第2章 基本UNIX实用程序

在这一章中，介绍几个最基本的UNIX实用程序。这些命令的操作对象都是文本文件中的文本数据。在后续的章节中，还介绍与文件系统有关的命令。其他的命令，如进程操作、IPC对象的操作，在需要时介绍。

UNIX中有很多文本文件的处理程序，如：more，less，pg，cat，hd，od，head，tail，sort，wc，grep，cut，paste，cb，pr，awk，……。这些命令普遍有下列特点：

（1）当不指定文件名（处理对象）时，从标准输入stdin获得数据；

（2）当指定文件名时，从文件中获取数据，而且可以同时指定多个文件；

（3）处理结果在标准输出stdout显示。

这些命令设计成这样的风格，使用户可以方便地利用UNIX的重定向和管道功能组合出所需要的命令。可以指定多个文件名的做法，与Shell的文件名通配符展开有关，在后面4.3节介绍。

Linux的重定向与管道：

重定向的作用：

将标准输出(即屏幕)的信息指向一个文件，或将标准输入(即键盘)的信息指向一个文件。重定向的代号有<和>。

输入重定向<：**write liang<msg.liang**

输出重定向>：

date > a.txt //重定向写入(覆盖)到a.txt

cal >> a.txt //重定向追加写入到a.txt

管道：管道将管道前面进程的输出结果作为管道后面的进程输入内容处理

语法：command1 | command2 [| commandN...]

ls | wc –l //统计当前目录下文件个数

2.1 more,less,pg:逐屏显示文件内容

more命令：以一页一页的形式显示文本内容，方便用户逐页阅读

more命令格式:

more [文件名]

表2-1 more命令的子命令

按 键	功 能
空格	显示下一屏
回车	上滚一行，当你所感兴趣的段落内容的一部分正好处于当前屏幕的尾部，另有一部分在下一页，可以连续按回车，将感兴趣的部分滚动上来
q	（Quit）退出程序，后面的内容不再显示
/pattern	搜索指定模式的字符串，模式描述使用正则表达式
/	继续查找指定模式的字符串
h	（Help）帮助信息。打印出 more 命令的所有功能列表
Ctrl+L	屏幕刷新

例子：

more server.c

more *.[ch]

ls -l | more

第一个命令，指定了一个文件**server.c**作为处理对象。第二个命令，指定多个文件作为处理对象，星号（*）是文件名通配符。方括号括起来的两个字符，是**UNIX**文件名通配符的一种描述，要求文件名有**.c**或者**.h**后缀。第三个命令，指定了**0**个处理对象，这样**more**从标准输入获取数据。这里的管道符（|），使标准输入来自于上个命令的标准输出。**ls –l**命令用于列出当前目录，在**4.4**节介绍。

