shell基础 1：文件安全与权限
这里面可能用到用户管理方面的知识，可查看贴：
http://bbs.chinaunix.net/forum/viewtopic.php?p=2923303#2923303

主要有以下内容：

代码：

```
文件权限位
基本命令
chmod
suid/guid
chown
chgrp
umask 算法和用法
符号链接
```

当创建一个文件的时候，系统保存了有关该文件的全部信息，包括：
- 文件的位置。
- 文件类型。
- 文件长度。
- 哪位用户拥有该文件，哪些用户可以访问该文件。
- i 节点。
- 文件的修改时间。
- 文件的权限位。

让我们用 touch 命令创建一个文件：

代码：

```
$ touch temp
```

创建了一个空文件，现在用 ls -l 命令查看该目录下文件的属性（我这里用中文版)：
如下：

代码：

```
[root@Linux_chenwy temp]# ls -l
总用量 36
-rw-r--r--    1 root     root        34890 10 月 19 20:17 httpd.conf
-rw-r--r--    1 root     root            0 10 月 19 20:16 temp
```

代码:

总用量 36：是 ls 所列出的入口占用空间的字节数(以 K 为单位)。
1 该文件硬链接的数目。
root：文件属主。
root：文件属组（一般是文件属主所在的缺省组。）
34890：字节来表示的文件长度，记住，不是 K 字节！
10 月 19 20:17：件的更新时间。
temp or httd.conf ：件名。

sunsroad 写到:

BTW:要检查该目录所有文件占用的空间应该用这个命令：du。

譬如说前面说的 36 是如何计算出来：

首先我们要先了解你所用的文件系统的 IO BLOCK（中文叫作簇）为多少，在你所使用的这个文件系统的 IO BLOCK 大小是 4096 Bytes。

他意义是文件系统最小的读写及分配单位，每次读写操作你都不能小于这个尺寸。即使你的文件是只有一个字节。而且文件在硬盘上的存储也是以这个为单位，就是说如果文件尺寸小于这个值，那么它在磁盘上占用的空间就是 4096 字节。

占用空间的具体算法是:（进一（文件尺寸/4096））×4096。根据这个你就可以计算出你所列举的例子中的文件的空间使用状况：34890 除以 4096，大约等于 8.5，进一法取得为 9，就是说文件在磁盘上占用了 9 个 BLOCk，每个 BLOCK 为 4K，所以这两个文件占用的空间就是 36K。

这个规则也适合于目录，不过不会出现为 0 的目录，即使是空目录

-rw-r--r-- ：这是该文件的权限位。

第一个横杠：指定文件类型,表示该文件是一个普通文件。(所创建的文件绝大多数都是普通文件或符号链接文件)。

除去最前面的横杠，一共是 9 个字符，他们分别对应 9 个权限位。通过这些权限位，可以设定用户对文件的访问权限。对这两个文件的精确解释是：

代码：

```
rw-：前三位，文件属主可读、写
r--：中间三位,组用户可读
r--：最后三位,其他用户只可读
```

在创建的时候并未给属主赋予执行权限，在用户创建文件时，系统不会自动地设置执行权限位。这是出于加强系统安全的考虑

BTW：文件的属主组并不一定就是所有者所在的缺省组，而可以是任何一个跟该文件所有者无关的用户组。为了方便，还是统称属主，属组和其它

文件类型

前面提到的第一条横杠，表示该文件是普通文件型
文件类型有七种，它可以从 ls -l 命令所列出的结果的第一位看出.

七种类型：

代码：

```
d 目录。
l 符号链接(指向另一个文件)。
s 套接字文件。
b 块设备文件。
c 字符设备文件。
p 命名管道文件。
- 普通文件，或者更准确地说，不属于以上几种类型的文件。
```

文件的权限位中中每一组字符中含有三个权限位：

代码：

```
r 读权限
w 写/更改权限
x 执行该脚本或程序的权限
```

如：

代码：

```
r-- --- --- 文文件属主可读，但不能写或执行
r-- r-- --- 文文件属主和属组用户(一般来说，是文件属主所在的缺省组)可读
r-- r-- r- - 文任何用户都可读，但不能写或执行
rwx r-- r- - 文文件属主可读、写、执行，属组用户和其他用户只可读
rwx r-x --- 文文件属主可读、写、执行，属组用户可读、执
rwx r-x r- x 文文件属主可读、写、执行，属组用户和其他用户可读、执行
rw- rw- --- 文文件属主和属组用户可读、写
rw- rw- r- - 文文件属主和属组用户可读、写，其他用户可读
rw- rw- --- 文文件属主和属组用户及其他用户读可以读、写，慎用这种权限
设置，因为任何用户都可以写入该文件
```

sunsroad 写到:

```
文件的所有者组并非是文件所有者所在的缺省组，而可以是任何一个跟该文件所有者无关的用户组。
```

使用 chmod 来改变权限位

这一命令有符号模式和绝对模式。

符号模式

chmod 命令的一般格式为：

chmod [who] operator [permission] filename

w h o 的含义是：

代码:

```
u 文件属主权限。
g 属组用户权限。
o 其他用户权限。
a 所有用户(文件属主、属组用户及其他用户)。
```

o p e r a t o r 的含义：

代码:

```
+ 增加权限。
- 取消权限。
= 设定权限。
```

permission的含义：

代码:

```
r 读权限。
w 写权限。
x 执行权限。
s 文件属主和组 set-ID。
t 粘性位*。
l 给文件加锁，使其他用户无法访问。
u,g,o 针对文件属主、属组用户及其他用户的操作。

*在列文件或目录时，有时会遇到“ t”位。“t”代表了粘性位。如果在一个目录上出现“t”位，这就意味着该目录中的文件只有其属主才可以删除，即使某个属组用户具有和属主同等的权限。不过有的系统在这一规则上并不十分严格。

如果在文件列表时看到“ t”，那么这就意味着该脚本或程序在执行时会被放在交换区(虚存)。
```

对"t"还没弄清楚这是"sunsroad"的解释：

sunsroad 写到:

```
"t"权限用在文件上面是没有意义的，不是什么在交换区的概念，它跟文件的执行没有关系，而主要是为了文件共享设置的。
```

例如

代码:

```
chmod a-x temp //rw- rw- rw- 收回所有用户的执行权限
chmod og-w temp //rw- r-- r- - 收回属组用户和其他用户的写权限
chmod g+w temp //rw- rw- r- - 赋予属组用户写权限
chmod u+x temp //rwx rw- r- - 赋予文件属主执行权限
chmod go+x temp //rwx rwx r- x 赋予属组用户和其他用户执行权限
```

举如

当创建 temp 文件时，它具有这样的权限：

代码:

```
-rw-r--r--    1 root     root           0 10 月 19 20:16 temp
```

如果要使属主和属组用户具有有执行权限，并取消其他用户(所有其他用户)的写权限，可以用：

代码:

```
$ chmod ug+x temp
$ chmod o-w temp
```

这样，该文件的权限变为：

代码:

```
-rwxr--r--    1 root     root           0 10 月 19 20:16 temp
```

现在已经使文件属主对 temp 文件具有读、写执行的权限,属组用户真有读写权限，其它用户没有权限了。
绝对模式

chm d 命令绝对模式的一般形式为：

chmod [mode] file

其中 m o d e 是一个八进制数。
在绝对模式中，权限部分有着不同的含义。每一个权限位用一个八进制数来代表，如

代码:

```
0 4 0 0 文件属主可读
0 2 0 0 文件属主可写
0 1 0 0 文件属主可执行

0 0 4 0 属组用户可读
0 0 2 0 属组用户可写
0 0 1 0 属组用户可执行

0 0 0 4 其他用户可读
0 0 0 2 其他用户可写
0 0 0 1 其他用户可执行
```

在设定权限的时候，只需按照上面查出与文件属主、属组用户和其他用户所具有的权限相对应的数字，并把它们加起来，就是相应的权限表示。
可以看出，文件属主、属组用户和其他用户分别所能够具有的最大权限值就是 7。

再来看看前面举的例子：

代码：

```
-rwxr--r--  1   root          0 10 月 19 20:16 temp
```

相应的权限是：

代码：

```
rwx-：0400 + 0200 +0100 (文件属主可读、写、执行) = 0 7 0 0
r--：0 0 4 0 (属组用户可读) = 0 0 4 0
r--：0 0 4 0 (属组用户可读) = 0 0 4 0
0 7 4 4
```

有一个计算八进制权限表示的更好办法，如下：

代码：

```
文件属主：r w x：4 + 2 + 1
属组用户：r w x：4 + 2 + 1
其他用户：r w x：4 + 2 + 1
```

这上面这相，更容易地计算出相应的权限值，只要分别针对文件属主、属组用户和其他用户把相应权限下面的数字加在一起就可以了。

temp 文件具有这样的权限：

代码：

```
r w x    r - - r - -
4+2+1  4    4
```

把相应权限位所对应的值加在一起，就是 7 4 4。

如：

代码:

```
chmod 666 rw- rw- rw- 赋予所有用户读和写的权限
chmod 644 rw- r-- r- - 赋予所有文件属主读和写的权限，所有其他用户读权限
chmod 744 rwx r-- r- - 赋予文件属主读、写和执行的权限，所有其他用户读的权限
chmod 664 rw- rw- r- - 赋予文件属主和属组用户读和写的权限，其他用户读权限
chmod 700 rwx --- --- 赋予文件属主读、写和执行的权限
chmod 444 r-- r-- r- - 赋予所有用户读权限
```

下面举一个例子，假定有一个名为 temp 的文件，具有如下权限：

代码:

```
-rw-rw-r--  1   root          0 10 月 19 20:16 test1
```

现在希望对该文件可读、写和执行， root 组用户对该文件只读，可以键入：

代码:

```
$chmod 740 test1
$ls -l
-rwxr-----  1   root          0 10 月 19 20:16 test1
```

如果文件可读、写和执行，对其他所有用户只读，用：

代码:

```
$chmod 744 test1
$ls -l
-rwxr--r--  1   root          0 10 月 19 20:16 test1
```

如果希望一次设置目录下所有文件的权限，可以用：

代码:

```
$chmod 664*
$ls -l
-rw-r--r--  1   root          0 10 月 19 20:16 test1
```

这将使文件属主和属组用户都具有读和写的权限，其他用户只具有读权限。

还可以通过使用- R 选项连同子目录下的文件一起设置：

代码：

```
chmod -R 664 /temp/*
```

这样就可以一次将/ temp 目录下的所有文件连同各个子目录下的文件的权限全部设置为文件属主和属组用户可读和写，其他用户只读。使用- R 选项一定要谨慎，只有在需要改变目录树下全部文件权限时才可以使用。

目录

目录的权限位和文件有所不同。目录的读权限位意味着可以列出其中的内容。写权限位意味着可以在该目录中创建文件，如果不希望其他用户在你的目录中创建文件，可以取消相应的写权限位。执行权限位则意味着搜索和访问该目录.

代码：

```
r :可以列出该目录中的文件
w:可以在该目录中创建或删除文件
x:可以搜索或进入该目录
```

权限文件属主属组用户其他用户

代码：

```
drwx rwx r- x ( 775 ) 属主读、写、执行，属组读、写、执行，其它组读、执行
drwx r-x r- - ( 754 ) 属主读、写、执行，属组读、执行，其它组读
drwx r-x r- x ( 755 ) 属主读、写、执行，属组读、执行，其它组读、执行
```

如果把属组用户或其他用户针对某一目录的权限设置为- - x，那么他们将无法列出该目录中的文件。如果该目录中有一个执行位置位的脚本或程序，只要用户知道它的路径和文件名，仍然可以执行它。用户不能够进入该目录并不妨碍他的执行。

目录的权限将会覆盖该目录中文件的权限。例如，如果目录 temp 具有如下的权限：

代码：

```
drwxr--r--  1   admin          0 10 月 19 20:16 temp
```

而目录下的文件 myfile 的权限为：

代码:

```
-rwxrwxrwx  1   admin          0 10 月 19 20:16 myfile
```

那么 admin 组的用户将无法编辑该文件，因为它所属的目录不具有这样的权限。

该文件对任何用户都可读，但由于它所在的目录并未给 admin 组的用户赋予执行权限，所以该组的用户都将无法访问该目录，他们将会得到“访问受限”的错误消息。

suid / guid

1、为什么要使用这种类型的脚本？

例如有几个着几个大型的数据库系统，对它们进行备份需要有系统管理权限。可以写几个脚本，并设置了它们的 g u i d，这样就可以指定的一些用户来执行这些脚本就能够完成相应的工作，而无须以数据库管理员的身份登录，以免不小心破坏了数据库服务器。通过执行这些脚本，他们可以完成数据库备份及其他管理任务，但是在这些脚本运行结束之后，他们就又回复到他们作为普通用户的权限。

2、查找 suid / guid 命令

有相当一些 U N I X 命令也设置了 s u i d 和 g u i d。如果想找出这些命令，可以进入/ b i n 或/ s b i n 目录，执行下面的命令：

代码:

```
$ ls -l | grep '^...s'
```

上面的命令是用来查找 s u i d 文件的；

代码:

```
$ ls -l | grep '^...s..s'
```

上面的命令是用来查找 s u i d 和 g u i d 的。

3、设置 UID

代码:

```
设置 s u i d：将相应的权限位之前的那一位设置为 4；
设置 g u i d：将相应的权限位之前的那一位设置为 2；
两者都置位：将相应的权限位之前的那一位设置为 4+2=6。
```

设置了这一位后 x 的位置将由 s 代替。
记住：在设置 s u i d 或 g u i d 的同时，相应的执行权限位必须要被设置。
例如，如果希望设置 g u i d，那么必须要让该用户组具有执行权限。

如果想要对文件 l o g i n[它当前所具有的权限为 rwx rw- r-- (741)]设置 s u i d,，可在使用 c h m o d 命令时在该权限数字的前面加上一个 4，即 chmod 4741，这将使该文件的权限变为 r w s rw- r - -。

代码:

```
$ chmod 4741 login
```

设置 suid / guid 的例子

代码:

```
    命令          结果          含义
chmod 4755   rws r-x r- x   文件被设置了 s u i d，文件属主具有读、写和执行的权限，其他用户具有读和执行的权限
chmod 6711   rws --s --x    文件被设置了 s u i d 和 g u i d，文件属主具有读、写和执行的权限，其他用户具有执行的权限
chmod 4764   rws rw- r- -   文件被设置了 s u i d，文件属主具有读、写和执行的权限，属组用户具有读和执行的权限，用户具有读权限
```

4、还可以使用符号方式来设置 s u i d / g u i d。如果某个文件具有这样的权限： rwx r-x r-x，那么可以这样设置其 s u i d：

代码:

```
chmod u+s <filename>
```

于是该文件的权限将变为： rws r-x r-x

在查找设置了 s u i d 的文件时，没准会看到具有这样权限的文件：rwS r-x r- x，其中 S 为大写。
它表示相应的执行权限位并未被设置，这是一种没有什么用处的 s u i d 设置，可以忽略它的存在。

注意，c h m o d 命令不进行必要的完整性检查，可以给某一个没用的文件赋予任何权限，但 chmod 命令并不会对所设置的权限组合做什么检查。因此，不要看到一个文件具有执行权限，

就认为它一定是一个程序或脚本。

chown 和 chgrp

当你创建一个文件时，你就是该文件的属主。一旦你拥有某个文件，就可以改变它的所有权，把它的所有权交给另外一个/ e t c / p a s s w d 文件中存在的合法用户。可以使用用户名或用户 I D 号来完成这一操作。

在改变一个文件的所有权时，相应的 s u i d 也将被清除，这是出于安全性的考虑。只有文件的属主和系统管理员可以改变文件的所有权。一旦将文件的所有权交给另外一个用户，就无法再重新收回它的所有权。如果真的需要这样做，那么就只有求助于系统管理员了。

1、c h o w n 命令的一般形式为：

代码:

```
chmod -R -h owner file
```

引用:

- R 选项意味着对所有子目录下的文件也都进行同样的操作。
- h 选项意味着在改变符号链接文件的属主时不影响该链接所指向的目标文件。

2、chown 举例

代码:

```
如：
# ls -l
drwxrwxr-x    2 sam      sam         4096 10月 26 19:48 sam
# chown gem sam
# ls -l
drwxrwxr-x    2 gem      sam         4096 10月 26 19:48 sam
```

文件 sam 的所有权现在由用户 sam 交给了用户 gem。

3、chgrp 举例

c h g r p 命令和 c h o w n 命令的格式差不多，下面给出一个例子。

代码:

```
# ls -l
drwxrwxr-x    2 gem      sam         4096 10月 26 19:48 sam
# chgrp group sam
# ls -l
drwxrwxr-x    2 gem      group       4096 10月 26 19:48 sam
```

现在把该文件 sam 所属的组由 sam 变为 group。

4、找出你所属于的用户组

如果你希望知道自己属于哪些用户组，可以用 ID 这个命令：

代码:

```
# su sam
$ id
uid=506(sam) gid=4(adm) groups=4(adm)
```

5、找出其他用户所属于的组

代码:

```
# id
uid=0(root)                                                    gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
查看当前用户所属组

# id gem
uid=507(gem) gid=507(group) groups=507(group),0(root),4(adm)
查看其它用户所用组：#id 用户名

# su sam
$ id gem
uid=507(gem) gid=507(group) groups=507(group),0(root),4(adm)
查看其它用户所属组
```

这里书上用 group，但我试过不能使
BTW：可以用#cat /etc/passwd 和#cat /etc/group 直接查看用户和组

umask

当最初登录到系统中时， u m a s k 命令确定了你创建文件的缺省模式。这一命令实际上和 c h m o d 命令正好相反。你的系统管理员必须要为你设置一个合理的 u m a s k 值，以确保你创建的文件具有所希望的缺省权限，防止其他非同组用户对你的文件具有写权限。

在已经登录之后，可以按照个人的偏好使用 u m a s k 命令来改变文件创建的缺省权限。相应的改变直到退出该 s h e l l 或使用另外的 u m a s k 命令之前一直有效。

一般来说，u m a s k 命令是在/ e t c / p r o f i l e 文件中设置的，每个用户在登录时都会引用这个文件，所以如果希望改变所有用户的 u m a s k，可以在该文件中加入相应的条目。如果希望永久性地设置自己的 u m a s k 值，那么就把它放在自己$ H O M E 目录下的. p r o f i l e 或. b a s h _ p r o f i l e 文件中。

如何计算 umask 值

u m a s k 命令允许你设定文件创建时的缺省模式，对应每一类用户(文件属主、属组、其他用户)存在一个相应的 u m a s k 值中的数字。对于文件来说，这一数字的最大值分别是 6。系统不允许你在创建一个文本文件时就赋予它执行权限，必须在创建后用 c h m o d 命令增加这一权限。目录则允许设置执行权限，这样针对目录来说， u m a s k 中各个数字最大可以到 7。该命令的一般形式为：

代码:

```
umask nnn
```

其中 n n n 为 u m a s k 置 0 0 0 - 7 7 7。

计算 u m a s k 值：

可以有几种计算 u m a s k 值的方法，通过设置 u m a s k 值，可以为新创建的文件和目录设置缺省权限。

与权限位相对应的 u m a s k 值。

代码:

```
umask     文件     目录
0     6     7
1      6     6
2     4     5
3     4     4
4     2     3
5     2     2
```

```
6     0     1
7     0     0
```

在计算 u m a s k 值时，可以针对各类用户分别按上面那张表中按照所需要的文件/目录创建缺省权限查找对应的 u m a s k 值。
例如，u m a s k 值

代码:

```
0     6     7
0     6     7
2     4     5
```

所以 002 所对应的文件和目录创建缺省权限分别为 6 6 4 和 7 7 5。

还有另外一种计算 u m a s k 值的方法。我们只要记住 u m a s k 是从权限中“拿走”相应的位即可。

代码:

```
umask     文件     目录
0     6     7
1      6      6
2     4     5
3     4     4
4     2     3
5     2     2
6     0     1
7     0     0
```

例如，对于 u m a s k 值 0 0 2，相应的文件和目录缺省创建权限是什么呢？

引用:

> 第一步，我们首先写下具有全部权限的模式，即 7 7 7 (所有用户都具有读、写和执行权限)。
>
> 第二步，在下面一行按照 u m a s k 值写下相应的位，在本例中是 0 0 2。
>
> 第三步，在接下来的一行中记下上面两行中没有匹配的位。这就是目录的缺省创建权限。
>
> 稍加练习就能够记住这种方法。
>
> 第四步，对于文件来说，在创建时不能具有文件权限，只要拿掉相应的执行权限比特即可。

这就是上面的例子，其中 u m a s k 值为 0 0 2：

引用:

1) 文件的最大权限 rwx rwx rwx (777)

2) umask 值为 0 0 2 - - - - - - -w-

3) 目录权限 rwx rwx r-x (775) 这就是目录创建缺省权限

4) 文件权限 rw- rw- r-- (664) 这就是文件创建缺省权限

下面是另外一个例子，假设这次 u m a s k 值为 0 2 2：

引用:

1) 文件的最大权限 rwx rwx rwx (777)

2) u m a s k 值为 0 2 2 - - - -w- -w-

3) 目录权限 rwx r-x r-x (755) 这就是目录创建缺省权限

4) 文件权限 rw- r-- r-- (644) 这就是文件创建缺省权限

如果想知道当前的 umask 值，可以使用 u m a s k 命令：

代码:

```
#su sam     /*切换到 sam 用户玩境下
#umask     /*查看 sam 的 umask
0022
前面多了个 0，是 suid/guid 的
```

代码:

```
$ touch file1
$ mkdir file2
$ ls -l
总用量 8
-rw-r--r--    1 sam      adm            0 10 月 30 19:27 file1
drwxr-xr-x    2 sam      adm         4096 10 月 30 19:28 file2
```

引用:

新建文件 file1 和目录 file2，查看新建文件和目录的默认权限,umask 为 022 时

目录权限 rwx r-x r-x (755)

```
文件权限 rw- r-- r-- (644)
```

更改 umask 默认值由 022 至 002

代码:

```
$ umask 002
$ touch file3
$ ls -l
总用量 12
-rw-r--r--    1 sam      adm            0 10 月 30 19:27 file1
drwxr-xr-x    2 sam      adm         4096 10 月 30 19:28 file2
-rw-rw-r--    1 sam      adm            0 10 月 30 19:34 file3
drwxrwxr-x    2 sam      adm         4096 10 月 30 19:34 file4
```

引用:

```
可以看到，新建文件和目录的默认权限改变了，umask 为 002 时
目录权限 rwx rwx r-x (775)
文件权限 rw- rw- r-- (664)
```

可以看见，$ umask 002 已生效

选项说明

代码:

```
bash

        -S

以 "u=rwx,g=rx,o=rx" 这种较人性的格式取代数字显示。
```

代码:

```
-p

当使用 -p 选项，但 mode 省略，输出格式为 umask mode (可以做为下 umask 指令使用)。当模式改变成功，或 mode 参数被省略，执行的状态值为 0。否则状态值为 1。
```

实例说明

bash

首先，让我们先显示目前环境的 umask 设定情况

代码:

```
# umask
0022
#
```

得到的数值为 "0022"。所以，建立新档案的预设权限是 644，目录则是 755。如果不习惯看数字，我们可以使用 -S 选项来显示设定值

代码:

```
# umask -S
u=rwx,g=rx,o=rx
#
```

实际建个档案与目录看看

代码:

```
# ls > fileA
# mkdir dirB
# ls -l
总计 8
drwxr-xr-x    2 root     root         4096 12 月 21 16:42 dirB
-rw-r--r--    1 root     root            6 12 月 21 16:42 fileA
#
```

可以看到文件和目录的不同

代码:

```
tcsh

$ umask
22
$
```

```
$ umask 000
$ umask
0
$
```

从以上，我们可以知道，tcsh 简单到连 0 都懒的显示...

要设置 umask 值，使文件所有者具有读写执行权限，属组的其他用户具有只读权限，除此之外的其他用户没有访问权限 (-rw-r-------)，请输入以下内容：

代码:

```
$ umask u=rwx,g=r,o=r
$ touch file5
$ mkdir file6
$ ls -l
总用量 16
-rw-r--r--    1 sam      adm            0 10月 30 19:48 file5
drwxr--r--    2 sam      adm         4096 10月 30 19:48 file6
```

要确定当前的 umask 设置，请键入：

代码:

```
$ umask -S
u=rwx,g=r,o=r
```

符号链接

存在两种不同类型的链接，软链接和硬链接。修改其中一个，硬连接指向的是节点(inode),而软连接指向的是路径(path)

软链接文件

软链接又叫符号链接，这个文件包含了另一个文件的路径名。可以是任意文件或目录，可以链接不同文件系统的文件。和 win 下的快捷方式差不多。链接文件甚至可以链接不存在的文件，这就产生一般称之为"断链"的问题(或曰“现象")，链接文件甚至可以循环链接自己。类似于编程语言中的递归。

命令格式：

代码：

```
ln [-s] source_path target_path
```

硬链接文件

info ln 命令告诉您，硬链接是已存在文件的另一个名字，硬连接的命令是

代码：

```
ln -d existfile newfile
```

引用：

> 硬链接文件有两个限制
>
> 1、不允许给目录创建硬链接；
>
> 2、只有在同一文件系统中的文件之间才能创建链接。

对硬链接文件进行读写和删除操作时候，结果和软链接相同。但如果我们删除硬链接文件的源文件，硬链接文件仍然存在，而且保留了愿有的内容。这时，系统就“忘记”了它曾经是硬链接文件。而把他当成一个普通文件。修改其中一个，与其连接的文件同时被修改

举例 说明：

代码：

```
$umask 022
$ cp /etc/httpd/conf/httpd.conf /usr/sam
```

原来前面做的试验，改变了系统默认的 umask 值，现在改回来为 022,
举个 httpd.conf 文件做例 子

代码：

```
$ ln httpd.conf httpd1.conf
$ ln -s httpd.conf httpd2.conf
```

第一条为硬链接，第二条为软链接

代码：

```
$ ls -li
```

代码：

```
总用量 80
1077669  -rw-r--r--   2 sam     adm       34890 10月 31 00:57 httpd1.conf
1077668  lrwxrwxrwx   1 sam     adm          10 10月 31 00:58 httpd2.conf
-> httpd.conf
1077669  -rw-r--r--   2 sam     adm       34890 10月 31 00:57 httpd.conf
```

可以看到，使用 ls -li，软连接只产生了 10 字节的快捷而已，硬连接却实实在在的的拷贝。**最前面的 inode 硬链接和源文件是一样的，而软链接不一样，具体看一下回复**这话有误，但先这么理解，具体请看下面的回复

对 http1.conf 进行编辑，可以发现 httpd.conf 也发生了一样的变化

代码：

```
$ rm httpd.conf
```

现在删除链接的源文件，来比较不同之处

代码：

```
$ ls -l
总用量 44
drw-r--r--   2 sam     adm       4096 10月 30 20:14 file6
-rw-r--r--   1 sam     adm      34890 10月 31 00:57 httpd1.conf
lrwxrwxrwx   1 sam     adm         10 10月 31 00:58 httpd2.conf ->
httpd.conf
```

发现，httpd2.conf 实际已经不存在了，是断链，而 httpd1.conf 变也了普通文件
转一个：更好理解

索引节点、硬连接和连接计数

索引节点 inode：

引用：

> Linux 为每个文件分配一个称为索引节点的号码 inode，可以将 inode 简单理解成一个指针，它永远指向本文件的具体存储位置。系统是通过索引节点(而不是文件名)来定位每一个文件。

例如：

假设我们在硬盘当前目录下建立了一个名为 mytext 文本文件，其内容只有一行：

代码：

```
This is my file.
```

代码：

```
1、当然这行文字一定是存储在磁盘数据区某个具体位置里(物理上要通过磁头号、柱面号和扇区号来描述，在本例中假设分别是 1、20、30)。

2、假设其 inode 是 262457，那么系统通过一段标准程序，就能将这个 inode 转换成存放此文件的具体物理地址(1 磁头、20 柱面、30 扇区)，最终读出文件的内容：“This is my file.”

3、所以 inode 是指向一个文件数据区的指针号码，一个 inode 对应着系统中唯一的一片物理数据区，而位于两个不同物理数据区的文件必定分别对应着两个不同的 inode 号码。
```

文件拷贝命令与硬链接的区别：

代码：

```
# cp /home/zyd/mytext newfile
```

在当前工作目录建立了一个新文件 newfile，其实际操作主要包括如下三步：

引用：

> 1、在当前目录中增加一个目录项，其文件名域填入 newfile，并分配了一个新的 inode，假设是 262456。
> 2、将原文件(在 1 磁头、20 柱面、30 扇区)的内容复制了一份到新的空闲物理块(假设是 1 磁头、20 柱面、31 扇区)。
> 3、填写一些其他关键信息，使系统通过这些信息及 inode 号码可以完成物理地址的转换。

所以文件复制要分配新的 inode 和新的数据区，虽然两个文件的内容是一样的。

硬连接 hardlink：

引用：

> 我们实际使用文件时一般是通过文件名来引用的。通过上面的讨论，我们知道：
>
> 1 个 inode 号码肯定和一片完全属于一个文件的数据区一一对应。那么一个文件系统中两个或更多个不同的文件名能否对应同一个文件呢？答案是肯定的。
>
> 我们知道 inode 号码是记录在文件名对应的目录项中的，我们可以使两个或多个文件的目录项具有相同的 inode 值，实际上就使它们对应着同一个文件。
>
> 有几个目录项具有相同的 inode 号，我们就说这个文件有几个硬连接(hardlink)，
>
> 对于普通文件，ls -l 命令的连接计数 count 域的数值就是本文件拥有的硬连接数。硬连接可以通过 ln 命令建立，

例如：

代码：

```
# ln /home/zyd/mytext hardlink_mytext
```

就建立了一个新的文件 hardlink_mytext，这个文件的 inode 同样是 262457。建立硬连接实际上只是增加了一个目录项，但并复制文件数据区，原文件的数据区由两个文件共享。这一方面能够节约大量磁盘空间，同时可以保证两个文件能同步更新。

代码：

```
'ls -il'可以显示文件的 inode(在下面最左边)：

262456 -rw-rw-r-- 1 zyd zyd 17 Nov 3 14:52 newfile
262457 -rw-rw-r-- 2 zyd zyd 17 Nov 3 14:50 hardlink_mytext
262457 -rw-rw-r-- 2 zyd zyd 17 Nov 3 14:50 mytext
```

连接计数 count：

前面我们介绍了，文件的连接计数域表明本系统中共有几个文件目录项的 inode 和本文件相同，也就是本文件共有几个硬连接。如上面的例子中 hardlink_mytext 和 mytext 文件的 count 值都是 2。

那么对于目录，其 count 域的含义是什么呢？目录的 count 同样表示共有多少个目录项指向此目录，不过要详细说明必须进一步解释 VFS 文件系统的结构，为简单起见，只要这样理解就行了：(count-2)等于本目录包含的直接子目录数(就是只包括儿子，不包括孙子啦！)。

代码：

例如：如果一个目录/abc 的 count 域为 5，那么/abc 目录一定包含 3 个子目录。

引用：

进一步说明：

硬连接文件实际上并不是一种新的文件类型，两个文件互为对方的硬连接。它们应该都是普通文件(谁能告诉我：其它类型的文件可以硬连接吗？)。两个文件除了名称或/和文件目录不同外，其它部分完全相同，更改了一个文件，另一个的文件长度、内容、更改时间等都将相应发生变化，更改了一个文件的权限位 mode，另一个也会发生同样的变化。

