and average point
END{print "Club student total points  :" tot
  print "Average Club Student points:" tot/NR}
```

第一行是！/bin/awk -f。这很重要，没有它自包含脚本将不能执行。这一行告之脚本系统中awk的位置。通过将命令分开，脚本可读性提高，还可以在命令之间加入注释。这里加入头信息和结尾的平均值。基本上这是一个一行脚本文件。

执行时，在脚本文件后键入输入文件名，但是首先要对脚本文件加入可执行权限。

```
$ chmod u+x student_tot.awk
$ student_tot.awk grade.txt
Student    Date     Member No. Grade  Age  Points   Max
Name       Joined                         Gained   Point Available
=========================================================================
M.Tansley   05/99     48311  Green    8     40     44
J.Lulu      06/99     48317  green    9     24     26
P.Bunny     02/99     48     Yellow   12    35     28
J.Troll     07/99     4842   Brown-3  12    26     26
L.Tansley   05/99     4712   Brown-2  12    30     28
Club student total points  :155
Average Club Student points:31
```

系统中运用的帐号核实程序检验数据操作人的数据输入，不幸的是这个程序有一点错误，或者应该说是“非文本特征”。如果一个记录被发现包含一个错误，它应该一次只打印一行

“ERROR*”，但实际上打印了许多这样的错误行。这会给帐号管理员造成误解，因此需要用awk脚本过滤出错误行的出现频率，使得每一个失败记录只对应一个错误行。

在awk实施过滤前先看看部分文件：

```
...
...
INVALID LCSD 98GJ23
ERROR*
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
ERROR*
ERROR*
ERROR*
ERROR*
PASS FIELD INVALID ON LDPS
ERROR*
ERROR*
PASS FIELD INVALID ON GHSI
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
ERROR*
```

awk脚本如下：

```
#!/bin/awk -f
# error_strip.awk
# to call: error_strip.awk <filename>
# strips out the ERROR* lines if there are more than one
# ERROR* lines after each failed record.

BEGIN { error_line="" }
# tell awk the whole is "ERROR*"
{ if ($0 == "ERROR*" && error_line == "ERROR*")

# go to next line
next;
  error_line = $0; print }
```

awk过滤结果如下：

```
$ strip.awk strip
INVALID LCSD 98GJ23
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
PASS FIELD INVALID ON LDPS
ERROR*
PASS FIELD INVALID ON GHSI
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
```

5. 在awk中使用FS变量

如果使用非空格符做域分隔符（FS）浏览文件，例如# 或：，编写这样的一行命令很容易，因为使用FS选项可以在命令行中指定域分隔符。

```
$ awk -F: 'awk {print $0}' input-file
```

使用awk脚本时，记住设置FS变量是在BEGIN部分。如果不这样做， awk将会发生混淆，不知道域分隔符是什么。

下述脚本指定FS变量。脚本从 /etc/passwd文件中抽取第 1和第5域，通过分号“:”分隔passwd文件域。第1域是帐号名，第5域是帐号所有者。

```
$ pg passwd.awk
!/bin/awk -f
# to call: passwd.awk /etc/passwd
# print out the first and fifth fields
BEGIN{
FS=":"}
{print $1,"\t",$5}
```

```
$ passwd.awk /etc/passwd
root      Special Admin login
xdm       Restart xdm Login
sysadm    Regular Admin login
daemon    Daemon Login for daemons needing permissions
```

腳本應該如下

[root@test ~]# cat passwd.awk

#!/bin/awk -f

#to call:passwd.awk /etc/passwd

#print out the first and fifth fields

BEGIN {

FS=":" }

{print $1,"\t",$5}

注意下各個括號的位置，否則會產生錯誤

#!/bin/awk –f 含義相當于告訴系統如下調用命令

awk –f passwd.awk /etc/passwd

6. 向awk脚本传值

向awk脚本传值与向awk一行命令传值方式大体相同，格式为：

```
awk script_file var=value input_file
```

下述脚本对比检查文件中域号和指定数字。这里使用了 NF变量MAX，表示指定检查的域号，使用双引号将域分隔符括起来，即使它是一个空格。

```
$ pg fieldcheck.awk
#!/bin/awk -f
# check on how many fields in a file
# name:fieldcheck.awk
# to call: fieldcheck MAX=n FS=<separator> filename
#
NF!=MAX{
print("line " NR " does not have" MAX " fields")}
```

如果以/etc/passwd作输入文件（passwd文件有7个域），运行上述脚本。参数格式如下：

```
$ fieldcheck.awk MAX=7 FS=":" /etc/passwd
```

使用前面一行脚本的例子，将之转换成 awk脚本如下：

```
$ pg name.awk
#!/bin/awk -f
# name: age.awk
# to call: age.awk AGE=n grade.txt
# prints ages that are lower than the age supplied on the comand line
{if ($5 < AGE)
  print $0}
```

文本包括了比实际命令更多的信息，没关系，仔细研读文本后，就可以精确知道其功能及如何调用它。

不要忘了增加脚本的可执行权限，然后将变量和赋值放在命令行脚本名字后、输入文件前执行。

```
$ age.awk AGE=10 grade.txt
M.Tansley   05/99     48311  Green    8     40   44
J.Lulu      06/99     48317  green    9     24   26
```

同样可以使用前面提到的管道命令传值，下述 awk脚本从du命令获得输入，并输出块和字节数。

```
$ pg duawk.awk
#!/bin/awk -f
# to call: du | duawk.awk
# prints file/direc's in bytes and blocks
BEGIN{
OFS="\t" ;
print "name" "\t\t","bytes","blocks\n"
print "============================="}
{print $2,"\t\t",$1*512,$1}
```

为运行这段脚本，使用du命令，并管道输出至awk脚本。

```
$ du | awkdu.awk
name                    bytes    blocks
=========================================
./profile.d             2048     4
./X11                   135680   265
./rc.d/init.d           27136    53
./rc.d/rc0.d            512      1
./rc.d/rc1.d            512      1
```

9.2.13 awk数组

前面讲述split函数时，提到怎样使用它将元素划分进一个数组。这里还有一个例子：

```
$ awk 'BEGIN {print split("123#456#678", myarray, "#")}'
3
```

在上面的例子中，split返回数组myarray下标数。实际上myarray数组为：

```
Myarray[1]="123"
Myarray[2]="456"
Myarray[3]="678"
```

数组使用前，不必定义，也不必指定数组元素个数。经常使用循环来访问数组。下面是一种循环类型的基本结构：

```
For (element in array ) print array[element]
```

对于记录“123#456#678”，先使用split函数划分它，再使用循环打印各数组元素。操作脚本如下：

```
$ pg arraytest.awk
#!/bin/awk -f
# name: arraytest.awk
# prints out an array
BEGIN{
record="123#456#789";
split(record, myarray,"#") }
  END { for (i in myarray) {print myarray[i]}}
```

要运行脚本，使用/dev/null作为输入文件。

```
$ arraytest.awk  /dev/null
123
456
789
```

数组和记录

上面的例子讲述怎样通过split函数使用数组。也可以预先定义数组，并使用它与域进行较测试，下面的例子中将使用更多的数组。

下面是从空手道数据库卸载的一部分数据，包含了学生级别及是否是成人或未成年人信息，有两个域，分隔符为（#），文件如下：

```
$ pg grade_student.txt
Yellow#Junior
Orange#Senior
Yellow#Junior
Purple#Junior
Brown-2#Junior
White#Senior
Orange#Senior
Red#Junior
Brown-2#Senior
Yellow#Senior
Red#Junior
Blue#Senior
Green#Senior
Purple#Junior
White#Junior
```

脚本功能是读文件并输出下列信息。

1) 俱乐部中Yellow、Orange和Red级别的人各是多少。

2) 俱乐部中有多少成年人和未成年人。

查看文件，也许20秒内就会猜出答案，但是如果记录超过 60个又怎么办呢？这不会很容易就看出来，必须使用awk脚本。

首先看看awk脚本，然后做进一步讲解。

```
$ pg belts.awk
!/bin/awk -f
# name: belts.awk
# to call: belts.awk grade2.txt
# loops through the grade2.txt file and counts how many
# belts we have in (yellow, orange, red)
# also count how many adults and juniors we have
#
# start of BEGIN
# set FS and load the arrays with our values
BEGIN{FS="#"
# load the belt colours we are interested in only
belt["Yellow"]
belt["Orange"]
belt["Red"]
# end of BEGIN
# load the student type
student["Junior"]
student["Senior"]
}
# loop thru array that holds the belt colours against field-1
# if we have a match, keep a running total
  {for (colour in belt)
    {if ($1==colour)
    belt[colour]++}}
# loop thru array that holds the student type against
```

```
# field-2 if we have a match, keep a running total
  {for (senior_or_junior in student)
    {if ($2==senior_or_junior)
    student[senior_or_junior]++}}

# finished processing so print out the matches..for each array
END{ for (colour in belt) print "The club has", belt[colour], colour,
"Belts"

for (senior_or_junior in student) print "The club
has",student[senior_or_junior]\
,senior_or_junior, "students"}
```

BEGIN部分设置FS为符号#，即域分隔符，因为要查找 Yellow、Orange和Red三个级别。然后在脚本中手工建立数组下标对学生做同样的操作。注意，脚本到此只有下标或元素，并没有给数组名本身加任何注释。初始化完成后， BEGIN部分结束。记住BEGIN部分并没有文件处理操作。

现在可以处理文件了。首先给数组命名为 color，使用循环语句测试域1级别列是否等于数组元素之一（Yellow、Orange或Red），如果匹配，依照匹配元素将运行总数保存进数组。

同样处理数组‘ Senior_or_junior’，浏览域2时匹配操作满足，运行总数存入 junior或senior的匹配数组元素。

END部分打印浏览结果，对每一个数组使用循环语句并打印它。

注意在打印语句末尾有一个 \符号，用来通知 awk（或相关脚本）命令持续到下一行，当输入一个很长的命令，并且想分行输入时可使用这种方法。运行脚本前记住要加入可执行权限。