ls -l 命令：

[root@localhost ~]$ ls -l /etc

total 1504

drwxr-xr-x. 3 root root   4096 Aug 8 11:12 abrt

drwxr-xr-x. 4 root root   4096 Aug 8 11:12 acpi

-rw-r--r--.   1 root root   16 Aug 8 11:19 adjtime

例子：

more /var/log/syslog //more 命令默认是整屏显示

```
pungki@dev-machine:~$ more /var/log/syslog
Jan 14 16:31:32 dev-machine kernel: imklog 5.8.11, log source = /proc/kmsg star
ed.
Jan 14 16:31:32 dev-machine rsyslogd: [origin software="rsyslogd" swVersion="5.
.11" x-pid="817" x-info="http://www.rsyslog.com"] start
Jan 14 16:31:32 dev-machine rsyslogd: rsyslogd's groupid changed to 103
Jan 14 16:31:32 dev-machine rsyslogd: rsyslogd's userid changed to 101
Jan 14 16:31:32 dev-machine rsyslogd-2039: Could not open output pipe '/dev/xco
sole' [try http://www.rsyslog.com/e/2039 ]
Jan 14 16:31:32 dev-machine kernel: [    0.000000] Initializing cgroup subsys c
uset
Jan 14 16:31:32 dev-machine kernel: [    0.000000] Initializing cgroup subsys c
u
Jan 14 16:31:32 dev-machine kernel: [    0.000000] Linux version 3.8.0-35-gener
c (buildd@allspice) (gcc version 4.7.3 (Ubuntu/Linaro 4.7.3-1ubuntu1) ) #50-Ubu
tu SMP Tue Dec 3 01:25:33 UTC 2013 (Ubuntu 3.8.0-35.50-generic 3.8.13.13)
Jan 14 16:31:32 dev-machine kernel: [    0.000000] KERNEL supported cpus:
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   Intel GenuineIntel
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   AMD AuthenticAMD
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   NSC Geode by NSC
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   Cyrix CyrixInstead
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   Centaur CentaurHauls
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   Transmeta GenuineTMx86
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   Transmeta TransmetaCPU
--More--(0%)
```

```
up to 0x37bfdfff @ [mem 0x01ffa000-0x01ffffff]
Jan 14 16:31:32 dev-machine kernel: [    0.000000] RAMDISK: [mem 0x34320000-0x36
187fff]
Jan 14 16:31:32 dev-machine kernel: [    0.000000] ACPI: RSDP 000e0000 00024 (v0
2 VBOX   )
Jan 14 16:31:32 dev-machine kernel: [    0.000000] ACPI: XSDT 3fff0030 00034 (v0
1 VBOX   VBOXXSDT 00000001 ASL  00000061)
Jan 14 16:31:32 dev-machine kernel: [    0.000000] ACPI: FACP 3fff00f0 000F4 (v0
4 VBOX   VBOXFACP 00000001 ASL  00000061)
Jan 14 16:31:32 dev-machine kernel: [    0.000000] ACPI: DSDT 3fff0410 01B96 (v0
1 VBOX   VBOXBIOS 00000002 INTL 20100528)
Jan 14 16:31:32 dev-machine kernel: [    0.000000] ACPI: FACS 3fff0200 00040
Jan 14 16:31:32 dev-machine kernel: [    0.000000] ACPI: SSDT 3fff0240 001CC (v0
1 VBOX   VBOXCPUT 00000002 INTL 20100528)
Jan 14 16:31:32 dev-machine kernel: [    0.000000] 131MB HIGHMEM available.
Jan 14 16:31:32 dev-machine kernel: [    0.000000] 891MB LOWMEM available.
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   mapped low ram: 0 - 37bfe00
0
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   low ram: 0 - 37bfe000
Jan 14 16:31:32 dev-machine kernel: [    0.000000] Zone ranges:
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   DMA      [mem 0x00010000-0x
00ffffff]
Jan 14 16:31:32 dev-machine kernel: [    0.000000]   Normal   [mem 0x01000000-0x
--More--(2%)
```

less命令格式:

less [文件名]

在Linux系统中的less命令和more的功能类似，但是，后退浏览的功能更强，可以直接使用键盘的上下箭头键，或者j，k，类似vi的光标定位键，以及 PageUp 键， PageDown 键， 或者 Ctrl+F 键、Ctrl+B键、Home键和End键，使用起来更方便，是对more命令的增强。在Linux中使用广泛。许多UNIX没有less命令，但是，这些UNIX中的more命令的增强功能融入了less命令的部分功能。

pg命令格式:

pg [文件名]

pg命令显示满一屏后，屏幕最后一行为冒号（:）提示符，显示暂停，等待按键命令，每个命令之后还需要按Enter键，这使得这个命令用起来比more麻烦。表2-2列出了可以使用的按键命令。

表2-2 pg命令的子命令

按　键	功　　能
回车	下一屏
l	（Line）上滚一行
q	（Quit）退出
/pattern	查找指定模式的字符串，模式描述用正则表达式规则
/	继续查找
h	（Help）帮助信息
Ctrl+L	屏幕刷新

比较而言，more比pg少按键，pg命令的每个子命令要比more多按Enter键。man命令也使用more或pg。一般来说，man先查找磁盘文件，再解压缩文件，最后用more或pg显示。根据满一屏后最后一行的提示，可以判断用的是pg还是more，然后使用相应的命令翻页或滚动。System V系统中，一般默认不太好用的pg为man的分屏浏览器。系统管理员可以设定man命令中使用more或pg。例如：在SCO UNIX中将文件/etc/default/man中的行