引用：

注意连接计数字段 count，互为硬连接的两个文件的 count 值都是 2，表明有两个 inode 指向同一文件的 inode。

当我们删除其中一个文件时，系统首先将(count-1)->count，如果结果是零，就将其目录项和数据区都删除，否则只将本目录项删除，数据区仍然保留，仍然可以通过另外的文件名访问。根据这个特性，可以通过为重要的文件建立硬连接的方法来防止其被误删除。

一个文件系统允许的 inode 节点数是有限的，如果文件数量太多，即使每个文件都是 0 字节的空文件，系统最终也会因为节点空间耗尽而不能再创建文件。所以当发现不能建立文件时首先要考虑硬盘数据区是否还有空间(可通过 du 命令)，其次还得检查节点空间。

引用：

互为硬连接的多个文件必须位于同一个文件系统上。根设备及任何一个需要 mount 才能挂接进来的分区、软盘、NFS、光驱等都是一个独立的文件系统，每个文件系统有一个相应的设备号，不同文件系统中具有相同 inode 节 点的文件间没有任何联系。系统则通过设备号和 inode 号的组合唯一确定一个文件。

Linux 之所以能支持多种文件系统，其实是由于 Linux 提供了一个虚拟文件系统 VFS，VFS 作为实际文件系统的上层软件，掩盖了实际文件系统底层的具体结构差异，为系统访问位于不同文件系统的文件提供了一个统一的接 口。

实际上许多文件系统并不具备 inode 结构，其目录结构也和以上的讨论不同，但通过 VFS，系统均为其提供了虚拟一致的 inode 和目录项结构。

所以，'ls -il'命令实际显示的 inode 应该是 VFS inode，也就是说，inode 是存在于内存中的数据结构，而不一定是实际的硬盘结构。

但为 Linux 量身定做的 ext2 文件系统具备实际的 inode 和连接型目录项结构,所以,对于 ext2 文件系统，可以认为我们上面讨论的关于硬连接的概念是完全正确的

shell 基础二：查找技巧,find 及 xargs 的使用。

由于 f i n d 具有强大的功能，所以它的选项也很多，其中大部分选项都值得我们花时间来了解一下。即使系统中含有网络文件系统(N F S)，f i n d 命令在该文件系统中同样有效，只你具有相应的权限。
在运行一个非常消耗资源的 f i n d 命令时，很多人都倾向于把它放在后台执行，因为遍历一个大的文件系统可能会花费很长的时间(这里是指 3 0 G 字节以上的文件系统)。

F i n d 命令的一般形式为：

代码:

```
find pathname -options [-print -exec -ok]
```

让我们来看看该命令的参数：

引用:

> pathname: find 命令所查找的目录路径。例如用.来表示当前目录，用/来表示系统根目录。
> -print： find 命令将匹配的文件输出到标准输出。
> -exec： find 命令对匹配的文件执行该参数所给出的 s h e l l 命令。相应命令的形式为'command' {} \;，注意{ }和\；之间的空格。
> -ok： 和- e x e c 的作用相同，只不过以一种更为安全的模式来执行该参数所给出的 s h e l l 命令，在执行每一个命令之前，都会给出提示，让用户来确定是否执行。

先了解一下 find 所带的参数，能实现哪些功能

find 命令选项

-name：按照**文件名**查找文件。
-perm：按照**文件权限**来查找文件。
-prune：使用这一选项可以使ｆｉｎｄ命令**不在当前指定的目录中查找**，如果同时使用-depth选项，那么-prune将被ｆｉｎｄ命令忽略。
-user： 按照**文件属主**来查找文件。
-group：按照**文件所属的组**来查找文件。

-mtime -n +n：按照**文件的更改时间**来查找文件， - n表示文件更改时间距现在n天以内，+ n表示文件更改时间距现在n天以前。Ｆｉｎｄ命令还有-ａｔｉｍｅ和-ｃｔｉｍｅ选项，但它们都和-ｍｔｉｍｅ选项。

-nogroup：查找**无有效所属组的文件**，即该文件所属的组在/ｅｔｃ/ｇｒｏｕｐｓ中不存在。
-nouser：查找**无有效属主的文件**，即该文件的属主在/ｅｔｃ/ｐａｓｓｗｄ中不存在。

-newer file1 ！file2：**查找更改时间比文件ｆｉｌｅ１新但比文件ｆｉｌｅ２旧的文件。**

-type 查找某一类型的文件，诸如：

代码：

```
b - 块设备文件。
d - 目录。
c - 字符设备文件。
p - 管道文件。
l - 符号链接文件。
f - 普通文件。
```

-size n：[c] 查找**文件长度为 n 块**的文件，带有ｃ时表示文件长度以字节计。
-depth：**在查找文件时，首先查找当前目录中的文件，然后再在其子目录中查找。**
-fstype：查找位于**某一类型文件系统中的文件**，这些文件系统类型通常可以在配置文件/ｅｔｃ/ｆｓｔａｂ中找到，该配置文件中包含了本系统中有关文件系统的信息。

引用：

> -mount：在查找文件时不跨越文件系统ｍｏｕｎｔ点。
>
> -follow：如果ｆｉｎｄ命令遇到符号链接文件，就跟踪至链接所指向的文件。
>
> -cpio：对匹配的文件使用ｃｐｉｏ命令，将这些文件备份到磁带设备中。

使用 exec 或 ok 来执行 shell 命令

引用：

> 使用 find 时，只要把想要的操作写在一个文件里，就可以用 exec 来配合 find 查找，很方便的

（在有些操作系统中只允许-exec选项执行诸如ls或ls -l这样的命令）。大多数用户使用这一选项是为了查找旧文件并删除它们。建议在真正执行rm命令删除文件之前，最好先用ls命令看一下，确认它们是所要删除的文件。

exec选项后面跟随着所要执行的命令或脚本，然后是一对儿{ }，一个空格和一个\，最后是一个分号。

为了使用exec选项，必须要同时使用print选项。如果验证一下find命令，会发现该命令只输出从当前路径起的相对路径及文件名。

例如：为了用ls -l命令列出所匹配到的文件，可以把ls -l命令放在find命令的-exec选项中

代码:

```
# find . -type f -exec ls -l {} \;
-rw-r--r--    1 root     root        34928 2003-02-25  ./conf/httpd.conf
-rw-r--r--    1 root     root        12959 2003-02-25  ./conf/magic
-rw-r--r--    1 root     root          180 2003-02-25  ./conf.d/README
```

上面的例子中，find命令匹配到了当前目录下的所有普通文件，并在-exec选项中使用ls -l命令将它们列出。

在/logs目录中查找更改时间在5日以前的文件并删除它们：

代码:

```
$ find logs -type f -mtime +5 -exec rm {} \;
```

记住，在shell中用任何方式删除文件之前，应当先查看相应的文件，一定要小心！当使用诸如mv或rm命令时，可以使用-exec选项的安全模式。它将在对每个匹配到的文件进行操作之前提示你。

在下面的例子中， find命令在当前目录中查找所有文件名以.LOG结尾、更改时间在5日以上的文件，并删除它们，只不过在删除之前先给出提示。

代码:

```
$ find . -name "*.conf"  -mtime +5 -ok rm {} \;
< rm ... ./conf/httpd.conf > ? n
```

按y键删除文件，按n键不删除。

任何形式的命令都可以在- e x e c 选项中使用。

在下面的例子中我们使用 g r e p 命令。f i n d 命令首先匹配所有文件名为“ passwd*”的文件，例如 passwd、passwd.old、passwd.bak，然后执行 grep 命令看看在这些文件中是否存在一个 sam 用户。

代码:

```
# find /etc -name "passwd*" -exec grep "sam" {} \;
sam:x:501:501::/usr/sam:/bin/bash
```

论坛里-exec 执行脚本的的例子
使用 find 命令查找某个时间段的 11 点到 12 点的 shell
find 命令的例子

查找当前用户主目录下的所有文件，下面两种方法都可以使用：

代码:

```
$ find $HOME -print
$ find ~ -print
```

为了在当前目录中文件属主具有读、写权限，并且文件所属组的用户和其他用户具有读权限的文件，可以用：

代码:

```
$ find . -type f -perm 644 -exec ls -l {} \;
```

为了查找系统中所有文件长度为 0 的普通文件，并列出它们的完整路径，可以用：

代码:

```
$ find / -type f -size 0 -exec ls -l {} \;
```

查找/var/logs 目录中更改时间在 7 日以前的普通文件，并在删除之前询问它们:

代码:

```
$ find /var/logs -type f -mtime +7 -ok rm {} \;
```

为了查找系统中所有属于 root 组的文件，可以用：

代码:

```
$find . -group root -exec ls -l {} \;
```

```
-rw-r--r--    1 root     root          595 10月 31 01:09 ./fie1
```

下面的 find 命令将删除当目录中访问时间在 7 日以来、含有数字后缀的 admin.log 文件。该命令只检查三位数字，所以相应文件的后缀不要超过 999。
先建几个 admin.log*的文件 ，才能使用下面这个命令

代码:

```
$ find . -name "admin.log[0-9][0-9][0-9]" -atime -7  -ok
rm {} \;
< rm ... ./admin.log001 > ? n
< rm ... ./admin.log002 > ? n
< rm ... ./admin.log042 > ? n
< rm ... ./admin.log942 > ? n
```

为了查找当前文件系统中的所有目录并排序，可以用：

代码:

```
$ find . -type d  |sort
```

为了查找系统中所有的 r m t 磁带设备，可以用：

代码:

```
$ find /dev/rmt -print
```

代码:

```
原书为：
为了查找当前文件系统中的所有目录并排序，可以用：
$ find . -type d  -loacl -mount |sort
已更正为：
$ find . -type d |sort
```

xargs

在使用 f i n d 命令的- e x e c 选项处理匹配到的文件时， f i n d 命令将所有匹配到的文件一起传递给 e x e c 执行。但有些系统对能够传递给 e x e c 的命令长度有限制，这样在 f i n d 命令运行几分钟之后，就会出现溢出错误。错误信息通常是“参数列太长”或“参数列溢出”。这就是 x a rg s 命令的用处所在，特别是与 f i n d 命令一起使用。

Find命令把匹配到的文件传递给xargs命令，而xargs命令每次只获取一部分文件而不是全部，不像-exec选项那样。这样它可以先处理最先获取的一部分文件，然后是下一批，并如此继续下去。

在有些系统中，使用-exec选项会为处理每一个匹配到的文件而发起一个相应的进程，并非将匹配到的文件全部作为参数一次执行；这样在有些情况下就会出现进程过多，系统性能下降的问题，因而效率不高；
而使用xargs命令则只有一个进程。另外，在使用xargs命令时，究竟是一次获取所有的参数，还是分批取得参数，以及每一次获取参数的数目都会根据该命令的选项及系统内核中相应的可调参数来确定。

来看看xargs命令是如何同find命令一起使用的，并给出一些例子。

下面的例子查找系统中的每一个普通文件，然后使用xargs命令来测试它们分别属于哪类文件

代码:

```
#find . -type f -print | xargs file
./.kde/Autostart/Autorun.desktop: UTF-8 Unicode English text
./.kde/Autostart/.directory:       ISO-8859 text\
......
```

在整个系统中查找内存信息转储文件(core dump)，然后把结果保存到/tmp/core.log文件中：

代码:

```
$ find / -name "core" -print | xargs echo "" >/tmp/core.log
```

上面这个执行太慢，我改成在当前目录下查找

代码:

```
#find . -name "file*" -print | xargs echo "" > /temp/core.log
# cat /temp/core.log
./file6
```

在当前目录下查找所有用户具有读、写和执行权限的文件，并收回相应的写权限：

代码:

```

# ls -l
drwxrwxrwx    2 sam      adm          4096 10月 30 20:14 file6
-rwxrwxrwx    2 sam      adm             0 10月 31 01:01 http3.conf
-rwxrwxrwx    2 sam      adm             0 10月 31 01:01 httpd.conf
```

```
# find . -perm -7 -print | xargs chmod o-w
# ls -l
drwxrwxr-x    2 sam      adm          4096 10月 30 20:14 file6
-rwxrwxr-x    2 sam      adm             0 10月 31 01:01 http3.conf
-rwxrwxr-x    2 sam      adm             0 10月 31 01:01 httpd.conf
```

用 **g r e p** 命令在所有的普通文件中搜索 **hostname** 这个词：

代码:

```
# find . -type f -print | xargs grep "hostname"
./httpd1.conf:#      different IP addresses or hostnames and have them handled by
the
./httpd1.conf:# VirtualHost: If you want to maintain multiple domains/hostnames
on your
```

用 **g r e p** 命令在当前目录下的所有普通文件中搜索 **hostnames** 这个词：

代码:

```
# find . -name \* -type f -print | xargs grep "hostnames"
./httpd1.conf:#      different IP addresses or hostnames and have them handled by
the
./httpd1.conf:# VirtualHost: If you want to maintain multiple domains/hostnames
on your
```

注意，在上面的例子中， \用来取消 f i n d 命令中的*在 s h e l l 中的特殊含义。

引用:

为了匹配$ H O M E 目录下的所有文件，下面两种方法都可以使用：

$ find $HOME -print

$ find ~ -print

引用:

为了查找当前文件系统中的所有目录并排序，可以用：

[code]$ find . -type d -print -local -mount |sort

```
# find . -type d | sort
.
./dir1
./file6
./.kde
./.kde/Autostart
./sam
./.xemacs

# ls -l
总用量 52
-rw-r--r-- 1 root root 0 10 月 31 18:06 admin.log001
-rw-r--r-- 1 root root 0 10 月 31 18:06 admin.log002
-rw-r--r-- 1 root root 0 10 月 31 18:06 admin.log042
-rw-r--r-- 1 root root 0 10 月 31 18:07 admin.log942
drwxr-xr-x 2 root root 4096 10 月 31 20:26 dir1
-rw-r--r-- 1 sam adm 0 10 月 31 01:07 fiel
drwxrwxr-x 2 sam adm 4096 10 月 31 20:25 file6
-rwxrwxr-x 2 sam adm 0 10 月 31 01:01 http3.conf
-rw-r--r-- 1 sam adm 34890 10 月 31 00:57 httpd1.conf
-rwxrwxr-x 2 sam adm 0 10 月 31 01:01 httpd.conf
drwxrwxr-x 2 gem group 4096 10 月 26 19:48 sam
-rw-r--r-- 1 root root 2792 10 月 31 20:19 temp
```

f i n d 命令配合使用 e x e c 和 x a rg s 可以使用户对所匹配到的文件执行几乎所有的命令。

下面是 find 一些常用参数的例子，有用到的时候查查就行了，像上面前几个贴子，都用到了其中的的一些参数，也可以用 man 或查看论坛里其它贴子有 find 的命令手册

1、使用 name 选项

文件名选项是 f i n d 命令最常用的选项，要么单独使用该选项，要么和其他选项一起使用。
可以使用某种文件名模式来匹配文件，记住要用引号将文件名模式引起来。
不管当前路径是什么，**如果想要在自己的根目录$ H O M E 中查找文件名符合* . t x t 的文件，**
使用~作为' p a t h n a m e 参数，波浪号~代表了你的$ H O M E 目录。

代码:

```
$ find ~ -name "*.txt" -print
```

想要在当前目录及子目录中查找所有的 ‘ * . t x t’ 文件，可以用：

代码:

```
$ find . -name "*.txt" -print
```

想要的当前目录及子目录中查找文件名以一个大写字母开头的文件，可以用：

代码：

```
$ find . -name "[A-Z]*" -print
```

想要在/ e t c 目录中查找文件名以 h o s t 开头的文件，可以用：

代码：

```
$ find /etc -name "host*" -print
```

想要查找$ H O M E 目录中的文件，可以用：

代码：

```
$ find ~ -name "*" -print 或 find . -print
```

要想让系统高负荷运行，就从根目录开始查找所有的文件。

代码：

```
$ find / -name "*" -print
```

如果想在当前目录查找文件名以两个小写字母开头，跟着是两个数字，最后是* . t x t 的文件，下面的命令就能够返回名为 a x 3 7 . t x t 的文件：$

代码：

```
$find . -name "[a-z][a-z][0--9][0--9].txt" -print
```

2、用 perm 选项

按照文件权限模式用- p e r m 选项。

按文件权限模式来查找文件的话。最好使用八进制的权限表示法。

如在当前目录下查找文件权限位为 7 5 5 的文件，即文件属主可以读、写、执行，其他用户可以读、执行的文件，可以用：

代码：

```
$ find . -perm 755 -print
```

还有一种表达方法：在八进制数字前面要加一个横杠-，表示都匹配，如-007 就相当于 777，-006 相当于 666

代码：

```
# ls -l
-rwxrwxr-x    2 sam      adm            0 10 月 31 01:01 http3.conf
-rw-rw-rw-    1 sam      adm        34890 10 月 31 00:57 httpd1.conf
-rwxrwxr-x    2 sam      adm            0 10 月 31 01:01 httpd.conf
drw-rw-rw-    2 gem      group       4096 10 月 26 19:48 sam
-rw-rw-rw-    1 root     root        2792 10 月 31 20:19 temp

# find . -perm 006
# find . -perm -006
./sam
./httpd1.conf
./temp
```

3、忽略某个目录

如果在查找文件时希望忽略某个目录，因为你知道那个目录中没有你所要查找的文件，那么可以使用-prune选项来指出需要忽略的目录。在使用-prune选项时要当心，因为如果你同时使用了-depth选项，那么-prune选项就会被find命令忽略。

如果希望在/apps目录下查找文件，但不希望在/apps/bin目录下查找，可以用：

代码：

```
$ find /apps -path "/apps/bin" -prune -o -print
```

-perm 选项中，我的解析
还有一种表达方法：在八进制数字前面要加一个横杠-，表示都匹配，如-007 就相当于 777，-006 相当于 666
不知对不对
补一个：使用 find 查找文件的时候怎么避开某个文件目录

比如要在/usr/sam 目录下查找不在 dir1 子目录之内的所有文件

代码：

```
find /usr/sam -path "/usr/sam/dir1" -prune -o -print
```

引用：

find [path ..] [expression] 在路径列表的后面的是表达式

-path "/usr/sam" -prune -o -print 是 -path "/usr/sam" -a -prune -o -print 的简写表达式按顺序求值，-a 和 -o 都是短路求值，与 shell 的 && 和 || 类似如果 -path "/usr/sam" 为真，则求值 -prune ，-prune 返回真，与逻辑表达式为真；否则不求值 -prune ，与逻辑表达式为假。 如果 -path "/usr/sam" -a -prune 为假，则求值 -print ，-print 返回真，或逻辑表达式为真；否则不求值 -print，或逻辑表达式为真。

这个表达式组合特例可以用伪码写为

代码:

```
if -path "/usr/sam"  then
        -prune
else
        -print
```

避开多个文件夹

引用:

```
find /usr/sam \( -path /usr/sam/dir1 -o -path /usr/sam/file1 \) -prune -o -print
```

圆括号表示表达式的结合。
\ 表示引用，即指示 shell 不对后面的字符作特殊解释，而留给 find 命令去解释其意义。

查找某一确定文件，-name 等选项加在-o 之后

代码:

```
#find /usr/sam  \(-path /usr/sam/dir1 -o -path /usr/sam/file1 \) -prune -o -name
"temp" -print
```

4、使用 user 和 nouser 选项

按文件属主查找文件，如在$ H O M E 目录中查找文件属主为 sam 的文件，可以用：

代码:

```
$ find ~ -user sam -print
```

在/ e t c 目录下查找文件属主为 u u c p 的文件：

代码:

```
$ find /etc -user uucp -print
```

为了查找属主帐户已经被删除的文件，可以使用- n o u s e r 选项。这样就能够找到那些属主在/ e t c / p a s s w d 文件中没有有效帐户的文件。在使用- n o u s e r 选项时，不必给出用户名； f i n d 命令能够为你完成相应的工作。
例如，希望在/ h o m e 目录下查找所有的这类文件，可以用：

代码:

```
$ find /home -nouser -print
```

5、使用 group 和 nogroup 选项

就像 u s e r 和 n o u s e r 选项一样，针对文件所属于的用户组， f i n d 命令也具有同样的选项，为了在/ a p p s 目录下查找属于 gem 用户组的文件，可以用：

代码:

```
$ find /apps -group gem -print
```

要查找没有有效所属用户组的所有文件，可以使用 n o g r o u p 选项。下面的 f i n d 命令从文件系统的根目录处查找这样的文件

代码:

```
$ find / -nogroup-print
```

6、按照更改时间或访问时间等查找文件

如果希望按照更改时间来查找文件，可以使用 m t i m e,atime 或 ctime 选项。如果系统突然没有可用空间了，很有可能某一个文件的长度在此期间增长迅速，这时就可以用 m t i m e 选项来查找这样的文件。
用减号-来限定更改时间在距今 n 日以内的文件，而用加号+来限定更改时间在距今 n 日以前的文件。
希望在系统根目录下查找更改时间在 5 日以内的文件，可以用：

代码:

```
$ find / -mtime -5 -print
```

为了在/ v a r / a d m 目录下查找更改时间在 3 日以前的文件，可以用：

代码:

```
$ find /var/adm -mtime +3 -print
```

论坛里的例 子
find 中的 -ctime 和 -mtime ,-atime 区别?
http://www.chinaunix.net/forum/viewtopic.php?t=15799

蜘蛛日记(九)::三个UNIX文件时间ctime,mtime,atime(转)
http://www.chinaunix.net/forum/viewtopic.php?t=92203
7、查找比某个文件新或旧的文件

如果希望查找更改时间比某个文件新但比另一个文件旧的所有文件，可以使用- n e w e r 选项。它的一般形式为：

代码:

```
newest_file_name ! oldest_file_name
```

其中，！ 是逻辑非符号。

查找更改时间比文件 sam 新但比文件 temp 旧的文件：

代码:

```
例：有两个文件
-rw-r--r--    1 sam      adm            0 10 月 31 01:07 fiel
-rw-rw-rw-    1 sam      adm        34890 10 月 31 00:57 httpd1.conf
-rwxrwxr-x    2 sam      adm            0 10 月 31 01:01 httpd.conf
drw-rw-rw-    2 gem      group       4096 10 月 26 19:48 sam
-rw-rw-rw-    1 root     root        2792 10 月 31 20:19 temp

# find -newer httpd1.conf  ! -newer temp -ls
1077669      0  -rwxrwxr-x    2  sam        adm                0  10  月   31
01:01 ./httpd.conf
1077671    4 -rw-rw-rw-   1 root     root        2792 10 月 31 20:19 ./temp
1077673    0 -rw-r--r--   1 sam      adm            0 10 月 31 01:07 ./fiel
```

查找更改时间在比 temp 文件新的文件：

代码:

```
$ find . -newer temp -print
```

8、使用 type 选项

在/etc目录下查找所有的目录，可以用：

代码:

```
$ find /etc -type d -print
```

在当前目录下查找除目录以外的所有类型的文件，可以用：

代码:

```
$ find . ! -type d -print
```

在/etc目录下查找所有的符号链接文件，可以用：

代码:

```
$ find /etc -type l -print
```

9、使用 size 选项

可以按照文件长度来查找文件，这里所指的文件长度既可以用块（block）来计量，也可以用字节来计量。以字节计量文件长度的表达形式为Nc；以块计量文件长度只用数字表示即可。在按照文件长度查找文件时，一般使用这种以字节表示的文件长度，在查看文件系统的大小，因为这时使用块来计量更容易转换。

在当前目录下查找文件长度大于1M字节的文件：

代码:

```
$ find . -size +1000000c -print
```

在/home/apache目录下查找文件长度恰好为100字节的文件：

代码:

```
$ find /home/apache -size 100c -print
```

在当前目录下查找长度超过10块的文件（一块等于512字节）：

代码:

```
$ find . -size +10 -print
```

论坛例子：
如何查找大小为 500K到 1000K之间的文件
http://bbs.chinaunix.net/forum/viewtopic.php?t=332268

10、使用depth选项

在使用f i n d命令时，可能希望先匹配所有的文件，再在子目录中查找。使用d e p t h选项就可以使f i n d命令这样做。这样做的一个原因就是，当在使用f i n d命令向磁带上备份文件系统时，希望首先备份所有的文件，其次再备份子目录中的文件。

在下面的例子中， f i n d命令从文件系统的根目录开始，查找一个名为C O N . F I L E的文件。它将首先匹配所有的文件然后再进入子目录中查找。

代码:

```
$ find / -name "CON.FILE" -depth -print
```

11、使用 mount 选项

在当前的文件系统中查找文件（不进入其他文件系统），可以使用 f i n d 命令的 m o u n t 选项。
从当前目录开始查找位于本文件系统中文件名以 X C 结尾的文件：

代码:

```
$ find . -name "*.XC" -mount -print
```

12、使用 cpio 选项

c p i o 命令可以用来向磁带设备备份文件或从中恢复文件。可以使用 f i n d 命令在整个文件系统中（更多的情况下是在部分文件系统中）查找文件，然后用 c p i o 命令将其备份到磁带上。如果希望使用 c p i o 命令备份/ e t c、/ h o m e 和/ a p p s 目录中的文件，可以使用下面所给出的命令，不过要记住你是在文件系统的根目录下：

代码:

```
#cd /
#find etc home apps -depthe -print | cpio -ivcdC65535 -o
```

在上面的例子中，应当注意到路径中缺少/。这叫作相对路径。之所以使用相对路径，是因为在从磁带中恢复这些文件的时候，可以选择恢复文件的路径。例如，可以将这些文件先恢复到另外一个目录中，对它们进行某些操作后，再恢复到原始目录中。如果在备份时使用了绝对路径，例如/ e t c，那么在恢复时，就只能恢复到/ e t c 目录中去，别无其他选择。在上面的例子中，我告诉 f i n d 命令首先进入/ e t c 目录，然后是/ h o m e 和/ a p p s 目录，先匹配这些目

录下
的文件，然后再匹配其子目录中的文件，所有这些结果将通过管道传递给 c p i o 命令进行备份。顺便说一下，在上面的例子中 c p i o 命令使用了 C 6 5 5 3 6 选项，本可以使用 B 选项，不过这样每块的大小只有 5 1 2 字节，而使用了 C 6 5 5 3 6 选项后，块的大小变成了 6 4 K 字节（6 5 5 3 6 / 1 0 2 4）

最后一项没有试验过

shell 基础三和四：后台(crontab,at,&,nohup)及(*,?,[]等)

- 设置 c r o n t a b 文件，并用它来提交作业。
- 使用 a t 命令来提交作业。
- 在后台提交作业。
- 使用 n o h u p 命令提交作业。

名词解释：
cron：系统调度进程。可以使用它在每天的非高峰负荷时间段运行作业，或在一周或一月中的不同时段运行。
At at 命令：使用它在一个特定的时间运行一些特殊的作业，或在晚一些的非负荷高峰时间段或高峰负荷时间段运行。
&：使用它在后台运行一个占用时间不长的进程。
Nohup：用它在后台运行一个命令，即使在用户退出时也不受影响
cron 和 crontab

c r o n 是系统主要的调度进程，可以在无需人工干预的情况下运行作业。c r o n t a b 命令允许用户提交、编辑或删除相应的作业。每一个用户都可以有一个 c r o n t a b 文件来保存调度信息。可以使用它运行任意一个 s h e l l 脚本或某个命令，每小时运行一次，或一周三次，这完全取决于你。每一个用户都可以有自己的 c r o n t a b 文件，但在一个较大的系统中，系统管理员一般会禁止这些文件，而只在整个系统保留一个这样的文件。系统管理员是通过 c r o n . d e n y 和 c r o n . a l l o w 这两个文件来禁止或允许用户拥有自己的 c r o n t a b 文件。

crontab 的域

为了能够在特定的时间运行作业，需要了解 c r o n t a b 文件每个条目中各个域的意义和格式。

引用：

> 下面就是这些域：
>
> 第 1 列分钟 1～5 9
>
> 第 2 列小时 1～2 3（0 表示子夜）
>
> 第 3 列日 1～3 1
>
> 第 4 列月 1～1 2
>
> 第 5 列星期 0～6（0 表示星期天）

```
第 6 列要运行的命令
```

下面是ｃｒｏｎｔａｂ的格式：

代码：

```
分< >时< >日< >月< >星期< >要运行的命令
```

其中< >表示空格。
Ｃｒｏｎｔａｂ文件的一个条目是从左边读起的，第一列是分，最后一列是要运行的命令，它位于星期的后面。

引用：

可以用横杠-来表示一个时间范围，例如你希望星期一至星期五运行某个作业，那么可以在星期域使用１－５来表示。
还可以在这些域中使用逗号“,”，例如你希望星期一和星期四运行某个作业，只需要使用１，４来表示。
可以用星号*来表示连续的时间段。如果你对某个表示时间的域没有特别的限定，也应该在该域填入*。该文件的每一个条目必须含有５个时间域，而且每个域之间要用空格分隔。
该文件中所有的注释行要在行首用＃来表示。

ｃｒｏｎｔａｂ文件例子：

代码：

```
30 21* * * /apps/bin/cleanup.sh
```

上面的例子表示每晚的２１：３０运行/ａｐｐｓ/ｂｉｎ目录下的ｃｌｅａｎｕｐ．ｓｈ。

代码：

```
45 4 1,10,22 * * /apps/bin/backup.sh
```

上面的例子表示每月１、１０、２２日的４：４５运行/ａｐｐｓ/ｂｉｎ目录下的ｂａｃｋｕｐ．ｓｈ。

代码：

```
10 1 * * 6,0 /bin/find -name "core" -exec rm {} \;
```

上面的例子表示每周六、周日的１：１０运行一个ｆｉｎｄ命令。

代码：

```
0,30 18-23 * * * /apps/bin/dbcheck.sh
```

上面的例子表示在每天１８：００至２３：００之间每隔３０分钟运行/ａｐｐｓ/ｂｉｎ目录下的ｄｂｃｈｅｃｋ．ｓｈ。

代码：

```
0 23 * * 6 /apps/bin/qtrend.sh
```

上面的例子表示每星期六的11：００ｐｍ运行/ａｐｐｓ/ｂｉｎ目录下的ｑｔｒｅｎｄ．ｓｈ。

你可能已经注意到上面的例子中，每个命令都给出了绝对路径。当使用ｃｒｏｎｔａｂ运行ｓｈｅｌｌ脚本时，要由用户来给出脚本的绝对路径，设置相应的环境变量。记住，既然是用户向ｃｒｏｎ提交了这些作业，就要向ｃｒｏｎ提供所需的全部环境。不要假定ｃｒｏｎ知道所需要的特殊环境，它其实并不知道。所以你要保证在ｓｈｅｌｌ脚本中提供所有必要的路径和环境变量，除了一些自动设置的全局变量。
如果ｃｒｏｎ不能运行相应的脚本，用户将会收到一个邮件说明其中的原因。
ｃｒｏｎｔａｂ命令的一般形式为：

代码：

```
Crontab [-u user] -e -l -r
```

其中：

引用：

> -u 用户名。
>
> -e 编辑ｃｒｏｎｔａｂ文件。
>
> -l 列出ｃｒｏｎｔａｂ文件中的内容。
>
> -r 删除ｃｒｏｎｔａｂ文件。

如果使用自己的名字登录，就不用使用-ｕ选项，因为在执行ｃｒｏｎｔａｂ命令时，该命令能够知道当前的用户。

创建一个新的 crontab 文件

在向ｃｒｏｎ进程提交一个ｃｒｏｎｔａｂ文件之前，要先设置环境变量ＥＤＩＴＯＲ。ｃｒｏｎ进程根据它来确定使用哪个编辑器编辑ｃｒｏｎｔａｂ文件。大部份的ＵＮＩＸ和ＬＩＮＵＸ用