```
$ belts.awk grade_student.txt
The club has 2 Red Belts
The club has 2 Orange Belts
The club has 3 Yellow Belts
The club has 7 Senior students
The club has 8 Junior students
```

For 循環語法

For (element in array) print array[element]

belt["yellow"]表示 belt[yellow]

在 awk 中数组叫做关联数组(associative arrays)，因为下标记可以是数也可以是串。awk 中的数组不必提前声明，也不必声明大小。数组元素用 0 或空串来初始化，这根据上下文而定。

1) 关联数组的下标

.. 把变量用作数组索引。

举例：

$ nawk '{name[x++]=$2};END{for(i=0;i<NR;i++)\

>; print i,name}' employees

0 Jones

1 Adams

2 Chang

3 Black

解释：在数组 name 里的下标是一个用户自定义变量 x。++显示了一个数值型的上下文。

.. 特殊 for 循环

在 for 循环无效的情况下，即当串被用作下标或者下标不是连续的数时，用特殊 for 循环来读取一个关联数组。特殊 for 循环把下标当作一个键来找到它的相关的值。

格式：

```
{for(item in arrayname){
print arrayname[item]}}
```

举例：

```
$ nawk '/^Tom/{name[NR]=$1};\
>; END{for(i=1;i<=NR;i++) print name}' db
Tom



Tom
Tom

Tommy
$ nawk '/^Tom/{name[NR]=$1};\
>; END{for(i in name){print name}}' db
Tom
Tom
Tommy
Tom
```

解释：这两个例子用 NR（行号）来做下标，因此匹配 Tom 的模式的行不连续，数组下标不连续。如果用传统 for 来循环打印会在数组没有值的地方打印空值。通过使用特殊 for 循环，只打印数组中有值的内容

For 循環有兩種方式，變量(數字)作下標；串用作下標。Split 函數連用，非常實用。。。。兩種下標方式，讀取就有所不同了。。

數組的使用是難點，和比較實用的地方。下面有個實例。

一个 a.file 文件

11 1.txt

20 1.txt

5 d.docx

5 y.doc

7 d.docx

要求的输出格式是这样的：

txt:31

docx:12

doc:5

awk '{split($2,a,".");b[a[2]]+=$1}END{for(i in b)print i":"b[i]}' a.file

awk -F '[.]+' '{a[$NF]+=$1}END{for (j in a) print j":"a[j]}' file

[root@test bin]# awk '{split($2,a);b[a[1]]+=$1}END{for (i in b) print i":"b[i]}'
a.file

d.docx:12

y.doc:5

1.txt:31

Split 函數的使用。要學會。

10.Sed 命令介紹

==========

sed是一个非交互性文本流编辑器。它编辑文件或标准输入导出的文本拷贝。标准输入可能是来自键盘、文件重定向、字符串或变量，或者是一个管道的文本。 sed可以做些什么呢？别忘了，Vi也是一个文本编辑器。 sed可以随意编辑小或大的文件，有许多 sed命令用来编辑、删除，并允许做这项工作时不在现场。 sed一次性处理所有改变，因而变得很有效，对用户来讲，最重要的是节省了时间。

本章内容有：

- 抽取域。
- 匹配正则表达式。
- 比较域。
- 增加、附加、替换。
- 基本的sed命令和一行脚本。

可以在命令行输入sed命令，也可以在一个文件中写入命令，然后调用 sed，这与awk基本相同。使用sed需要记住的一个重要事实是，无论命令是什么， sed并不与初始化文件打交道，它操作的只是一个拷贝，然后所有的改动如果没有重定向到一个文件，将输出到屏幕。

因为sed是一个非交互性编辑器，必须通过行号或正则表达式指定要改变的文本行。

本章介绍sed用法和功能。本章大多编写的是一行命令和小脚本。这样做可以慢慢加深对sed用法的了解，取得宝贵的经验，以便最终自己编出大的复杂 sed脚本。

和grep与awk一样， sed是一种重要的文本过滤工具，或者使用一行命令或者使用管道与grep与awk相结合。

10.1 sed怎样读取数据

sed从文件的一个文本行或从标准输入的几种格式中读取数据，将之拷贝到一个编辑缓冲区，然后读命令行或脚本的第一条命令，并使用这些命令查找模式或定位行号编辑它。重复此过程直到命令结束。

10.2 调用sed

调用sed有三种方式：在命令行键入命令；将 sed命令插入脚本文件，然后调用 sed；将sed命令插入脚本文件，并使 sed脚本可执行。

使用sed命令行格式为：

```
sed [选项] sed命令 输入文件。
```

记住在命令行使用sed命令时，实际命令要加单引号。 sed也允许加双引号。

使用sed脚本文件，格式为：

```
sed [选项] -f sed脚本文件 输入文件
```

要使用第一行具有sed命令解释器的sed脚本文件，其格式为：

```
sed脚本文件 [选项] 输入文件
```

不管是使用shell命令行方式或脚本文件方式，如果没有指定输入文件， sed从标准输入中接受输入，一般是键盘或重定向结果。

sed选项如下：

n 不打印；sed不写编辑行到标准输出，缺省为打印所有行（编辑和未编辑）。p命令可以用来打印编辑行。

c 下一命令是编辑命令。使用多项编辑时加入此选项。如果只用到一条 sed命令，

此选项无用，但指定它也没有关系。

f 如果正在调用sed脚本文件，使用此选项。此选项通知 sed一个脚本文件支持所有的 sed命令，例如： sed -f myscript.sed input_file，这里myscript.sed即为支持sed命令的文件。

10.2.1 保存sed输出

由于不接触初始化文件，如果想要保存改动内容，简单地将所有输出重定向到一个文件即可。下面的例子重定向 sed命令的所有输出至文件‘ myoutfile’，当对结果很满意时使用这种方法。

```
$ sed 'some-sed-commands' input-file > myoutfile
```

10.2.2 使用sed在文件中查询文本的方式

sed浏览输入文件时，缺省从第一行开始，有两种方式定位文本：

1) 使用行号，可以是一个简单数字，或是一个行号范围。

2) 使用正则表达式，怎样构建这些模式请参见第 7章。

表10-1给出使用sed定位文本的一些方式。

表10-1 使用sed在文件中定位文本的方式

x	x为一行号，如1
x,y	表示行号范围从x到y，如2，5表示从第2行到第5行
/pattern/	查询包含模式的行。例如/disk/或/[a-z]/
/pattern/pattern/	查询包含两个模式的行。例如/disk/disks/
pattern/,x	在给定行号上查询包含模式的行。如/ribbon/,3
x,/pattern/	通过行号和模式查询匹配行。 3./vdu/
x,y!	查询不包含指定行号x和y的行。1,2!

10.2.3 基本sed编辑命令

表10-2列出了Sed的编辑命令。

表10-2 sed编辑命令

p	打印匹配行
=	显示文件行号
a\	在定位行号后附加新文本信息
i\	在定位行号后插入新文本信息
d	删除定位行
c\	用新文本替换定位文本
s	使用替换模式替换相应模式
r	从另一个文件中读文本
w	写文本到一个文件
q	第一个模式匹配完成后推出或立即推出
l	显示与八进制 ASCII代码等价的控制字符
{}	在定位行执行的命令组
n	从另一个文件中读文本下一行，并附加在下一行
g	将模式2粘贴到/pattern n/
y	传送字符
n	延续到下一输入行；允许跨行的模式匹配语句

如果不特别声明，sed例子中使用下述文本文件quote.txt。

```
$ pg quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

10.3 sed和正则表达式

sed识别任何基本正则表达式和模式及其行匹配规则。记住规则之一是：如果要定位一特殊字符，必须使用（\）屏蔽其特殊含义，如有必要请参照第 7章正则表达式。第7章使用的所有正则表达式在sed中都是合法的。

10.4 基本sed编程举例

下面通过例子实际检验一下 sed的编辑功能。

10.4.1 使用p（rint）显示行

print命令格式为[address[，address]P。显示文本行必须提供sed命令行号。

```
$ sed '2p' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

错误在哪儿？原意只打印第二行，但是却打印了文件中所有行，为此需使用 -n选项，显示打印定位（匹配）行。

```
$ sed -n '2p' quote.txt
It was an evening of splendid music and company.
```

10.4.2 打印范围

可以指定行的范围，现打印 1到3行，用逗号分隔行号。

```
$ sed -n '1,3p' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
```

打印 sed –n ‘[start [,end] p’ inputFile

打印第 start 行，或從 start,end 行

sed –n ‘/string/’p inputFile　打印查找，匹配的行; p 字符要跟在正则表達式后面

可以這樣理解這兩個選項 -n　next 跳過其它的行 –p print 打印

10.4.3 打印模式

假定要匹配单词Neave，并打印此行，方法如下。使用模式 /pattern/格式，这里为/Neave/。

```
$ sed -n '/Neave/'p quote.txt
The local nurse Miss P.Neave was in attendance.
```

10.4.4 使用模式和行号进行查询

为编辑某个单词浏览一个文件时， sed返回包含指定单词的许多行。怎样使返回结果更精确以满足模式匹配呢？可以将行号和模式结合使用。下面这个例子，假定要改动文件 quote.txt 最后一行中的单词the，使用sed查询the，返回两行：

```
$ sed -n '/The/'p quote.txt
The honeysuckle band played all night long for only $90.
The local nurse Miss P.Neave was in attendance.
```

使用模式与行号的混合方式可以剔除第一行，格式为 line_number,/pattern/。逗号用来分隔行号与模式开始部分。为达到预期结果，使用 4,/the/。意即只在第四行查询模式 the，命令如下：

```
$ sed -n '4,/The/'p quote.txt
The local nurse Miss P.Neave was in attendance.
```

用行號與模式來精確匹配 x,/pattern/

10.4.5 匹配元字符

匹配元字符 $前，必须使用反斜线\屏蔽其特殊含义。模式为/\$/ p。