PAGER=/usr/bin/pg

改为

PAGER=/usr/bin/more

那么，man将使用more。

2.2 cat:列出文本文件内容

cat命令用于列出文本文件的内容，功能和DOS中的TYPE命令类似。

基本用法：cat［文件名］

例如：

cat try.c

指定处理对象一个，打印文本文件try.c的内容。

cat > try.txt

指定处理对象0个，程序从标准输入（键盘）获取数据，直到按Ctrl+D键标志输入结束。程序输出被重定向到文件try.txt。这是一种很简单的建立新文件的方法。

cat tryl.c try2.c try.h

命令行参数:3个，指定处理对象为3个文件，程序顺序打印出3个文件的内容。

cat tryl.c try2.c try.h > trysrc

cat makefile *.[ch] > src

上述两个命令都是将多个文件，输出重定向到一个文件，完成文件合并。cat命令得名于“串接”（catenate），在指定多个文件时，cat就是将这些文件串接起来打印。

假设我们有test1.txt、test2.tx和test3.txt，并且内容如下；

```
cat test1.txt
123456
Northwest University

cat test2.txt
Happy New Year

cat test3.txt
Happy Birthday

cat test1.txt test2.txt test3.txt > test4.txt
more test4.txt
123456
Northwest University
Happy New Year
Happy Birthday
```

2.3 od:列出文件每个字节的内容

od得名于octal dump，八进制打印。

格式：od [-x][-c] [filename]

例子：

od a.dat

以八进制打印出文件。

od命令的-x选项，使用十六进制打印文件内容。由于正好两个十六进制数字描述出一个字节，所以十六进制打印经常被使用。od命令的-c选项，对可打印字符打印出字符，对不可打印字符打印出ASCII码。

当不指定处理对象时，od命令从标准输入获取数据。下边的组合命令可以查出字符abcdABCD的ASCII码。echo命令将命令行参数输出到标准输出。

echo abcdABCD | od -x

```
$ echo abcdABCD | od –x
0000000  6162 6364 4142 4344 0a00
$
```

2.4 head与tail:打印文件头或尾

head和tail的用法类似，可以打印出指定文件头部或者尾部的一部分内容。UNIX的选项一般以减号开头，这里减号后边的数字是期望看到的行数。对head和tail命令，若未指定行数，默认为显示10行。

【例2-2】 使用head和tail命令的例子。

$ head -15 ab.c

（显示文件ab.c中前15行）

$ head -23 a.c b.c c.c | more

（显示三个文件各自的前23行共显示69行）

$ tail -20 liu.mail

（打印出文件尾部的20行，看看邮件尾部的发信者签名）

$ netstat –s –p tcp | head –5

tcp:

5902873 packets sent

5118107 data packets (3423705271 bytes)

35445 data packets (18222213 bytes) retransmitted

329697 ack-only packets (35578 delayed)

tail命令有个重要的选项-f(forever)经常会被用到。具体用法为：

tail –f *filename*

功能：实时显示文件新追加的内容

【例2-3】 使用tail命令实时显示文件新追加的内容。

tail –f /usr/adm/pppd.log

上述命令列出文件/usr/adm/pppd.log的尾部（UNIX文件系统路径名分割用正斜线/，与DOS中使用反斜线\不同）。文件尾部的10行打印完毕之后，tail命令并不退出，继续等待。UNIX允许多个活动程序同时操作磁盘上的同一个文件。如果系统中其他的活动进程会在pppd.log文件的尾部追加，那么，tail会实时地打印出这些追加的内容。随着其他进程对pppd.log文件的不断追加，tail命令会随着系统其他进程对文件追加的进度，及时打印出这些信息。停止tail进程，按下Ctrl+C键。

2.5 wc:字计数

字计数（word count）命令wc，可以列出文件中一共有多少行，有多少个单词，多少字符，当指定的文件数大于1时，最后还列出一个合计。

格式： wc [-l][-c][-w] [filename]

-1：文件行数；

-c：文件字符数；

-w：文件单词数。

【例2-4】使用wc命令的例子。

wc sum.c (1个文件)

wc x.c makefile stat.sh (多个文件)