户都使用 v i，如果你也是这样，那么你就编辑$ H O M E 目录下的. p r o f i l e 文件，在其中加入这样一行：

代码:

```
EDITOR=vi; export EDITOR
```

然后保存并退出。

创建一个名为< u s e r > c r o n 的文件，其中< u s e r >是用户名，例如， samcron。在该文件中加入如下的内容。

代码:

```
#(put your own initials here) echo the date to the console every
#15 minutes between 6pm and 6am
0,15,30,45 18-06 * * * /bin/echo 'date' > /dev/console
```

保存并退出。确信前面 5 个域用空格分隔。
在上面的例子中，系统将每隔 1 5 分钟向控制台输出一次当前时间。如果系统崩溃或挂起，从最后所显示的时间就可以一眼看出系统是什么时间停止工作的。在有些系统中，用 t t y 1 来表示控制台，可以根据实际情况对上面的例子进行相应的修改。
为了提交你刚刚创建的 c r o n t a b 文件，可以把这个新创建的文件作为 c r o n 命令的参数：

代码:

```
$su sam
crontab samcron
```

为了方便演示，切换到 sam 用户环境下，然后用 crontab samcron 提交给 c r o n 进程，它将每隔 1 5 分钟运行一次。

引用:

> 同时，新创建文件的一个副本已经被放在/ v a r / s p o o l / c r o n 目录中，文件名就是用户名（即 sam）。

引用:

```
#su
# cat /var/spool/cron/sam
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (samcron installed on Wed Nov 10 21:41:55 2004)
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie Exp $)
```

```
#(put your own initials here) echo the date to the console every
#15 minutes between 6pm and 6am
0,15,30,45 18-06 * * * /bin/echo 'date' > /dev/console
```

回到 root 下，查看/var/spool/cron/sam

列出 crontab 文件

为了列出 c r o n t a b 文件，可以用：

代码:

```
$ crontab -l
```

引用:

```
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (samcron installed on Wed Nov 10 21:41:55 2004)
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37 vixie Exp $)
#(put your own initials here) echo the date to the console every
#15 minutes between 6pm and 6am
0,15,30,45 18-06 * * * /bin/echo 'date' > /dev/console
```

你将会看到和上面类似的内容。可以使用这种方法在$ H O M E 目录中对 c r o n t a b 文件做一备份：

代码:

```
$ crontab -l > $HOME/mycron
```

这样，一旦不小心误删了 c r o n t a b 文件，可以用上一节所讲述的方法迅速恢复。

编辑 crontab 文件

如果希望添加、删除或编辑 c r o n t a b 文件中的条目，而 E D I TO R 环境变量又设置为 v i，那么就可以用 v i 来编辑 c r o n t a b 文件，相应的命令为：

代码:

```
$ crontab -e
```

可以像使用 v i 编辑其他任何文件那样修改 c r o n t a b 文件并退出。如果修改了某些条目或

添加了新的条目，那么在保存该文件时， cron 会对其进行必要的完整性检查。如果其中的某个域出现了超出允许范围的值，它会提示你。
例如，加入下面的一条：

引用：

```
#DT:delete core files,at 3:30am on 1,7,14,21,26 days of each month
30 3 1,7,14,21,26 * * /bin/find -name "core" -exec rm {} \;
```

现在保存并退出。最好在 crontab 文件的每一个条目之上加入一条注释，这样就可以知道它的功能、运行时间，更为重要的是，知道这是哪位用户的作业。
现在让我们使用前面讲过的 crontab -l 命令列出它的全部信息：

引用：

```
#(put your own initials here) echo the date to the console every
#15 minutes between 6pm and 6am
0,15,30,45 18-06 * * * /bin/echo 'date' > /dev/console


#DT:delete core files,at 3:30am on 1,7,14,21,26 days of each month
30 3 1,7,14,21,26 * * /bin/find -name "core" -exec rm {} \;
```

删除 crontab 文件

为了删除 crontab 文件，可以用：

代码：

```
$ crontab -r
```

恢复丢失的 crontab 文件

如果不小心误删了 crontab 文件，假设你在自己的$HOME 目录下还有一个备份，那么可以将其拷贝到/var/spool/cron/<username>，其中<username>是用户名。如果由于权限问题无法完成拷贝，可以用：

代码：

```
$ crontab <filename>
```

其中，<filename>是你在$HOME 目录中副本的文件名。
建议在自己的$HOME 目录中保存一个该文件的副本。编辑副本，然后重新提交新的文件。
有些 crontab 的变体有些怪异，所以在使用 crontab 命令时要格外小心。如果遗漏了

任何选项，crontab可能会打开一个空文件，或者看起来像是个空文件。这时敲delete键退出，不要按<Ctrl-D>，否则你将丢失crontab文件。

at命令

at命令允许用户向cron守护进程提交作业，使其在稍后的时间运行。一旦一个作业被提交，at命令将会保留所有当前的环境变量，包括路径，不象crontab，只提供缺省的环境。该作业的所有输出都将以电子邮件的形式发送给用户，除非你对其输出进行了重定向，绝大多数情况下是重定向到某个文件中。

和crontab一样，根用户可以通过/etc目录下的at.allow和at.deny文件来控制哪些用户可以使用at命令，哪些用户不行。不过一般来说，对at命令的使用不如对crontab的使用限制那么严格。

at命令的基本形式为：

代码：

```
at [-f script] [-m -l -r] [time] [date]
```

其中，

引用：

-f：script 是所要提交的脚本或命令。

-l：列出当前所有等待运行的作业。atq命令具有相同的作用。

-r：清除作业。为了清除某个作业，还要提供相应的作业标识（ID）；有些UNIX变体只接受atrm作为清除命令。

-m：作业完成后给用户发邮件。

time：at命令的时间格式非常灵活；可以是H、HH.HHMM、HH：MM或H：M，其中H和M分别是小时和分钟。还可以使用a．m .或p．m .。

date：日期格式可以是月份数或日期数，而且at命令还能够识别诸如today、tomorrow这样的词。

使用at命令提交命令或脚本

使用at命令提交作业有几种不同的形式，可以通过命令行方式，也可以使用at命令提示符。一般来说在提交若干行的系统命令时，使用at命令提示符方式，在提交shell脚本时，使用命令行方式。

命令行方式：

代码：

```
at [-f script] [-m -l -r] [time] [date]
```

提示符方式：

以在 a t 命令后面跟上日期/时间并回车。然后就进入了 a t 命令提示符，这时只需逐条输入相应的命令，然后按‘ < C T R L - D >’退出。

1、例一：提示符方式

代码：

```
# su sam
$ at 10:40
warning: commands will be executed using (in order) a) $SHELL b) login shell c)
/bin/sh
at> find /etc -name "passwd" -print
at> <EOT>
job 1 at 2004-11-02 10:40
```

其中， < E O T >就是< C T R L - D >。在 10:40 系统将执行一个简单的 f i n d 命令。提交的作业被分配了一个唯一标识 job 1。该命令在完成以后会将全部结果以邮件的形式发送给我。

下面这些日期/时间格式都是 a t 命令可以接受的：

代码：

```
at 5.00am May23
at 11.20pm
at now +2 hour
at 9am tomorrow
at 15:00 May24
at now + 10 minutes
```

2、例二：命令行方式

如果希望向 a t 命令提交一个 s h e l l 脚本，使用其命令行方式即可。在提交脚本时使用- f 选项。

如：

代码：

```
$ touch db_table.sh
$ at 3:00pm tomorrow -f db_table.sh
warning: commands will be executed using (in order) a) $SHELL b) login shell c)
/bin/sh
```

```
job 3 at 2004-11-02 15:00
```

在上面的例子中，一个叫做 d b _ t a b l e . s h 的脚本将在 2004-11-02 15:00 运行。俺的机子时间不对。

3、还可以使用 e c h o 命令向 a t 命令提交作业：

代码:

```
$ echo find /etc -name "passwd" -print | at now +1 minute
warning: commands will be executed using (in order) a) $SHELL b) login shell c)
/bin/sh
job 4 at 2004-11-01 19:07
```

列出所提交的作业

一个作业被提交后，可以使用 at -l 命令来列出所有的作业：

代码:

```
$ at -l
1       2004-11-02 10:40 a sam
3       2004-11-02 15:00 a sam
4       2004-11-01 19:07 a sam
```

其中，第一行是作业标识，后面是作业运行的日期/时间。最后一列 a 代表 a t。
还可以使用 a t q 命令来完成同样的功能，它是 a t 命令的一个链接。
直接>atq,相当于>at -l

当提交一个作业后，它就被拷贝到/ v a r / s p o o l / a t 目录中，准备在要求的时间运行。

代码:

```
# pwd
/var/spool/at
# ls -l
```

清除一个作业

清除作业的命令格式为：
atrm [job no] 或 at -r [job no]

要清除某个作业，首先要执行 at -l 命令，以获取相应的作业标识，然后对该作业标识使用 at -r

命令，清除该作业。

代码:

```
$ at -l
1       2004-11-02 10:40 a sam
3       2004-11-02 15:00 a sam
4       2004-11-01 19:07 a sam
$at -r 3
$at -l
1       2004-11-02 10:40 a sam
4       2004-11-01 19:07 a sam
```

有些系统使用 at-r [job no]命令清除作业。

&命令

当在前台运行某个作业时，终端被该作业占据；而在后台运行作业时，它不会占据终端。可以使用&命令把作业放到后台执行。

代码:

```
该命令的一般形式为：
命令&
```

在后台运行作业时要当心：需要用户交互的命令不要放在后台执行，因为这样你的机器就会在那里傻等。

不过，作业在后台运行一样会将结果输出到屏幕上，干扰你的工作。如果放在后台运行的作业会产生大量的输出，最好使用下面的方法把它的输出重定向到某个文件中：

代码:

```
command >out.file 2>&1 &
```

在上面的例子中，2>&1 表示所有的标准输出和错误输出都将被重定向到一个叫做out.file 的文件中。

当你成功地提交进程以后，就会显示出一个进程号，可以用它来监控该进程，或杀死它。

关于>&2、2>&1 等重定向的详细解释！

http://bbs.chinaunix.net/forum/viewtopic.php?t=16361

例一：

查找名为“httpd.conf”的文件，并把所有标准输出和错误输出重定向到 f i n d . d t 的文件中：

代码:

```
# find /etc/httpd/ -name "httpd.conf" -print >find.dt 2>&1 &
[2] 7832
[1]   Done                    find /etc/ -name "httpd.conf" -print >find.dt 2>&1 &
```

成功提交该命令之后，系统给出了它的进程号 7832。

代码:

```
# cat find.dt
/etc/httpd/conf/httpd.conf
[2]+  Done                    find /etc/httpd/ -name "httpd.conf" -print >find.dt
2>&1 &
```

查看 find.dt,可以看到执行结果

例二：
在后台执行脚本，如：有一个叫 psl 的脚本
$ps psl &
[7878]

用 ps 命令查看进程
用提交命令时所得到的进程号来监控它的运行。用 p s 命令和 g r e p 命令列出这个进程：

代码:

```
# ps -x |grep 7832
7868 pts/0    S      0:00 grep 7832
```

如果系统不支持 ps x 命令，可以用：

代码:

```
# ps -ef |grep 7832
root      7866  7790  0 23:40 pts/0    00:00:00 grep 7832
```

在用 p s 命令列出进程时，它无法确定该进程是运行在前台还是后台。

杀死后台进程

杀死后台进程可以使用 k i l l 命令。当一个进程被放到后台运行时， s h e l l 会给出一个进程号，我们可以根据这个进程号，用 k i l l 命令杀死该进程。该命令的基本形式为：

代码:

```
kill -signal [process_number]
```

现在暂且不要考虑其中的各种不同信号。
在杀进程的时候，执行下面的命令(你的进程号可能会不同)并按回车键。系统将会给出相应的信息告诉用户进程已经被杀死。

代码:

```
$kill 7832
```

如果系统没有给出任何信息，告诉你进程已经被杀死，那么不妨等一会儿，也许系统正在杀该进程，如果还没有回应，就再执行另外一个 k i l l 命令，这次带上一个信号选项：

代码:

```
$kill - 9 7868
```

如果用上述方法提交了一个后台进程，那么在退出时该进程将会被终止。为了使后台进程能够在退出后继续运行，可以使用 n o h u p 命令。

kill 这段俺没法验证，到后看到再说

nohug 命令

如果你正在运行一个进程，而且你觉得在退出帐户时该进程还不会结束，那么可以使用 n o h u p 命令。该命令可以在你退出帐户之后继续运行相应的进程。N o h u p 就是不挂起的意思(n ohang up)。
该命令的一般形式为：

代码:

```
nohup command &
```

使用 nohup 命令提交作业

如果使用 n o h u p 命令提交作业，那么在缺省情况下该作业的所有输出都被重定向到一个名为 n o h u p . o u t 的文件中，除非另外指定了输出文件：

代码:

```
nohup command > myout.file 2>&1
```

在上面的例子中，输出被重定向到 m y o u t . f i l e 文件中。

让我们来看一个例子，验证一下在退出帐户后相应的作业是否能够继续运行。我们先提交一个名为 p s 1 的日志清除进程：

代码:

```
$nobup ps1
```

现在退出该 s h e l l，再重新登录，然后执行下面的命令：

代码:

```
$ps x |grep ps1
```

我们看到，该脚本还在运行。如果系统不支持 ps x 命令，使用 ps -ef|grep ps1 命令。
一次提交几个作业

如果希望一次提交几个命令，最好能够把它们写入到一个 s h e l l 脚本文件中，并用 n o h u p 命令来执行它。
例如，下面的所有命令都用管道符号连接在一起；我们可以把这些命令存入一个文件，并使该文件可执行。

代码:

```
cat /home/accounts/qrt_0499 | /apps/bin/trials.awk | sort | lp
$cat > quarterend
cat /home/accounts/qtr_0499 | /apps/bin/trials.awk | sort | lp
<ctrl-D>
```

现在让它可执行：

代码:

```
$ chmod 744 quarterend
```

我们还将该脚本的所有输出都重定向到一个名为 q t r. o u t 的文件中。

代码:

```
nobup ./quarterend > qtr.out 2>
```

后台运行作业的:
有时我们必须要对大文件进行大量更改，或执行一些复杂的查找，这些工作最好能够在系统负荷较低时执行。
创建一个定时清理日志文件或完成其他特殊工作的脚本，这样只要提交一次，就可以每天晚上运行，而且无需你干预，只要看看相应的脚本日志就可以了。C r o n 和其他工具可以使系统管理任务变得更轻松。
***，？，[...]，[!...]**等

引用：

> • 匹配文件名中的任何字符串。
> • 匹配文件名中的单个字符。
> • 匹配文件名中的字母或数字字符。

下面就是这些特殊字符：

引用：

> * 匹配文件名中的任何字符串，包括空字符串。
> ？ 匹配文件名中的任何单个字符。
> [...] 匹配[ ]中所包含的任何字符。
> [!...] 匹配[ ]中非感叹号！之后的字符。

当ｓｈｅｌｌ遇到上述字符时，就会把它们当作特殊字符，而不是文件名中的普通字符，这样用户就可以用它们来匹配相应的文件名。

1、*：使用星号*可以匹配文件名中的任何字符串。就不用多说了，和 win 下差不多
2、？：使用可以匹配文件名中的任何单个字符。和 win 差不多

3、[]：使用[. . .]可以用来匹配方括号[ ]中的任何字符。可以使用一个横杠-来连接两个字母或数字，以此来表示一个范围。

1)列出以 i 或 o 开头的文件名：

代码：

```
#ls [io]*
```

2)列出 log.开头、后面跟随一个数字、然后可以是任意字符串的文件名：

代码：

```
#ls log.[0-9]*
```

3)与例二相反，列出 log.开头、后面不跟随一个数字、然后可以是任意字符串的文件名

代码：

```
#ls log.[!0-9]*
```

4)列出所有以 LPS 开头、中间可以是任何两个字符，最后以 1 结尾的文件名：

代码:

```
#ls LPS??1
```

5)列出所有以大写字母开头的文件名：

代码:

```
$ ls [A-Z]*
```

6)列出所有以小写字母开头的文件名：

代码:

```
$ ls [a-z]*
```

7)为了列出所有以数字开头的文件名：

代码:

```
$ ls [0-9]*
```

8)列出所有以. 开头的文件名（隐含文件，例如. p r o f i l e、. r h o s t s、. h i s t o r y 等）：

代码:

```
$ ls .*
```

shell基础五：输入和输出(echo,read,cat,管道,tee,重定向等)

在看这个之前，像俺这样没有基础的，得先看完网中人的《shell 十三问》的前三章，在置顶处，所以前面 echo 的含义,参数，及基础用法等就不说了。

我下面的所有环境都在在 REDHAT LINUX9 下试验的

在 LINUX 中，要使转义符生效，需加参数-e

从 echo 的变量开始说起

如：e c h o 命令输出转义符以及变量。

代码:

```
# echo -e "\007your home is $HOME , you are connected on `tty`"
your home is /root , you are connected on /dev/pts/1
# echo -e "\ayour home is $HOME , you are connected on `tty`"
your home is /root , you are connected on /dev/pts/1
#
```

引用：

> 本例中
>
> \007 或\a 你可以让终端铃响一声
>
> 显示出$ H O M E 目录，
>
> 并且可以让系统执行 t t y 命令(注意，该命令用键盘左上角的符号，法语中的抑音符引起来，
>
> 不是单引号)。

在 e c h o 命令输出之后附加换行，可以使用\ n 选项：

代码：

```
$ cat echod
#!/bin/sh
echo -e "this echo's 3 new lines\n\n\n"
echo "OK"
```

编辑一个新 echod，如上内容，然后运行输出如下：

代码：

```
$ ./echod
this echo's 3 new lines



OK
$
```

在 e c h o 语句中使用跳格符，记住别忘了加反斜杠\：

代码：

```
$ echo -e "here is a tab\there are two tabs\t\tok"
here is a tab    here are two tabs                ok
$
```

把一个字符串输出到文件中，使用重定向符号>。
在下面的例子中一个字符串被重定向到一个名为 m y f i l e 的文件中：

代码：

```
$ echo "The log files have all been done"> myfile
```

或者可以追加到一个文件的末尾，这意味着不覆盖原有的内容：

代码：

```
$ echo "$LOGNAME carried them out at `date`">>myfile
```

现在让我们看一下 m y f i l e 文件中的内容：

引用：

```
The log files have all been done
sam carried them out at 六 11 月 13 12:54:32 CST 2004
```

引号是一个特殊字符，所以必须要使用反斜杠 \ 来使 s h e l l 忽略它的特殊含义。
假设你希望使用 e c h o 命令输出这样的字符串：“/ d e v / r m t 0”，那么我们只要在引号前面加上反斜杠\即可：

代码：

```
$ echo "\"/dev/rmt0"\"
"/dev/rmt0"
$
```

其它用法：

--〉'echo'用法收集
http://bbs.chinaunix.net/forum/viewtopic.php?t=424904

ANSI控制码
http://bbs.chinaunix.net/forum/viewtopic.php?t=207837&highlight=%B7%C9%BB%D2%B3%C8
其它：可以自己练习

代码：

```
[sam@chenwy sam]$ read name
sam
[sam@chenwy sam]$ echo $name
sam
[sam@chenwy sam]$ read name surname
sam ch
[sam@chenwy sam]$ echo $name surname
```

```
sam surname
[sam@chenwy sam]$ read name surname
sam ch yiir
[sam@chenwy sam]$ echo $name
sam
[sam@chenwy sam]$ echo $surname
ch yiir
```

代码:

```
[sam@chenwy sam]$ cat var_test
#!/bin/sh
#var_test
echo -e "First Name :\c"
read name
echo -e "Middle Name :\c"
read middle
echo -e "Last name :\c"
read surname
```

var_test 文件内容如上

代码:

```
[sam@chenwy sam]$ ./var_test
First Name :wing
Middle Name :er
Last Name:chenwy
```

运行var_test文件

请问上面是不是把三个值赋给name,middle,surname三个变量了？？？

用read可以倒着读一个文件?
http://www.chinaunix.net/forum/viewtopic.php?t=939
cat：显示文件内容，创建文件，还可以用它来显示控制字符。

注意:在文件分页符处不会停下来；会一下显示完整个文件。因此，可以使用 m o r e 命令或把 c a t 命令的输出通过管道传递到另外一个具有分页功能的命令中，使用命令 less file 可实现相同的功能。

如下形式

代码:

```
$ cat myfile | more
或
$ cat myfile | pg
```

c a t 命令的一般形式为：

代码:

```
cat [options] filename1 ... filename2 ...
```

1、显示名为 m y f i l e 的文件：

代码:

```
$ cat myfile
```

2、显示 m y f i l e 1、m y f i l e 2、m y f i l e 3 这三个文件，可以用：

代码:

```
$ cat myfile1 myfile2 myfile3
```

3、创建一个包含上述三个文件的内容，名为 b i g f i l e 的文件，可以用输出重定向到新文件中：

代码:

```
$ cat myfile1 myfile2 myfile3 > bigfile
```

4、如果 cat 的命令行中没有参数,输入的每一行都立刻被 cat 命令输出到屏幕上,输入完毕后按 < C T R L - D >结束

代码:

```
$ cat
Hello world
Hello world
<ctrl+d>
```

```
$
```

5、新建文件

代码：

```
$cat >myfile
This is great
<ctrl-d>
$cat myfile
This is great
```

cat：参数选项

使用方式：

代码：

```
cat [-AbeEnstTuv] [--help] [--version] fileName
```

说明：把档案串连接后传到基本输出（萤幕或加 > fileName 到另一个档案）

参数：

引用：

```
-n 或 --number 由 1 开始对所有输出的行数编号
-b 或 --number-nonblank 和 -n 相似，只不过对于空白行不编号
-s 或 --squeeze-blank 当遇到有连续两行以上的空白行，就代换为一行的空白行
-v 或 --show-nonprinting 显示非打印字符
```

例：
显示时加上行号

代码：

```
$cp /etc/httpd/conf/httpd /usr/sam
$ cat -n httpd.conf
```

把 httpd.conf 的内容加上行号后输入 httpd1.conf 这个文件里

代码：

```
$cat -n httpd.conf > httpd1.conf
```

对文件 httpd.conf 加上行号(空白不加)后显示

代码:

```
$ cat -b httpd.conf
```

把 textfile1 和 textfile2 的档案内容加上行号（空白行不加）之后将内容附加到 textfile3 里。

代码:

```
$ cat -b textfile1 textfile2 >> textfile3
```

清空/etc/test.txt 档案内容

代码:

```
$cat /dev/null > /etc/test.txt
```

使用 sed 与 cat 除去空白行

代码:

```
$ cat -s /etc/X11/XF86Config | sed '/^[[:space:]]*$/d'
```

-s项我试了一下，不成功，不知是不是用错了

其它参数来自：(这个我没试)
http://bbs.chinaunix.net/forum/viewtopic.php?t=438463&highlight=cat

cat 还可以在您查看包含如制表符这样的非打印字符的文件时起帮助作用。您可以用以下选项来显示制表符：

引用:

```
* -T 将制表符显示为 ^I


* -v 显示非打印字符，除了换行符和制表符，它们使用各自效果相当的“控制序列”。例如，当您处理一个在 Windows 系统中生成的文件时，这个文件将使用 Control-M（^M）来标记行的结束。对于代码大于 127 的字符，它们的前面将会被加上 M-（表示“meta”），这与其
```

它系统中在字符前面加上 Alt- 相当。

* -E 在每一行的结束处添加美元符（$）。

显示非打印字符

代码:

```
$ cat -t /etc/X11/XF86Config
...
# Multiple FontPath entries are allowed (they are concatenated together)
# By default, Red Hat 6.0 and later now use a font server independent of
# the X server to render fonts.
^IFontPath^I"/usr/X11R6/lib/X11/fonts/TrueType"
^IFontPath^I"unix/:7100"
EndSection
...
```

代码:

```
$ cat -E /etc/X11/XF86Config
...
# Multiple FontPath entries are allowed (they are concatenated together)$
# By default, Red Hat 6.0 and later now use a font server independent of$
# the X server to render fonts.$
$
FontPath "/usr/X11R6/lib/X11/fonts/TrueType"$
FontPath "unix/:7100"$
$
EndSection$
...
```

代码:

```
$ cat -v /etc/X11/XF86Config
...
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@M-|M-8^X^@^@^@
P^@^O"M-X^O M-@^M^@^@^@M-^@^O"M-@M-k^@M-8*^@
@M-^H$M-@M-9|A(M-@)M-yM-|M-sM-*M-hW^A^@^@j^@
M-|M-sM-%1M-@M-9^@^B^@^@M-sM-+fM-^A= ^@ ^@
F^@^@ ^@M-9^@^H^@^@M-sM-$M-G^E(I!M-@M-^?
```

```
^IM-A5^@^@^D^@PM-^]M-^\X1M-H%^@^@^D^@tyM-G
…
```

cat -n 应该还可以吧

tee：读取标准输入的数据，并将其内容输出成文件。

语　　法：tee [-ai][--help][--version][文件…]

补充说明：tee 指令会从标准输入设备读取数据，将其内容输出到标准输出设备,同时保存成文件。我们可利用 tee 把管道导入的数据存成文件，甚至一次保存数份文件。

参　　数：-a 附加到既有文件的面，而非覆盖它。如果给予 tee 指令的文件名称已经存在，预设会覆盖该文件的内容。加上此参数，数据会新增在该文件内容的最面，而不会删除原先之内容。

-i 忽略中断信号

--help 在线帮助

--version 显示版本信息

例一：

列出文本文件 slayers.story 的内容，同时复制 3 份副本，文件名称分别为 ss-copy1、ss-copy2、ss-copy3：

代码:

```
$ cat slayers.story |tee ss-copy1 ss-copy2 ss-copy3
```

例一： 把列出当前目录，并把结果结到 myfile 里

代码:

```
$ls -l |tee myfile
```

管道：可以通过管道把一个命令的输出传递给另一个命令作为输入。管道用竖杠|表示。它的一般形式为：

代码:

```
命令 1 |命令 2
其中|是管道符号。
```

上例就是

标准输入、输出和错误

当我们在ｓｈｅｌｌ中执行命令的时候，每个进程都和三个打开的文件相联系，并使用文件描述符来引用这些文件。由于文件描述符不容易记忆， ｓｈｅｌｌ同时也给出了相应的文件名。
下面就是这些文件描述符及它们通常所对应的文件名：

引用：

> 文件文件描述符
>
> 输入文件—标准输入 0：它是命令的输入，缺省是键盘，也可以是文件或其他命令的输出。
>
> 输出文件—标准输出 1：它是命令的输出，缺省是屏幕，也可以是文件。
>
> 错误输出文件—标准错误 2：这是命令错误的输出，缺省是屏幕，同样也可以是文件。

如果没有特别指定文件说明符，命令将使用缺省的文件说明符（你的屏幕，更确切地说是你的终端）。

系统中实际上有 1 2 个文件描述符，但是正如我们在上表中所看到的， 0、1、2 是标准输入、输出和错误。可以任意使用文件描述符 3 到 9。

在执行命令时，可以指定命令的标准输入、输出和错误，要实现这一点就需要使用文件重定向。表 5 - 1 列出了最常用的重定向组合，并给出了相应的文件描述符。
在对标准错误进行重定向时，必须要使用文件描述符，但是对于标准输入和输出来说，这不是必需的。

代码：

```
常用文件重定向命令
command > filename 把把标准输出重定向到一个新文件中
command >> filename 把把标准输出重定向到一个文件中(追加)
command 1 > fielname 把把标准输出重定向到一个文件中
command > filename 2>&1 把把标准输出和标准错误一起重定向到一个文件中
command 2 > filename 把把标准错误重定向到一个文件中
command 2 >> filename 把把标准输出重定向到一个文件中(追加)
command >> filename 2>&1 把把标准输出和标准错误一起重定向到一个文件中(追加)
command < filename >filename2 把ｃｏｍｍａｎｄ命令以ｆｉｌｅｎａｍｅ文件作为标准输入，以ｆｉｌｅｎａｍｅ２文件
作为标准输出
command < filename 把ｃｏｍｍａｎｄ命令以ｆｉｌｅｎａｍｅ文件作为标准输入
command << delimiter 把从标准输入中读入，直至遇到ｄｅｌｉｍｉｔｅｒ分界符
command <&m 把把文件描述符 m 作为标准输入
command >&m 把把标准输出重定向到文件描述符 m 中
command <&- 把关闭标准输入
```

例子

[转载]常用文件重定向命令 (这篇网中人的回复好精彩啊^-^)
http://bbs.chinaunix.net/forum/viewtopic.php?t=191375

关于>&2、2>&1 等重定向的详细解释!
http://bbs.chinaunix.net/forum/viewtopic.php?t=16361

转贴：UNIX管道和重定向功能在系统备份中的妙用
http://www.chinaunix.net/forum/viewtopic.php?t=17925
exec:

e x e c 命令可以用来替代当前 s h e l l；换句话说，并没有启动子 s h e l l。使用这一命令时任何现有环境都将会被清除，并重新启动一个 s h e l l。它的一般形式为：
exec command
其中的 c o m m a n d 通常是一个 s h e l l 脚本。

我所能够想像得出的描述 e x e c 命令最贴切的说法就是：当这个脚本结束时，相应的会话可能就结束了。e x e c 命令的一个常见用法就是在用户的. p r o f i l e 最后执行时，用它来执行一些用于增强安全性的脚本。如果用户的输入无效，该
s h e l l 将被关闭，然后重新回到登录提示符。e x e c 还常常被用来通过文件描述符打开文件。e x e c 在对文件描述符进行操作的时候（也只有在这时），它不会覆盖你当前的 s h e l l。

可以看网中人《shell 十三问》第六节：
6) exec 跟 source 差在哪？

能把十三问一次性看完最好，不过对我来说还是有些难度，今天才弄清楚第四问，看了好久才明白，目前为止，看完 1，2，3，4，及 11

exec:

e x e c 命令可以用来替代当前 s h e l l；换句话说，并没有启动子 s h e l l。使用这一命令时任何现有环境都将会被清除，并重新启动一个 s h e l l。它的一般形式为：
exec command

其中的 c o m m a n d 通常是一个 s h e l l 脚本。

e x e c 在对文件描述符进行操作的时候，它不会覆盖你当前的 s h e l l。

这章到此为止了
1、变量一定得用""
2、处理顺序要搞清楚:这两行一定要牢牢记在脑中

> **引用:**

命令格式

command-name options argument

处理过程：

shell 会依据 IFS(Internal Field Seperator) 将 command line 所输入的文字给拆解为"字段"(word)。

然后再针对特殊字符(meta)先作处理，

最后再重组整行 command line 。

3、例子：

空格的好理解,但 CR 字符不好理解,如'',""

代码:

```
$ A='B
> C
> '
$ echo "$A"
B
C
$ echo $A
B C
```

echo 的$A 加上 soft quote 后，得出的结果不同了，
第一个是断行字符(new line),取消了 CR 和 IFS 的功能
第二个应该是一个空格了,仅取消 CR 功能，而保留 IFS 功能
第三个是 CR

原因如下：

然而，由于 echo $A 时的变量没至于 soft quote 中，因此当变量替换完成后并作命令行重组时，<enter> 会被解释为 IFS （空格键），而不是解释为 New Line （换行符）字符。

而在 escape 中

代码:

```
$ A=B\
> C\
>
$ echo $A
BC
$ echo "$A "
BC
```

得出的结果是 BC，原因：
<enter> 键本身在 shell meta 中的特殊性，在 \ 跳脱后面，仅仅取消其 CR 功能，而不会保留其 IFS 功能（空格）。 因此就是(NULL)

因此在上面两个例子中 <enter> 键所产生的字符有四种：

引用：

```
CR （结束命令）
IFS （空格）
NL(New Line) （断行）
NULL （空）
```

不知我的理解是否正确，还望各位指点，呵呵,我感觉这样说好像更容易理解 
上面理解了，下面就不难了，找个地方放一下，俺怕自己给忘了

而接下来的例子中，则要理解 shell meta 与 command meta

有些 meta ，都是有特殊用途的，比如 { } ，但在 awk 中 却要用 { } 来区分出 awk 的命令区段(BEGIN, MAIN, END)，也就是双方都用到了{ }