```
$ sed -n '/\$/'p quote.txt
The honeysuckle band played all night long for only $90.
```

10.4.6 显示整个文件

要打印整个文件，只需将行范围设为第一行到最后一行 1,$。$意为最后一行。

```
$ sed -n '1,$p' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

10.4.7 任意字符

匹配任意字母，后跟任意字母的 0次或多次重复，并以ing结尾，模式为/.*ing/。可以使用这个模式查询以ing结尾的任意单词。

```
$ sed -n '/.*ing/'p quote.txt
It was an evening of splendid music and company.
```

注意 p 出現的位置在 正則表達式與行號匹配 時有所不同

sed -n '/any.$/'p quote.txt　　sed 查找的是整行，這樣寫匹配的是行尾

sed -n '/.*ing /'p quote.txt　　這樣寫才是匹配其中某個單詞，字母

10.4.8 首行

要打印文件第一行，使用行号：

```
$ sed -n '1p' quote.txt
The honeysuckle band played all night long for only $90.
```

10.4.9 最后一行

要打印最后一行，使用$。$是代表最后一行的元字符。

```
$ sed -n '$p' quote.txt
The local nurse Miss P.Neave was in attendance.
```

10.4.10 打印行号

要打印行号，使用等号=。打印模式匹配的行号，使用格式/pattern/=。

```
$ sed  -e '/music/=' quote.txt
The honeysuckle band played all night long for only $90.
2
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

整个文件都打印出来，并且匹配行打印了行号。如果只关心实际行号，使用 -e选项。

```
$ sed  -n '/music/=' quote.txt
2
```

如果只打印行号及匹配行，必须使用两个 sed命令，并使用e选项。第一个命令打印模式匹配行，第二个使用=选项打印行号，格式为sed -n -e /pattern/p -e /pattern/=。

```
$ sed -n -e '/music/p' -e '/music/=' quote.txt
It was an evening of splendid music and company.
2
```

10.4.11 附加文本

要附加文本，使用符号a\，可以将指定文本一行或多行附加到指定行。如果不指定文本放置位置，sed缺省放在每一行后面。附加文本时不能指定范围，只允许一个地址模式。文本附加操作时，结果输出在标准输出上。注意它不能被编辑，因为 sed执行时，首先将文件的一行文本拷贝至缓冲区，在这里 sed编辑命令执行所有操作（不是在初始文件上），因为文本直接输出到标准输出，sed并无拷贝。

要想在附加操作后编辑文本，必须保存文件，然后运行另一个 sed命令编辑它。这时文件的内容又被移至缓冲区。

附加操作格式如下：

```
[address]a\
text\
text\
...
text
```

地址指定一个模式或行号，定位新文本附加位置。 a\ 通知sed对a\后的文本进行实际附加操作。观察格式，注意每一行后面有一斜划线，这个斜划线代表换行。 sed执行到这儿，将创建一新行，然后插入下一文本行。最后一行不加斜划线， sed假定这是附加命令结尾。

当附加或插入文本或键入几个 sed命令时，可以利用辅助的 shell提示符以输入多行命令。这里没有这样做，因为可以留给使用者自己编写，并且在一个脚本文件中写这样的语句更适宜。现在马上讲述sed脚本文件。另外，脚本可以加入空行和注释行以增加可读性。

10.4.12 创建sed脚本文件

要创建脚本文件append.sed，输入下列命令：

```
$ pg append.sed
#!/bin/sed -f
/company/ a\
Then suddenly it happened.
```

保存它，增加可执行权限：

```
$ chmod u+x append.sed
```

运行，

```
$ append.sed quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Then suddenly it happened.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

如果返回‘file not found’，试在脚本前加入.\。

现在查看其具体功能。第一行是sed命令解释行。脚本在这一行查找sed以运行命令，这里定位在/bin。

第二行以/company/开始，这是附加操作起始位置。 a\通知sed这是一个附加操作，首先应插入一个新行。第三行是附加操作要加入到拷贝的实际文本。

附加操作：[address]a\ 具體爲插入一個新行，執行后面的附加動作

现在查看其具体功能。第一行是sed命令解释行。脚本在这一行查找sed以运行命令，这里定位在/bin。

第二行以/company/开始，这是附加操作起始位置。 a\通知sed这是一个附加操作，首先应插入一个新行。第三行是附加操作要加入到拷贝的实际文本。

输出显示附加结果。如果要保存输出，重定向到一个文件。

10.4.13 插入文本

插入命令类似于附加命令，只是在指定行前面插入。和附加命令一样，它也只接受一个地址。下面是插入命令的一般格式。地址是匹配模式或行号：

也可以使用行号指定文本插入位置，插入位置在模式或指定行号 4之前。脚本如下：

```
#!/bin/sed -f
4 i\
Utter confusion followed.
```

sed '4 i\Utter confusion followed.' quote.txt 會在第四行前面位置插入該行，成爲新的第四行

10.4.14 修改文本

修改命令将在匹配模式空间的指定行用新文本加以替代，格式如下：

```
[address[,address] c\
text\
text\
text\
...
text
```

将第一行The honeysuckle band played all night long for only $90替换为The office Dibble band played well。首先要匹配第一行的任何部分，可使用模式‘ /Honeysuckle/’。sed脚本文件为change.sed。内容如下：

```
$ pg change.sed
#!/bin/sed -f
# change.sed
/honeysuckle/ c\
The Office Dibble band played well.
```

运行它，不要忘了给脚本增加可执行权限。 chmod u+x change.sed。

```
$ change.sed quote.txt
The Office Dibble band played well.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

像插入动作一样，可以使用行号代替模式，两种方式完成相同的功能。

```
#!/bin/sed -f
3 c\
The Office Dibble band played well.
```

sed '3 c\The office dibble band play well.' quote.txt

可以对同一个脚本中的相同文件进行修改、附加、插入三种动作匹配和混合操作。下面是一个带有注释的脚本例子。

```
$ pg mix.sed
#!/bin/sed -f
# this is a comment line, all comment starts with a #
# name: mix.sed


# this is the change on line 1
1 c\
The Dibble band were grooving.
# let's now insert a line
/evening/ i\
They played some great tunes.
# change the last line, a $ means last line
$ c\
Nurse Neave was too tipsy to help.
```

```
# stick in a new line before the last line
3 a\
Where was the nurse to help?
```

运行它，结果如下：

```
$ mix.sed quote.txt
The Dibble band were grooving.
They played some great tunes.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
Where was the nurse to help?
Nurse Neave was too tipsy to help.
```

10.4.15 删除文本

sed删除文本格式：

```
[address[, address]]d
```

地址可以是行的范围或模式，让我们看几个例子。

删除第一行；1d意为删除第一行。

```
$ sed '1d' quote.txt
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

删除第一到第三行：

```
$ sed '1,3d' quote.txt
The local nurse Miss P.Neave was in attendance.
```

删除最后一行：

```
$ sed '$d' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
```

也可以使用正则表达式进行删除操作。下面的例子删除包含文本‘ Neave’的行。

```
$ sed '/Neave/d' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
```

10.4.16 替换文本

替换命令用替换模式替换指定模式，格式为：

```
[address[, address]] s/ pattern-to-find /replacement-pattern/[g p w n]
```

s选项通知sed这是一个替换操作，并查询 pattern-to-find，成功后用replacement-pattern替换它。

替换选项如下：

g 缺省情况下只替换第一次出现模式，使用 g选项替换全局所有出现模式。

p 缺省sed将所有被替换行写入标准输出，加 p选项将使-n选项无效。-n选项不打印输出结果。

w 文件名 使用此选项将输出定向到一个文件。

让我们看几个例子。替换 night为NIGHT，首先查询模式 night，然后用文本 NIGHT替换它。

```
$ sed 's/night/NIGHT/' quote.txt
The honeysuckle band played all NIGHT long for only $90.
```

要从 $90中删除 $符号（记住这是一个特殊符号，必须用 \屏蔽其特殊含义），在 replacement-pattern部分不写任何东西，保留空白，但仍需要用斜线括起来。在 sed中也可以这样删除一个字符串。

```
$ sed 's/\$//' quote.txt
The honeysuckle band played all night long for only 90.
```

要进行全局替换，即替换所有出现模式，只需在命令后加 g选项。下面的例子将所有 The 替换成Wow！。

```
$ sed 's/The/Wow!/g' quote.txt
Wow! honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
Wow! local nurse Miss P.Neave was in attendance.
```

将替换结果写入一个文件用 w选项，下面的例子将 splendid替换为SPLENDID的替换结果写入文件 sed.out：

```
$ sed s/splendid/SPLENDID/w sed.out' quote.txt
```

注意要将文件名括在sed的单引号里。文件结果如下：

```
$ pg sed.out
It was an evening of SPLENDID music and company.
```

让我们看几个例子。替换 night为NIGHT，首先查询模式 night，然后用文本 NIGHT替换它。

```
$ sed 's/night/NIGHT/' quote.txt
The honeysuckle band played all NIGHT long for only $90.
```

要从 $90中删除 $符号（记住这是一个特殊符号，必须用 \屏蔽其特殊含义），在 replacement-pattern部分不写任何东西，保留空白，但仍需要用斜线括起来。在 sed中也可以这样删除一个字符串。

```
$ sed 's/\$//' quote.txt
The honeysuckle band played all night long for only 90.
```

要进行全局替换，即替换所有出现模式，只需在命令后加 g选项。下面的例子将所有 The 替换成Wow！。

```
$ sed 's/The/Wow!/g' quote.txt
Wow! honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
Wow! local nurse Miss P.Neave was in attendance.
```

将替换结果写入一个文件用 w选项，下面的例子将 splendid替换为SPLENDID的替换结果写入文件 sed.out：

```
$ sed s/splendid/SPLENDID/w sed.out' quote.txt
```