以下是上机操作实例。

$ wc *.[ch] makefile *.sh

1912	6532	49143	auth.c
1227	4038	32394	ccp.c
860	2558	22487	chap.c
124	695	4702	chap.h
762	2129	17159	fsm.c
144	792	5237	fsm.h
2168	7487	56500	lcp.c
87	288	2035	magic.c
1827	5833	44234	main.c
306	1901	11841	md5.c
58	349	3048	md5.h
390	1343	9138	multilink.c
1545	5220	37149	options.c

738	4142	28320	pppd.h
876	2712	18623	utils.c
347	1324	9673	makefile
78	735	5125	ppp.sh
1344	948	078	356808 total

列出的内容，第一列为文本文件的行数。第二列，是“词”数。第三列，是字符数。最后一列是文件名。当处理对象为两个或者更多时，最后一行有个合计。

```
$ who | wc -l
10
```

2.6 sort:对文件内容排序

sort可针对文本文件的内容，以行为单位来排序。

格式：sort [-n] [filename]

默认的排序方法是将文件每行作为一个整体，按照ASCII码字符串的方式比较，从小到大排列。字符串比较时，注意会出现32 > 123。

例子：

cat testfile #文件原有排序
Test 30
Hello 95
Linux 85

sort testfile #重排结果
Hello 95
Linux 85
Test 30

格式：sort [-n] [filename]

-n：文件内容按数值升序排序，而不是按照字符串比较规则

例子：

cat number.txt	sort number.txt	sort -n number.txt
1	1	1
10	10	2
19	11	5
11	19	10
2	2	11
5	5	19

【例2-5】使用sort命令的例子。

$ ls -s | sort | tail –10（默认地按照字符串方式比较进行排序）

```
 44 main.c
 48 auth.c
 56 lcp.c
1268 BUGS.report
1720 paper.pdf
202712 document.pdf
27052 disk.img
27056 linux-src.tar.Z
3532 pppd.log
total 263724
```

$ ls -s | sort -n | tail –10（-n选项对于数字按照算术值大小排序）

```
 40 options.c
 44 main.c
 48 auth.c
 56 lcp.c
1268 BUGS.report
```

1720 paper.pdf

3532 pppd.log

27052 disk.img

27056 linux-src.tar.Z

202712 document.pdf

2.7 tee:三通

将从标准输入stdin得到的数据抄送到标准输出stdout显示，同时存入磁盘文件中。这一功能类似水管或电线的T形三通，命令取名tee。

许多程序员都有过这样的经历，编写源程序，编译后运行，运行过程中有许多打印，但是，打印的信息较多，很快地滚动过屏幕，程序员又想再重新看看已经滚过屏幕的信息，这时，就可以使用tee命令。

格式：tee [-a] filename

-a：不覆盖filename原有信息

例: **./myap | tee myap.log**

例子：

free -h | tee mem.txt #free命令显示系统内存使用信息

	total	used	free	shared	buff/cache	available
Mem:	1.8G	164M	1.2G	9.6M	387M	1.5G
Swap:	2.0G	0B	2.0G			

cat mem.txt

	total	used	free	shared	buff/cache	available
Mem:	1.8G	164M	1.2G	9.6M	387M	1.5G
Swap:	2.0G	0B	2.0G			

2.8 正则表达式的概念

在UNIX的文本文件处理中，广泛地使用正则表达式（regular expressions）的概念。

正则表达式：指对于一个字符串模式的描述，通常用在模式匹配操作中，寻找与所设条件相符字符串。

grep awk sed等命令中使用。

1. 正则表达式中的特殊字符

正则表达式的特殊字符，共6个：

. * [\ ^ $

除此之外的其他字符与其自身匹配。如：hello，bye。

2. 单字符正则表达式

长的、复杂的正则表达式是由单字符正则表达式构成的。

（1）普通字符。除了前边列出的六个特殊字符外，其他字符与其自身匹配，如：a与a匹配，b与B不匹配，?与j不匹配。

（2）转义字符（\）。在特殊字符前，增加反斜线，则丧失字符的特殊含义，与其自身匹配。

\. * \$ \^ \[\\

（3）句点（·）。句点匹配任意单字符。

（4）单字符集合的定义。使用方括号。左方括号与其后的右方括号一起定义一个集合，描述一个字符。

① 在左方括号（[）与右方括号（]）之间的字符为集合的内容，如：单字符正则表达式［abcd］与a、b、c、d中任一字符匹配。

② 可以用减号定义一个区间。如[a-d]、[A-Z]、[a-zA-Z0-9]。若减号在最后，则失去表示区间的意义，如：[ad-]只与3个字符a、d、-之一匹配。

③ 可以用^表示补集。若^在开头，则表示与集合内字符之外的任意其他单字符匹配，如:[^a-z]匹配任一非小写字母。若^不在开头，则失去其表示补集的特殊意义。如:[a-z^]能匹配27个单字符。