如果输入下例命令就会出错：

代码：

```
$ awk {print $0} 1.txt
```

这是因为 { } 在 shell 中并没关闭，那 shell 就将 {print $0} 视为 command block ，而不是 awk 的参数，但同时又没有" ；"符号作命令区隔，因此就出现 awk 的语法错误结果。

要解决之，可用 hard quote ：

代码：

```
$ awk '{print $0}' 1.txt
```

将原本的 {、<space>、$(注三)、} 这几个 shell meta 关闭， 避免掉在 shell 中遭到处理，而完整的成为 awk 参数中的 command meta 。
(注三：而其中的 $0 是 awk 内建的 field number ，而非 awk 的变量，awk 自身的变量无需使用 $ 。)

要是理解了 hard quote 的功能，再来理解 soft quote 与 escape 就不难：

代码：

```
awk "{print \$0}" 1.txt
awk \{print\ \$0\} 1.txt
```

第一行：由于 soft quote 中没法关闭$，因此用\来关闭$meta
第二行：\{关闭{,\ （\空格，关闭空格键）,\$,\}就不用说了

如果 awk 的 $0 的 0 值是从另一个 shell 变量读进

比方说：已有变量 $A 的值是 0 ，那如何在 command line 中解决 awk 的 $$A 呢？ 那么 hard quoe 就不可行了：
代码:

代码：

```
$ awk '{print $$A}' 1.txt
```

因为 $A 的 $ 在 hard quote 中是不能替换变量的。

可以使用如下几种方案：

代码：

```
A=0
awk "{print \$$A}" 1.txt
awk \{print\ \$$A\} 1.txt
awk '{print $'$A'}' 1.txt
awk '{print $'"$A"'}' 1.txt     # 注："$A" 包在 soft quote 中
```

上面得注意$和''和""的包含位置

shell基础 67：执行顺序(||及&&，{}及())及正则表表达式

引用：

```
$ unset A
$ [ -n "$A" ] && [ "$A" -lt 100 ] || echo 'too big!'
too big!
```

为何上面的结果也可得到呢？ 做个记号

网中人第十问
10) && 与 || 差在哪？
http://bbs.chinaunix.net/forum/viewtopic.php?t=218853&show_type=new&postdays=0&postorder=asc&start=60

第六问：
6) exec 跟 source 差在哪？
http://www.chinaunix.net/forum/viewtopic.php?t=194191

总结：
fork:在子行程中的环境如何变更，均不会影响父行程的环境。

正常来说，当我们执行一个 shell script 时，其实是先产生一个 sub-shell 的子行程，然后 sub-shell 再去产生命令行的子行程。
即我们正常运行一个脚本时：

代码:

```
./my.script
```

source：所谓 source 就是让 script 在当前 shell 内执行、而不是产生一个 sub-shell 来执行。
由于所有执行结果均于当前 shell 内完成，若 script 的环境有所改变，当然也会改变当前环境了

代码:

```
source ./my.script
或：
. ./my.script
```

（）和{ }

引用:

> 如果希望把几个命令合在一起执行， s h e l l提供了两种方法。既可以在当前 s h e l l也可以在子 s h e l l中执行一组命令。
>
> 1、()：
>
> 为了在当前 s h e l l中执行一组命令，可以用命令分隔符隔开每一个命令，并把所有的命令用圆括号（）括起来。
>
> 它的一般形式为：
>
> **代码:**

```
（命令 1;命令 2;. . .）
```

2、{}：

使用{ }来代替（），那么相应的命令将在子ｓｈｅｌｌ中作为一个整体被执行，只有在{ }中所有命令的输出作为一个整体被重定向时，其中的命令才被放到子ｓｈｅｌｌ中执行，否则在当前ｓｈｅｌｌ执行。它的一般形式为：

代码:

```
{命令 1;命令 2;. . . }
```

此段有误，现更下如下：

代码:

```
(cmd1;cmd2;...;cmdN)#在一个子 shell 里执行一组命令
{cmd1;cmd2;...;cmdN}# 在当前 shell 里执行一组命令
这是一个基本概念


[jason@firewall jason]$ A=1;echo $A;{ A=2; };echo $A
1
2
[jason@firewall jason]$ A=1;echo $A;( A=2; );echo $A
1
1

{ A=2; }改变了当前 shell 变量的值
( A=2; )未改变当前 shell 变量的值
```

可查看原链接
http://bbs.chinaunix.net/forum/viewtopic.php?show_type=&p=3386007#3386007
多谢指出，呵呵

例一：上面的例子中：

代码:

```
$ comet month_end.txt || exit
```

现在如果该脚本执行失败了，我希望先给自己发个邮件，然后再退出，可以用下面的方法来实现：

代码：

```
$ comet month_end || (echo "Comet did no work" | mail sam ; exit)
```

上例中由于 只使用了命令分隔符而没有把它们组合在一起， shell将直接执行最后一个命令（exit）。

例二：下面是原来的那个例子：

代码：

```
$ sort quarter_end.txt > quarter.sorted && lp quarter.sorted
```

使用命令组合的方法，如果sort命令执行成功了，先将输出文件拷贝到一个日志区，然后再打印。

代码：

```
$ sort quarter_end.txt > quarter.sorted && (cp quarter.sorted /logs/quarter.sorted; lp quarter.sorted)
```

shell十三问第七问：
7) ( ) 与 { } 差在哪？
http://bbs.chinaunix.net/forum/viewtopic.php?t=218853&show_type=new&postdays=0&postorder=asc&start=45

代码：

```
• 匹配行首与行尾。
• 匹配数据集。
• 只匹配字母和数字。
• 匹配一定范围内的字符串集。
```

当从一个文件或命令输出中抽取或过滤文本时，可以使用正则表达式（ RE），正则表达式是一些特殊或不很特殊的字符串模式的集合。

基本元字符集及其含义

网中人 写到：

```
abc: 表示 abc 三個連續的字符, 但彼此獨立而非集合. (可簡單視為三個 char. set)

(abc): 表示 abc 這三個連續字符的集合. (可簡單視為一個 char. set)
```

```
a|b: 表示單一字符, 或 a 或 b .
(abc|xyz): 表示或 abc 或 xyz 這兩個 char. set 之一. (註二)
[abc]: 表示單一字符, 可為 a 或 b 或 c . (與 wildcard 之 [abc] 原理相同)
[^abc]: 表示單一字符, 不為 a 或 b 或 c 即可. (與 wildcard 之 [!abc] 原理相同)
. : 表示任意單一字符. (與 wildcard 之 ? 原理相同)
```

代码:

```
^ 只只匹配行首
$ 只只匹配行尾
* 只一个单字符后紧跟*，匹配 0 个或多个此单字符
[ ] 只匹配[ ]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[ 1 2 3 4 5 ]
\ 只用来屏蔽一个元字符的特殊含义。因为有时在 s h e l l 中一些元字符有
特殊含义。\可以使其失去应有意义
. 只匹配任意单字符
p a t t e r n \ { n \ } 只用来匹配前面 p a t t e r n 出现次数。n 为次数
p a t t e r n \ { n，\ } m 只含义同上，但次数最少为 n
p a t t e r n \ { n，m \ } 只含义同上，但 p a t t e r n 出现次数在 n 与 m 之间
现在详细讲解其中特殊含义。
```

1、使用句点匹配单字符

例一：beng.n:以 b e g 开头，中间夹一个任意字符。
例二：. . . .X C. . . .：共 10 个字符，前四个之后为 XC
例三：列出所有用户都有写权限的目录或文件 ：

代码:

```
ls -l |grep ...x..x..x
```

2、行首以^匹配字符串或字符序列
^只允许在一行的开始匹配字符或单词。

例如，使用 ls -l 命令，并匹配目录。

代码:

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

3、在行尾以$匹配字符串或字符

可以说$与^正相反，它在行尾匹配字符串或字符， $符号放在匹配单词后。

例一：列出文件 httpd1.conf 中所有以单词 common 结尾的行

代码:

```
$grep common$ httpd1.conf
或
$cat httpd1.conf | grep common$
```

例二：匹配所有空行:^ $

例三：只返回包含一个字符的行：^.$

4、用\屏蔽一个特殊字符的含义

下列字符一般可以认为是特殊字符：

代码:

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

引用:

如：

\ .

反斜杠后面的字符不再是特殊字符，而是一个普通字符，即句点。

引用:

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：

\ ^

引用:

在正则表达式中匹配以* . p a s 结尾的所有文件：

\ * \ . p a s

即可屏蔽字符*的特定含义。

5、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\ { \ }，

引用：

> 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 - 2 5 5 中任意整数。

例一：匹配字母 A 出现两次，并以 B 结尾：

代码：

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

匹配值为 A A B

例二：匹配 A 至少 4 次：

代码：

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

可以得结果 A A A A B 或 A A A A A A A B，但不能为 A A A B。

例三：如给出出现次数范围，例如 A 出现 2 次到 4 次之间：

代码：

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

则结果为 A A B、A A A B、A A A A B，而不是 A B 或 A A A A A B 等。

例四：假定从下述列表中抽取代码：

引用：

> 1234XC9088
>
> 4523XX9001
>
> 0011XA9912
>
> 9931Xc3445

格式如下：前 4 个字符是数字，接下来是 x x，最后 4 个也是数字，操作如下：
[0 - 9] \ { 4 \ }X X[0 - 9] \ { 4 \ }

引用：

> 具体含义如下：

1) 匹配数字出现 4 次。

2) 后跟代码 x x。

3) 最后是数字出现 4 次。

结果如下

引用:

```
1234XC9088 -no match
4523XX9001 -match
0011XA9912 -no match
9931Xc3445 -no match
```

经常使用的正则表达式举例

代码:

```
^ 对行首
$ 对行尾
^ [ t h e ] 对以 t h e 开头行
[ S s ] i g n a [ l L ] 对匹配单词 s i g n a l、s i g n a L、S i g n a l、S i g n a L
[Ss]igna[lL]\. 对同上，但加一句点
[ m a y M A Y ] 对包含 m a y 大写或小写字母的行
^ U S E R $ 对只包含 U S E R 的行
[tty]$ 对以 t t y 结尾的行
\ . 对带句点的行
^ d . . x . . x . . x 对对用户、用户组及其他用户组成员有可执行权限的目录
^ [ ^ l ] 对排除关联目录的目录列表
[ . * 0 ] 对 0 之前或之后加任意字符
[ 0 0 0 * ] 对 0 0 0 或更多个
[ iI] 对大写或小写 I
[ i l ] [ n N ] 对大写或小写 i 或 n
[ ^ $ ] 对空行
[ ^ . * $ ] 对匹配行中任意字符串
^ . . . . . $ 对包括 6 个字符的行
[a- zA-Z] 对任意单字符
[ a - z ] [ a - z ] * 对至少一个小写字母
[ ^ 0 - 9 \ $ ] 对非数字或美元标识
[ ^ 0 - 0 A - Z a - z ] 对非数字或字母
[ 1 2 3 ] 对 1 到 3 中一个数字
[ D d ] e v i c e 对单词 d e v i c e 或 D e v i c e
D e . . c e 对前两个字母为 D e，后跟两个任意字符，最后为 c e
```

\ ^ q 对以^ q 开始行
^ . $ 对仅有一个字符的行
^\.[0-9][0-9] 对以一个句点和两个数字开始的行
' " D e v i c e " ' 对单词 d e v i c e
D e [V v] i c e \ . 对单词 D e v i c e 或 d e v i c e
[0 - 9] \ { 2 \ } - [0 - 9] \ { 2 \ } - [0 - 9] \ { 4 \ } 对日期格式 d d - m m - y y y y
[0 - 9] \ { 3 \ } \ . [0 - 9] \ { 3 \ } \ . [0 - 9] \ { 3 \ } \ . [0 - 9] \ { 3 \ } 对 I P 地址格式 nnn. nnn.nnn.nnn
[^ . * $] 对匹配任意行

shell基础八：文本过滤工具（grep）

网中人 写到:

比方以 grep 来说，在 Linux 上你可找到 grep, egrep, fgrep 这几个程序，其差异大致如下:

* grep:

传统的 grep 程序，在没有参数的情况下，只输出符合 RE 字符串之句子. 常见参数如下:

-v: 逆反模示，只输出"不含" RE 字符串之句子.

-r: 递归模式，可同时处理所有层级子目录里的文件.

-q: 静默模式，不输出任何结果(stderr 除外. 常用以获取 return value，符合为 true，否则为 false .)

-i: 忽略大小写.

-w: 整词比对，类似 \<word\> .

-n: 同时输出行号.

-c: 只输出符合比对的行数.

-l: 只输出符合比对的文件名称.

-o: 只输出符合 RE 的字符串. (gnu 新版独有，不见得所有版本都支持.)

-E: 切换为 egrep .

* egrep:

为 grep 的扩充版本，改良了许多传统 grep 不能或不便的操作. 比方说:

- grep 之下不支持 ? 与 + 这两种 modifier，但 egrep 则可.

- grep 不支持 a|b 或 (abc|xyz) 这类"或一"比对，但 egrep 则可.

- grep 在处理 {n,m} 时，需用 \{ 与 \} 处理，但 egrep 则不需.

诸如此类的... 我个人会建议能用 egrep 就不用 grep 啦... ^_^

* fgrep:
不作 RE 处理，表达式仅作一般字符串处理，所有 meta 均失去功能.

g r e p 一般格式为：

代码：

```
grep [选项]基本正则表达式[文件]
这里基本正则表达式可为字符串。
```

单引号双引号
在 g r e p 命令中输入字符串参数时，最好将其用双引号括起来。
在调用模式匹配时，应使用单引号。

例如：“m y s t r i n g”。这样做有两个原因，一是以防被误解为 s h e l l 命令，二是可以用来查找多个单词组成的字符串。
在调用变量时，也应该使用双引号，诸如： g r e p “$ M Y VA R” 文件名，如果不这样，将没有返回结果。

常用的 g r e p 选项有：

引用：

-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。

开始讨论之前，先生成一个文件，插入一段文本，并在每列后加入< Ta b >键，g r e p 命令示例中绝大多数将以此为例，其命名为 d a t a . . f。生成一个文件，d a t a . f 的记录结构如下：

引用：

第 1 列：城市位置编号。

```
第 2 列：月份。
第 3 列：存储代码及出库年份。
第 4 列：产品代号。
第 5 列：产品统一标价。
第 6 列：标识号。
第 7 列：合格数量。
```

文件内容如下：

代码:

```
$ cat data.f
48      Dec     3BC1977 LPSX    68.00   LVX2A   138
483     Sept    5AP1996 USP     65.00   LVX2C   189
47      Oct     3ZL1998 LPSX    43.00   KVM9D   512
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
```

1、查询多个文件

在所有文件中查询单词“ sort it”

代码:

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

2、 行匹配

1)显示包含“4 8”字符串的文本：

代码:

```
$ grep  "48"data.f
```

2)输出匹配行的总数

代码:

```
$ grep -c "48"data.f
4
```

g r e p 返回数字 4，表示：包含字符串“4 8”的有 4 行。

3)行数

显示满足匹配模式的所有行行数：

代码:

```
$ grep -n "48"data.f
```

行数在输出第一列，后跟包含 4 8 的每一匹配行。

4)显示非匹配行

显示所有不包含 4 8 的各行

代码:

```
$ grep -v "48"data.f
```

5)精确匹配

可能大家已注意到，在上一例中，抽取字符串“ 4 8”，返回结果包含诸如 4 8 4 和 4 8 3 等包含“4 8”的其他字符串，实际上应精确抽取只包含 4 8 的各行。

使用 g r e p 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 4 8，方法如下：

代码:

```
$grep "48\>" data.f
```

引用:

> 另一种方法我试过，好像不行：
>
> 注意在每个匹配模式中抽取字符串后有一个< Ta b >键，所以应操作如下：
>
> < Ta b >表示点击 t a b 键。
>
> $grep "48<tab>" data.f

6)大小写敏感

缺省情况下， g r e p 是大小写敏感的，如要查询大小写不敏感字符串，必须使用- i 开关。在 d a t a . f 文件中有月份字符 S e p t，既有大写也有小写，要取得此字符串大小写不敏感查询，方法如下：

代码:

```
$grep -i "48" data.f
```

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 g r e p 中使用的专有模式与一些 s h e l l 命令的特殊方式相混淆。

1、模式范围

抽取代码为 4 8 4 和 4 8 3 的城市位置，可以使用[ ]来指定字符串范围。

代码:

```
$ 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
```

2、不匹配行首

使行首不是 4 或 8，可以在方括号中使用^记号。

代码:

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

如果是字符串 48

代码:

```
$ grep -v "^[^48]" data.f
```

3、设置大小写

使用- i 开关可以屏蔽月份 S e p t 的大小写敏感

代码:

```
[sam@chenwy sam]$ grep -i "sept" data.f
483     Sept    5AP1996 USP     65.00   LVX2C   189
216     sept    3ZL1998 USP     86.00   KVM9E   234
```

也可以用另一种方式[ ]模式抽取各行包含 S e p t 和 s e p t 的所有信息。

代码:

```
[sam@chenwy sam]$ grep '[sS]ept' data.f
```

如果要抽取包含 S e p t 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：

代码:

```
[sam@chenwy sam]$ grep '[sS]ept' data.f | grep 48
```

```
483     Sept    5AP1996 USP     65.00   LVX2C   189
```

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

4、匹配任意字符

如果抽取以 K 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：

代码:

```
[sam@chenwy sam]$ 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：

代码:

```
[sam@chenwy sam]$ grep '[A-Z]..C' 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
```

5、日期查询

一个常用的查询模式是日期查询。先查询所有以 5 开始以 1 9 9 6 或 1 9 9 8 结尾的所有记录。使用模式 5 . . 1 9 9 [6 , 8]。这意味着第一个字符为 5，后跟两个点，接着是 1 9 9，剩余两个数字是 6 或 8。

代码:

```
[sam@chenwy sam]$ grep '5..199[6,8]' data.f
483     Sept    5AP1996 USP     65.00   LVX2C   189
483     may     5PA1998 USP     37.00   KVM9D   644
```

6、范围组合

必须学会使用[ ]抽取信息。假定要取得城市代码，第一个字符为 0-9，第二个字符在 0 到 5 之间，第三个字符在 0 到 6 之间，使用下列模式即可实现。

代码:

```
[sam@chenwy sam]$ grep '[0-9][0-5[0-6]' data.f
48      Dec     3BC1977 LPSX    68.00   LVX2A   138
483     Sept    5AP1996 USP     65.00   LVX2C   189
47      Oct     3ZL1998 LPSX    43.00   KVM9D   512
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
```

这里返回很多信息，有想要的，也有不想要的。参照模式，返回结果是正确的，因此这里

代码:

```
[sam@chenwy sam]$ grep '^[0-9][0-5[0-6]' data.f
219     dec     2CC1999 CAD     23.00   PLV2C   68
216     sept    3ZL1998 USP     86.00   KVM9E   234
```

这样可以返回一个预期的正确结果。

以下要注意有无边界字符的区别

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：

代码:

```
[sam@chenwy sam]$ 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，注意有无边界字符的区别。

同样，抽取记录使之包含数字 9 9 9（三个 9），方法如下：

代码:

```
[sam@chenwy sam]$ 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 重复出现两次或三次：

代码:

```
[sam@chenwy sam]$ grep '9\{3\}' data.f
219     dec     2CC1999 CAD     23.00   PLV2C   68
[sam@chenwy sam]$ grep '9\{2\}' data.f
483     Sept    5AP1996 USP     65.00   LVX2C   189
47      Oct     3ZL1998 LPSX    43.00   KVM9D   512
219     dec     2CC1999 CAD     23.00   PLV2C   68
484     nov     7PL1996 CAD     49.00   PLV2C   234
```

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：

代码:

```
[sam@chenwy sam]$ cat myfile
83
888883
8884
```

```
88883
[sam@chenwy sam]$ grep '8\{2,6\}3' myfile
888883
88883
```

8、使用 grep 匹配“与”或者“或”模式

g r e p 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 2 1 9 或 2 1 6，方法如下：

代码:

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

9、空行

结合使用^和$可查询空行。使用- c 参数显示总行数：

代码:

```
[sam@chenwy sam]$ grep -c '^$' myfile
```

使用- n 参数显示实际在哪一行：

代码:

```
[sam@chenwy sam]$ grep -c '^$' myfile
```

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：

代码:

```
[sam@chenwy sam]$ grep '\.' myfile
```

或者是一个双引号：

代码:

```
[sam@chenwy sam]$ grep '\"' myfile
```

以同样的方式，如要查询文件名 c o n f t r o l l . c o n f（这是一个配置文件），脚本如下：

代码:

```
[sam@chenwy sam]$ grep 'conftroll\.conf' myfile
```

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。

代码:

```
[sam@chenwy sam]$ grep '^[a-z]\{1,6\}\.[A-Z]\{1,2\}' filename
```

这个写法我不知道有没有错 

12 查询 IP 地址

要查看 n n n . n n n 网络地址，如果忘了第二部分中的其余部分，只知有两个句点，例如 n n n . n n . .。要抽取其中所有 nnn.nnn IP 地址，使用[0 - 9] \ { 3 \ } \ . [0 - 0 \ { 3 \ } \。含义是任意数字出现 3 次，后跟句点，接着是任意数字出现 3 次，后跟句点。

代码:

```
[0-9]\{3\}\.[0-9]\{3\}\.'
```

1、类名

g r e p 允许使用国际字符模式匹配或匹配模式的类名形式。
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

引用:

```
[ [ : u p p e r : ] ] [ A - Z ] [ [ : a l n u m : ] ] [ 0 - 9 a - zA-Z]
[ [ : l o w e r : ] ] [ a - z ] [ [ : s p a c e : ] ] 空格或 t a b 键
[ [ : d i g i t : ] ] [ 0 - 9 ] [ [ : a l p h a : ] ] [ a - z A - Z ]
```

例一：取以 5 开头，后跟至少两个大写字母：

代码:

```
$grep '5[[:upper:]][[:upper]]' data.f
```

取以 P 或 D 结尾的所有产品代码：

代码:

```
grep '[[:upper:]][[:upper:]][P,D]' data.f
```

2、使用通配符 * 的匹配模式

代码:

```
$cat testfile
looks
likes
looker
long
```

试试如下：

代码:

```
grep "l.*s" testfile
```

如在行尾查询某一单词，试如下模式：

代码:

```
grep "ng$" testfile
```

这将在所有文件中查询行尾包含单词 ng 的所有行。

3、系统 grep

文件 passwd

代码:

```
[root@Linux_chenwy sam]# grep "sam" /etc/passwd
sam:x:506:4::/usr/sam:/bin/bash
```

上述脚本查询/ e t c / p a s s w d 文件是否包含 sam 字符串

如果误输入以下脚本：

代码:

```
[root@Linux_chenwy sam]# grep "sam" /etc/password
grep: /etc/password: 没有那个文件或目录
```

将返回 g r e p 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 g r e p 命令- s 开关，可屏蔽错误信息。
返回命令提示符，而没有文件不存在的错误提示。

代码:

```
[root@Linux_chenwy sam]# grep -s "sam" /etc/password
```

如果 g r e p 命令不支持- s 开关，可替代使用以下命令：

代码:

```
[root@Linux_chenwy sam]# grep "sam" /tec/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/ d e v / n u l l 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。

如要保存 g r e p 命令的查询结果，可将命令输出重定向到一个文件。

代码:

```
[root@Linux_chenwy sam]# grep "sam" /etc/passwd >/usr/sam/passwd.out
[root@Linux_chenwy sam]# cat /usr/sam/passwd.out
sam:x:506:4::/usr/sam:/bin/bash
```

脚本将输出重定向到目录/ t m p 下文件 p a s s w d . o u t 中。

使用 ps 命令

使用带有 ps x 命令的 g r e p 可查询系统上运行的进程。ps x 命令意为显示系统上运行的所有进程列表。要查看 D N S 服务器是否正在运行（通常称为 n a m e d），方法如下：

代码:

```
[root@Linux_chenwy sam]# ps ax|grep "named"
2897 pts/1    S      0:00 grep named
```

输出也应包含此 g r e p 命令，因为 g r e p 命令创建了相应进程， ps x 将找到它。在 g r e p 命令中使用- v 选项可丢弃 p s 命令中的 g r e p 进程。如果 ps x 不适用于用户系统，替代使用 ps -ef。这里，由于我没有 DNS 服务，因而只有 grep 进程。

对一个字符串使用 grep

g r e p 不只应用于文件，也可应用于字符串。为此使用 e c h o 字符串命令，然后对 g r e p 命令使用管道输入。

代码:

```
[root@Linux_chenwy sam]# STR="Mary Joe Peter Pauline"
[root@Linux_chenwy sam]# echo $STR | grep "Mary"
Mary Joe Peter Pauline
```

匹配成功实现。

代码:

```
[root@Linux_chenwy sam]# echo $STR | grep "Simon"
```

因为没有匹配字符串，所以没有输出结果。

4、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：

代码:

```
[root@Linux_chenwy sam]# vi grepstrings
[root@Linux_chenwy sam]# cat grepstrings
484
47
```

代码:

```
[root@Linux_chenwy sam]# egrep -f grepstrings data.f
47      Oct     3ZL1998 LPSX    43.00   KVM9D   512
484     nov     7PL1996 CAD     49.00   PLV2C   234
```

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。

代码:

```
[root@Linux_chenwy sam]# 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。

代码:

```
$who |egrep (louise|matty|pauline)
```

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：

代码:

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

如果要查询一个文件列表，包括shutdown、shutdowns、reboot和rebo

o t s，使用 e g r e p 可容易地实现。

代码:

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

shell基础九：awk [精华]

终于看经常用的到的 awk 了 

下面没有讲述 a w k 的全部特性，也不涉及 a w k 的深层次编程，仅讲述使用 a w k 执行行操作及怎样从文本文件和字符串中抽取信息。

引用:

内容有：

- 抽取域。
- 匹配正则表达式。
- 比较域。
- 向 a w k 传递参数。
- 基本的 a w k 行操作和脚本。

a w k 语言的最基本功能是在文件或字符串中基于指定规则浏览和抽取信息。a w k 抽取信息后，才能进行其他文本操作。完整的 a w k 脚本通常用来格式化文本文件中的信息。

1 调用 awk

有三种方式调用 a w k，第一种是命令行方式，如：

代码:

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

这里，c o m m a n d s 是真正的 a w k 命令。
上面例子中， [- F 域分隔符]是可选的，因为 a w k 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 p a s s w d 文件，此文件各域以冒号作为分隔符，则必须指明- F 选项，如：

代码:

```
awk -F:  'commands' input-file(s)
```

第二种方法是将所有 a w k 命令插入一个文件，并使 a w k 程序可执行，然后用 a w k 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：

代码:

```
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。

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，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。

awk读文件记录的方式

引用:

```
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
```

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。

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。

代码:

```
$ cat grade.txt
M.Tans 5/99 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.Tansl 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。

代码:

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

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。

代码:

```
$ 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：

引用：

```
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
```

这里我怎么看不明白，汗

4．打印所有记录

代码：

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

a w k 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。

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 f i e l d - 1 和 f i e l d - 4，因此可以使用$ 1 和$ 4，但不要忘了加逗号以分隔域。

代码：

```
$ awk '{print $1,$4}' grade.txt
M.Tans Green
J.Lulu green
P.Bunny Yellow
J.Troll Brown-3
L.Tansl 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 t a b 键加以划分。为加入 t a b 键，使用 t a b 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 n a m e 和 b e l t 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 B E G I N 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 a w k 查看第一条记录前，信息头被打印。

代码：

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

```
M.Tans    Green
J.Lulu    green
P.Bunny   Yellow
J.Troll   Brown-3
L.Tansl   Brown-2
```

7．打印信息尾

如果在末行加入 end of report 信息，可使用 E N D 语句。E N D 语句在所有文本处理动作执行完之后才被执行。E N D 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。

代码：

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

8. awk 错误信息提示

几乎可以肯定，在使用 a w k 时，将会在命令中碰到一些错误。a w k 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， a w k 将返回：

代码：

```
$ 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 a w k 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 a w k 错误时，可相应查找：

引用：

- 确保整个 a w k 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 a w k 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：

代码:

```
$ awk 'END {print NR}' grades.txt
awk: cmd. line:2: fatal: cannot open file `grades.txt' for reading (没有那个文件或目录)
```

9.awk 键盘输入

如果在命令行并没有输入文件 g r a d e . t x t，将会怎样？

代码:

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

B E G I N 部分打印了文件头，但 a w k 最终停止操作并等待，并没有返回 s h e l l 提示符。这是因为 a w k 期望获得键盘输入。因为没有给出输入文件， a w k 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<Ct r l - D >键。如果敲入了正确的域分隔符， a w k 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。

2.3awk 中正则表达式及其操作

在 g r e p 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 a w k 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 G r e e n，使用/ G r e e n / 可以查出单词 G r e e n 的出现情况。

2.4 元字符

这里是 a w k 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。

代码:

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

这里有两个字符第 7 章没有讲到，因为它们只适用于 a w k 而不适用于 g r e p 或 s e d。它们是：

引用:

+ 使用+匹配一个或多个字符。

? 匹配模式出现频率。例如使用/X Y?Z/匹配 X Y Z 或 Y Z。

条件操作符

a w k 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 i f 语句。a w k 中 i f 后面的条件用（）括起来。

观察文件 g r a d e . t x t，如果只要显示 b r o w n 腰带级别可知其所在域为 f i e l d - 4，这样可以写出表达式{if($4~/brown/) print }意即如果 f i e l d - 4 包含 b r o w n，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。

代码:

```
[root@Linux_chenwy sam]# awk '{if($4~/Brown/) print $0}' grade.txt
J.Troll 07/99 4842 Brown-3 12 26 26
L.Tansl 05/99 4712 Brown-2 12 30 28
```

匹配记录找到时，如果不特别声明， a w k 缺省打印整条记录。使用 i f 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 b r o w n，就打印它：

代码:

```
[root@Linux_chenwy sam]# awk '$0 ~ /Brown/' grade.txt
J.Troll 07/99 4842 Brown-3 12 26 26
L.Tansl 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 4 8，文件中有许多学生序号包含 4 8，如果在 f i e l d - 3 中查询序号 4 8，a w k 将返回所有序号带 4 8 的记录：

代码:

```
[root@Linux_chenwy sam]# awk '{if($3~/48/) print$0}' grade.txt
M.Tans 5/99 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 4 8，使用等号= =，并用单引号括起条件。例如$ 3

代码:

```
[root@Linux_chenwy sam]# awk '$3=="48" {print$0}' grade.txt
P.Bunny 02/99 48 Yellow 12 35 28
[root@Linux_chenwy sam]# awk '{if($3=="48") print$0}' grade.txt
P.Bunny 02/99 48 Yellow 12 35 28
```

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 b r o w n 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 b r o w n 腰带级别的记录并打印它。

注意，缺省情况下， a w k 将打印所有匹配记录，因此这里不必加入动作部分。

代码:

```
[root@Linux_chenwy sam]# awk '$0 !~ /Brown/' grade.txt
M.Tans 5/99 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 f i e l d - 4 进行不匹配操作，方法如下：

代码:

```
[root@Linux_chenwy sam]# awk '{if($4~/Brown/) print $0}' grade.txt
J.Troll 07/99 4842 Brown-3 12 26 26
L.Tansl 05/99 4712 Brown-2 12 30 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 b r o w n，将只匹配 ‘b r o w n 而不匹配 b r o w n - 2 和 b r o w n - 3，当然，如果想要查询非 b r o w n - 2 的腰带级别，可做如下操作：