注意要将文件名括在sed的单引号里。文件结果如下：

```
$ pg sed.out
It was an evening of SPLENDID music and company.
```

sed -p 's/The/Wow!/w sed.out' quote.txt

10.5 使用替换修改字符串

如果要附加或修改一个字符串，可以使用（ &）命令，&命令保存发现模式以便重新调用它，然后把它放在替换字符串里面。这里给出一个修改的设计思路。先给出一个被替换模式，然后是一个准备附加在第一个模式后的另一个模式，并且后面带有 &，这样修改模式将放在匹配模式之前。例如，sed语句s/nurse/"Hello"&/p 的结果如下：

```
$ sed -n 's/nurse/"Hello" &/p' quote.txt
The local "Hello" nurse Miss P.Neave was in attendance.
```

原句是文本行The local nurse Miss P.Neave was in attendance。

记住模式中要使用空格，因为输出结果表明应加入空格。

还有一个例子：

```
$ sed -n 's/played/from Hockering &/p' quote.txt
The honeysuckle band from Hockering played all night long for only $90.
```

原句是The honeysuckle band played all night long for only $90。相信这种修改动作已经讲解得很清楚了。

sed -n 's/played/from Hockering &/p' sed/quote.txt

sed -n 's/played/& from Hockering /p' sed/quote.txt

&號表示發現的部分，可以放置在前，后面位置

10.6 将sed结果写入文件命令

像使用>文件重定向发送输出到一个文件一样，在 sed命令中也可以将结果输入文件。格式有点像使用替换命令：

```
[address[, address]]w filename
```

‘w’选项通知sed将结果写入文件。filename是自解释文件名。下面有两个例子。

```
$ sed '1,2 w filedt' quote.txt
```

文件quote.txt输出到屏幕。模式范围即 1，2行输出到文件filedt。

```
$ pg filedt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
```

下面例子中查询模式Neave，匹配结果行写入文件filedht。

```
$ sed '/Neave/ w dht' quote.txt

$ pg dht
The local nurse Miss P.Neave was in attendance.
```

10.7 从文件中读文本

处理文件时，sed允许从另一个文件中读文本，并将其文本附加在当前文件。此命令放在模式匹配行后，格式为：

```
address r filename
```

这里r通知sed将从另一个文件源中读文本。filename是其文件名。

现在创建一个小文件sedex.txt，内容如下：

```
$ pg sedex.txt
Boom boom went the music.
```

将sedex.txt内容附加到文件quote.txt的拷贝。在模式匹配行/company/后放置附加文本。本例为第三行。注意所读的文件名需要用单引号括起来。

```
$ sed '/company./r sedex.txt' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Boom boom went the music.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

sed '/company./r sedex.txt' quote.txt

sed '/music/r sedex.txt' quote.txt 兩個命令輸出結果是一樣的，模式匹配后自動將讀取的文件輸出到下一行。

10.8 匹配后退出

有时需要在模式匹配首次出现后退出 sed，以便执行其他处理脚本。退出命令格式为：

```
address q
```

下面的例子假定查询模式/.a.*/，意为任意字符后跟字符 a，再跟任意字符0次或任意多次。查看文本文件，然后在下列行产生下列单词：

Line 1. band
Line 2. bad
Liner3. was
Line 4. was

查询首次出现模式，然后退出。需要将 q放在sed语句末尾。

```
$ sed '/.a.*/q' quote.txt
The honeysuckle band played all night long for only $90.
```

sed '/.a.*/q' quote.txt 查詢首次匹配后就會退出，不再繼續處理。

10.9 显示文件中的控制字符

当从其他系统下载文件时，有时要删除整个文件的控制字符（非打印字符），从菜单中捕获一个应用的屏幕输出有时也会将控制字符输出进文件，怎样知道文件中是否有控制字符？使用cat -v filename命令，屏幕会乱叫，且到处都是一些垃圾字符，这可以确知文件中包含有控制字符，如果有兴趣可以观察一下这些字符以便于更加确认它们是控制字符。

一些系统中使用cat filename而不是cat-v来查看非打印字符。

sed格式为：

```
[address, [address]]l
```

‘l’ 意为列表。

一般情况下要列出整个文件，而不是模式匹配行，因此使用 l要从第一到最后一行。模式范围1，$即为此意。

如果cat一个文件，发现实际上包含有控制字符。

```
$ cat -v func.txt
This is is the F1 key:^[OP
This is the F2 key:^[OQ
```

现在运行sed命令，观察输出结果。

```
$ sed -n '1,$l' func.txt
This is is the F1 key:\033OP$
This is the F2 key:\033OQ$
```

sed找到并显示了两个控制字符。\033代表退格键，OP为F1键值，放在退格键后。第二行也是如此。

各系统控制字符键值可能不同，主要取决于其映射方式（例如使用 terminfo或termcap）。如果要在文本文件中插入控制字符F1键，使用vi查看其键值，操作如下：

- 启动vi。
- 进入插入模式。
- 按下<Ctrl>键，然后按<v>键 （出现a^）。
- 释放上述两个键。
- 按下F1键（显示[OP]。
- 按下<ESC>键（显示F1键值）。

sed -n '1,$l' func.txt 顯示控制字符，但與 cat –v func.txt 命令中顯示的控制字符不同。不明白。。。

10.10 使用系统sed

前面已经讲述了sed的基本功能，但是在脚本或命令行中使用 sed真正要做的是修改或删除文件或字符串中文本。下面运用前面学过的知识讲述这一点。

10.10.1 处理控制字符

使用sed实现的一个重要功能是在另一个系统中下载的文件中剔除控制字符。

下面是传送过来的文件（dos.txt）的部分脚本。必须去除所有可疑字符，以便于帐号所有者使用文件。

```
$ pg dos.txt
```

```
12332##DISO##45.12^M
00332##LPSO##23.11^M
01299##USPD##34.46^M
```

可采取以下动作：

1) 用一个空格替换所有的（##）符号。

2) 删除起始域中最前面的0（00）。

3) 删除行尾控制字符（^M）。

一些系统中，回车符为^@和^L，如果遇到一些怪异的字符，不必担心，只要是在行尾并且全都相同就可以。

按步执行每一项任务，以保证在进行到下一任务前得到理想结果。使用输入文件 dos.txt。

任务1。 删除所有的#字符很容易，可以使用全局替换命令。这里用一个空格替换两个或更多的#符号。

```
$ sed 's/##*//g' dos.txt
12332 DISO 45.12^M
00332 LPSO 23.11^M
01299 USPD 34.46^M
```

任务2。 删除所有行首的0。使用^符号表示模式从行首开始，^0*表示行首任意个0。模式s/^0*//g设置替换部分为空，即为删除模式，正是要求所在。

```
$ sed 's/^0*//g' dos.txt
12332##DISO##45.12^M
332##LPSO##23.11^M
1299##USPD##34.46^M
```

任务3。 最后去除行尾 ^M符号，为此需做全局替换。设置替换部分为空。模式为：‘s/^m//g’，注意‘^M’，这是一个控制字符。

要产生控制字符（^M），需遵从前面产生F1键同样的处理过程。步骤如下；键入 sed s/，然后按住<Ctrl>键和v键，释放v键，再按住^键，并保持<Ctrl>键不动，再释放两个键，最后按<return>键。下面命令去除行尾^M字符。

```
$ sed 's/^M//g' dos.txt
```

分步测试预想功能对理解整个过程很有帮助。用 sed在移到下一步前测试本步功能及结果很重要。如果不这样，可能会有一大堆包含怪异字符的意料外的结果。

将所有命令结合在一起，使用管道将 cat命令结果传入一系列 sed命令，sed命令与上面几步精确过滤字符的sed相同。

```
$ cat dos.txt | sed 's/^0*//g' | sed 's/^M//g' | sed 's/##*/ /g'
12332 DISO 45.12
332 LPSO 23.11
1299 USPD 34.46
```

现在文件对帐号管理者可用。

可以将命令放在文件里，然后运行它。下面即为转换脚本。

```
$ pg dos.sed
#!/bin/sed -f
# name: dos.sed
# to call: dos.sed dos.txt

# get rid of the hash marks
```

cat -v dos.txt|sed 's/##*//g'|sed 's/^0*//g'|sed 's/\^M//'

```
s/##//g

# now get rid of the leading zeros
s/^0*//g
# now get rid of the carriage return
# the ^M is generated like we did for the F1 key.
s/\^M//g
```

通过仅指定一个sed命令可以将命令行缩短，本书后面部分介绍脚本中 sed的用法。

^M 標志是 windows 系統的分行標識符，在 linux 下可以這樣產生

Ctrl+V 鍵同時按下，先松開 V 鍵，再緊接著按下 M 鍵，再兩個按鍵同時松下。就會產生。可以按光標移動看每次跑動的格數爲一格說明是正確的，或者在 vi 下保存，使用 cat 正常查看不到時說明正確。

解決 windows 系統的^M 字符問題可以用 dos2unix 命令轉換文本

10.10.2 处理报文输出

当从数据库中执行语句输出时，一旦有了输出结果，脚本即可做进一步处理。通常先做一些整理，下面是一个sql查询结果。

```
Database        Size (MB)       Date Created
------------------------------------------
GOSOUTH         2244            12/11/97
TRISUD          5632            8/9/99

(2 rows affected)
```

为了使用上述输出信息做进一步自动处理，需要知道所存数据库名称，为此需执行以下操作：

1) 使用s/-*//g删除横线------。
2) 使用/^$/d删除空行。
3) 使用$d删除最后一行
4) 使用1d删除第一行。
5) 使用awk {print $1}打印第一列。

命令如下，这里使用了 cat，并管道传送结果到sed命令。

```
$ cat sql.txt | sed 's/--*//g' | sed '/^$/d' | sed '$d' | sed '1d' | awk
'{print $1}'
GOSOUTH
TRISUD
```

10.10.3 去除行首数字

对接下来卸载的这个文件实施的操作是去除行首所有数字，每个记录应以 UNH或UND开头，而不是UNH或UND前面的数字。文件如下：