④ 正则表达式的特殊字符，在方括号内时，仅代表它们自己。如：[*.]可以匹配2个单字符。

3. 单字符正则表达式的组合

（1）简单串结。正则表达式abc是三个单字符正则表达式的串结，仅能匹配字符串abc。

正则表达式[A-Z].[0-9].，是四个单字符正则表达式的串结，要求字符串的第一个字符是大写字母，第三个字符是数字，第二个和第四个字符可以是任意字符。

正则表达式[Mm]akefile匹配Makefile或者makefile。

正则表达式a\[i]*3\.14匹配字符串：a[i]*3.14

正则表达式a[i]*3.14并不匹配字符串a[i]*3.14

（2）星号（*）。单字符正则表达式后跟星号（*），则匹配此单字符正则表达式的0次或任意多次出现。

（3）行尾符$和行首符^。$只有出现在正则表达式最尾部时才有特殊意义，否则与其自身匹配。类似地，^只有出现在正则表达式最首部时才有特殊意义，否则与其自身匹配。

对于C语言的程序员来说，应当注意正则表达式的这些处理。

【例2-6】 正则表达式中符号*的作用。

（1）正则表达式12*4。

1234　　不匹配

1224　　匹配

12224　　匹配

14　　匹配

此例中*作用于它左面的单字符正则表达式2。注意正则表达式12*4与字符串14是匹配的。

（2）正则表达式[A-Z][0-9]*。此例中*作用于它左侧的单字符正则表达式为[0-9]，代表：

[A-Z]

[A-Z][0-9]

[A-Z][0-9][0-9]

[A-Z][0-9][0-9][0-9]

…

与A、A1、C45、D768匹配，与b64512、T546t不匹配。

（3）正则表达式[Cc]hapter *[1-4]，在*号前有一个空格。允许数字1～4之前有多个或者0个空格。可匹配Chapter2、chapter 3，等等。

类似地，正则表达式：

a\[i] *= *b\[j] ** *c\[k]

可以匹配字符串a[i]=b[j]*c[k]，并允许等号和星号两侧有多个或者0个空格。

【例2-7】 正则表达式中$和^的作用。

（1）123$ 匹配文件中行尾的123，不在行尾的123字符与正则表达式123$不匹配。

（2）$123与字符串$123匹配。

（3）.$匹配行尾的任意字符。

（4）正则表达式^Hello匹配行首的Hello字符串，不在行首的Hello串不匹配。

（5）正则表达式Hel^lo与字符串Hel^lo匹配。

（6）^号后跟4个空格，仅匹配行首的连续4个空格。

注意：正则表达式规则与文件名匹配规则是不同的。一般来说，正则表达式规则用于文本处理的场合，文件名匹配规则用于文件处理的场合。星号（*）、问号（?）、句点，在正则表达式和文件名通配符中有不同的解释。

2.9 grep,egrep与fgrep:在文件中查找字符串

grep（global regular expression print）命令是一种文本过滤程序，按照正则表达式的规则，筛选出含有指定模式字符串的文本行。语法：

grep *pattern file-list*

1. grep

如果指定的文件数>1，则当查找到指定字符串时，整个行，连同该行处的文件名一起显示。如果指定的文件数≤1，则只列出含有指定模式的整个行的内容，但不显示文件名。

【例2-8】 grep命令的使用。

```
grep O_RDWR /usr/include/*.h
```

用于查找C语言中宏定义O_RDWR在哪些头文件中定义，查找范围为多个文件。类似的命令用于查找C语言的struct类型定义，等等。

```
grep routed /etc/tcp        指定文件数=1
who | grep liang            指定的文件数=0
```

例子：

ls -l | grep '^a'