代码:

```
[root@Linux_chenwy sam]# awk '$4!="Brown-2" {print $0}' grade.txt
M.Tans 5/99 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
```

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 f i e l d - 6 是否小于最高分 f i e l d - 7，在输出结果中，加入这一改动很容易。

代码:

```
[root@Linux_chenwy sam]# awk '{if($6 < $7) print $0}' grade.txt
M.Tans 5/99 48311 Green 8 40 44
J.Lulu 06/99 48317 green 9 24 26
```

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。

代码:

```
[root@Linux_chenwy sam]# awk '{if($6 <= $7) print $1}' grade.txt
M.Tans
J.Lulu
J.Troll
```

6．大于

代码:

```
[root@Linux_chenwy sam]# awk '{if($6 > $7) print $1}' grade.txt
P.Bunny
L.Tansl
```

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 g r e e n 的所有记录，不论其大小写，表达式应为 ‘ / [G g] r e e n /’

代码:

```
[root@Linux_chenwy sam]# awk '/[Gg]reen/' grade.txt
M.Tans 5/99 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。

代码:

```
[root@Linux_chenwy sam]# awk '$1 ~ /^...a/' grade.txt
M.Tans 5/99 48311 Green 8 40 44
L.Tansl 05/99 4712 Brown-2 12 30 28
```

9．或关系匹配

为抽取级别为 y e l l o w 或 b r o w n 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。

代码:

```
[root@Linux_chenwy sam]# 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.Tansl 05/99 4712 Brown-2 12 30 28
```

上面例子输出所有级别为 Ye l l o w 或 B r o w n 的记录。

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

代码:

```
[root@Linux_chenwy sam]# awk '/^M/' grade.txt
M.Tans 5/99 48311 Green 8 40 44
```

10. 行首

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

代码:

```
[root@Linux_chenwy sam]# awk '/^M/' grade.txt
```

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：

引用:

> && AND ： 语句两边必须同时匹配为真。
>
> || O R：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. B u n n y 且级别为 Ye l l o w，使用表达式($ 1 = = " P. B u n n y " & &$ 4 = = " Ye l l o w ")，意为& &两边匹配均为真。完整命令如下：

代码:

```
[root@Linux_chenwy sam]# 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 Ye l l o w 或 B r o w n，使用或命令。意为“ | |”符号两边的匹配模式之一

或全部为真。

代码:

```
[root@Linux_chenwy sam]# 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.Tansl 05/99 4712 Brown-2 12 30 28
```

原来不一定得加 print,下面我自己对例一二做了一下

代码:

```
1
[root@Linux_chenwy sam]# awk '$4~/Brown/' grade.txt
J.Troll 07/99 4842 Brown-3 12 26 26
L.Tansl 05/99 4712 Brown-2 12 30 28
```

代码:

```
2
[root@Linux_chenwy sam]# awk '$3=="48"' grade.txt
P.Bunny 02/99 48 Yellow 12 35 28
```

代码:

```
[root@Linux_chenwy sam]# awk '$3="48"' grade.txt
M.Tans 5/99 48 Green 8 40 44
J.Lulu 06/99 48 green 9 24 26
P.Bunny 02/99 48 Yellow 12 35 28
J.Troll 07/99 48 Brown-3 12 26 26
L.Tansl 05/99 48 Brown-2 12 30 28
```

2 中，我把=和==写错了，呵呵，一个是赋值，一个是等于

awk 内置变量

a w k 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。

引用:

> awk 内置变量
>
> A R G C 命令行参数个数
>
> A R G V 命令行参数排列
>
> E N V I R O N 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

引用：

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)。

NF、NR 和 FILENAME

要快速查看记录个数，应使用 N R。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 N R 将打印输入文件的记录个数。print NR 放在 E N D 语法中。

代码:

```
[root@chenwy sam]# awk 'END{print NR}' grade.txt
5
```

如：所有学生记录被打印，并带有其记录号。使用 N F 变量显示每一条读记录中有多少个域，并在 E N D 部分打印输入文件名。

[root@chenwy sam]# awk '{print NF,NR,$0} END{print FILENAME}' grade.txt

代码:

```
7 1 M.Tans 5/99 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.Tansl 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 B r o w n 级别记录。使用 A N D 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 B r o w n，最后打印结果。

代码:

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

N F 的一个强大功能是将变量$ P W D 的返回值传入 a w k 并显示其目录。这里需要指定域分隔符/。

代码:

```
[root@chenwy sam]# echo $PWD | awk -F/ ' {print $NF}'
sam
```

另一个例子是显示文件名。

代码:

```
[root@chenwy sam]# echo "/usr/local/etc/rc.sybase" | awk -F/ '{print $NF}'
rc.sybase
```

如果不指定域分割符，返回的如下：

代码:

```
[root@chenwy sam]# echo $PWD | awk  '{print $NF}'
/usr/sam
[root@chenwy sam]# echo "/usr/local/etc/rc.sybase" | awk '{print $NF}'
/usr/local/etc/rc.sybase
```

awk 操作符

在 a w k 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。

引用:

```
= += *= /= %= ^= 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 a w k 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 n a m e = $ n，这里 n a m e 为调用的域变量名， n 为实际域号。例如设置学生域名为 n a m e，级别域名为 b e l t，操作为 n a m e = $ 1 ; b e l t s = $ 4。注意分号的使用，它分隔 a w k 命令。下面例子中，重新赋值学生名域为 n a m e，级别域为 b e l t s。查询级别为 Ye l l o w 的记录，并最终打印名称和级别。

代码:

```
[sam@chenwy sam]$ 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 B E G I N 中给变量名赋值。

2) 在关系操作中使用实际数值。

通常在 B E G I N 部分赋值是很有益的，可以在 a w k 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 2 7 点以下的学生。

用引号将数字引用起来是可选的，“2 7”、2 7 产生同样的结果。

代码:

```
[sam@chenwy sam]$ awk '{if ($6<$7) print $0}' grade.txt
M.Tans 5/99 48311 Green 8 40 44
J.Lulu 06/99 48317 green 9 24 26
```

第二个例子中给数字赋以变量名 B A S E L I N E 和在 B E G I N 部分给变量赋值，两者意义相同。

代码:

```
[sam@chenwy sam]$ 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 a w k 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 a w k 复本。a w k 会在变量 N R 或 N F 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 . Ta n s l e y 的目前级别分域，使其数值从 4 0 减为 3 9，使用赋值语句$ 6 = $ 6 - 1，当然在实施修改前首先要匹配域名。

代码:

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

代码:

```
[sam@chenwy sam]$ awk '{if($1=="M.Tans") {$6=$6-1;print $1,$6,$7}}'
grade.txt
M.Tans 39 44
```

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．Tr o l l中加入字母，使其成为J．L．Tr o l l，表达式为$ 1 = " J．L．Tr o l l "，记住字符串要使用双秒号（ " "），并用圆括号括起整个语法。

代码:

```
[sam@chenwy sam]$ awk '{if($1=="J.Troll") $1="J.L.Troll"; print $1}' grade.txt
M.Tans
J.Lulu
P.Bunny
J.L.Troll
L.Tansl
```

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。

代码:

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

不知道为什么，我这里多了一个空行？

6．创建新的输出域

在ａ ｗ ｋ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。

代码:

```
[sam@chenwy sam]$ awk 'BEGIN{print "Name\tDifference"}{if($6<$7)
{$8=$7-$6;print $1,$8}}' grade.txt
```

```
Name    Difference
M.Tans 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：

代码:

```
[sam@chenwy sam]$ awk 'BEGIN{print "Name\tDifference"}{if($6<$7)
{diff=$7-$6;print $1,diff}}' grade.txt
Name    Difference
M.Tans 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 t o t a l，表达式为 t o t a l + = $ 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 a w k 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 t o t + = $ 6，t o t 即为 a w k 浏览的整个文件的域 6 结果总和。所有记录读完后，在 E N D 部分加入一些提示信息及域 6 总和。不必在 a w k 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。

代码:

```
[sam@chenwy sam]$ awk '(tot+=$6); END{print "Club student total points :" tot}'
grade.txt
M.Tans 5/99 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.Tansl 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。

代码:

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

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。
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。

代码:

```
[sam@chenwy sam]$ ls -l | awk '/^[^d]/ {print $9"\t"$5} {tot+=$5} END {print
"total KB:" tot}'
.....................
total KB:174144
```

内置的字符串函数

代码:

```
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。
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。
现在看一看 awk 中这些字符串函数的功能。

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 4 8 4 2 到 4 8 9 9：

代码:

```
[root@Linux_chenwy root]# cd /usr/sam
[root@Linux_chenwy sam]# awk 'gsub(/4842/,4899){print $0}' grade.txt
J.Troll 07/99 4899 Brown-3 12 26 26
```

代码:

```
[root@Linux_chenwy sam]# awk 'gsub(/4842/,4899)' grade.txt
J.Troll 07/99 4899 Brown-3 12 26 26
```

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 B u n n y 中 n y 出现的第一位置，即字符个数。

代码:

```
[root@Linux_chenwy sam]# 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 . Tr o l l 返回名字及其长度，即人名构成的字符个数

代码:

```
[root@Linux_chenwy sam]# 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。

代码:

```
[root@Linux_chenwy sam]# awk 'BEGIN{print length("A FEW GOOD MEN")}'
14
```

4. match

m a t c h 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 A N C D 中查找 d。因其不存在，所以返回 0。第二个例子在 A N C D 中查找 D。因其存在，所以返回 A N C D 中 D 出现的首位置字符数。第三个例子在学生 J . L u l u 中查找 u。

代码:

```
[root@Linux_chenwy sam]# awk 'BEGIN{print match("ANCD",/d/)}'
0
[root@Linux_chenwy sam]# awk 'BEGIN{print match("ANCD",/D/)}'
4
[root@Linux_chenwy sam]# 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。

代码:

```
[root@Linux_chenwy sam]# awk 'BEGIN {print split("123-456-789",pats_array,"-")}'3
```

还有一个例子使用不同的分隔符。

代码:

```
[root@Linux_chenwy sam]# awk 'BEGIN {print split("123#456#789",myarray,"#")}'
3
```

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

代码:

```
myarray[1]=123
myarray[2]=456
myarray[3]=789
```

结尾部分讲述数组概念。

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：‘pO Ped pope pill’。

如：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。

代码:

```
[root@Linux_chenwy sam]# awk '$1=="J.Troll" sub(/26/,"29",$0)' grade.txt
```

```
M.Tans 5/99 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.Tansl 05/99 4712 Brown-2 12 30 28
```

7. substr

s u b s t r 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：

代码:

```
[root@Linux_chenwy sam]# awk '$1=="L.Tansl" {print substr($1,1,3)}' grade.txt
L.T
```

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， a w k 将从起始位置返回所有字符，要抽取 L Ta n s l - e y 的姓，只需从第 3 个字符开始返回长度为 7。可以输入长度 9 9，a w k 返回结果相同。

代码:

```
[root@Linux_chenwy sam]# awk '$1=="L.Tansl" {print substr($1,1,99)}' grade.txt
L.Tansl
```

s u b s t r 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：

代码:

```
[root@Linux_chenwy sam]# awk '{print substr($1,3)}' grade.txt
Tans
Lulu
Bunny
Troll
Tansl
```

还有一个例子，在 B E G I N 部分定义字符串，在 E N D 部分返回从第 t 个字符开始抽取的子串。

代码:

```
[root@Linux_chenwy sam]# awk 'BEGIN{STR="A FEW GOOD MEN"}END{print
substr(STR,7)}' grade.txt
GOOD MEN
```

8. 从 s h e l l 中向 a w k 传入字符串

a w k 脚本大多只有一行，其中很少是字符串表示的。大多要求在一行内完成 a w k 脚本，这一点通过将变量传入 a w k 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 s t a n d - b y 传入 a w k，返回其长度。

代码:

```
[root@Linux_chenwy sam]# echo "Stand-by" | awk '{print length($0)}'
8
```

设置文件名为一变量，管道输出到 a w k，返回不带扩展名的文件名。

代码:

```
[root@Linux_chenwy sam]# STR="mydoc.txt"
[root@Linux_chenwy sam]# echo $STR|awk '{print substr($STR,1,5)}'
mydoc
```

设置文件名为一变量，管道输出到 a w k，只返回其扩展名。

代码:

```
[root@Linux_chenwy sam]# STR="mydoc.txt"
[root@Linux_chenwy sam]# echo $STR|awk '{print substr($STR,7)}'
txt
```

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 a w k 中失掉其特殊含义。

代码:

```
awk 中使用的屏蔽序列
\ b 退格键
\ t t a b 键
\ f 走纸换页
\ d d d 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 t a b 键，后跟两个新行，再打印 May Day，但这次使用八进制数 1 0 4、1 4 1、1 7 1、分别代表 D、a、y。

代码:

```
[root@chenwy sam]# awk 'BEGIN {print"\n\May\tDay\n\nMay\t\104\141\171"}'

May     Day

May     Day
```

注意，\ 1 0 4 为 D 的八进制 A S C I I 码，\ 1 4 1 为 a 的八进制 A S C I I 码，等等。

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 t a b 键以外没有任何格式。a w k 提供函数 p r i n t f，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 p r i n t f 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。
p r i n t f 函数基本语法是 p r i n t f（[格式控制符]，参数），格式控制字符通常在引号里。

printf 修饰符

代码:

```
- 左对齐
Wi d t h 域的步长，用 0 表示 0 步长
. p r e c 最大字符串长度，或小数点右边的位数
表 9-7 awk printf 格式
% c A S C I I 字符
% d 整数
% e 浮点数，科学记数法
% f 浮点数，例如（1 2 3 . 4 4）
% g a w k 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 A S C I I 码中 6 5 的等价值。管道输出 6 5 到 a w k。p r i n t f 进行 A S C I I 码字符转换。这里也加入换行，因为缺省情况下 p r i n t f 不做换行动作。

代码:

```
A[sam@chenwy sam]$ echo "65" | awk '{printf "%c\n",$0}'
A
```

按同样方式使用 a w k 得到同样结果。

代码:

```
[sam@chenwy sam]$ 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‘ 9 9 9’的输出结果。整数传入后被加了六个小数点。

代码:

```
[sam@chenwy sam]$ 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， 1 5 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。

代码:

```
[root@chenwy sam]# awk '{printf "%-15s %s\n",$1,$3}' grade.txt
M.Tans          48311
J.Lulu          48317
P.Bunny         48
J.Troll         4842
L.Tansl         4712
```

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 p r i n t 加入头信息。如果愿意，也可使用 p r i n t f。

代码:

```
[root@chenwy sam]# awk 'BEGIN{print "Name\t\tS.Number"}{printf "%-15s
%s\n",$1,$3}' grade.txt
Name            S.Number
M.Tans          48311
J.Lulu          48317
P.Bunny         48
J.Troll         4842
L.Tansl         4712
```

3.向一行 a w k 命令传值

在查看 a w k 脚本前，先来查看怎样在 a w k 命令行中传递变量。

在 a w k 执行前将值传入 a w k 变量，需要将变量放在命令行中，格式如下：

代码:

```
awk 命令变量=输入文件值
```

（后面会讲到怎样传递变量到 a w k 脚本中）。
下面的例子在命令行中设置变量 A G E 等于 1 0，然后传入 a w k 中，查询年龄在 1 0 岁以下的所有学生。

代码:

```
[root@chenwy sam]# awk '{if ($5<AGE) print $0}' AGE=10 grade.txt
M.Tans 5/99 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 a w k 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 a w k，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（1 0 2 4 块）而不是 d f 的文件头，然后对命令行与 ‘ i f ($ 4 < T R I G G E R)’ 上变量 T R I G G E R 中指定
的值进行查询测试。

代码:

```
[root@chenwy sam]# df -k|awk '{if($4<TRIGGER) print $6"\t"$4}'
TRIGGER=560000
/boot   458589
/dev/shm        99352
```

代码:

```
[root@chenwy sam]# df -k|awk '($4~/^[0-9]/) {if($4<TRIGGER) print $6"\t"$4}'
TRIGGER=5600000
/       2610716
/boot   458589
/dev/shm        99352
```

($4~/^[0-9]/)好像没什么用

在系统中使用 df -k 命令，产生下列信息：

代码:

```
[root@chenwy sam]# df -k
文件系统             1K-块       已用     可用 已用% 挂载点
/dev/sda2            5162828   2289804   2610764  47% /
/dev/sda1             497829     13538    458589   3% /boot
none                   99352         0     99352   0% /dev/shm
```

如果系统中 d f 输出格式不同，必须相应改变列号以适应工作系统。
当然可以使用管道将值传入 a w k。本例使用 w h o 命令， w h o 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。

代码:

```
[sam@chenwy sam]$ who |awk '{print $1" is logged on"}'
root is logged on
root is logged on
[sam@chenwy sam]$ who
root     :0           Nov 23 20:17
root     pts/0        Nov 23 20:25 (:0.0)
```

a w k 也允许传入环境变量。下面的例子使用环境变量 HOME 支持当前用户目录。可从 pwd 命令管道输出到 a w k 中获得相应信息。

代码:

```
[sam@chenwy sam]$ pwd | awk '{if ($1==derr) print $1}' derr=$HOME
/usr/sam
```

4. awk 脚本文件

可以将 a w k 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 a w k 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 a w k 可执行文件。像原来做的一样，将学生目前级别分相加 awk ‘（t o t + = $ 6） END{print "club student total points：" t o t }’ g r a d e . t x t。
创建新文件 s t u d e n t _ t o t . a w k，给所有 a w k 程序加入 a w k 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 a w k 程序。文本如下：

代码:

```
[sam@chenwy sam]$ cat student_tot.awk
#!/bin/awk -f
#all commnet lines must start with a hash '#'
#name:students_tots.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 proessing now let's print the total and average point
END{
print "Club student total points :" tot
print "Average Club Student Points:" tot/NR}
```

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。

代码:

```
[sam@chenwy sam]$ chmod u+x student_tot.awk
[sam@chenwy sam]$./student_tot.awk grade.txt
Student   Date   Member   No.   Grade Age   Points Max
Name      Joined                        Gained   Point Available
==============================================================
M.Tans 5/99 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.Tansl 05/99 4712 Brown-2 12 30 28
Club student total points :155
Average Club Student Points:31
```

过滤相同行：

如有一个文件 strip 中有多条重复错误提法：

代码:

```
[sam@Linux_chenwy sam]$ cat strip
etreiytrpytyu
ERROR*
ERROR*
ERROR*
ERROR*
IUEWROPYJRTMELUYK
```

```
ERROR*
ERROR*
ERROR*
ERROR*
ERROR*
ERROR*
EWUTIRWJYHT
ERROR*
ERROR*
JGIOERYO56ERU
ERROR*
ERROR*
ERROR*
JGEORYKP65EKU;YK,
```

现在用 a w k 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。awk 脚本如下：

代码：

```
[sam@Linux_chenwy sam]$ cat error_strip.awk
#!/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}
```

执行结果如下：

代码：

```
[sam@Linux_chenwy sam]$ ./error_strip.awk strip
etreiytrpytyu
ERROR*
IUEWROPYJRTMELUYK
ERROR*
```

```
EWUTIRWJYHT
ERROR*
JGIOERYO56ERU
ERROR*
JGEORYKP65EKU;YK,
```

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。

代码:

```
$awk -F: '{print $0}' inputfile
```

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。

我举的例子是第七个域:

代码:

```
[sam@Linux_chenwy sam]$ awk -F: '{print $1,"\t",$7}' passwd
root      /bin/bash
bin       /sbin/nologin
daemon    /sbin/nologin
adm       /sbin/nologin
lp        /sbin/nologin
sync      /bin/sync
.................................
```

这是不用脚本的，后面的结果省略

现使用脚本如下：

代码:

```
[sam@Linux_chenwy sam]$ cat passwd.awk
#!/bin/awk -f
#to call:passwd.awk /etc/passwd
#print out the first and seventh fields
BEGIN{
FS=":"}
{print $1,"\t",$7}
```

结果如下：

代码：

```
[sam@Linux_chenwy sam]$ chmod u+x passwd.awk
[sam@Linux_chenwy sam]$ ./passwd.awk passwd
root      /bin/bash
bin       /sbin/nologin
daemon    /sbin/nologin
adm       /sbin/nologin
lp        /sbin/nologin
sync      /bin/sync
.....................................
```

6．向 a w k 脚本传值

向 a w k 脚本传值与向 a w k 一行命令传值方式大体相同，格式为：

代码：

```
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 N F 变量 M A X，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。
脚本如下：

代码：

```
[sam@Linux_chenwy sam]$ cat 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 NF 中的值不等于最大 MAX 值，则打印出"哪一行的域总数不是 max"

如果以/ e t c / p a s s w d 作输入文件（p a s s w d 文件有 7 个域），运行上述脚本。参数格式如下：

代码：

```
[sam@Linux_chenwy sam]$ chmod u+x fieldcheck.awk
[sam@Linux_chenwy sam]$ ./fieldcheck.awk MAX=7 FS=":" passwd
```

正好 7 个域，如果改成 6，就会显示不同结果，试试看？

使用前面一行脚本的例子，将之转换成 a w k 脚本如下：

代码:

```
[sam@Linux_chenwy sam]$ cat name.awk
#!/bin/awk -f
#name:age.awk
#to call:age.awk AGE=n grade.txt
#print 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。

代码:

```
[sam@Linux_chenwy sam]$ chmod u+x name.awk
[sam@Linux_chenwy sam]$ ./name.awk AGE=10 grade.txt
M.Tans 5/99 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 a w k 脚本从 d u 命令获得输入，并输出块和字节数。

代码:

```
[root@Linux_chenwy sam]# cat 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}
```

使用 du 的结果如下

代码:

```
[root@Linux_chenwy sam]# du
12      ./.kde/Autostart
16      ./.kde
```

```
8       ./.xemacs
4       ./sam
4       ./dir1
4       ./file6
184     .
```

执行：

代码：

```
[root@Linux_chenwy sam]# du | ./duawk.awk
name                bytes   blocks

==================================
./.kde/Autostart                        6144    12
./.kde                      8192    16
./.xemacs                           4096    8
./sam                       2048    4
./dir1                      2048    4
./file6                     2048    4
.                           94208   184
```

代码：

```
OFS="\t";
```

是什么意思了，好累，晚上再研究，谁解答一下更好

数组

前面讲述 s p l i t 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：

代码：

```
[sam@Linux_chenwy          sam]$          awk          'BEGIN          {print
split("123#456#789",myarray,"#")}'
3
```

实际上 m y a r r a y 数组为

代码：

```
Myarray[1]="123"
Myarray[2]="456"
Myarray[3]="789"
```

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：

代码：

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

对于记录“ 1 2 3 # 4 5 6 # 6 7 8”，先使用 s p l i t 函数划分它，再使用循环打印各数组元素。操作脚本如下：

代码：

```
[sam@Linux_chenwy sam]$ cat 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/ d e v / n u l l 作为输入文件。

代码：

```
sam@Linux_chenwy sam]$chmod u+x arraytest.awk
[sam@Linux_chenwy sam]$ ./arraytest.awk /dev/null
123
456
789
[sam@Linux_chenwy sam]$
```

数组和记录

上面的例子讲述怎样通过 s p l i t 函数使用数组。也可以预先定义数组，并使用它与域进行比较测试，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的信息，有两个域，分隔符为（ #），文件如下：

代码：

```
[sam@Linux_chenwy sam]$ cat grade_student.txt
Yellow#Junior
Orange#Senior
Yellor#Junior
Purple#Junior
```

```
Brown-2#Junior
White#Senior
Orange#Senior
Red#Junior
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。

代码:

```
[sam@Linux_chenwy sam]$ cat 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 部分设置 FS 为符号#，即域分隔符

BEGIN{FS="#"

#Load the belt colours we are interested in only
#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。

belt["Yellow"]
```

```
belt["Orange"]
belt["Red"]
#end of BEGIN
#load the student type
student["Junior"]
student["Senior"]
}

##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。

#loop thru array that holds the belt colours against field-1
#if we have a match,keep a running total

#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。

{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 runing total

#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。

{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部分打印浏览结果，对每一个数组使用循环语句并打印它。

END{for (colour in belt )print "The club has ",belt[colour],colour,"Belts"

#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。

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

运行脚本前记住要加入可执行权限

代码:

```
[sam@Linux_chenwy sam]$ chmod u+x belts.awk
[sam@Linux_chenwy sam]$ ./belts.awk grade_student.txt
The club has  3 Red Belts
The club has  2 Orange Belts
The club has  2 Yellow Belts
The club has  7 Senior student
The club has  9 Junior student
```

shell 基础十：sed

sed 用法介绍

s e d 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。

引用:

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

可以在命令行输入 s e d 命令，也可以在一个文件中写入命令，然后调用 s e d，这与 a w k 基本相同。使用 s e d 需要记住的一个重要事实是，无论命令是什么， s e d 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 s e d 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。
本文介绍 s e d 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 s e d 用法的了解，取得宝贵的经验，以便最终自己编出大的复杂 s e d 脚本。
和 g r e p 与 a w k 一样， s e d 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 g r e p 与 a w k 相结合。

1 sed 怎样读取数据

s e d 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。

2 调用 sed

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

使用 s e d 命令行格式为：

代码：

```
sed [选项] s e d 命令输入文件。
```

记住在命令行使用 s e d 命令时，实际命令要加单引号。s e d 也允许加双引号。

使用 s e d 脚本文件，格式为：

代码：

```
sed [选项] -f sed 脚本文件输入文件
```

要使用第一行具有 s e d 命令解释器的 s e d 脚本文件，其格式为：

代码：

```
s e d 脚本文件[选项]  输入文件
```

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

引用：

> s e d 选项如下：
> n 不打印；s e d 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 s e d 命令，此选项无用，但指定它也没有关系。
> f 如果正在调用 s e d 脚本文件，使用此选项。此选项通知 s e d 一个脚本文件支持所有的 s e d 命令，例如：sed -f myscript.sed input_file，这里 m y s c r i p t . s e d 即为支持 s e d 命令的文件。

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 s e d 命令的所有输出至文件 ‘ m y o u t f i l e’，当对结果很满意时使用这

种方法。

代码:

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

2.2 使用 sed 在文件中查询文本的方式

s e d 浏览输入文件时，缺省从第一行开始，有两种方式定位文本：

引用:

> 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

下面是使用 s e d 定位文本的一些方式。

代码:

```
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
/ p a t t e r n / 查询包含模式的行。例如/ d i s k /或/[a-z]/
/ p a t t e r n / p a t t e r n / 查询包含两个模式的行。例如/ d i s k / d i s k s /
p a t t e r n / , x 在给定行号上查询包含模式的行。如/ r i b b o n / , 3
x , / p a t t e r n / 通过行号和模式查询匹配行。3 . / v d u /
x , y ! 查询不包含指定行号 x 和 y 的行。1 , 2 !
```

2.3 基本 sed 编辑命令

代码:

```
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 A S C I I 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
```

sed 和正则表达式

s e d 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

s e d 例子中使用下述文本文件 q u o t e . t x t。

代码:

```
[sam@Linux_chenwy sam]$ cat 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

1 使用 p（rint）显示行

只打印第二行，用-n

代码:

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

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。

代码:

```
[sam@Linux_chenwy sam]$ 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 23:00.
```

3 打印模式

假定要匹配单词 N e a v e，并打印此行，方法如下。使用模式/ p a t t e r n /格式，这里为/ N e a v e /。

代码:

```
[sam@Linux_chenwy sam]$ 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.
```

4 使用模式和行号进行查询

可以将行号和模式结合使用。假定要改动文件 q u o t e . t x t 最后一行中的单词 t h e，使用 s e d 查询 t h e，返回两行：

代码:

```
[sam@Linux_chenwy sam]$ 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 l i n e _ n u m b e r, / p a t t e r n /。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 , / t h e /。意即只在第四行查询模式 t h e，命令如下：

代码:

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

上面有错，其实是把第四行后的都打出来了

这个模式应该哪果指定行找不到符合条件的，就从下一行开始查找，直到找到为止，并把，找到行之前的全部打打印出来。

如果指定行本身就符合条伯，把本行及后面的行的全部打印出来

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。

代码:

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

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。

代码:

```
[sam@Linux_chenwy sam]$ 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

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 i n g 结尾，模式为/ . * i n g /。可以使用这个模式查询以 i n g 结尾的任意单词。

代码:

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

8 首行

要打印文件第一行，使用行号：

代码:

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

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。

代码:

```
[sam@Linux_chenwy sam]$ sed -n '$p' quote.txt
The local nurse Miss P.Neave was in attendance.
```

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/ p a t t e r n / =。

代码:

```
[sam@Linux_chenwy sam]$ 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 23:00.
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。

代码:

```
[sam@Linux_chenwy sam]$ sed -n '/music/=' quote.txt
2
```

如果只打印行号及匹配行，必须使用两个 s e d 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/=

代码:

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

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， s e d 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 s e d 执行时，首先将文件的一行文本拷贝至缓冲区，在这里 s e d 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，s e d 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 s e d 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：

代码:

```
[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 s e d 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。s e d 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， s e d 假定这是附加命令结尾。

当附加或插入文本或键入几个 s e d 命令时，可以利用辅助的 s h e l l 提示符以输入多行命令。
当附加或插入文本或键入几个 s e d 命令时，可以利用辅助的 s h e l l 提示符以输入多行命令。
创建 sed 脚本文件

创建脚本文件 a p p e n d . s e d：
第一行是 s e d 命令解释行。脚本在这一行查找 s e d 以运行命令，这里定位在/ b i n。
第二行以/ c o m p a n y /开始，这是附加操作起始位置。a \通知 s e d 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。

代码:

```
[sam@chenwy sam]$ cat append.sed
#!/bin/sed -f
```

```
/company/ a\
Then suddenly it happed.
```

保存它，增加可执行权限,运行

代码：

```
[sam@chenwy sam]chmod u+x append.sed
[sam@chenwy sam]$ ./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 happed.
Too bad the disco floor fell through at 23:00.
The local nurse Miss P.Neave was in attendance.
```

或直接用命令行：

代码：

```
[sam@chenwy sam]$ sed "/company/a\Then suddenly it happened." 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

[sam@chenwy sam]$ sed "/company/i\utter confusion followed." quote.txt
The honeysuckle band played all night long for only $90.
utter confusion followed.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:00.
The local nurse Miss P.Neave was in attendance.

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。
如在 a t t e n d a n c e 结尾的行前插入文本 utter confusion followed。

代码：

```
[sam@chenwy sam]$ sed "/company/i\Utter confusion followed." quote.txt
```

也可以指定行：

代码：

```
[sam@chenwy sam]$ cat insert.sed
#!/bin/sed -f
4 i\
Utter confusion followed.
```

执行结果

代码：

```
[sam@chenwy sam]$ chmod u+x insert.sed
[sam@chenwy sam]$ ./insert.sed 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 23:00.
Utter confusion followed.
The local nurse Miss P.Neave was in attendance.
```

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：

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：

代码：

```
[sam@chenwy sam]$ cat change.sed
#!/bin/sed -f
3 c\
The office Dibble band played well.
```

代码：

```
[sam@chenwy sam]$ chmod u+x change.sed
[sam@chenwy sam]$ ./change.sed quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
The office Dibble band played well.
The local nurse Miss P.Neave was in attendance.
```

或命令行：

代码：

```
[sam@chenwy sam]$ sed "/honeysuck/c\The Office Dibble band played well." 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

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。

代码:

```
[sam@chenwy sam]$ cat mix.sed
#!/bin/sed -f
1 c\
The Dibble band were grooving.

/evening/ i\
They played some great tunes.

3 a\
Where was the nurse to help?
```