```
$ pg UNH.txt
12345UND SPLLFC 234344
9999999UND SKKLT   3423
1UND SPLLY  434
...
```

使用基本正则表达式完成这个操作。 [0-9]代表行首任意数字，替换部分为空格是为了确保删除前面的匹配模式，即数字。

```
$ sed 's/^[0-9]//g' UNH.txt
UND SPLLFC 234344
UND SKKLT   3423
UND SPLLY  434
```

sed 's/^[0-9]*//g' UNH.txt

10.10.4 附加文本

当帐户完成设置一个文件时，帐号管理者可能要在文件中每个帐号后面加一段文字，下面是此类文件的一部分：

```
$ pg ok.txt
AC456
AC492169
AC9967
AC88345
```

任务是在每一行末尾加一个字符串‘passed’。

使用$命令修改各域会使工作相对容易些。首先需要匹配至少两个或更多的数字重复出现，这样将所有的帐号加进匹配模式。

```
$ sed 's/[0-9][0-9]*/& Passed/g' ok.txt
AC456 Passed
AC492169 Passed
AC9967 Passed
AC88345 Passed
```

修改數據，sed 命令進行的是向下方向移動的橫向對比操作，針對整個文本文件

awk 命令進行的是橫向移動的向下分域對比操作，針對每一條記錄(橫排)

10.10.5 从shell向sed传值

要从命令行中向sed传值，值得注意的是用双引号，否则功能不执行。

```
$ NAME="It's a go situation"
$ REPLACE="GO"
$ echo $NAME | sed "s/go/$REPLACE/g"
It's a GO situation
```

10.10.6 从sed输出中设置shell变量

从sed输出中设置 shell变量是一个简单的替换过程。运用上面的例子，创建 shell变量NEW-NAME，保存上述sed例子的输出结果。

```
$ NAME="It's a go situation"
$ REPLACE="GO"
$ NEW_NAME=`echo $NAME | sed "s/go/$REPLACE/g"`
$ echo $NEW_NAME
It's a GO situation
```

1.從 shell 命令行中向 sed 傳值要用雙引號

2.從 sed 輸出中設置 shell 變量要考慮命令執行順序，這里要用反引號，讓命令執行后賦值

3.sed 不區分大小寫，從上例可以看出

10.11 快速一行命令

下面是一些一行命令集。([]表示空格，[ ]表示tab键)

‘s/\.$//g’	删除以句点结尾行
‘-e /abcd/d’	删除包含abcd的行
‘s/[][][]*/[]/g’	删除一个以上空格，用一个空格代替
‘s/^[][]*//g’	删除行首空格
‘s/\.[][]*/[]/g’	删除句点后跟两个或更多空格，代之以一个空格
‘/^$/d’	删除空行
‘s/^.//g’	删除第一个字符
‘s/COL\(...\)//g’	删除紧跟COL的后三个字母
‘s/^\///g’	从路径中删除第一个\
‘s/[]/[]//g’	删除所有空格并用tab键替代
‘S/^[]//g’	删除行首所有tab键
‘s/[]*//g’	删除所有tab键

實例

在结束这一章前，看看一行脚本的一些例子。

1. 删除路径名第一个\符号

将当前工作目录返回给sed，删除第一个\:

```
cd /usr/local
$ echo $PWD| sed 's/^\///g'
$ usr/local
```

2. 追加/插入文本

将"Mr Willis"字串返回给sed并在Mr后而追加"Bruce"。

```
$ echo "Mr Willis" | sed 's/Mr /& Bruce/g'
Mr Bruce Willis
```

3. 删除首字符

sed删除字符串“accounts.doc”首字符。

```
$ echo "accounts.doc" | sed 's/^.//g'
$ ccounts.doc
```

4. 删除文件扩展名

sed删除“accounts.doc”文件扩展名。

```
$ echo "accounts.doc" | sed 's/.doc//g'
accounts
```

5. 增加文件扩展名

sed附加字符串“.doc”到字符串“accounts”。

```
$ echo "accounts" | sed 's/$/.doc/g'
accounts.doc
```

6. 替换字符系列

如果变量x含有下列字符串:

```
$ x="Department+payroll%Building G"
$ echo $x
$ Department+payroll%Building G
```

如果要实现下列转换:

+ to ‘of’
% to ‘located’

sed命令是:

```
$ echo $x | sed 's/\+/ of /g' | sed 's/\%/ Located at /g'
Department of payroll Located Building G
```

Sed 表達式里的所有特殊符號都要轉義，，單引號在這里并不能屏蔽其特殊含義。

10.12 小结

sed是一个强大的文本过滤工具。使用sed可以从文件或字符串中抽取所需信息。正像前面讲到的，sed不必写太长的脚本以取得所需信息。本章只讲述了sed的基本功能，但使用这些功能就可以执行许多任务了。

如果使用sed对文件进行过滤，最好将问题分成几步，分步执行，且边执行边测试结果。经验告诉我们，这是执行一个复杂任务的最有效方式。

下面有個網頁實例，

aaa正在搜索调试

bbb正在下载C#.NET 开发的 MIS 打印程序.rar

要取出 aaa、bbb 等等！

1.grep -oP '(?<=color="#0000b0">)[^<]*' file

2.sed 's/.*#0000b0">\(.*\)/\1/' file|awk -F '<' '{print $1}

3.sed 's/.*#0000b0">\(.*\)/\1/' file | sed 's/<\/font.*//'

4.sed 's/.*#0000b0">\([^<]*\).*/\1/' urfile

5.sed 's!</font.*$!!;s!^.*b0">!!' urfile

s/...\(..\).../\1/　這個格式，語法

\1 代表什么？

第11章　合并与分割

Sort ,uniq,join,paste, ,cut ,.split　　的用法

本章内容有：

- 实用的分类（sort）操作。
- uniq。
- join。
- cut 。
- paste。
- split。

11.1　sort用法

sort命令将许多不同的域按不同的列顺序分类。当查阅注册文件或为另一用户对下载文件重排文本列时，sort工具很方便。实际上，使用其他 UNIX工具时，已假定工作文件已经被分过类。无论如何，分类文件比不分类文件看起来更有意义。

11.1.2　sort选项

sort命令的一般格式为：

```
sort -cmu -o output_file [other options] +pos1 +pos2 input_files
```

下面简要介绍一下sort的参数：

-c 测试文件是否已经分类。

-m 合并两个分类文件。

-u 删除所有复制行。

-o 存储sort结果的输出文件名。

其他选项有：

-b 使用域进行分类时，忽略第一个空格。

-n 指定分类是域上的数字分类。

-t 域分隔符；用非空格或tab键分隔域。

-r 对分类次序或比较求逆。

+n n为域号。使用此域号开始分类。

n n为域号。在分类比较时忽略此域，一般与 +n一起使用。

post1 传递到m，n。m为域号，n为开始分类字符数；例如4，6意即以第5域分类，从第7个字符开始。

11.1.3 保存输出

-o选项保存分类结果，然而也可以使用重定向方法保存。下面例子保存结果到 results.out：

```
$ sort video.txt >results.out
```

11.1.4 sort启动方式

缺省情况下，sort认为一个空格或一系列空格为分隔符。要加入其他方式分隔，使用 -t选项。

sort执行时，先查看是否为域分隔设置了 -t选项，如果设置了，则使用它来将记录分隔成域0、域1等等；如果未设置，用空格代替。缺省时 sort将整个行排序，指定域号的情况例外。

11.1.5 sort对域的参照方式

关于sort的一个重要事实是它参照第一个域作为域 0，域1是第二个域，等等。sort也可以使用整行作为分类依据。为防止混淆，对于此文件用户应按如下方式参照域并做分类依据：

Field 0	Field 1	Field 2	Field 3
Star Wars	HK	301	4102
A Few Good Men	KL	445	5851

sort将定位各域，因此应把域0作为分类键0，域1作为分类键1等等。

sort 默認是對行進行分類，排列； 可以使用按域進行分類，-t 指定域分類符，不指定域時默認會返回基于第一域 0 的結果

下面是文件video.txt的清单，包含了上个季度家电商场的租金情况。各域为：（1）名称，（2）供货区代码，（3）本季度租金，（4）本年租金。域分隔符为冒号。为此对此例需使用‘-t’选项。文件如下：

```
$ pg video.txt
Boys in Company C:HK:192:2192
Alien:HK:119:1982
The Hill:KL:63:2972
Aliens:HK:532:4892
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Toy Story:HK:239:3972
```

使用 sort –c video.txt 測試文件是否已分類，沒有分類會返回錯誤，否則會回到提示符

11.1.8 sort分类求逆

如果要逆向sort结果，使用-r选项。在通读大的注册文件时，使用逆向 sort很方便。下面是按域0分类的逆向结果。

```
$ sort -t: -r video.txt
Toy Story:HK:239:3972
The Hill:KL:63:2972
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Boys in Company C:HK:192:2192
Aliens:HK:532:4892
Alien:HK:119:1982
```

11.1.9 按指定域分类

有时需要只按第 2域（分类键 1）分类。这里为重排报文中供应区代码，使用 t1，意义为按分类键1分类。下面的例子中，所有供应区代码按分类键 1分类；注意分类键 2和3对应各域也被分类。

```
$ sort -t: +1  video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
Toy Story:HK:239:3972
Star Wars:HK:301:4102
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
The Hill:KL:63:2972
```

sort +n 選項無法在 centos 下單獨使用，可以這樣 sort +n –n 詳見后

新命令使用-k 選項可以替用，sort -k start,end files (关键字开始列/域，关键字结束列/域)； k 選項可以只加起始列，且這個列/域是從 1 開始算起，而不是從 0。

上面的命令可改爲

sort –t: -k2 video.txt

-n 選項完全可以用-k 選項替用

11.1.10 数值域分类

依此类推，要按第三分类键分类，使用 t3。但是因为这是数值域，即为数值分类，可以使

用-n选项。下面例子为按年租金分类命令及结果：