通过管道过滤ls -l输出的内容，只显示以a开头的行

grep 'test' aa bb cc

显示在aa，bb，cc文件中匹配test的行

2. egrep

扩展的grep，在描述模式时，使用扩展的正则表达式，egrep对基本的正则表达式进行了扩展，可以用括号（）和表示“或”的符号|，其次，还定义了和正则表达式中的星号*地位类似的+和?。*号表示它左边的单字符正则表达式的0次或多次重复，对应地，+号表示1次或多次，?表示0次或1次。

语法：**egrep [范本模式] [文件或目录]**

[范本模式]：查找的字符串规则。

[文件或目录]：查找的目标文件或目录。

【例2-9】 egrep的扩展正则表达式的使用。

egrep可以使用扩展的正则表达式，下面是几个扩展正则表达式以及它们可以匹配的字符串示例。

(xy)*　　可匹配空字符串，xy，xyxy，xyxyxy，等。

(pink|green)　　与pink或green匹配。

[0-9]+　　不匹配空字符串，匹配长度至少为1的数字串。

a?　　匹配0个或1个a。

下面两个命令按扩展的正则表达式规则检索字符串。

```
egrep '(SEEK_|IPC_)' *.h
egrep '[0-9]:[0-9][0-9] (client|server)$'
```

egrep在指定模式方面比grep更灵活，但算法需要稍多的处理时间。

3．fgrep

快速grep，按字符串搜索而不是按模式搜索。fgrep运算速度快，适合于从大量的数据中进行检索。但它只能指定字符串，不可按模式查找。

```
cat test.txt
Hello 95
Test 30
Linux 85
```

```
fgrep -n 'Hello' test.txt
1：Hello 95
```

4．选项

grep/fgrep/egrep有若干选项用以控制输出格式。

-n 显示时每行前面显示行号。

-i 字母比较时忽略字母的大小写。

-v 显示所有不包含模式的行。

【例2-10】 grep选项的使用。

（1）grep -n __DATE__ *.c。在所有的后缀为.c的文件中查找含有正则表达式__DATE__的行，并打印行号。当文件数超过1个时，除了输出行号，还输出文件名。

（2）grep -v '[Dd]isable' device.stat>device.active。将文件device.stat中取消所有含有指定模式的行，生成新文件device.active。

（3）grep -i richard telnos。在文件telnos中检索字符串richard，不顾字母的大小写。

（4）grep '[0-9]*' chapter1。由于[0-9]*与空字符串匹配，上述命令打印出chapter1文件中所有行，而不是仅打印出含数字的行。正确的用法应当是：

grep '[0-9][0-9]*' chapter1。

打印出文件chapterl中所有含有数字的行。或者，使用egrep的扩展正则表达式：

egrep '[0-9]+' chapter1

这里给出了grep命令的3个常用选项，UNIX的grep命令共有十几个选项。前面介绍的命令中，也介绍了一些命令选项。这些选项都以减号开头。丰富的选项，为命令提供了丰富的功能选择，尽管有些功能并不经常用到。UNIX风格的大部分命令都是在一个命令行内，通过命令行参数的形式提供程序处理所需要的数据、处理对象以及描述处理方法的命令选项。程序开始运行后，就不再需要任何其他的交互式输入，直到命令执行完毕。

2.10 awk:文本处理语言

- awk是一种对文本文件进行过滤的程序，通过逐行扫描筛选出满足指定条件的文本行，并且能够生成报表。
- awk可以指定0到多个文件作为处理对象。
- awk支持条件控制、循环控制、变量定义、函数等功能。

- 用法1: awk 'Program' file-list

  用法2: awk -f ProgramFile file-list

- awk程序的写法是：

condition {action}

awk对满足“条件（*condition*）”的行，执行大括号中指定的这些“动作（*action*）”。

- **内置程序变量：是awk文本处理程序中不需要定义就可以直接使用的变量。**

awk编程中的内置变量

变　　量	含　　义
NR	当前记录的记录编号(No. of Record)
$0	当前记录
$1，$2，……	当前记录中的域
FILENAME	当前输入的文件名

描述“条件”时，有几种方法：

（1）如果不指定任何条件，那么对文本文件的所有行进行处理。

（2）使用与C语言类似的关系运算符。

awk编程中的运算符

运算符	意　　义	运算符	意　　义
<	小于	>=	大于或等于
<=	小于或等于	!=	不等于
==	等于	||	条件或
>	大于	&&	条件与

（3）正则表达式的模式匹配。模式描述方法为：/pattern/

（4）特殊的条件：BEGIN和END。BEGIN之后的动作，在awk开始处理所有文本行之前执行。同样，END之后的动作，在awk处理完所有文本行之后执行。

描述“动作”时，简单的用法有：

print 变量1， 变量2，……

printf("格式串"，变量1， 变量2，……)