代码:

```
[sam@chenwy sam]$ chmod u+x mix.sed
[sam@chenwy sam]$ ./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 23:00.
Where was the nurse to help?
The local nurse Miss P.Neave was in attendance.
```

删除文本

s e d 删除文本格式：

代码:

```
[ a d d r e s s [, a d d r e s s ] ] d
```

删除第一行；1 d 意为删除第一行。

代码:

```
[sam@chenwy sam]$ sed '1d' quote.txt
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:00.
The local nurse Miss P.Neave was in attendance.
```

删除第一到第三行：

代码：

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

删除最后一行：

代码：

```
[sam@chenwy sam]$ 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 23:00.
```

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。

代码：

```
[sam@chenwy sam]$ 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 23:00.
```

替换文本

替换命令用替换模式替换指定模式，格式为：

代码：

```
[ a d d r e s s [，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：

引用：

> 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 s e d 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。

如替换 n i g h t 为 N I G H T，首先查询模式 n i g h t，然后用文本 N I G H T 替换它。

代码:

```
[sam@chenwy sam]$ sed 's/night/NIGHT/' 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

要从$ 9 0 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 r e p l a c e m e n t - p a t t e r n 部分不写任何东西，保留空白，但仍需要用斜线括起来。在 s e d 中也可以这样删除一个字符串。

代码:

```
[sam@chenwy sam]$ sed 's/\$//' 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

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 T h e 替换成 Wo w!。

代码:

```
[sam@chenwy sam]$ 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 23:00.
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 s p l e n d i d 替换为 S P L E N D I D 的替换结果写入文件 s e d . o u t：

代码:

```
[sam@chenwy sam]$ sed 's/splendid/SPLENDID/w sed.out' 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

注意要将文件名括在 s e d 的单引号里。文件结果如下：

代码:

```
[sam@chenwy sam]$ cat sed.out
It was an evening of SPLENDID music and company.
```

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。

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， s e d 语句 s/nurse/"Hello"&/p 的结果如下

代码:

```
[sam@chenwy sam]$ 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：

代码:

```
[sam@chenwy sam]$ 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。

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 s e d 命令中也可以将结果输入文件。格式有点像使用替换命令：

代码:

```
[ a d d r e s s [，address]]w filename
```

‘w’选项通知 s e d 将结果写入文件。f i l e n a m e 是自解释文件名。

下面有两个例子。

代码:

```
[sam@chenwy sam]$ sed '1,2 w filedt' 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

文件 q u o t e . t x t 输出到屏幕。模式范围即 1，2 行输出到文件 f i l e d t。

代码:

```
[sam@chenwy sam]$ cat 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 N e a v e，匹配结果行写入文件 f i l e d h t。

代码:

```
[sam@chenwy sam]$ sed '/Neave/ w dht' 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

代码:

```
[sam@chenwy sam]$ cat dht
The local nurse Miss P.Neave was in attendance.
```

从文件中读文本

处理文件时， s e d 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：

代码:

```
address r filename
```

这里 r 通知 s e d 将从另一个文件源中读文本。f i l e n a m e 是其文件名。

现在创建一个小文件 s e d e x . t x t，内容如下：

代码:

```
[sam@chenwy sam]$ echo "Boom boom went the music" >sedex.txt
[sam@chenwy sam]$ cat sedex.txt
Boom boom went the music
```

将 s e d e x . t x t 内容附加到文件 q u o t e . t x t 的拷贝。在模式匹配行/ c o m p a n y / 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。

代码:

```
[sam@chenwy sam]$ 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 23:00.
The local nurse Miss P.Neave was in attendance.
```

匹配后退出

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

代码:

```
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。查询首次出现模式，然后退出。需要将 q 放在 s e d 语句末尾。

代码:

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

显示文件中的控制字符

1、$vi dos.txt

进入 vi 后，用 ctrl+v 再用 ctrl+M 产生控制字符^M 不知对不对

使用 cat -v filename 命令查看编辑好的文件

代码:

```
[sam@chenwy sam]$ cat -v dos.txt
12332##DISO##45.12^M
00332##LPSO##23.14^M
01299##USPD##34.46^M
```

s e d 格式为：

代码:

```
[ a d d r e s s, [ a d d r e s s ] ] 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。

代码:

```
[sam@chenwy sam]$ sed -n '1,$l' dos.txt
12332##DISO##45.12\r$
00332##LPSO##23.14\r$
01299##USPD##34.46\r$
```

处理控制字符

使用 s e d 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（ d o s . t x t）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。

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。

代码:

```
[sam@chenwy sam]$ sed 's/##/ /g' dos.txt
12332 DISO 45.12
00332 LPSO 23.14
01299 USPD 34.46
```

。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。

代码:

```
[sam@chenwy sam]$ sed 's/##/ /g;s/^0*/ /g' dos.txt
12332 DISO 45.12
332 LPSO 23.14
1299 USPD 34.46
```

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。**
在命令行里也必须用^M 控制字符耶！？

代码:

```
[sam@chenwy sam]$ sed 's/##/ /g;s/^0*/ /g;s/^M/ /g' dos.txt
12332 DISO 45.12
332 LPSO 23.14
1299 USPD 34.46
```

或

代码:

```
[sam@chenwy sam]$ cat dos.txt | sed 's/^0*/ /g' | sed 's/^M/ /g' | sed 's/###/ /g'
```

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 s q l 查询结果。

代码:

```
[sam@chenwy sam]$ cat data.txt
Database Size(MB) DataCreated
----------------------------
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 1 d 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 c a t，并管道传送结果到 s e d 命令。

代码:

```
[sam@chenwy sam]$ cat data.txt |sed 's/--*/ /g' | sed '/^$/d' | sed '$d' | sed '1d'
| awk '{print $1}'
GOSOUTH
TRISUD
```

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：

代码:

```
[sam@chenwy sam]$ cat ok.txt
AC456
AC492169
AC9967
```

```
AC88345
```

任务是在每一行末尾加一个字符串‘ p a s s e d’。
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。

代码:

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

从 shell 向 sed 传值

要从命令行中向 s e d 传值，值得注意的是用双引号，否则功能不执行。

代码:

```
[sam@chenwy sam]$ NAME="It's a go situation"
[sam@chenwy sam]$ REPLACE="GO"
[sam@chenwy sam]$ echo $NAME | sed "s/go/$REPLACE/g"
It's a GO situation
```

从 sed 输出中设置 shell 变量

从 s e d 输出中设置 s h e l l 变量是一个简单的替换过程。运用上面的例子，创建 s h e l l 变量 N E W- N A M E，保存上述 s e d 例子的输出结果。

代码:

```
[sam@chenwy sam]$ NAME="It's a go situation"
[sam@chenwy sam]$ REPLACE="GO"
[sam@chenwy sam]$ NEW_NAME=`echo $NAME | sed "s/go/$REPLACE/g"`
[sam@chenwy sam]$ echo $NEW_NAME
It's a GO situation
```

这里的`是键盘左上角那个`

下面是一些一行命令集。（[ ]表示空格， [ ]表示 t a b 键）

引用:

‘s / \ . $ / / g’ 删除以句点结尾行

‘-e /abcd/d’ 删除包含 a b c d 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 /CO L \ ( . . . \ ) / / g’  删除紧跟 C O L 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 t a b 键替代
‘S / ^ [ ] / / g’  删除行首所有 t a b 键
‘s / [ ] * / / g’  删除所有 t a b 键
```

1. 删除路径名第一个\符号

将当前工作目录返回给 s e d，删除第一个\：

代码:

```
[sam@chenwy sam]$ echo $PWD |sed 's/^\///g'
usr/sam
```

2. 追加/插入文本

将"Mr Wi l l i s "字串返回给 s e d 并在 M r 后而追加" B r u c e "。

代码:

```
[sam@chenwy sam]$ echo "Mr Willis" |sed 's/Mr /& Bruce/g'
Mr  BruceWillis
```

3. 删除首字符

s e d 删除字符串“a c c o u n t s . d o c”首字符。

代码:

```
[sam@chenwy sam]$ echo "accounts.doc" |sed 's/^.//g'
ccounts.doc
```

4. 删除文件扩展名

s e d 删除“a c c o u n t s . d o c”文件扩展名。

代码:

```
[sam@chenwy sam]$ echo "accounts.doc"|sed 's/.doc//g'
accounts
```

5．增加文件扩展名
s e d 附加字符串“．d o c”到字符串“a c c o u n t s”。

代码:

```
[sam@chenwy sam]$ 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：

代码:

```
[sam@chenwy sam]$ x="Department+playroll&Building G"
[sam@chenwy sam]$ echo $x
Department+playroll&Building G
```

如果要加入 of,located,并去掉+,&实现下列转换：

代码:

```
[sam@chenwy sam]$ echo $x |sed 's/\+/ of /g' |sed 's/\&/ Located at /g'
Department of playroll Located at Building G
```

把+用 of 替换，&用 located at 替换

——————————————
sed完

转一个贴了，不知到有没有人转过

挑选编辑器
在 UNIX 世界中有很多文本编辑器可供我们选择。思考一下 -- vi、emacs 和 jed 以及很多其它工具都会浮现在脑海中。我们都有自己已逐渐了解并且喜爱的编辑器（以及我们喜爱的组合键）。有了可信赖的编辑器，我们可以轻松处理任何数量与 UNIX 有关的管理或编程任务。

虽然交互式编辑器很棒，但却有其限制。尽管其交互式特性可以成为强项，但也有其不足之处。考虑一下需要对一组文件执行类似更改的情形。您可能会本能地运行自己所喜爱的编辑器，然后手工执行一组烦琐、重复和耗时的编辑任务。然而，有一种更好的方法。

进入 sed
如果可以使编辑文件的过程自动化，以便用“批处理”方式编辑文件，甚至编写可以对现有文件进行复杂更改的脚本，那将太好了。幸运的是，对于这种情况，有一种更好的方法 -- 这种更好

的方法称为 "sed"。

sed 是一种几乎包括在所有 UNIX 平台（包括 Linux）的轻量级流编辑器。sed 有许多很好的特性。首先，它相当小巧，通常要比您所喜爱的脚本语言小很多倍。其次，因为 sed 是一种流编辑器，所以，它可以对从如管道这样的标准输入接收的数据进行编辑。因此，无需将要编辑的数据存储在磁盘上的文件中。因为可以轻易将数据管道输出到 sed，所以，将 sed 用作强大的 shell 脚本中长而复杂的管道很容易。试一下用您所喜爱的编辑器去那样做。

GNU sed

对 Linux 用户来说幸运的是，最好的 sed 版本之一恰好是 GNU sed，其当前版本是 3.02。每一个 Linux 发行版都有（或至少应该有）GNU sed。GNU sed 之所以流行不仅因为可以自由分发其源代码，还因为它恰巧有许多对 POSIX sed 标准便利、省时的扩展。另外，GNU 没有 sed 早期专门版本的很多限制，如行长度限制 -- GNU 可以轻松处理任意长度的行。

最新的 GNU sed

在研究这篇文章之时我注意到：几个在线 sed 爱好者提到 GNU sed 3.02a。奇怪的是，在 ftp.gnu.org（有关这些链接，请参阅参考资料）上找不到 sed 3.02a，所以，我只得在别处寻找。我在alpha.gnu.org 的 /pub/sed 中找到了它。于是我高兴地将其下载、编译然后安装，而几分钟后我发现最新的 sed 版本却是 3.02.80 -- 可在alpha.gnu.org 上 3.02a 源代码旁边找到其源代码。安装完 GNU sed 3.02.80 之后，我就完全准备好了。

alpha.gnu.org

alpha.gnu.org（请参阅参考资料）是新的和实验性 GNU 源代码的所在地。然而，您还会在那里发现许多优秀、稳定的源代码。出于某种原因，不是许多 GNU 开发人员忘记将稳定的源代码移至 ftp.gnu.org，就是它们的 "beta" 期间格外长（2 年!）。例如，sed 3.02a 已有两年，甚至 3.02.80 也有一年，但它们仍不能（在 2000 年 8 月写本文章时）在 ftp.gnu.org 上获得。

正确的 sed

在本系列中，将使用 GNU sed 3.02.80。在即将出现的本系列后续文章中，某些（但非常少）最高级的示例将不能在 GNU sed 3.02 或 3.02a 中使用。如果您使用的不是 GNU sed，那么结果可能会不同。现在为什么不花些时间安装 GNU sed 3.02.80 呢？那样，不仅可以为本系列的余下部分作好准备，而且还可以使用可能是目前最好的 sed。

sed 示例

sed 通过对输入数据执行任意数量用户指定的编辑操作（“命令”）来工作。sed 是基于行的，因此按顺序对每一行执行命令。然后，sed 将其结果写入标准输出 (stdout)，它不修改任何输入文件。

让我们看一些示例。头几个会有些奇怪，因为我要用它们演示 sed 如何工作，而不是执行任何有用的任务。然而，如果您是 sed 新手，那么理解它们是十分重要的。下面是第一个示例：

代码:

```
$ sed -e 'd' /etc/services
```

如果输入该命令，将得不到任何输出。那么，发生了什么？
在该例中，用一个编辑命令 'd' 调用 sed。sed 打开 /etc/services 文件，将一行读入其模式缓冲区，执行编辑命令（“删除行”），然后打印模式缓冲区（缓冲区已为空）。然后，它对后面的每一行重复这些步骤。这不会产生输出，因为 "d" 命令除去了模式缓冲区中的每一行！

在该例中，还有几件事要注意。首先，根本没有修改 /etc/services。这还是因为 sed 只读取在命令行指定的文件，将其用作输入 -- 它不试图修改该文件。第二件要注意的事是 sed 是面向行的。'd' 命令不是简单地告诉 sed 一下子删除所有输入数据。相反，sed 逐行将 /etc/services 的每一行读入其称为模式缓冲区的内部缓冲区。一旦将一行读入模式缓冲区，它就执行 'd' 命令，然后打印模式缓冲区的内容（在本例中没有内容）。我将在后面为您演示如何使用地址范围来控制将命令应用到哪些行 -- 但是，如果不使用地址，命令将应用到所有行。

第三件要注意的事是括起 'd' 命令的单引号的用法。养成使用单引号来括起 sed 命令的习惯是个好注意，这样可以禁用 shell 扩展。

另一个 sed 示例
下面是使用 sed 从输出流除去 /etc/services 文件第一行的示例：

代码：

```
$ sed -e '1d' /etc/services | more
```

地址范围
现在，让我们看一下如何指定地址范围。在本例中，sed 将删除输出的第 1 到 10 行：

代码：

```
$ sed -e '1,10d' /etc/services | more
```

当用逗号将两个地址分开时，sed 将把后面的命令应用到从第一个地址开始、到第二个地址结束的范围。在本例中，将 'd' 命令应用到第 1 到 10 行（包括这两行）。所有其它行都被忽略。

带规则表达式的地址
现在演示一个更有用的示例。假设要查看 /etc/services 文件的内容，但是对查看其中包括的注释部分不感兴趣。如您所知，可以通过以 '#' 字符开头的行在 /etc/services 文件中放置注释。为了避免注释，我们希望 sed 删除以 '#' 开始的行。以下是具体做法：

代码：

```
$ sed -e '/^#/d' /etc/services | more
```

让我们分析发生的情况。

要理解 '/^#/d' 命令，首先需要对其剖析。首先，让我们除去 'd' -- 这是我们前面所使用的同一个删除行命令。新增加的是 '/^#/' 部分，它是一种新的规则表达式地址。规则表达式地址总是由斜杠括起。它们指定一种 模式，紧跟在规则表达式地址之后的命令将仅适用于正好与该特定模式匹配的行。

因此，'/^#/' 是一个规则表达式。但是，它做些什么呢？很明显，现在该复习规则表达式了。

规则表达式复习

可以使用规则表达式来表示可能会在文本中发现的模式。您在 shell 命令行中用过 '*' 字符吗？这种用法与规则表达式类似，但并不相同。下面是可以在规则表达式中使用的特殊字符：

引用：

字符 描述

与行首匹配

与行末尾匹配

与任一个字符匹配

将与前一个字符的零或多个出现匹配

[ ] 与 [ ] 之内的所有字符匹配

感受规则表达式的最好方法可能是看几个示例。所有这些示例都将被 sed 作为合法地址接受，这些地址出现在命令的左边。下面是几个示例：

引用：

规则

表达式 描述

/./ 将与包含至少一个字符的任何行匹配

/../ 将与包含至少两个字符的任何行匹配

/^#/ 将与以 '#' 开始的任何行匹配

/^$/ 将与所有空行匹配

/}^/ 将与以 '}'（无空格）结束的任何行匹配

/} *^/ 将与以 '}' 后面跟有零或多个空格结束的任何行匹配

/[abc]/ 将与包含小写 'a'、'b' 或 'c' 的任何行匹配

/^[abc]/ 将与以 'a'、'b' 或 'c'开始的任何行匹配

在这些示例中，鼓励您尝试几个。花一些时间熟悉规则表达式，然后尝试几个自己创建的规则表达式。可以如下使用 /^#/：

代码：

```
$ sed -e '/^#/d' /etc/services | more
```

这将导致 sed 删除任何匹配的行。删除以#开头的行

另一个例子：

代码：

```
$ sed -n -e '/^#/p' /path/to/my/test/file | more
```

请注意新的 '-n' 选项，该选项告诉 sed 除非明确要求打印模式空间，否则不这样做。您还会注意到，我们用 'p' 命令替换了 'd' 命令，如您所猜想的那样，这明确要求 sed 打印模式空间。就这样，将只打印匹配部分。打印以#开头的行

有关地址的更多内容

目前为止，我们已经看到了行地址、行范围地址和 ^# 地址。但是，还有更多的可能。我们可以指定两个用逗号分开的规则表达式，sed 将与所有从匹配第一个规则表达式的第一行开始，到匹配第二个规则表达式的行结束（包括该行）的所有行匹配。例如，以下命令将打印从包含 "BEGIN" 的行开始，并且以包含 "END" 的行结束的文本块：

代码：

```
$ sed -n -e '/BEGIN/,/^END/p' /my/test/file | more
```

如果没发现 "BEGIN"，那么将不打印数据。如果发现了 "BEGIN"，但是在这之后的所有行中都没发现 "END"，那么将打印所有后续行。发生这种情况是因为 sed 面向流的特性 -- 它不知道是否会出现 "END"。

C 源代码示例

如果只要打印 C 源文件中的 main() 函数，可输入：

代码：

```
$ sed -n -e '/main[[:space:]]*(/,/^}/p' sourcefile.c | more
```

以 main 后面跟空格或制表键，以(开头，)结尾的

该命令有两个规则表达式 '/main[[:space:]]*(/' 和 '/^)/'，以及一个命令 'p'。第一个规则表达式将与后面依次跟有任意数量的空格或制表键以及开始圆括号的字符串 "main" 匹配。这应该与一般 ANSI C main() 声明的开始匹配。

在这个特别的规则表达式中，出现了 '[[:space:]]' 字符类。这只是一个特殊的关键字，它告诉 sed 与 TAB 或空格匹配。如果愿意的话，可以不输入 '[[:space:]]'，而输入 '['，然后是空格字母，然后是 -V，然后再输入制表键字母和 ']' -- Control-V 告诉 bash 要插入“真正”的制表键，而不是执行命令扩展。使用 '[[:space:]]' 命令类（特别是在脚本中）会更清楚。

好，现在看一下第二个 regexp。'/^)' 将与任何出现在新行行首的 ')' 字符匹配。如果代码的格式很好，那么这将与 main() 函数的结束花括号匹配。如果格式不好，则不会正确匹配 -- 这是执行模式匹配任务的一件棘手之事。

因为是处于 '-n' 安静方式，所以 'p' 命令还是完成其惯有任务，即明确告诉 sed 打印该行。试着对 C 源文件运行该命令 -- 它应该输出整个 main() { } 块，包括开始的 "main()" 和结束的 ')'。

替换！

让我们看一下 sed 最有用的命令之一，替换命令。使用该命令，可以将特定字符串或匹配的规则表达式用另一个字符串替换。下面是该命令最基本用法的示例：

代码：

```
$ sed -e 's/foo/bar/' myfile.txt
```

上面的命令将 myfile.txt 中每行第一次出现的 'foo'（如果有的话）用字符串 'bar' 替换，然后将该文件内容输出到标准输出。请注意，我说的是每行第一次出现，尽管这通常不是您想要的。在进行字符串替换时，通常想执行全局替换。也就是说，要替换每行中的所有出现，如下所示：

代码：

```
$ sed -e 's/foo/bar/g' myfile.txt
```

在最后一个斜杠之后附加的 'g' 选项告诉 sed 执行全局替换。

关于 's///' 替换命令，还有其它几件要了解的事。首先，它是一个命令，并且只是一个命令，在所有上例中都没有指定地址。这意味着，'s///' 还可以与地址一起使用来控制要将命令应用到哪些行，如下所示：

代码：

```
$ sed -e '1,10s/enchantment/entrapment/g' myfile2.txt
```

上例将导致用短语 'entrapment' 替换所有出现的短语 'enchantment'，但是只在第一到第十行（包括这两行）上这样做。

代码：

```
$ sed -e '/^$/,/^END/s/hills/mountains/g' myfile3.txt
```

该例将用 'mountains' 替换 'hills'，但是，只从空行开始，到以三个字符 'END' 开始的行结束（包括这两行）的文本块上这样做。

关于 's///' 命令的另一个妙处是 '/' 分隔符有许多替换选项。如果正在执行字符串替换，并且规则表达式或替换字符串中有许多斜杠，则可以通过在 's' 之后指定一个不同的字符来更改分隔符。例如，下例将把所有出现的 /usr/local 替换成 /usr：

代码：

```
$ sed -e 's:/usr/local:/usr:g' mylist.txt
```

在该例中，使用冒号作为分隔符。如果不指定分隔符，则变成了如下：

代码：

```
$ sed -e 's/usr/local/usrg' mylist.txt
```

这样就不能执行了
如果需要在规则表达式中指定分隔符字符，可以在它前面加入反斜杠。

规则表达式混乱
目前为止，我们只执行了简单的字符串替换。虽然这很方便，但是我们还可以匹配规则表达式。例如，以下 sed 命令将匹配从 '<' 开始、到 '>' 结束、并且在其中包含任意数量字符的短语。下例将删除该短语（用空字符串替换）：

代码：

```
$ sed -e 's/<.*>//g' myfile.html
```

这是要从文件除去 HTML 标记的第一个很好的 sed 脚本尝试，但是由于规则表达式的特有规则，它不会很好地工作。原因何在？当 sed 试图在行中匹配规则表达式时，它要在行中查找最长的匹配。在我的前一篇 sed 文章中，这不成问题，因为我们使用的是 'd' 和 'p' 命令，这些命令总要删除或打印整行。但是，在使用 's///' 命令时，确实有很大不同，因为规则表达式匹配的整个部分将被目标字符串替换，或者，在本例中，被删除。这意味着，上例将把下行：

代码：

```
<b>This</b> is what <b>I</b> meant.

变成：
```

```
meant.
我们要的不是这个，而是：
This is what I meant.
```

幸运的是，有一种简便方法来纠正该问题。我们不输入“'<' 字符后面跟有一些字符并以 '>' 字符结束”的规则表达式，
而只需输入一个“'<' 字符，后面跟有任意数量非 '>' 字符，并以 '>' 字符结束”的规则表达式。这将与最短、而不是最长的可能性匹配。新命令如下：

代码：

```
$ sed -e 's/<[^>]*>//g' myfile.html
```

在上例中，'[^>]' 指定“非 '>'”字符，其后的 '*' 完成该表达式以表示“零或多个非 '>' 字符”。对几个 html 文件测试该命令，将它们管道输出 "more"，然后仔细查看其结果。

更多字符匹配

'[ ]' 规则表达式语法还有一些附加选项。要指定字符范围，只要字符不在第一个或最后一个位置，就可以使用 '-'，如下所示：

引用：

'[a-x]*'

这将匹配零或多个全部为 'a'、'b'、'c'...'v'、'w'、'x' 的字符。另外，可以使用 '[:space:]' 字符类来匹配空格。以下是可用字符类的相当完整的列表：

字符类 描述

[:alnum:] 字母数字 [a-z A-Z 0-9]

[:alpha:] 字母 [a-z A-Z]

[:blank:] 空格或制表键

[:cntrl:] 任何控制字符

[:digit:] 数字 [0-9]

[:graph:] 任何可视字符（无空格）

[:lower:] 小写 [a-z]

[:print:] 非控制字符

[:punct:] 标点字符

[:space:] 空格

[:upper:] 大写 [A-Z]

```
[:xdigit:] 十六进制数字 [0-9 a-f A-F]
```

尽可能使用字符类是很有利的，因为它们可以更好地适应非英语 locale（包括某些必需的重音字符等等）.

高级替换功能

我们已经看到如何执行简单甚至有些复杂的直接替换，但是 sed 还可以做更多的事。实际上可以引用匹配规则表达式的部分或全部，并使用这些部分来构造替换字符串。作为示例，假设您正在回复一条消息。下例将在每一行前面加上短语 "ralph said: "：

代码:

```
$ sed -e 's/.*/ralph said: &/' origmsg.txt
```

输出如下：

代码:

```
ralph said: Hiya Jim, ralph said: ralph said:
I sure like this sed stuff! ralph said:
```

该例的替换字符串中使用了 '&' 字符，该字符告诉 sed 插入整个匹配的规则表达式。因此，可以将与 '.*' 匹配的任何内容（行中的零或多个字符的最大组或整行）插入到替换字符串中的任何位置，甚至多次插入。这非常好，但 sed 甚至更强大。

那些极好的带反斜杠的圆括号

's///' 命令甚至比 '&' 更好，它允许我们在规则表达式中定义区域，然后可以在替换字符串中引用这些特定区域。作为示例，假设有一个包含以下文本的文件：

代码:

```
foo bar oni eeny meeny miny larry curly moe jimmy the weasel
```

现在假设要编写一个 sed 脚本，该脚本将把 "eeny meeny miny" 替换成 "Victor eeny-meeny Von miny" 等等。要这样做，首先要编写一个由空格分隔并与三个字符串匹配的规则表达式。

代码:

```
'.* .* .*'
```

现在，将在其中每个感兴趣的区域两边插入带反斜杠的圆括号来定义区域：

代码：

```
'\(.*\) \(.*\) \(.*\)'
```

除了要定义三个可在替换字符串中引用的逻辑区域以外，该规则表达式的工作原理将与第一个规则表达式相同。下面是最终脚本：

代码：

```
$ sed -e 's/\(.*\) \(.*\) \(.*\)/Victor \1-\2 Von \3/' myfile.txt
```

如您所见，通过输入 '\x'（其中，x 是从 1 开始的区域号）来引用每个由圆括号定界的区域。输入如下：

代码：

```
Victor foo-bar Von oni Victor eeny-meeny Von miny Victor larry-curly Von moe
Victor jimmy-the Von weasel
```

随着对 sed 越来越熟悉，您可以花最小力气来进行相当强大的文本处理。您可能想如何使用熟悉的脚本语言来处理这种问题 -- 能用一行代码轻易实现这样的解决方案吗？

组合使用

在开始创建更复杂的 sed 脚本时，需要有输入多个命令的能力。有几种方法这样做。首先，可以在命令之间使用分号。例如，以下命令系列使用 '=' 命令和 'p' 命令，'=' 命令告诉 sed 打印行号，'p' 命令明确告诉 sed 打印该行（因为处于 '-n' 模式）。

代码：

```
$ sed -n -e '=;p' myfile.txt
```

无论什么时候指定了两个或更多命令，都按顺序将每个命令应用到文件的每一行。在上例中，首先将 '=' 命令应用到第 1 行，然后应用 'p' 命令。接着，sed 继续处理第 2 行，并重复该过程。虽然分号很方便，但是在某些场合下，它不能正常工作。另一种替换方法是使用两个 -e 选项来指定两个不同的命令：

代码：

```
$ sed -n -e '=' -e 'p' myfile.txt
```

然而，在使用更为复杂的附加和插入命令时，甚至多个 '-e' 选项也不能帮我们的忙。对于复杂的多行脚本，最好的方法是将命令放入一个单独的文件中。然后，用 -f 选项引用该脚本文件：

代码：

```
$ sed -n -f mycommands.sed myfile.txt
```

这种方法虽然可能不太方便，但总是管用。

一个地址的多个命令

有时，可能要指定应用到一个地址的多个命令。这在执行许多 's///' 以变换源文件中的字和语法时特别方便。要对一个地址执行多个命令，可在文件中输入 sed 命令，然后使用 '{ }' 字符将这些命令分组，如下所示：

代码：

```
1,20{    s/[LI]inux/GNU\/Linux/g    s/samba/Samba/g    s/posix/POSIX/g }
```

上例将把三个替换命令应用到第 1 行到第 20 行（包括这两行）。还可以使用规则表达式地址或者二者的组合：

代码：

```
1,/^END/{        s/[LI]inux/GNU\/Linux/g        s/samba/Samba/g        s/posix/POSIX/g    p }
```

该例将把 '{ }' 之间的所有命令应用到从第 1 行开始，到以字母 "END" 开始的行结束（如果在源文件中没发现 "END"，则到文件结束）的所有行。

附加、插入和更改行

既然在单独的文件中编写 sed 脚本，我们可以利用附加、插入和更改行命令。这些命令将在当前行之后插入一行，在当前行之前插入一行，或者替换模式空间中的当前行。它们也可以用来将多行插入到输出。插入行命令用法如下：

代码：

```
i\ This line will be inserted before each line
```

如果不为该命令指定地址，那么它将应用到每一行，并产生如下的输出：

代码：

```
This line will be inserted before each line line 1 here
This line will be inserted before each line line 2 here
This line will be inserted before each line line 3 here
This line will be inserted before each line line 4 here
```

如果要在当前行之前插入多行，可以通过在前一行之后附加一个反斜杠来添加附加行，如下所示：

代码：

```
i\ insert this line\ and this one\ and this one\ and, uh, this one too.
```

附加命令的用法与之类似，但是它将把一行或多行插入到模式空间中的当前行之后。其用法如下：

代码：

```
a\ insert this line after each line.  Thanks! :)
```

另一方面，“更改行”命令将实际替换模式空间中的当前行，其用法如下：

代码：

```
c\ You're history, original line! Muhahaha!
```

因为附加、插入和更改行命令需要在多行输入，所以将把它们输入到一个文本 sed 脚本中，然后通过使用 '-f' 选项告诉 sed 执行它们。使用其它方法将命令传递给 sed 会出现问题。
文本转换
第一个实际脚本将 UNIX 风格的文本转换成 DOS/Windows 格式。您可能知道，基于 DOS/Windows 的文本文件在每一行末尾有一个 CR（回车）和 LF（换行），而 UNIX 文本只有一个换行。有时可能需要将某些 UNIX 文本移至 Windows 系统，该脚本将为您执行必需的格式转换。

```
$ sed -e 's/$/\r/' myunix.txt > mydos.txt
```

在该脚本中，'$' 规则表达式将与行的末尾匹配，而 '\r' 告诉 sed 在其之前插入一个回车。在换行之前插入回车，立即，每一行就以 CR/LF 结束。请注意，仅当使用 GNU sed 3.02.80 或以后的版本时，才会用 CR 替换 '\r'。如果还没有安装 GNU sed 3.02.80，请在我的第一篇 sed 文章中查看如何这样做的说明。

我已记不清有多少次在下载一些示例脚本或 C 代码之后，却发现它是 DOS/Windows 格式。虽然很多程序不在乎 DOS/Windows 格式的 CR/LF 文本文件，但是有几个程序却在乎 -- 最著名的是 bash，只要一遇到回车，它就会出问题。以下 sed 调用将把 DOS/Windows 格式的文本转换成可信赖的 UNIX 格式：