```
$ sort -t: +3n  video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
The Hill:KL:63:2972
Toy Story:HK:239:3972
Star Wars:HK:301:4102
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
```

如果不加-n，结果会怎样？这里假定按第 3域分类，找出最好的季度租金。因为是分类键2，所以使用t2。

```
$ sort -t: +2 video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
Toy Story:HK:239:3972
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Aliens:HK:532:4892
The Hill:KL:63:2972
```

观察结果，分类进行了，但不是预想的结果，因为第 3域为数值域。当然这个结果也是某种类型的排列，录像机 The Hill应该在第二行，但结果是： sort只查看第3域每个数值的第一个数，并按其分类，然后再按第二个数依次下去。

记住按数值域分类要加-n，这样才会得到预想结果。

-n 按數值域分類，不加時，數值會先只比較數值的第一個數，相同時再比較第二個，這樣產生的結果不准備，因此 n 在數值域分類時不可以省的。

sort –t: -k4n video.txt

sort –t: -k3n video.txt

11.1.11 唯一性分类

有时，原文件中有重复行，这时可以使用 -u选项进行唯一性（不重复）分类以去除重复行，本例中Alien有相同的两行。带重复行的文件如下，其中 Alien插入了两次：

```
$ pg video.txt
Boys in Company C:HK:192:2192
Alien:HK:119:1982
The Hill:KL:63:2972
Aliens:HK:532:4892
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Alien:HK:119:1982
```

使用-u选项去除重复行，不必加其他选项， sort会自动处理。

```
$ sort -u video.txt
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
```

11.1.12 使用k的其他sort方法

sort还有另外一些方法指定分类键。可以指定 k选项，第1域（分类键）以1开始。不要与前面相混淆。我经常使用这个选项。因为我习惯于第一域为数值 1，这样使用sort时用同样的数值做分类依据会更有意义。其他选项也可以使用 k，主要用于指定分类域开始的字符数目。

要在第1域进行分类，可以使用-k4，这是按年租金分类的次序。

```
$ sort -t: -k4  video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
The Hill:KL:63:2972
Star Wars:HK:301:4102
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
```

11.1.13 使用k做分类键排序

可以指定分类键次序。先以第 4域，再以第1域分类，命令为-k4 -k1，也可以反过来，以便在文件首行显示最高年租金，方法如下：

```
$ sort -t: -r -k4 -k1 video.txt
A Few Good Men:KL:445:5851
Aliens:HK:532:4892
Star Wars:HK:301:4102
The Hill:KL:63:2972
Boys in Company C:HK:192:2192
Alien:HK:119:1982
```

11.1.14 指定sort序列

可以指定分类键顺序，也可以使用 -n选项指定不使用哪个分类键进行查询。看下面的 sort命令：

```
sort +0 -2 +3
```

该命令意即开始以域0分类，忽略域2，然后再使用域3分类。

sort -t: +0 -2 +3 video.txt 這條命令可以執行，真是怪。。測試發現，在出現加號時同時要出線減號，命令就不會報錯了。。如：sort -t: +3 -2 video.txt

11.1.15 pos用法

指定开始分类的域位置的另一种方法是使用如下格式：

sort +field_number .characters_in

意即从field_number开始分类，但是要在此域的第 characters_in个字符开始。这里是一个例子，供应区代码加入一些后缀。如：

```
$ pg video.txt
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
The Hill:KL23:63:2972
Aliens:HK11:532:4892
```

```
Star Wars:HK38:301:4102
A Few Good Men:KL87:445:5851
Toy Story:HK65:239:3972
```

要只使用供应区代码后缀部分将文件分类，其命令为 +1.2，意即以第1域最左边第3个字符开始分类，其具体含义及脚本如下：

	Field 0	Field 1	Field 2	Field 3
	Aliens	H K 1 1	532	4892
Characters in:		0 1 2 3		

```
$ sort -t: +1.2 video.txt
Aliens:HK11:532:4892
The Hill:KL23:63:2972
Star Wars:HK38:301:4102
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
Toy Story:HK65:239:3972
A Few Good Men:KL87:445:5851
```

sort –t: +1.2 -2 video.txt

sort 應用

11.1.16 使用head和tail将输出分类

分类操作时，不一定要显示整个文件或一页以查看 sort结果中的第一和最后一行。如果只显示最高年租金，按第4域分类-k4并求逆，然后使用管道只显示 sort输出的第一行，此命令为 head，可以指定查阅行数。如果只有第一行，则为 head -1：

```
$ sort -t: -r -k4  video.txt | head -1
A Few Good Men:KL:445:5851
```

要查阅最低年租金，使用 tail命令与head命令刚好相反，它显示文件倒数几行。 1为倒数一行，2为倒数两行等等。查阅最后一行为 tail -1。结合上述的 sort命令和 tail命令显示最低年租金：

```
$ sort -t: -r -k4 video.txt | tail -1
Alien:HK:119:1982
```

可以使用head或tail查阅任何大的文本文件， head用来查阅文件头，基本格式如下：

head [how_many_lines_to_display] file_name

Tail用来查阅文件尾，基本格式为：

tail [how_many_lines_to_display] file_name

如果使用head或tail时想省略显示行数，缺省时显示 10行。

要查阅文件前20行：

```
$ head -20 file_name
```

要查阅文件后7行：

```
$ tail -7 file_name
```

11.1.17 awk使用sort输出结果

对数据分类时，对sort结果加一点附加信息很有必要，对其他用户尤其如此。使用 awk可以轻松完成这一功能。比如说采用上面最低租金的例子，需要将 sort结果管道输出到awk，不要忘了用冒号作域分隔符，显示提示信息和实际数据。

```
$ sort -t: -r -k4 video.txt|tail -1 | awk -F: '{print "Worst rental
",$1,"has been rented "$3}'

Worst rental Alien has been rented 119
```

11.1.18 将两个分类文件合并

将文件合并前，它们必须已被分类。合并文件可用于事务处理和任何种类的修改操作。下面这个例子，因为忘了把两个家电名称加入文件，它们被放在一个单独的文件里，现在将之并入一个文件。分类的合并格式为‘ sort -m sorted_file1 sorted_file2，下面是包含两个新家电名称的文件列表，它已经分类完毕：

```
$ pg video2.txt
Crimson Tide:134:2031
Die Hard:152:2981
```

使用-m +o。将这个文件并入已存在的分类文件 video.sort，要以名称域进行分类，实际上没有必要加入+o，但为了保险起见，还是加上的好。

```
$ sort -t: -m +0 video2.txt video.sort
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
Crimson Tide:134:2031
Die Hard:152:2981
A Few Good Men:KL:445:5851
Star Wars:HK:301:4102
The Hill:KL:63:2972
```

使用-m 選項，合并兩個文件輸出 sort –m sorted_file1 sorted_file2

11.2 系统sort

sort可以用来对/etc/passwd文件中用户名进行分类。这里需要以第 1域即注册用户名分类，然后管道输出结果到awk，awk打印第一域。

```
$ cat passwd | sort -t: +0 | awk -F":" '{print $1}'
adm
bin
daemon
...
...
```

sort还可以用于df命令，以递减顺序打印使用列。下面是一般 df输出。

```
$ df
Filesystem   1024-blocks Used   Available Capacity Mounted on
/dev/hda5    495714      291027 179086    62%      /
/dev/hda1    614672      558896 55776     91%      /dos
```

使用-b选项，忽略分类域前面的空格。使用域 4 (+4)，即容量列将分类求逆，最后得出文件系统自由空间的清晰列表。

```
$ df | sort -b -r +4
Filesystem   1024-blocks Used   Available Capacity Mounted on
/dev/hda1    614672      558896 55776     91%      /dos
/dev/hda5    495714      291027 179086    62%      /
```

在一个文本文件中存入所有 IP地址的拷贝，这样查看本机 IP地址更容易一些。有时如果在管理员权限下，就需要将此文件分类。将 IP地址按文件中某种数值次序分类时，需要指定域分隔符为句点。这里只需关心 IP地址的最后一段。分类应从此域即域 3开始，未分类文件如下：

```
$ pg iplist
193.132.80.123  dave tansley
193.132.80.23   HP printer 2nd floor
193.132.80.198  JJ. Peter's scanner
193.132.80.38   SPARE
193.132.80.78   P.Edron
```

分类后结果如下：

```
$ sort -t.  +3n iplist
193.132.80.23   HP printer 2nd floor
193.132.80.38   SPARE
193.132.80.78   P.Edron
193.132.80.123  dave tansley
193.132.80.198  JJ. Peter's scanner
```

==========

11.3 uniq用法

uniq用来从一个文本文件中去除或禁止重复行。一般uniq假定文件已分类，并且结果正确。我们并不强制要求这样做，如果愿意，可以使用任何非排序文本，甚至是无规律行。

可以认为uniq有点像sort命令中唯一性选项。对，在某种程度上讲正是如此，但两者有一个重要区别。sort的唯一性选项去除所有重复行，而uniq命令并不这样做。重复行是什么？在uniq里意即持续不断重复出现的行，中间不夹杂任何其他文本，现举例如下：

```
$ pg myfile.txt
May Day
May Day
May Day
Going Down
May Day.
```

uniq将前三个May Day看作重复副本，但是因为第4行有不同的文本，故并不认为第五行持续的May Day为其副本。uniq将保留这一行。

命令一般格式：

uniq -u d c -f input-file output-file

其选项含义：

-u 只显示不重复行。

-d 只显示有重复数据行，每种重复行只显示其中一行

-c 打印每一重复行出现次数。

-f n为数字，前n个域被忽略。

uniq 默認爲去掉連接重復的行，-u 不顯示重復的行，.-u 選項在 uniq 里指去掉連續出現的行，在不連續出現時不會去掉，uniq 使用其它選項時，也均是計算不連續出現的情況；當直接不加任何選項是，默認爲將重復的行壓縮成一行。

注意：sort –u myfile.txt 與 uniq –u myfile.txt 結果是不一樣的，sort –u 是真正的唯一。。

一些系统不识别-f选项，这时替代使用-n。

使用本节开始时的文本，创建文件myfile.txt，在此文件上运行uniq命令。

```
$ uniq myfile.txt
May Day
Going Down
May Day
```

注意第5行保留下来，其文本为最后一行May Day。如果运行sort -u，将只返回May Day和Going Down。

使用-c选项显示行数，即每个重复行数目。本例中，行May Day重复出现三次。

```
$ uniq -c myfile.txt
      3 May Day
      1 Going Down
      1 May Day
```

1. 不唯一

使用-d显示重复出现的不唯一行：

```
$ uniq -d myfile.txt
May Day
```

2. 对特定域进行测试

使用-n只测试一行一部分的唯一性。例如 -5意即测试第5域后各域唯一性。域从 1开始记数。

如果忽略第1域，只测试第2域唯一性，使用-n2，下述文件包含一组数据，其中第 2域代表组代码。

```
$ pg parts.txt
AK123 OP
DK122 OP
EK999 OP
```

运行uniq，将返回所有行。因为这个文件每一行都不同。

```
$ uniq -c parts.txt
      1 AK123 OP
      1 DK122 OP
      1 EK999 OP
```

如果指定测试在第1域后，结果就会不同。uniq会比较三个相同的OP，因此将返回一行。

```
$ uniq -f2 parts.txt
AK123 OP
```

如果‘-f’返回错误，替代使用：

```
$ uniq -n2 parts.txt
AK123 OP
```

uniq –f2 parts.txt 測試域的唯一性，或者針對某個域進行唯一性比對

==========

11.4 join用法

join用来将来自两个分类文本文件的行连在一起。如果学过 SQL语言，可能会很熟悉join命令。

下面讲述join工作方式。这里有两个文件 file1和file2，当然已经分类。每个文件里都有一些元素与另一个文件相关。由于这种关系， join将两个文件连在一起，这有点像修改一个主文件，使之包含两个文件里的共同元素。

文本文件中的域通常由空格或 tab键分隔，但如果愿意，可以指定其他的域分隔符。一些系统要求使用join时文件域要少于20，为公平起见，如果域大于20，应使用DBMS系统。

为有效使用join，需分别将输入文件分类。

其一般格式为：

join [options] input-file1 input-file2.

让我们看看它的可用选项列表：

an n为一数字，用于连接时从文件 n中显示不匹配行。例如， -a1显示第一个文件的不匹配行，-a2为从第二个文件中显示不匹配行。

o n.m n为文件号，m为域号。1.3表示只显示文件1第三域，每个n，m必须用逗号分隔，如1.3，2.1。

j n m n为文件号，m为域号。使用其他域做连接域。

t 域分隔符。用来设置非空格或tab键的域分隔符。例如，指定冒号做域分隔符 -t:。

现有两个文本文件，其中一个包含名字和街道地址，称为name.txt，另一个是名字和城镇，为town.txt。

```
$ pg names.txt
M.Golls 12 Hidd Rd
P.Heller The Acre
P.Willey 132 The Grove
T.Norms 84 Connaught Rd
K.Fletch 12 Woodlea

$ pg town.txt
M.Golls Norwich NRD
P.Willey Galashiels GDD
T.Norms Brandon BSL
K.Fletch Mildenhall MAF
```

连接两个文件

连接两个文件，使得名字支持详细地址。例如 M.Golls记录指出地址为12 Hidd Rd。连接域为域0——名字域。因为两个文件此域相同， join将假定这是连接域：

```
$ join names.txt town.txt
M.Golls  12 Hidd Rd Norwich NRD
P.Willey 132 The Grove Galashiels GDD
T.Norms  84 Connaught Rd Brandon BSL
K.Fletch 12 Woodlea Mildenhall MAF
```

好，工作完成。缺省join删除或去除连接键的第二次重复出现，这里即为名字域。

1. 不匹配连接

如果一个文件与另一个文件没有匹配域时怎么办？这时 join不可以没有参数选项，经常指定两个文件的-a选项。下面的例子显示匹配及不匹配域。

```
$ join -a1  -a2 names.txt town.txt
M.Golls  12 Hidd Rd Norwich NRD
P.Heller The Acre
P.Willey 132 The Grove Galashiels GDD
T.Norms  84 Connaught Rd Brandon BSL
K.Fletch 12 Woodlea Mildenhall MAF
```

输出表明P.Heller不匹配第二个文件中任何一个记录。再运行这个命令，但指定只显示第一个文件中不匹配行：

```
$ join -a1 names.txt town.txt
```

2. 选择性连接

使用-o选项选择连接域。例如要创建一个文件仅包含人名及城镇， join执行时需要指定显示域。方式如下：

使用1.1显示第一个文件第一个域，2.2显示第二个文件第二个域，其间用逗号分隔。命令为：

```
$ join  -o 1.1,2.2 names.txt town.txt
M.Golls Norwich
P.Willey Galashiels
T.Norms Brandon
K.Fletch Mildenhall
```

要创建此新文件，将输出结果重定向到一个文件即可。

```
$ join  -o 1.1,2.2 names.txt town.txt >towns.txt
```

使用-jn m进行其他域连接，例如用文件1域3和文件域2做连接键，命令为：

```
join -j1 3 -j2 2 file1 file2
```

下面观察一个具体实例。有两个文件：

```
$ pg pers
P.Jones Office Runner   ID897
S.Round UNIX admin      ID666
L.Clip  Personl Chief   ID982

$ pg pers2
Dept2C  ID897   6 years
Dept3S  ID666   2 years
Dept5Z  ID982   1 year
```

文件pers包括名字、工作性质和个人ID号。文件pers2包括部门、个人ID号及工龄。连接应使用文件pers中域4，匹配文件pers2中域2，命令及结果如下：

```
$ join -j1 4 -j2 2 pers pers2
ID897 P.Jones Office Runner Dept2C 6 years
ID666 S.Round UNIX admin Dept3S    2 years
ID982 L.Clip Personl Chief Dept5Z  1 year
```

使用join应注意连接域到底是哪一个，比如说你认为正在访问域4，但实际上join应该访问域5，这样将不返回任何结果。如果是这样，用awk检查域号。例如，键入$ awk '{print $4}'文件名，观察其是否匹配假想域。

Join –o 1.1 2.2 name.txt town.txt

Join –j 1.1 –j 2.2 name.txt town.txt

第 1 個文件的第 1 域與第 2 個文件的第 2 域連接

Join –a1 [–a2] name.txt town.txt 顯示第 1 個文件 和第 2 個文件的不匹配選項

sort –m 的連接，只是把一個文件追加到另一個文件后，相似記錄放緊接著放

11.5 cut用法

cut用来从标准输入或文本文件中剪切列或域。剪切文本可以将之粘贴到一个文本文件。下一节将介绍粘贴用法。

cut一般格式为：

```
cut [options] file1 file2
```

下面介绍其可用选项：

-c list 指定剪切字符数。

-f field 指定剪切域数。

-d 指定与空格和tab键不同的域分隔符。

-c用来指定剪切范围，如下所示：

-c1，5-7 剪切第1个字符，然后是第5到第7个字符。

-c1-50 剪切前50个字符。

-f 格式与-c相同。

-f 1，5 剪切 第1域，第5域。

-f1，10-12 剪切第1域，第10域到第12域。

参照上一节中的文件‘pers’，现在从'pers'文件中剪切文本。使用冒号做其域分隔符。

```
$ pg pers
P.Jones:Office Runner:ID897
S.Round:UNIX admin:ID666
L.Clip:Personl Chief:ID982
```

11.5.1 使用域分隔符

文件中使用冒号“:”为域分隔符，故可用-d选项指定冒号，如-d:。如果有意观察第3域，可以使用-f3。要抽取ID域。可使用命令如下：

```
$ cut -d: -f3 pers
ID897T
ID666
ID982
```

11.5.2 剪切指定域

cut命令中剪切各域需用逗号分隔，如剪切域1和3，即名字和ID号，可以使用：

```
$ cut -d: -f1,3 pers
P.Jones:ID897
S.Round:ID666
L.Clip:ID982
```

要从文件/etc/passwd中剪切注册名及缺省根目录，需抽取域1和域3：

```
$ cut -d: -f1,6 /etc/passwd
gopher:/usr/lib/gopher-data
ftp:/home/ftp
peter:/home/apps/peter
dave:/home/apps/dave
...
```

使用-c选项指定精确剪切数目。这种方法需确切知道开始及结束字符。通常我不用这种方法，除非在固定长度的域或文件名上。

当信息文件传送到本机时，查看部分文件名就可以识别文件来源。要得到这条信息需抽取文件名后三个字符。然后才决定将之存在哪个目录下。下面的例子显示文件名列表及相应cut命令：

```
2231DG
2232DP
2236DK
```

```
$ ls 223*|cut -c4-6
1DG
2DP
6DK
```

如果使用ls -l命令作部分输出，情况将不同。需使用 -c选项。

```
-rw-r--r--   1 dave   admin        56 Apr 26 20:40 tr2.txt
-rw-r--r--   1 dave   admin        71 Apr 26 21:20 trpro.txt
```

要剪切字符，须计算 ls -l列表中的字符数。如显示权限用 cut -c1-10。然而这种方法可能相当慢，因此需要使用其他工具将相应信息抽取出来。要剪切谁正在使用系统的用户信息，方法如下：

```
$ who -u| cut -c1-8
root
dave
```

Cut 命令 –d 指定域分隔符 –f 指定剪切哪個域

11.6 paste用法

cut用来从文本文件或标准输出中抽取数据列或者域，然后再用 paste可以将这些数据粘贴起来形成相关文件。粘贴两个不同来源的数据时，首先需将其分类，并确保两个文件行数相同。

paste将按行将不同文件行信息放在一行。缺省情况下， paste连接时，用空格或tab键分隔新行中不同文本，除非指定 -d选项，它将成为域分隔符。

paste格式为；