```
$ sed -e 's/.$//' mydos.txt > myunix.txt
```

该脚本的工作原理很简单：替代规则表达式与一行的最末字符匹配，而该字符恰好就是回车。我们用空字符替换它，从而将其从输出中彻底删除。如果使用该脚本并注意到已经删除了输出中每行的最末字符，那么，您就指定了已经是 UNIX 格式的文本文件。也就没必要那样做了！

反转行

下面是另一个方便的小脚本。与大多数 Linux 发行版中包括的 "tac" 命令一样，该脚本将反转文件中行的次序。"tac" 这个名称可能会给人以误导，因为 "tac" 不反转行中字符的位置（左和右），而是反转文件中行的位置（上和下）。用 "tac" 处理以下文件：

```
foo bar oni
```

....将产生以下输出：

```
oni bar foo
```

可以用以下 sed 脚本达到相同目的：

```
$ sed -e '1!G;h;$!d' forward.txt > backward.txt
```

如果登录到恰巧没有 "tac" 命令的 FreeBSD 系统，将发现该 sed 脚本很有用。虽然方便，但最好还是知道该脚本为什么那样做。让我们对它进行讨论。

反转解释

首先，该脚本包含三个由分号隔开的单独 sed 命令：'1!G'、'h' 和 '$!d'。现在，需要好好理解用于第一个和第三个命令的地址。如果第一个命令是 '1G'，则 'G' 命令将只应用第一行。然而，还有一个 '!' 字符 -- 该 '!' 字符忽略该地址，即，'G' 命令将应用到除第一行之外的所有行。'$!d' 命令与之类似。如果命令是 '$d'，则将只把 'd' 命令应用到文件中的最后一行（'$' 地址是指定最后一行的简单方式）。然而，有了 '!' 之后，'$!d' 将把 'd' 命令应用到除最后一行之外的所有行。现在，我们所要理解的是这些命令本身做什么。

当对上面的文本文件执行反转脚本时，首先执行的命令是 'h'。该命令告诉 sed 将模式空间（保存正在处理的当前行的缓冲区）的内容复制到保留空间（临时缓冲区）。然后，执行 'd' 命令，该命令从模式空间中删除 "foo"，以便在对这一行执行完所有命令之后不打印它。

现在，第二行。在将 "bar" 读入模式空间之后，执行 'G' 命令，该命令将保留空间的内容 ("foo\n") 附加到模式空间 ("bar\n")，使模式空间的内容为 "bar\n\foo\n"。'h' 命令将该内容放回保留空间保护起来，然后，'d' 从模式空间删除该行，以便不打印它。

对于最后的 "oni" 行，除了不删除模式空间的内容（由于 'd' 之前的 '$!'）以及将模式空间的内容（三行）打印到标准输出之外，重复同样的步骤。

现在，要用 sed 执行一些强大的数据转换。

sed QIF 魔法
过去几个星期，我一直想买一份 Quicken 来结算我的银行帐户。Quicken 是一个非常好的金融程序，当然会成功地完成这项工作。但是，经过考虑之后，我觉得自己可以轻易编写某个软件来结算我的支票簿。我想，毕竟，我是个软件开发人员！

我开发了一个很好的小型支票簿结算程序（使用 awk），它通过分析包含我的所有交易的文本文件的语法来计算余额。略微调整之后，我将其改进，以便可以象 Quicken 那样跟踪不同的贷款和借款类别。但是，我还要添加一个特性。最近，我将帐户转移到一家有联机 Web 帐户界面的银行。有一天，我注意到，这家银行的 Web 站点允许以 Quicken 的 .QIF 格式下载我的帐户信息。我马上觉得，如果可以将该信息转换成文本格式，那就太棒了。

两种格式的故事
在查看 QIF 格式之前，先看一下我的 checkbook.txt 格式：

28 Aug 2000 food - - Y Supermarket 30.94 25 Aug 2000 watr - 103 Y Check 103 52.86

在我的文件中，所有字段都由一个或多个制表符分开，每个交易占据一行。日期之后的下一个字段列出支出类型（如果是收入项，则为 "-"）。第三个字段列出收入类型（如果是支出项，则为 "-"）。然后，是一个支票号字段（如果为空，则还是 "-"），一个交易完成字段（"Y" 或 "N"），一个注释和一个美元金额字段。现在，让我们看一下 QIF 格式。当用文本查看器查看下载的 QIF 文件时，它看起来如下：

!Type:Bank D08/28/2000 T-8.15 N PCHECKCARD SUPERMARKET ^ D08/28/2000 T-8.25 N PCHECKCARD PUNJAB RESTAURANT ^ D08/28/2000 T-17.17 N PCHECKCARD SUPERMARKET

浏览过文件之后，不难猜出其格式 -- 忽略第一行，其余的格式如下：

D<数据>

T<交易量>
N<支票号>
P<描述>
^ （这是字段分隔符）

开始处理
在处理象这样重要的 sed 项目时，不要气馁 -- sed 允许您将数据逐渐修改成最终形式。在进行当中，可以继续细化 sed 脚本，直到输出与预期的完全一样为止。无需在试第一次时就保证其完全正确。

要开始，首先创建一个名为 "qiftrans.sed" 的文件，然后开始修改数据：

```
1d /^^/d s/[[:cntrl:]]//g
```

第一个 '1d' 命令删除第一行，第二个命令从输出除去那些讨厌的 '^' 字符W詈笠恍谐 ノ募 锌赡芄嬖诘娜魏慰刂谱址 ＜热辉诖 硗铁次募 袷剑 蚁胂 谧型居龅饺魏慰刂谱址姆缦铡5侥壳拔 梗 磺兴忱 O衷冢 蚋没 窘疟局刑砑右恍¬ 耆δ埽?

```
1d /^^/d s/[[:cntrl:]]//g /^D/ {
s/^D\(.*\)/\1\tOUTY\tINNY\t/
s/^01/Jan/ s/^02/Feb/
s/^03/Mar/ s/^04/Apr/
s/^05/May/ s/^06/Jun/
s/^07/Jul/ s/^08/Aug/
s/^09/Sep/ s/^10/Oct/
s/^11/Nov/ s/^12/Dec/
s:^\(.*\)/\(.*\)/\(.*\):\2 \1 \3: }
```

首先，添加一个 '/^D/' 地址，以便 sed 只在遇到 QIF 数据字段的第一个字符 'D' 时才开始处理。当 sed 将这样一行读入其模式空间时，将按顺序执行花括号中的所有命令。

花括号中的第一个命令将把如下行：

```
D08/28/2000
```

变换成：

08/28/2000 OUTY INNY

当然，现在的格式还不完美，但没关系。我们将在进行过程中逐渐细化模式空间的内容。后面 12 行的最后效果是将数据变换成三个字母的格式，最后一行从数据中除去三个斜杠。最后得到这一行：

Aug 28 2000 OUTY INNY

OUTY 和 INNY 字段是占位符，以后将被替换。现在还不能确定它们，因为如果美元金额为负，将把 OUTY 和 INNY 设置成 "misc" 和 "-"，但是，如果美元金额为正，将分别把它们更改成 "-" 和 "inco"。既然还没有读入美元金额，所以，需要暂时使用占位符。

细化
现在进一步细化：

```
1d /^^/d s/[[:cntrl:]]//g /^D/ {
s/^D\(.*\)/\1\tOUTY\tINNY\t/
s/^01/Jan/ s/^02/Feb/
s/^03/Mar/ s/^04/Apr/
s/^05/May/ s/^06/Jun/
s/^07/Jul/ s/^08/Aug/
s/^09/Sep/ s/^10/Oct/
s/^11/Nov/ s/^12/Dec/
s:^\(.*\)/\(.*\)/\(.*\):\2 \1 \3:
N N N
s/\nT\(.*\)\nN\(.*\)\nP\(.*\)/NUM\2NUM\t\tY\t\t\3\tAMT\1AMT/
s/NUMNUM/-/ s/NUM\([0-9]*\)NUM/\1/
s/\([0-9]\),/\1/ }
```

后七行有些复杂，所以将详细讨论它们。首先，连续使用三个 'N' 命令。'N' 命令告诉 sed 将下一行读入输入中，然后将其附加到当前模式空间。这三个 'N' 命令导致将下三行附加到当前模式空间缓冲区，现在这一行看起来如下：

28 Aug 2000 OUTY INNY \nT-8.15\nN\nPCHECKCARD SUPERMARKET

sed 的模式空间变得很难看 -- 需要除去额外的新行，并执行某些附加的格式化。要这样做，将使用替代命令。要匹配的模式为：

'\nT.*\nN.*\nP.*'

这将与后面依次跟有 'T'、零或多个字符、新行、'N'、任何数量的字符、新行、'P'、以及任何数量字符的新行匹配。呀！这个规则表达式将与刚刚附加到模式空间的三行的全部内容匹配。但我们要重新格式化该区域，而不是整个替换它。美元金额、支票号（如果有的话）和描述需要出现在替换字符串中。要这样做，我们用带有反斜杠的圆括号括起那些“感兴趣部分”，以便可以在替换字符串中引用它们（使用 '\1'、'\2\ 和 '\3' 来告诉 sed 将它们插入到何处）。以下是最后的命令：

s/\nT\(.*\)\nN\(.*\)\nP\(.*\)/NUM\2NUM\t\tY\t\t\3\tAMT\1AMT/

该命令将我们的行变换成：

28 Aug 2000 OUTY INNY NUMNUM Y CHECKCARD SUPERMARKET AMT-8.15AMT

虽然该行正变得好一些，但是，有几件事一看就有点...啊...有趣。首先是那个愚蠢的 "NUMNUM" 字符串 -- 其目的何在？如果查看 sed 脚本的后两行，就会发现其目的，后两行将把 "NUMNUM" 替换成 "-"，而把 "NUM"<number>"NUM" 替换成 <number>。如您所见，用愚蠢的标记括起支票号允许我们在该字段为空时方便地插入一个 "-"。

结束尝试
最后一行除去数字后的逗号。它把如 "3,231.00" 这样的美元金额转换成我使用的格式 "3231.00"。现在，让我们看一下最终脚本：

最终的“QIF到文本”脚本 1d /^^/d s/[[:cntrl:]]//g /^D/
{ s/^D\(.*\)/\1\tOUTY\tINNY\t/
s/^01/Jan/ s/^02/Feb/ s/^03/Mar/ s/^04/Apr/ s/^05/May/
s/^06/Jun/ s/^07/Jul/ s/^08/Aug/ s/^09/Sep/ s/^10/Oct/
s/^11/Nov/ s/^12/Dec/ s:^\(.*\)/\(.*\)/\(.*\):\2 \1 \3:
N N N s/\nT\(.*\)\nN\(.*\)\nP\(.*\)/NUM\2NUM\t\tY\t\t\3\tAMT\1AMT/
s/NUMNUM/-/ s/NUM\([0-9]*\)NUM/\1/ s/\([0-9]\),/\1/
/AMT-[0-9]*.[0-9]*AMT/b fixnegs
s/AMT\(.*\)AMT/\1/ s/OUTY/-/ s/INNY/inco/
b done :fixnegs s/AMT-\(.*\)AMT/\1/ s/OUTY/misc/
s/INNY/-/ :done }

附加的十一行使用替代和一些分支功能来美化输出。首先看一下这行：

/AMT-[0-9]*.[0-9]*AMT/b fixnegs

该行包含一个格式为 "/regexp/b label" 的分支命令。如果模式空间与规则表达式匹配，sed 将分支到 fixnegs 标号。您应该可以轻易找到该标号，它在代码中为 ":fixnegs"。如果规则表达式不匹配，则以常规方式继续处理下一个命令。

既然您理解该命令本身的工作原理，让我们看一下分支。如果看一下分支规则表达式，将看到它与后面依次跟有 '-'、任意数量的数字、一个 '.'、任意数量的数字和 'AMT' 的字符串 'AMT' 匹配。就象我确信您已猜到一样，该规则表达式专门处理负的美元金额。在这之前，用 'ATM' 括起美元金额，以便以后可以轻易找到它。因为规则表达式只与以 '-' 开始的美元金额匹配，所以，该分支只在恰巧处理借款时才发生。如果正处理贷款，应该将 OUTY 设置成 'misc'，将 INNY 设置成 '-'，并且应该除去贷款数量前面的负号。如果跟踪代码的流程，将看到实际情况正是这样。如果不执行分支，则用 '-' 替换 OUTY，用 'inco' 替换 INNY。完成了！现在输出行是完美的：

28 Aug 2000 misc - - Y CHECKCARD SUPERMARKET -8.15
看完，对什么时候使用什么选项还是有些糊涂

man sed

代码:

```
-n, --quiet, --silent

              suppress automatic printing of pattern space

前面举过例子：
[sam@Linux_chenwy sam]$ sed -n '2p' quote.txt
It was an evening of splendid music and company.
如果没有-n,就把其它的都打印出来了
```

代码:

```
-e script, --expression=script

              add the script to the commands to be executed
```

```
呵，跟多个表达式啊
[sam@Linux_chenwy sam]$ sed -n -e '/music/p' -e '/music/='  quote.txt
It was an evening of splendid music and company.
2
```

代码：

```
-f script-file, --file=script-file

              add the contents of script-file to the commands to be executed
脚本文件
```

代码：

```
      -i[suffix], --in-place[=suffix]

              edit files in place (makes backup if extension supplied)
寂寞烈火说过：是直接更更改原文件，不过最好用重定向
```

-l N, --line-length=N

specify the desired line-wrap length for the `l' command

-r, --regexp-extended

use extended regular expressions in the script.

-s, --separate

consider files as separate rather than as a single continuous
long stream.

-u, --unbuffered

load minimal amounts of data from the input files and flush the
output buffers more often

--help display this help and exit

-V, --version
output version information and exit

If no -e, --expression, -f, or --file option is given, then the first non-option argument is taken as the sed script to interpret. All remaining arguments are names of input files; if no input files are specified, then the standard input is read.

E-mail bug reports to: bonzini@gnu.org . Be sure to include the word ``sed'' somewhere in the ``Subject:'' field.
不定期补充一下在论坛看到的sed的实例：

例一：sed分域
http://bbs.chinaunix.net/forum/24/20041207/461745.html

1C2
1C3
1C31
1C32
1C4
2C3
2C4
1D1
1D10
1D12
1D2
1D3
1D31
1RC2
1RC20
1RC21
1RC3
1RC31
1WR1
1WR2
1WR20
1WR21
1WR23
…

排序后
[file2.txt]
1C2
1C3

1C4
1C31
1C32
2C3
2C4
1D1
1D2
1D3
1D10
1D12
1D31
1RC2
1RC3
1RC20
1RC21
1RC31
1WR1
1WR2
1WR20
1WR21
1WR23
…

规律：将每行分成三部分： “数字 1” “字符串” “数字 2”
第一、三字段按numberic顺序排序，中间部分按字母排序

第二个字段为主关键字，第三个字段为次关键字， 然后是第一个字段

代码:

```
$ cat file |sed 's/^\([0-9]*\)\([A-Z]*\)\([0-9]*\)/\1 \2 \3 /g' |sort +1 -2 +2n
+0  -1
```

以空格划分域再排序

例二：处理日期
http://bbs.chinaunix.net/forum/24/20041207/462196.html

123456 345678 2005.05.06 123456
123456 234567 2003.5.6 234567
345555 987644 2003.4.23 543333
555555 999999 2004.11.5 999999

要将第四列数据变成正常的年月日，将 2003.5.6 变成 2003.05.0;

2003.4.23 变成 2003.04.23； 2004.11.5 变成 2004.11.05

代码:

```
cat file | sed -e :a -e 's/-\([0-9]\)\([- ]\)/-0\1\2/;ta'
```

or

代码:

```
cat file | sed 's/-\([0-9]\)-/-0\1-/; s/-\([0-9]\) /-0\1 /'
```

基础 11：文件分类、合并和分割 (sort,uniq,join,cut,paste,split)

引用:

- 实用的分类（s o r t）操作。
- uniq。
- join。
- cut。
- paste。
- split。

sort 用法

s o r t 命令选项很长，下面仅介绍各种选项。

选项

s o r t 命令的一般格式为：

代码:

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

下面简要介绍一下 s o r t 的参数：

引用:

-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 s o r t 结果的输出文件名。
```

其他选项有：

引用：

```
-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 t a b 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。
```

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 r e s u l t s . o u t：

代码：

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

启动方式
缺省情况下， s o r t 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 s o r t 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 s o r t 将整个行排序，指定域号的情况例外。

下面是文件 v i d e o . t x t 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：

代码：

```
[sam@chenwy sam]$ cat 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
```

sort 对域的参照方式

关于 s o r t 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。s o r t 也可以使用整行作为分类依据。

文件是否已分类

怎样分辨文件是否已分类？如果只有 3 0 行，看看就知道了，但如果是 4 0 0 行呢，使用 s o r t - c 通知 s o r t 文件是否按某种顺序分类。

代码:

```
[sam@Linux_chenwy sam]$ sort -c video.txt
sort: video.txt:2: disorder: Alien:HK:119:1982
```

结果显示未分类，

现在将之分类，再试一次：

代码:

```
[sam@Linux_chenwy sam]$ sort -t:  video.txt >video2.txt
[sam@Linux_chenwy sam]$ sort -c video2.txt
[sam@Linux_chenwy sam]$
```

返回提示符表明已分类。然而如果测试成功，返回一个信息行会更好。

基本 sort

最基本的 s o r t 方式为 sort filename，按第一域进行分类（分类键 0）。实际上读文件时 s o r t 操作将行中各域进行比较，这里返回基于第一域 s o r t 的结果

代码:

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

sort 分类求逆

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

代码:

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

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 t 1，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。

代码:

```
[sam@Linux_chenwy sam]$ 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
```

前几个第二域都是 HK，第三域：119,192,301,489,532,63，按第一个数字分了，因此必须指定多域及数值域

数值域分类

依此类推，要按第三分类键分类，使用 t 3。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:

代码:

```
[sam@Linux_chenwy sam]$ 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
```

如果不指定 n,如下

代码:

```
[sam@Linux_chenwy sam]$ 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
```

ｏｒｔ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。

代码:

```
[sam@Linux_chenwy sam]$ sort -t: +2n video.txt
The Hill:KL:63:2972
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
```

数值域倒序：

代码:

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

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：

代码:

```
[sam@Linux_chenwy sam]$ echo "Aliens:HK:532:4892" >> video.txt
[sam@Linux_chenwy sam]$ cat 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
Aliens:HK:532:4892
```

使用- u 选项去除重复行，不必加其他选项， s o r t 会自动处理。

代码:

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

使用 k 的其他 sort 方法

s o r t 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。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。
使用- k 4，按年租金分类的次序。

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k4 video.txt
A alien:HK:119:1982
Alien:HK:119:1982
Boys in Company C:HK:192:2192
A the Hill:KL:63:2972
The Hill:KL:63:2972
Toy Story:HK:239:3972
Star Wars:HK:301:4102
Aliens:HK:532:4892
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
```

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：

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k4 -k1  video.txt
AAlien:HK:119:1982
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
```

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k4 -k1 -r  video.txt
A Few Good Men:KL:445:5851
Aliens:HK:532:4892
Star Wars:HK:301:4102
Toy Story:HK:239:3972
The Hill:KL:63:2972
Boys in Company C:HK:192:2192
Alien:HK:119:1982
AAlien:HK:119:1982
```

这里-r 是对第四域反排序？

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k1  video.txt
AAlien:HK:119:1982
A Few Good Men:KL:445:5851
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
Star Wars:HK:301:4102
The Hill:KL:63:2972
Toy Story:HK:239:3972
```

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k1 -k4  video.txt
AAlien:HK:119:1982
A Few Good Men:KL:445:5851
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
Star Wars:HK:301:4102
The Hill:KL:63:2972
Toy Story:HK:239:3972
```

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k1 -k4 -r  video.txt
Toy Story:HK:239:3972
The Hill:KL:63:2972
Star Wars:HK:301:4102
Boys in Company C:HK:192:2192
Aliens:HK:532:4892
Alien:HK:119:1982
A Few Good Men:KL:445:5851
AAlien:HK:119:1982
```

对第一域进行反排序?

换成第 3 域

代码:

```
[sam@Linux_chenwy sam]$ sort -t:  +2nr -k1 -r  video.txt
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
Star Wars:HK:301:4102
Toy Story:HK:239:3972
Boys in Company C:HK:192:2192
Alien:HK:119:1982
AAlien:HK:119:1982
The Hill:KL:63:2972
```

对第三域进行倒序，再对第一域排序，最后把第一域倒序?

代码:

```
[sam@Linux_chenwy sam]$ sort -t:  +2nr -k1  video.txt
```

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

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 s o r t 命令：

代码:

```
[sam@Linux_chenwy sam]$ sort +0 -2 +3 video.txt
```

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。

基本 sort

最基本的 s o r t 方式为 sort filename，按第一域进行分类（分类键 0）。实际上读文件时 s o r t 操作将行中各域进行比较，这里返回基于第一域 s o r t 的结果

代码:

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

sort 分类求逆

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

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -r video.txt
Toy Story:HK:239:3972
The Hill:KL:63:2972
Star Wars:HK:301:4102
Boys in Company C:HK:192:2192
```

```
Aliens:HK:532:4892
Alien:HK:119:1982
A Few Good Men:KL:445:5851
```

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 t 1，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。

代码:

```
[sam@Linux_chenwy sam]$ 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
```

前几个第二域都是 HK，第三域：119,192,301,489,532,63，按第一个数字分了，因此必须指定多域及数值域

数值域分类

依此类推，要按第三分类键分类，使用 t 3。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：

代码:

```
[sam@Linux_chenwy sam]$ 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
```

如果不指定 n,如下

代码:

```
[sam@Linux_chenwy sam]$ 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
```

o r t 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。

代码:

```
[sam@Linux_chenwy sam]$ sort -t: +2n video.txt
The Hill:KL:63:2972
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
```

数值域倒序：

代码:

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

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 A l i e n 有相同的两行。带重复行的文件如下，其中 A l i e n 插入了两次：

代码:

```
[sam@Linux_chenwy sam]$ echo "Aliens:HK:532:4892" >> video.txt
[sam@Linux_chenwy sam]$ cat 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
Aliens:HK:532:4892
```

使用- u 选项去除重复行，不必加其他选项， s o r t 会自动处理。

代码:

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

使用 k 的其他 sort 方法

s o r t 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。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。

使用- k 4，按年租金分类的次序。

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k4 video.txt
A alien:HK:119:1982
Alien:HK:119:1982
Boys in Company C:HK:192:2192
A the Hill:KL:63:2972
The Hill:KL:63:2972
Toy Story:HK:239:3972
Star Wars:HK:301:4102
Aliens:HK:532:4892
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
```

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：

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k4 -k1  video.txt
AAlien:HK:119:1982
```

```
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
```

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k4 -k1 -r  video.txt
A Few Good Men:KL:445:5851
Aliens:HK:532:4892
Star Wars:HK:301:4102
Toy Story:HK:239:3972
The Hill:KL:63:2972
Boys in Company C:HK:192:2192
Alien:HK:119:1982
AAlien:HK:119:1982
```

这里-r 是对第四域反排序?

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k1  video.txt
AAlien:HK:119:1982
A Few Good Men:KL:445:5851
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
Star Wars:HK:301:4102
The Hill:KL:63:2972
Toy Story:HK:239:3972
```

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k1 -k4  video.txt
AAlien:HK:119:1982
A Few Good Men:KL:445:5851
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
```

```
Star Wars:HK:301:4102
The Hill:KL:63:2972
Toy Story:HK:239:3972
```

代码:

```
[sam@Linux_chenwy sam]$ sort -t: -k1 -k4 -r  video.txt
Toy Story:HK:239:3972
The Hill:KL:63:2972
Star Wars:HK:301:4102
Boys in Company C:HK:192:2192
Aliens:HK:532:4892
Alien:HK:119:1982
A Few Good Men:KL:445:5851
AAlien:HK:119:1982
```

对第一域进行反排序？

换成第 3 域

代码:

```
[sam@Linux_chenwy sam]$ sort -t:  +2nr -k1 -r  video.txt
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
Star Wars:HK:301:4102
Toy Story:HK:239:3972
Boys in Company C:HK:192:2192
Alien:HK:119:1982
AAlien:HK:119:1982
The Hill:KL:63:2972
```

对第三域进行倒序，再对第一域排序，最后把第一域倒序？

代码:

```
[sam@Linux_chenwy sam]$ sort -t:  +2nr -k1  video.txt
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
Star Wars:HK:301:4102
Toy Story:HK:239:3972
Boys in Company C:HK:192:2192
AAlien:HK:119:1982
Alien:HK:119:1982
The Hill:KL:63:2972
```

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 s o r t 命令：

代码:

```
[sam@Linux_chenwy sam]$ sort +0 -2 +3 video.txt
```

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

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：

代码:

```
sort +field_number.characters_in
```

意即从 f i e l d _ n u m b e r 开始分类，但是要在此域的第 c h a r a c t e r s _ i n 个字符开始。

如：

代码:

```
[sam@chenwy sam]$ cat video.txt
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
The Hill:KL223:63:2972
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

代码:

```
[sam@chenwy sam]$ sort -t: +1.2 video.txt
Aliens:HK11:532:4892
The Hill:KL223:63:2972
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
```

比较一下加 n,呵呵，其实区码并不需要加 n

代码:

```
[sam@chenwy sam]$ sort -t: +1.2n video.txt
Aliens:HK11:532:4892
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
```

转一贴子：sort

http://www.gnu.org/manual/

Jacek Artymiak（jacek@artymiak.com）
自由作家和顾问
2003 年 3 月 6 日

通过使用 sort 和 tsort，而不是采取使用 Perl 或 Awk 的较复杂的解决方案，可以节省时间，同时还能避免令人头疼的问题。Jacek Artymiak 将向您说明如何做到这一点。
尽管可以用 Perl 或 Awk 编写高级排序应用程序，但并非总是有此必要，而且这样的工作也常常令人感到头疼。使用 sort 命令，您同样可以实现您所需的大多数功能，而且更容易，它可以对多个文件中的行进行排序、合并文件甚至可以查看是否有必要对它们进行排序。您可以指定排序键（用于比较的行中的一部分），也可不指定，后一种情况下 sort 就比较所有行。

因此，如果您想对密码文件进行排序，就可以使用下列命令（请注意，您不能将输出直接发送到输入文件，因为这会破坏输入文件。这就是为何您需要将它发送到临时文件中，然后将该文件重命名为 /etc/passwd 的原因，如下所示）。

1、清单 1. 简单排序

代码:

```
$ su -
# sort /etc/passwd > /etc/passwd-new
# mv /etc/passwd-new /etc/passwd
```

2、有关 sort 和 tsort 的更多信息
通过打开有关排序操作的 GNU 手册页来学习手册页中的内容，或者通过在命令行中输入 man sort 或 man tsort 在新的终端窗口的手册页或信息页中查看这些选项。
如果您想倒转排序的次序，则应当使用 -r 选项。您还可以用 -u 选项来禁止打印相同的行。

3、sort 的一个非常实用的特性是它用字段键进行排序的能力。字段是一个文本字符串，通过某个字符与其它字段分隔开。例如，/etc/passwd 中的字段是用冒号（:）分隔的。因此，如

果愿意的话，您可以按照用户标识、组标识、注释字段、主目录或 shell 对 /etc/passwd 进行排序。要做到这一点，请使用 -t 选项，其后跟着用作分隔符的字符，接着是用作排序键的字段编号，再跟作为键的最后一个字段的编号；
例如，

代码:

```
sort -t : -k 5,5 /etc/passwd
```

按照注释字段对密码文件进行排序，该字段中存储了完整的用户名（如“John Smith”）。

而

代码:

```
sort -t : -k 3,4 /etc/passwd
```

同时使用用户标识和组标识对同一个文件进行排序。如果您省略了第二个数字，那么 sort 会假定键是从给定的字段开始，一直到每一行的末尾。动手试一试，并观察其中的区别（当数字排序看上去有错时，请添加 -g 选项）。

还要注意的是，空白过渡是缺省的分隔符，因此，如果字段已经用空白字符分隔了，那么您可以省略分隔符，只使用 -t（另注：字段的编号是从 1 开始的）。

5、为了更好地进行控制，您可以使用键和偏移量。偏移量是用点与键相分隔的，比如在 -k 1.3,5.7 中，表示排序键应当从第 1 个字段的第 3 个字符开始，到第 5 个字段的第 7 个字符结束（偏移量也是从 1 开始编号的）。何时会用得着偏移量呢？嗯，我时常用它来对 Apache 日志进行排序；键和偏移量表示法让我跳过了日期字段。

6、另一个要关注的选项是 -b，它告知 sort 忽略空白字符（空格、跳格等等）并将行中的第一个非空白字符当做是排序键的开始。还有，如果您使用该选项，那么将从第一个非空白字符开始计算偏移量（当字段分隔符不是空白字符，且字段可能包含以空白字符开头的字符串时，这非常有用）。

引用:

> 可以用下面这些选项来进一步修改排序算法：
>
> -d（只将字母、数字和空白用作排序键）、
>
> -f（关闭大小写区分，认为小写和大写字符是一样的）、
>
> -i（忽略非打印的 ASCII 字符）、
>
> -M（使用三个字母的月份名称缩写：JAN、FEB、MAR … 来对行进行排序）和
>
> -n（只用数字、- 和逗号或另外一个千位分隔符对行进行排序）。
>
> 这些选项以及 -b 和 -r 选项可以用作键编号的一部分，
>
> 在这种情况下，它们只适用于该键而非全局，其作用就跟在键定义外使用它时一样。

以键编号的用法为例，请考虑：

代码：

```
sort -t: -k 4g,4 -k 3gr,3 /etc/passwd
```

这条命令将按照组标识对 passwd 文件进行排序，而在组内按照用户标识进行逆向排序。

7、如果您所使用的键不能用来确定哪一行是在先，那么它也可以解决这类平局问题。增加一个解决平局问题的提示，请添加另一个 -k 选项，让它跟在字段和（可选的）偏移量后面，使用与前面用于定义键相同的表示法；
例如，

代码：

```
sort -k 3.4,4.5 -k 7.3,9.4 /etc/passwd
```

对行进行排序时，使用从第 3 个键的第 4 个字符开始到第 4 个键的第 5 个字符结束的键，然后再采用从第 7 个字段的第 3 个字符到第 9 个字段的第 4 个字符结束的键来解决上述难题。

8、最后一组选项处理输入、输出和临时文件。例如，-c 选项，当它用于 sort -c < file 中时，它检查输入文件是否已进行了排序（您也可以使用其它选项），如果已进行了排序，则报告一个错误。这样，在处理可能需要花很长时间进行排序的大型文件之前，可以很方便地对其进行检查。当您将 -u 选项和 -c 选项一起使用时，会被解释为一个请求：检查输入文件中不存在两个相同的行。

9、当您处理大型文件时还有一个很重要的 -T 选项，它用于为临时文件（这些临时文件在 sort 完成工作之后会被除去）指定其它目录，而不是缺省的 /tmp 目录。

10、您可以使用 sort 来同时处理多个文件，这样做的方式基本上有两种：首先可以使用 cat 来并置它们，如下所示：

代码：

```
cat file1 file2 file3 | sort > outfile
```

或者，可以使用下面这个命令：

代码：

```
sort -m file1 file2 file3 > outfile
```

第二种情况有个条件：在将所有输入文件一起进行 sort -m 之前，每个文件都必须经过排序。这看起来似乎是个不必要的负担，但事实上这加快了工作速度并节约了宝贵的系统资源。对了，别忘了 -m 选项。在这里您可以使用 -u 选项来禁止打印相同的行。

11、如果需要某种更深奥的排序方法，您可能要查看 tsort 命令，该命令对文件执行拓扑排序。拓扑排序和标准 sort 之间的差别如清单 2 所示（您可以从参考资料下载 happybirthday.txt）。

清单 2. 拓扑排序和标准排序之间的差别

代码:

```
$ cat happybirthday.txt

Happy Birthday to You!

Happy Birthday to You!

Happy Birthday Dear Tux!

Happy Birthday to You!
```

代码:

```
$ sort happybirthday.txt

Happy Birthday Dear Tux!

Happy Birthday to You!

Happy Birthday to You!

Happy Birthday to You!
```

代码:

```
$ tsort happybirthday.txt

Dear

Happy

to
```

```
Tux!

Birthday

You!
```

当然，对于 tsort 的使用来说，这并非一个非常有用的演示，只是举例说明了这两个命令输出的不同。

tsort 通常用于解决一种逻辑问题，即必须通过观察到的部分次序预测出整个次序；例如（来自 tsort 信息页中）：

代码：

```
tsort <<EOF a b c d e f b c d e EOF
```

会产生这样的输出

代码：

```
a
b
c
d
e
f
```

tsort 的没试过，最后一个我不行耶，不知道咋回事

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- k 4 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：

代码：

```
[sam@chenwy sam]$ sort -t: -k4r video.txt
A Few Good Men:KL87:445:5851
Aliens:HK11:532:4892
Star Wars:HK38:301:4102
Toy Story:HK65:239:3972
```

```
The Hill:KL223:63:2972
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
```

代码：

```
[sam@chenwy sam]$ sort -t: -k4r video.txt | head -1
A Few Good Men:KL87:445:5851
```

代码：

```
[sam@chenwy sam]$ sort -t: -k4r video.txt | head -2
A Few Good Men:KL87:445:5851
Aliens:HK11:532:4892
```

要查阅最低年租金，使用 t a i l 命令与 h e a d 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 s o r t 命令和 t a i l 命令显示最低年租金：

代码：

```
[sam@chenwy sam]$ sort -t: -k4r video.txt | tail -1
Alien:HK57:119:1982
```

代码：

```
[sam@chenwy sam]$ sort -t: -k4r video.txt | tail -2
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
```

可以使用 h e a d 或 t a i l 查阅任何大的文本文件， h e a d 用来查阅文件头，基本格式如下：

代码：

```
head [how_many_lines_to_display] file_name
```

Ta i l 用来查阅文件尾，基本格式为：

代码：

```
tail [how_many_lines_to_display] file_name
```

如果使用 h e a d 或 t a i l 时想省略显示行数，缺省时显示 1 0 行。

要查阅文件前 2 0 行：

代码:

```
[sam@chenwy sam]$ head -20 passwd
```

要查阅文件后 10 行：

代码:

```
[sam@chenwy sam]$ tail -10 passwd
```

awk 使用 sort 输出结果

对数据分类时，对 s o r t 结果加一点附加信息很有必要，对其他用户尤其如此。使用 a w k 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 s o r t 结果管道输出到 a w k，不要忘了用冒号作域分隔符，显示提示信息和实际数据。

代码:

```
[sam@chenwy sam]$ sort -t: -r -k4 video.txt |tail -1 | awk -F: '{print "Worst
rental", $1, "has been rented" $3}'
Worst rental Alien has been rented119
```

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：

代码:

```
[sam@chenwy sam]$ cat 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 v i d e o . s o r t，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。

代码:

```
[sam@chenwy sam]$ sort -t: -m +0 video2.txt video.txt
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
Crimson Tide:134:2031
Die Hard:152:2981
The Hill:KL223:63:2972
```

```
Aliens:HK11:532:4892
Star Wars:HK38:301:4102
A Few Good Men:KL87:445:5851
Toy Story:HK65:239:3972
```

系统 sort

s o r t 可以用来对/ e t c / p a s s w d 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 a w k，a w k 打印第一域。

代码:

```
[sam@chenwy sam]$ cat passwd | sort -t: +0 | awk -F: '{print $1}'
adm
apache
bin
chenwy
daemon
desktop
.......
```

s o r t 还可以用于 d f 命令，以递减顺序打印使用列。下面是一般 d f 输出。

代码:

```
[sam@chenwy sam]$ df
文件系统             1K-块       已用     可用 已用% 挂载点
/dev/sda2            5162828   2289460   2611108  47% /
/dev/sda1             497829     13538    458589   3% /boot
none                   99352         0     99352   0% /dev/shm
```

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。

代码:

```
[sam@chenwy sam]$ df | sort -b -r +4
文件系统             1K-块       已用     可用 已用% 挂载点
/dev/sda2            5162828   2289460   2611108  47% /
/dev/sda1             497829     13538    458589   3% /boot
none                   99352         0     99352   0% /dev/shm
```

在一个文本文件中存入所有 I P 地址的拷贝，这样查看本机 I P 地址更容易一些。有时如果管理

员权限下，就需要将此文件分类。将 I P 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 I P 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：

代码:

```
[sam@chenwy sam]$ vi iplist
[sam@chenwy sam]$ cat 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：

代码:

```
[sam@chenwy sam]$ 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
```

sort 结束

uniq 用法

u n i q 用来从一个文本文件中去除或禁止重复行。一般 u n i q 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 u n i q 有点像 s o r t 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。s o r t 的唯一性选项去除所有重复行，而 u n i q 命令并不这样做。重复行是什么？在 u n i q 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：

代码:

```
[sam@chenwy sam]$ cat myfile.txt
May Day
May Day
May Day
Going DOwn
May Day
May Day.
May Day
```

u n i q 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。u n i q 将保留这一行。
命令一般格式：

代码：

```
$uniq -u d c -f input-file out-file
```

引用：

> 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。
>
> 一些系统不识别- f 选项，这时替代使用- n。

创建文件 m y f i l e . t x t，在此文件上运行 u n i q 命令。

代码：

```
[sam@chenwy sam]$ uniq myfile.txt
May Day
Going DOwn
May Day
May Day.
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。

连续重复出现
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

代码：

```
[sam@chenwy sam]$ uniq -c myfile.txt
      3 May Day
      1 Going DOwn
      1 May Day
      1 May Day.
      1 May Day
```

1．不唯一
使用- d 显示重复出现的不唯一行：

代码:

```
[sam@chenwy sam]$ uniq -d myfile.txt
May Day
```

代码:

```
[sam@chenwy sam]$ uniq -u myfile.txt
Going DOwn
May Day
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。

代码:

```
[sam@chenwy sam]$ cat parts.txt
AK123 OPP Y13
DK122 OPP Y24
EK999 OPP M2
```

代码:

```
[sam@chenwy sam]$ cat parts.txt
AK123 33 46 6u OPP ty yu
DK122 5h 67 y8 OPP ty yu
EK999 56 56 78 IIY ty yu
```

运行 u n i q，将返回所有行。因为这个文件每一行都不同。

代码:

```
[sam@chenwy sam]$ cat parts.txt
1 AK123 33 46 6u OPP ty yu
1 DK122 5h 67 y8 OPP ty yu
1 EK999 56 56 78 IIY ty yu
```

如果指定测试在第 4 域后，结果就会不同。u n i q 会比较三个相同的 O PP，因此将返回一行。

代码:

```
[sam@chenwy sam]$ uniq -f4 -c parts.txt
      2 AK123 33 46 6u OPP ty yu
      1 EK999 56 56 78 IIY ty yu
```

指定第 5 域，即从第 6 域开始比较：

代码：

```
[sam@chenwy sam]$ uniq -f5 -c parts.txt
      3 AK123 33 46 6u OPP ty yu
```

如果‘- f’返回错误，替代-n 使用：
转：uniq:

进行排序之后，您会发现有些行是重复的。有时候该重复信息是不需要的，可以将它除去以节省磁盘空间。不必对文本行进行排序，但是您应当记住 uniq 在读取行时会对它们进行比较并将只除去两个或更多的连续行。下面的示例说明了它实际上是如何工作的：

清单 1. 用 uniq 除去重复行

代码：

```
$ cat happybirthday.txt

Happy Birthday to You!

Happy Birthday to You!

Happy Birthday Dear Tux!

Happy Birthday to You!
```

代码：

```
$ sort happybirthday.txt

Happy Birthday Dear Tux!

Happy Birthday to You!

Happy Birthday to You!

Happy Birthday to You!
```

代码：

```
$ sort happybirthday.txt | uniq

Happy Birthday Dear Tux!

Happy Birthday to You!
```

警告：请不要使用 uniq 或任何其它工具从包含财务或其它重要数据的文件中除去重复行。在这种情况下，重复行几乎总是表示同一金额的另一个交易，将它除去会给会计部造成许多困难。千万别这么干！

有关 uniq 的更多信息
本系列文章介绍了文本实用程序，它对在手册页和信息页找到的信息作了补充。如果您打开新的终端窗口并输入 man uniq 或 info uniq，或者打开新的浏览器窗口并查看位于 gnu.org 的 uniq 手册页，那么就可以了解更多的相关信息。

如果您希望您的工作轻松点，比如只显示唯一的或重复的行，那么该怎么办呢？您可以用 -u（唯一）和 -d（重复）选项来做到这一点，例如：

清单 2. 使用 -u 和 -d 选项

代码：

```
$ sort happybirthday.txt | uniq -u

Happy Birthday Dear Tux!
```

代码：

```
$ sort happybirthday.txt | uniq -d

Happy Birthday to You!
```

您还可以用 -c 选项从 uniq 中获取一些统计信息：

清单 3. 使用 -c 选项

代码：

```
$ sort happybirthday.txt | uniq -uc

      1 Happy Birthday Dear Tux!
```

代码：

```
$ sort happybirthday.txt | uniq -dc

      3 Happy Birthday to You!
```

就算 uniq 对完整的行进行比较，它仍然会很有用，但是那并非该命令的全部功能。特别方便的是：使用 -f 选项，后面跟着要跳过的字段数，它能够跳过给定数目的字段。当您查看系统日志时这非常有用。通常，某些项要被复制许多次，这使得查看日志很难。使用简单的 uniq 无法完成任务，因为每一项都以不同的时间戳记开头。但是如果您告诉它跳过所有的时间字段，您的日志一下子就会变得更加便于管理。试一试 uniq -f 3 /var/log/messages，亲眼看看。

还有另一个选项 -s，它的功能就像 -f 一样，但是跳过给定数目的字符。您可以一起使用 -f 和 -s。uniq 先跳过字段，再跳过字符。如果您只想使用一些预先设置的字符进行比较，那么该怎么办呢？试试看 -w 选项。

join 用法

j o i n 用来将来自两个分类文本文件的行连在一起。

下面讲述 j o i n 工作方式。这里有两个文件 f i l e 1 和 f i l e 2，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， j o i n 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 t a b 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 j o i n 时文件域要少于 2 0，为公平起见，如果域大于 2 0，应使用 D B M S 系统。为有效使用 j o i n，需分别将输入文件分类。

其一般格式为：

代码：

```
join [options] input-file1 input-file2
```

引用：

选项：

an n 为一数字，用于连接时从文件 n 中显示不匹配行。例如， - a 1 显示第一个文件的不匹配行，- a 2 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 t a b 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 n a m e．t x t，另一个是名字和城镇，
为 t o w n．t x t。

代码:

```
[sam@chenwy sam]$ cat 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
```

代码:

```
[sam@chenwy sam]$ cat town.txt
M.Golls Norwich NRD
P.Willey Galashiels GDD
T.Norms Brandon BSL
K.Fletch Mildenhall MAF
K.Firt Mitryl Mdt
```

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．G o l l s 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， j o i n 将假定这是连接域：

代码:

```
[sam@chenwy sam]$ 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
```

缺省 j o i n 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。

代码:

```
[sam@chenwy sam]$ 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
K.Firt Mitryl Mdt
```

代码:

```
[sam@chenwy sam]$ join -a1  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
```

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：

代码:

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

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：

代码:

```
[sam@chenwy sam]$ cat pers
P.Jones Office Runner ID897
S.Round UNIX admin ID666
```

```
L.Clip Personl Chief ID982
```

代码：

```
[sam@chenwy sam]$ cat pers2
Dept2C ID897 6 years
Dept3S ID666 2 years
Dept5Z ID982 1 year
```

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

代码：

```
[sam@chenwy sam]$ 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 j o i n 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 j o i n 应该访问域 5，这样将不返回任何结果。如果是这样，用 a w k 检查域号。例如，键入$ awk '{print $4}'文件名，观察其是否匹配假想域。

——————————————

cut 用法

c u t 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。

c u t 一般格式为：

代码：

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

引用：

> 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 t a b 键不同的域分隔符。
>
> - c 用来指定剪切范围，如下所示：
>
> - c 1，5-7 剪切第 1 个字符，然后是第 5 到第 7 个字符。

```
-c1-50 剪切前 5 0 个字符。
-f 格式与- c 相同。
-f 1，5 剪切第 1 域，第 5 域。
- f 1，10-12 剪切第 1 域，第 1 0 域到第 1 2 域。
```

现在从' p e r s '文件中剪切文本。

代码:

```
[sam@chenwy sam]$ cat pers
P.Jones Office Runner ID897
S.Round UNIX admin ID666
L.Clip Personl Chief ID982
```

使用域分隔符

文件中使用空格“ ”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- f 3。要抽取 I D 域。可使用命令如下：

代码:

```
[sam@chenwy sam]$ cut -d" " -f3 pers
Runner
admin
Chief
```

剪切指定域

c u t 命令中剪切各域需用逗号分隔，如剪切域 1 和 3，即名字和 I D 号，可以使用：

代码:

```
[sam@chenwy sam]$ cut -d" " -f1,3 pers
P.Jones Runner
S.Round admin
L.Clip Chief
```

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 c u t 命令：

代码:

```
[sam@chenwy sam]$ cat pers2
Dept2C ID897 6 years
```

```
Dept3S ID666 2 years
Dept5Z ID982 1 year
```

代码:

```
[sam@chenwy sam]$ cut -c4-8,11-12 pers2
t2C I97
t3S I66
t5Z I82
```

要剪切谁正在使用系统的用户信息，方法如下：

代码:

```
[sam@chenwy sam]$ who -u|cut -c1-8
root
root
```

paste 用法

c u t 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 p a s t e 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。p a s t e 将按行将不同文件行信息放在一行。缺省情况下， p a s t e 连接时，用空格或 t a b 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。

p a s t e 格式为；

代码:

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

引用:

选项含义如下：

-d 指定不同于空格或 t a b 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：

代码:

```
[sam@chenwy sam]$ cut -d" " -f 2 pers2 >pas1
[sam@chenwy sam]$ cat pas1
ID897
ID666
```

```
ID982
```

代码：

```
[sam@chenwy sam]$ cut -d" " -f1 pers >pas2
[sam@chenwy sam]$ cat pas2
P.Jones
S.Round
L.Clip
```

基本 p a s t e 命令将之粘贴成两列：

代码：

```
[sam@chenwy sam]$ paste pas1 pas2
ID897   P.Jones
ID666   S.Round
ID982   L.Clip
```

指定列

通过交换文件名即可指定哪一列先粘：

代码：

```
[sam@chenwy sam]$ paste pas2 pas1
P.Jones ID897
S.Round ID666
L.Clip  ID982
```

使用不同的域分隔符

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

代码：

```
[sam@chenwy sam]$ paste -d: pas2 pas1
P.Jones:ID897
S.Round:ID666
L.Clip:ID982
```

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 I D 号。

代码：

```
[sam@chenwy sam]$ paste -s pas2 pas1
P.Jones S.Round L.Clip
ID897    ID666    ID982
```

paste 命令管道输入

p a s t e 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：

代码:

```
[sam@chenwy sam]$ ls | paste -d" " - - - - -
1.bak 1.txt append.sed backll.ee change.sed
data.f data.txt delete_me_and_die dht dir1
.....................
```

一行显示四个文件，以空格分开

代码:

```
[sam@chenwy sam]$ ls | paste -d: - - - - -
1.bak:1.txt:append.sed:backll.ee:change.sed
data.f:data.txt:delete_me_and_die:dht:dir1
.....................
```

一行显示四个文件，以冒号:分开

也可以以一列格式显示输出：

代码:

```
[sam@chenwy sam]$ ls | paste -d" " -
1.bak
1.txt
append.sed
backll.ee
.................
一行显示一个文件
```

split 用法

s p l i t 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 v i 或其他工具诸如 s o r t 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。

s p l i t 命令一般格式：

代码:

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

这里 o u t p u t - f i l e - s i z e 指的是文本文件被分割的行数。
s p l i t 查看文件时，o u t p u t - f i l e - s i z e 选项指定将文件按每个最多 1 0 0 0 行分割。
如果有个文件有 38 行，那么将分割成 3 个文件，分别有
10、10、10、8 行。每个文件格式为 x [a a]到 x [z z]，x 为文件名首字母， [a a]、[z z]为文件名剩余部分顺序字符组合，下面的例子解释这一点。
如 passwd 有 38 行：

代码:

```
[sam@chenwy split]$ ls -l
总用量 8
-rw-r--r--    1 sam      sam         1649 12 月  4 11:13 passwd
-rw-rw-r--    1 sam      sam           84 12 月  4 11:19 split1
```

代码:

```
[sam@chenwy split]$ split -10 passwd
[sam@chenwy split]$ ls -l
总用量 24
-rw-r--r--    1 sam      sam         1649 12 月  4 11:13 passwd
-rw-rw-r--    1 sam      sam           84 12 月  4 11:19 split1
-rw-rw-r--    1 sam      sam          368 12 月  4 11:24 xaa
-rw-rw-r--    1 sam      sam          474 12 月  4 11:24 xab
-rw-rw-r--    1 sam      sam          495 12 月  4 11:24 xac
-rw-rw-r--    1 sam      sam          312 12 月  4 11:24 xad
```

生成了四个文件，前三个文件每个文件 10 行，最后一个 8 行，分割分的文件名自动产生，格式为 x[a-a][z-z]

再如 split 有 6 行：

代码:

```
[sam@chenwy split]$ cat split1
this is line1
this is line2
this is line3
this is line4
this is line5
this is line6
```

按每个文件 1 行分割，命令为：

代码:

```
[sam@chenwy split]$ split -1 split1
[sam@chenwy split]$ ls -l
总用量 32
-rw-r--r--    1 sam      sam         1649 12 月  4 11:13 passwd
-rw-rw-r--    1 sam      sam           84 12 月  4 11:19 split1
-rw-rw-r--    1 sam      sam           14 12 月  4 11:25 xaa
-rw-rw-r--    1 sam      sam           14 12 月  4 11:25 xab
-rw-rw-r--    1 sam      sam           14 12 月  4 11:25 xac
-rw-rw-r--    1 sam      sam           14 12 月  4 11:25 xad
-rw-rw-r--    1 sam      sam           14 12 月  4 11:25 xae
-rw-rw-r--    1 sam      sam           14 12 月  4 11:25 xaf
```

文件有 6 行，split 按每个文件 1 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：

代码:

```
[sam@chenwy split]$ cat xaa
this is line1
[sam@chenwy split]$ cat xaf
this is line6
```

————————————————

使用 head 和 tail 以块方式读取文本流

没实践过，有兴趣的自己试试

假定您想只处理文件的一部分，譬如头几行或后几行，那您该怎么做呢？请使用 head（它将头 10 行发送至标准输出）或 tail（它将后 10 行发送至标准输出）。

您可以通过使用 -n 选项改变这些命令发送至其标准输出的行数（当然，输出结果将随 XF86Config 文件的内容而不同）：

清单 1. 将 XF86Config 中选定行数的内容发送至标准输出

代码:

```
$ head -n 4 /etc/X11/XF86Config
```

引用:

```
# File generated by anaconda.
```

```
#
**********************************************************************

# Refer to the XF86Config(4/5) man page for details about the format of

# this file.
```

代码:

```
$ tail -n 4 /etc/X11/XF86Config
```

引用:

```
Modes "1600x1200"

ViewPort 0 0

EndSubsection

EndSection
```

如果您想让 head 或 tail 以字节而不是以行为单位，那该怎么办呢？您可以用 -c 选项代替 -n 选项。因此，要显示前 200 个字符，请使用

代码:

```
head -c 200 file
```

或者使用

代码:

```
tail -c 200 file
```

来显示后 200 个字符。如果数字后面跟有 b（表示块（block）），那么这个数字将被乘以 512。类似地，跟有 k（表示千字节（kilobyte））表示用 1024 去乘给定的数字，而跟有 m（表示兆字节（megabyte））表示用 1048576 字节去乘给定的数字。

请记住，

代码:

```
head file1 file2 file3
```

和

代码:

```
cat file1 file2 file3 | head
```

之间有重大差别。前者将打印每个文件指定行数的内容，不同文件的内容之间用头信息隔开，头信息以 ==> 后跟文件名开头。
后者将打印由 cat 命令后所列文件组成的输入流中指定行数的内容，但将把输入流作为单个文件对待。
可以使用 -q（表示静默（quiet））选项关闭文件名头信息。与 -q 相反的是 -v（表示详列（verbose））。

假如您要处理的文件在处理期间一直在发生变化（比如，当您让 head 或 tail 读取来自正在被另一个命令写入的文件的数据时，就是这种情况），请使用 -f 选项让 tail 持续读取来自指定文件的数据并将这些数据发送至 tail 自己的标准输出中。通过管道发送数据时该选项会被忽略。因此，cat file | tail -f 将不会得到所期望的结果，但 tail -f file 则可以。

（如果 tail 正在读取的文件不止一个，那么各行内容之间将用标准头信息隔开，以指明它们来自哪个文件，标准头信息以 ==> 开头。）

这个选项用于监视系统日志再合适不过，譬如，在单独的终端窗口（或单独的控制台）中执行的 tail -f /var/log/access.log 将持续打印每次点击后新添加的 Apache 访问日志条目，一直到您用 Ctrl-C 停止它为止。

通过组合使用 head 和 tail，可以从文件的中间部分读取给定长度的一块数据！下面说明如何做到：假定您想从文件开头算起第 1000 字节处开始读取一块 789 字节的数据。可以使用

代码:

```
cat file | head -c 1788 | tail -c 789
```

来解决这一问题。

使用 tac 命令逆序排序文件

如果您想对文件中的各行进行逆序排序，该怎么做呢？这就要用到 tac 命令。（请注意，tac 由 cat 逆序拼写而成。）该命令对所列出的文件中的各行或各段进行逆序排序。

该命令不能逆序排序各文件的顺序 — 这个任务您得自己做，以逆向顺序在 tac 命令后列出各文件即可。作为说明 tac 的工作原理的示例，对一些文件使用 ls -l | tail 和 ls -l | tail | tac，比较其结果。

代码:

```
[sam@chenwy split]$ ls -l |tail
总用量 32
-rw-r--r--  1 sam  sam  1649 12 月  4 11:13 passwd
-rw-rw-r--  1 sam  sam    84 12 月  4 11:19 split1
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xaa
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xab
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xac
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xad
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xae
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xaf
```

正序

代码:

```
[sam@chenwy split]$ ls -l |tail | tac
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xaf
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xae
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xad
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xac
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xab
-rw-rw-r--  1 sam  sam    14 12 月  4 11:25 xaa
-rw-rw-r--  1 sam  sam    84 12 月  4 11:19 split1
-rw-r--r--  1 sam  sam  1649 12 月  4 11:13 passwd
```

倒序

代码:

```
[sam@chenwy split]$ tac split1
this is line6
this is line5
this is line4
this is line3
this is line2
this is line1
```

对文件操作

shell基础 12:tr [保留]

关于 tr

t r 用来从标准输入中通过替换或删除操作进行字符转换。t r 主要用于删除文件中控制字符或进

行字符转换。使用 t r 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。t r 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。

下面讲述：

引用：

- 大小写转换。
- 去除控制字符。
- 删除空行。

带有最常用选项的 t r 命令格式为：

代码：

```
tr-c-d-s["string1_to_translate_from"]["string2_to_translate_to"]input_file
```

这里：

引用：

-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 A S C I I。

-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。

I n p u t - f i l e 是转换文件名。虽然可以使用其他格式输入，但这种格式最常用。

字符范围

使用 t r 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。

引用：

[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 A S C I I 字符。

[O*n] 表示字符 O 重复出现指定次数 n。因此[O * 2]匹配 O O 的字符串。

大部分 t r 变种支持字符类和速记控制字符。

字符类格式为[：c l a s s]，包含数字、希腊字母、空行、小写、大写、c n t r l 键、空格、点记符、图形等等。

下表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[“\ 0 1 2”]或 “\ 0 1 2”，t r 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， t r 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。

tr 中特定控制字符的不同表达方式

代码:

```
速记符含义八进制方式
\ a Ctrl-G 铃声\ 0 0 7
\ b Ctrl-H 退格符\ 0 1 0
\f Ctrl-L 走行换页\ 0 1 4
\n Ctrl-J 新行\ 0 1 2
\ r Ctrl-M 回车\ 0 1 5
\t Ctrl-I tab 键\ 0 11
\ v Ctrl-X \ 0 3 0
```

——————————————————
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 v i 编辑器中，偶尔按住一个键不放。

代码:

```
[sam@chenwy split]$ cat opps.txt
And the cowwwwws went homeeeeeeeeeeeee
Or did theyyyyyyyyyyyyy
```

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 t r 命令。

代码:

```
[sam@chenwy split]$ tr -s "[a-z]" < opps.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 c a t 命令，再将结果管道输出至 t r，结果是一样的。

代码:

```
[sam@chenwy split]$ cat opps.txt | tr -s "[a-z]"
And the cows went home
```

```
Or did they
```

删除空行

要删除空行，可将之剔出文件。下面是一个文件 p l a n e . t x t。文本间有许多空行。

代码:

```
[sam@chenwy split]$ cat plane.txt
plane.txt
9879932 Spitfire




190992 Lancaster




238991 Typhoon
```

使用- s 来做这项工作。换行的八进制表示为\ 0 1 2，命令为：

代码:

```
[sam@chenwy split]$ tr -s "[\012]" < plane.txt
plane.txt
9879932 Spitfire
190992 Lancaster
238991 Typhoon
```

也可以使用换行速记方式\ n。

代码:

```
[sam@chenwy split]$ tr -s "[\n]" < plane.txt
plane.txt
9879932 Spitfire
190992 Lancaster
238991 Typhoon
```

大写到小写

除了删除控制字符，转换大小写是 t r 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，t r 从一个包含大小写字母的字符串中接受输入。

代码:

```
[sam@chenwy split]$ 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[：l o w e r：]和[：u p p e r：]。

代码:

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

删除指定字符

偶尔会从下载文件中删除只包含字母或数字的列。需要结合使用- c 和- s 选项完成此功能。下面的文件包含一个星期的日程表。任务是从其中删除所有数字，只保留日期。日期有大写，也有小写格式。因此需指定两个字符范围[a - z]和[A - Z]，命令 tr -cs "[a-z][A-Z]""[\012*]" 将文件每行所有不包含在[a - z]或[A - Z]（所有希腊字母）的字符串放在字符串 1 中并转换为一新行。- s 选项表明压缩所有新行， - c 表明保留所有字母不动。原文件如下，后跟 t r 命令：

代码:

```
[sam@chenwy split]$ cat diary.txt
mondy 10:50
Tuesday 15:00
wednesday 15:30
thurday 10:30
Fridya 09:20
```

代码:

```
[sam@chenwy split]$ tr -cs "[a-z][A-Z]" "[\012*]" <diary.txt
mondy
Tuesday
wednesday
thurday
Fridya
```

转换控制字符

t r 的第一个功能就是转换控制字符，特别是从 d o s 向 U N I X 下载文件时，忘记设置 f t p

关于回车换行转换的选项时更是如此。
下面是故意没有设置转换开关的一个文本文件，是关于文具需求的一部分内容。使用 c a t- v 显示控制字符。

代码:

```
[sam@chenwy split]$ cat -v stat.tr
Boxes paper      12^M
Clips metal      50^M
Pencils-meduim   10^M
^Z
```

猜想‘中间空的是’是 t a b 键。每一行以 C t r l - M 结尾，文件结尾 C t r l - Z，以下是改动方法。
使用- s 选项，查看 A S C I I 表。^的八进制代码是 1 3 6，^ M 是 0 1 5，t a b 键是 0 11，^ Z 是 0 3 2 ,下面将按步骤完成最终功能。

用新行替换每行末尾的^ M，并用\ n 去除^ Z，输入要来自于临时工作文件 s t a t . t m p。将结果重定向到临时工作文件 s t a t . t m p。

代码:

```
[sam@chenwy split]$ tr -s "[\015\032]" "\n" <stat.tr >stam.tmp
[sam@chenwy split]$ cat -v stam.tmp
Boxes paper      12
Clips metal      50
Pencils-meduim   10
```

快速转换
如果需要删除文件中^ M，并代之以换行。使用命令：

代码:

```
[sam@chenwy split]$ tr -s "[\015]" "\n" < stat.tr |cat -v
Boxes paper      12
Clips metal      50
Pencils-meduim   10
^Z
```

或者用下述命令得同样结果。

代码:

```
[sam@chenwy split]$ tr -s "[\015]" "\n" < stat.tr >stat1.tr
[sam@chenwy split]$ cat stat1.tr
Boxes paper      12
```

```
Clips metal     50
Pencils-meduim  10
```

也可以用下述命令：

代码：

```
[sam@chenwy split]$ tr -s "[\r]" "\n" < stat.tr
```

代码：

```
[sam@chenwy split]$ tr -s "\r" "\n" < stat.tr
```

另一个一般的 D o s 到 U N I X 转换是命令：

代码：

```
[sam@chenwy split]$ tr -s "[\015\032]" "[\012*]" < stat.tr
Boxes paper     12
Clips metal     50
Pencils-meduim  10
```

将删除所有 ^ M 和 ^ Z，代之以换行。

要删除所有的 t a b 键，代之以空格，使用命令：

代码：

```
[sam@chenwy split]$ tr -s "[\011]" "[\040*]" < stat.tr >temp.txt
[sam@chenwy split]$cat -v temp.txt
Boxes paper 12^M
Clips metal 50^M
Pencils-meduim 10^M
^Z
```

替换 p a s s w d 文件中所有冒号，代之以 t a b 键，可以增加可读性。将冒号引起来，指定替换字符串中 t a b 键八进制值 0 11，下面是 p a s s w d 文件，后跟 t r 命令结果：

代码：

```
[sam@chenwy split]$ tr -s "[:]" "[\t]" < passwd
root    x       0       0       root    /root   /bin/bash
bin     x       1       1       bin     /bin    /sbin/nologin
daemon  x       2       2       daemon  /sbin   /sbin/nologin
.....................
```

或

代码:

```
[sam@chenwy split]$ tr -s "[:]" "[\011]" < passwd
```

匹配多于一个字符

可以使用[c h a r a c t e r * n]格式匹配多于一个字符。下述文件列出系统硬盘信息，其中包含了系统已经注册的和未识别的。第一列是数字，如果不全是 0，表明第二列相应硬盘已经注册。

有时全部为 0 看起来很烦人，找个吸引人注意力的符号来代替它，以便一眼就能看出哪个硬盘已注册，哪个不可识别。原文件如下：

代码:

```
[sam@chenwy split]$ cat hdisk.txt
15566 hdisk3
456554 hdisk2
0000 hdisk1
```

从文件列表中知道，有一个硬盘未注册，因此用星号代替所有的 0。模式为[0 * 4]，意即匹配至少 4 个 0，替换字符串为星号，过滤命令及结果如下：

代码:

```
[sam@chenwy split]$ tr "[0*4]" "*" < hdisk.txt
15566 hdisk3
456554 hdisk2
**** hdisk1
```

但我发现加上[]后结果不对了

代码:

```
[sam@chenwy split]$ tr "[0*4]" "[*]" < hdisk.txt
15566 hdisk3
456554 hdisk2
]]]] hdisk1
```