```
paste -d -s -file1 file2
```

选项含义如下：

-d 指定不同于空格或tab键的域分隔符。例如用@分隔域，使用-d@。

-s 将每个文件合并成行而不是按行粘贴。

- 使用标准输入。例如ls -l |paste ，意即只在一列上显示输出。

从前面的剪切中取得下述两个文件：

```
$ pg pas1
ID897
ID666
ID982


$ pg pas2
P.Jones
S.Round
L.Clip
```

基本paste命令将之粘贴成两列：

```
$ paste pas1 pas2
ID897   P.Jones
ID666   S.Round
ID982   L.Clip
```

paste 命令可以在兩個文件沒有相關性的條件下合并，默認是按行合并，-s 選項是將每個文件合并成行(按列合并).-d 指定域分隔符 join 命令的合并時必需要有

相關聯的地方才行。

11.6.1 指定列

通过交换文件名即可指定哪一列先粘：

```
$ paste pas2 pas1
P.Jones ID897
S.Round ID666
L.Clip  ID982
```

11.6.2 使用不同的域分隔符

要创建不同于空格或tab键的域分隔符，使用-d选项。下面的例子用冒号做域分隔符。

```
$ paste -d: pas2 pas1
P.Jones:ID897
S.Round:ID66
```

要合并两行，而不是按行粘贴，可以使用 -s选项。下面的例子中，第一行粘贴为名字，第二行是ID号。

```
$ paste -s pas2 pas1
P.Jones S.Round L.Clip
ID897   ID666   ID982
```

11.6.3 paste命令管道输入

paste命令还有一个很有用的选项（ -）。意即对每一个（ -），从标准输入中读一次数据。使用空格作域分隔符，以一个4列格式显示目录列表。方法如下：

```
$ pwd
$ /etc
$ ls | paste -d" " - - - -
init.d rc rc.local rc.sysinit
rc0.d rc1.d rc2.d rc3.d
rc4.d rc5.d rc6.d
```

也可以以一列格式显示输出：

```
$ ls | paste -d"" -
init.d
rc
rc.local
rc.sysinit
rc0.d
rc1.d
...
```

11.7 split用法

split用来将大文件分割成小文件。有时文件越来越大，传送这些文件时，首先将其分割可能更容易。使用vi或其他工具诸如sort时，如果文件对于工作缓冲区太大，也会存在一些问题。因此有时没有选择余地，必须将文件分割成小的碎片。

split命令一般格式：

```
split -output_file-size input-filename output-filename
```

这里output-file-size指的是文本文件被分割的行数。split查看文件时，output-file-size选项指定将文件按每个最多1000行分割。如果有个文件有2800行，那么将分割成3个文件，分别有1000、1000、800行。每个文件格式为x[aa]到x[zz]，x为文件名首字母，[aa]、[zz]为文件名剩余部分顺序字符组合，下面的例子解释这一点。

假定文件bigone.txt有2800行，split命令产生下列文件：

```
$ split bigone.txt
xaa
xab
xac
```

文件大小为：

Size	Filename
1000	xaa
1000	xab
800	xac

可以使用output-file-size选项来分割文件。以下为一个6行文件。

```
$ pg split1
this is line1
this is line2
this is line3
this is line4
this is line5
this is line6
```

按每个文件2行分割，命令为：

```
$ split -2 split1
```

观察其结果。

```
$ ls -lt |head
total 205
-rw-r--r--   1 dave     admin          28 Apr 30 13:12 xaa
-rw-r--r--   1 dave     admin          28 Apr 30 13:12 xab
-rw-r--r--   1 dave     admin          28 Apr 30 13:12 xac
    ...
```

文件有6行，split按每个文件两行进行了分割，并按字母顺序命名文件。为进一步确信操作成功，观察一个新文件内容：

```
$ pg xac
this is line5
this is line6
```

split -3 video.txt video3.txt 會在生成的文件名后自動加 aa-zz 的后綴

第12章 tr 用 法

12.1 关于tr

tr用来从标准输入中通过替换或删除操作进行字符转换。 tr主要用于删除文件中控制字符或进行字符转换。使用tr时要转换两个字符串：字符串1用于查询，字符串2用于处理各种转换。tr刚执行时，字符串1中的字符被映射到字符串2中的字符，然后转换操作开始。

本章内容有：

- 大小写转换。
- 去除控制字符。
- 删除空行。

带有最常用选项的tr命令格式为：

```
tr -c -d -s ["string1_to_translate_from"] ["string2_to_triamnsulta_te_to"]
file
```

这里：

-c 用字符串1中字符集的补集替换此字符集，要求字符集为 ASCII。

-d 删除字符串1中所有输入字符。

-s 删除所有重复出现字符序列，只保留第一个；即将重复出现字符串压缩为一个字符串。

tr 作用

大小寫轉換

去除控制字符

刪除空行

標准格式如 tr “[a-z]” [A-Z]” inputfile

12.1.1 字符范围

使用tr时，可以指定字符串列表或范围作为形成字符串的模式。这看起来很像正则表达式，但实际上不是。指定字符串1或字符串2的内容时，只能使用单字符或字符串范围或列表。

[a-z] a-z内的字符组成的字符串。

[A-Z] A-Z内的字符组成的字符串。

[0-9] 数字串。

/octal 一个三位的八进制数，对应有效的 ASCII字符。

[O*n] 表示字符O重复出现指定次数n。因此[O*2]匹配OO的字符串。

大部分tr变种支持字符类和速记控制字符。字符类格式为 [: class]，包含数字、希腊字母、空行、小写、大写、 cntrl键、空格、点记符、图形等等。表 12-1包括最常用的控制字符的速记方式及三位八进制引用方式。

当用一个单字符替换一个字符串或字符范围时，注意字符并不放在方括号里（ []）。一些系统也可以使用方括号，例如可以写成 [“\012”]或 “\012”，tr也允许不加引号，因此命令中看到单引号而不是双引号时也不要感到奇怪。

像大多数系统工具一样，tr也受特定字符的影响。因此如果要匹配这些字符，需使用反斜

线屏蔽其特殊含义。例如，用\{指定花括号左边可以屏蔽其特殊含义。

表12-1　tr中特定控制字符的不同表达方式

速 记 符	含　义	八进制方式
\a	Ctrl-G 铃声	\007
\b	Ctrl-H 退格符	\010
\f	Ctrl-L 走行换页	\014
\n	Ctrl-J 新行	\012
\r	Ctrl-M 回车	\015
\t	Ctrl-I tab键	\011
\v	Ctrl-X	\030

12.1.2　保存输出

要保存输出结果，需将之重定向到一个文件。下面的例子重定向输出到文件　results.txt。输入文件是cops.txt。

```
$ tr -s "[a-z]"< oops.txt >results.txt
```

现在看一些例子。

12.1.3　去除重复出现的字符

下面文件包含了一些打印错误。这种情况时常发生，例如在 vi编辑器中，偶尔按住一个键不放。

```
$ pg oops.txt
And the cowwwwws went homeeeeeeee
Or did theyyyy
```

如果要去除重复字母或将其压缩在一起，使用 -s选项。因为都是字母，故使用[a-z]。输入文件重定向到tr命令。

```
$ tr -s "[a-z]"< oops.txt
And the cows went home
Or did they
```

所有重复字符被压缩成一个。如果使用 cat命令，再将结果管道输出至tr，结果是一样的。

```
$ cat oops.txt | tr -s "[a-z]"
And the cows went home
Or did they
```

12.1.4　删除空行

要删除空行，可将之剔出文件。下面是一个文件 plane.txt。文本间有许多空行。

```
$ pg plane.txt
987932  Spitfire


190992  Lancaster


238991  Typhoon
```

使用-s来做这项工作。换行的八进制表示为\012，命令为：

```
$ tr -s "[\012]" < plane.txt
987932  Spitfire
190992  Lancaster
238991  Typhoon
```

也可以使用换行速记方式\n，这里用单引号（通常用双引号）。

```
$ tr -s ["\n"] < plane.txt
987932  Spitfire
190992  Lancaster
238991  Typhoon
```

12.1.5 大写到小写

除了删除控制字符，转换大小写是tr最常用的功能。为此需指定即将转换的小写字符 [a-z]和转换结果[A-Z]。

第一个例子，tr从一个包含大小写字母的字符串中接受输入。

```
$ echo "May Day, May Day, Going Down.." | tr "[a-z]" "[A-Z]"
MAY DAY, MAY DAY, GOING DOWN..
```

同样，也可以使用字符类[：lower：]和[：upper：]。

```
$ echo "May Day, May Day, Going Down.." | tr "[:lower:]" "[:upper:]"
MAY DAY, MAY DAY, GOING DOWN..
```

将文本文件大写转换为小写并输出至一个新文件，格式为：

cat file-to-translate | tr "[A-Z]" "[a-z]" > new-file-name

这里file-to-translate保存即将转换的文件，new-file-name为保存结果的新文件名。例如：

cat myfile | tr "[A-Z]" "[a-z]" > lower_myfile

12.1.6 小写到大写

转换小写到大写与上一节大写到小写过程刚好相反。以下有两个例子：

```
$ echo " Look for the route, or make the route" | tr "[a-z]" "[A-Z]"
 LOOK FOR THE ROUTE, OR MAKE THE ROUTE

$ echo "May Day, May Day, Going Down.." | tr "[:upper:]" "[:lower:]"
may day, may day, going down..
```

将文本文件从小写转换为大写并将结果存入一个新文件，格式为：

cat file-to-translate | tr "[a-z]" "[A-Z]" > new-file-name

file-to-translate保存即将转换的文件，new-file-name保存结果文件，例如：

cat myfile | tr "[a-z]" "[A-Z]" > upper_myfile