print将用逗号隔开的变量打印，用空格隔开。

printf的用法和C语言里的printf函数用法类似，可以灵活地使用printf的格式控制字符串。

【例2-11】使用awk命令的例子。

$ date

Thu May 27 22:02:22 BEIJING 2004

$ date | awk '{print $4}'

（未指定条件，处理所有的文本行）

22:02:42

$ who

zhang ttylb Sep 29 11:20

liang ttyla Sep 29 11:53

zhang ttylf Sep 29 12:04

feng tty1c Sep 29 12:54

$ who | awk '/^ *zhang / {printf("%s ", $2)}'

（仅处理含有正则表达式字符串的文本行。由于printf的格式字符串尾不含\n，程序执行完之后，不换行，导致下个命令的提示符$在打印行的行尾）

tty1b tty1f $

$ ls -s | awk '$1 > 2000 { print $2 }'

（这里的描述条件为：第一列的取值大于2000的文本行）

disk.img

document.pdf

linux-src.tar.Z

pppd.log

$ cat list.awk

（事先编辑好的awk程序文件，程序中含有三组“条件 {动作}”描述。其中：BEGIN的动作有三个，程序执行时，awk处理所有文本行之前，执行BEGIN指定的三个动作；处理完所有文本行之后，执行END指定的一个动作；最后一个程序块，未指定任何条件，对所有文本行执行这个动作。在这段程序中使用了内置变量FILENAME和NR，$0）

```
BEGIN {
   printf("=====================================\n")
   printf("FILENAME %s\n", FILENAME)
   printf("-------------------------------\n")
}

END { printf("=====================================\n") }

{ printf("%3d: %s\n", NR, $0) }
```

$ awk -f list.awk md5.c

```
=====================================
FILENAME md5.c
-------------------------------
  1:
  2: #include "md5.h"
  3:
```

```
 4: /* forward declaration */
 5: static void Transform ();
 6:
 7: /* F, G, H and I are basic MD5 functions */
 8: #define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
 ……
298:   buf[2] += c;
299:   buf[3] += d;
300: }
301:
```

==========

2.11 sed:流编辑

sed（stream editor）是一个流编辑程序。当指定的处理对象为0个文件时，它从标准输入获取输入字符流，否则，将文件中的数据作为输入字符流。对输入字符流进行编辑处理，加工处理后再输送到标准输出。

用法1：sed '命令' 文件名列表

用法2：sed -f 文件名 文件名列表

这两种用法的区别和awk命令类似。

sed 命令动作说明：

a：新增，c：取代， d：删除，i：插入，

p：打印，s：取代

例子：在testfile文件第四行后添加一行，并将结果输出到标准输出

```
cat testfile    #查看testfile 中的内容
HELLO LINUX!
Linux is a free unix-type opterating system.
This is a linux testfile!
Linux test
sed -e 4a\newline testfile    #使用sed 在第四行后添加新字符串
HELLO LINUX! #testfile文件原有的内容
Linux is a free unix-type opterating system.
This is a linux testfile!
Linux test
newline
```

例子：将 /etc/passwd 的内容列出并且列印行号，同时，请将第 2~5 行删除！

```
nl /etc/passwd | sed '2,5d'
1 root:x:0:0:root:/root:/bin/bash
6 sync:x:5:0:sync:/sbin:/bin/sync
7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
.....
```

例子：将 /etc/passwd 的内容列出并且列印行号，同时，将第2-5行的内容取代成为 No 2-5 number

```
nl /etc/passwd | sed '2,5c No 2-5 number'
1 root:x:0:0:root:/root:/bin/bash
No 2-5 number
6 sync:x:5:0:sync:/sbin:/bin/sync
.....
```

【例2-12】使用sed命令的例子。

（1）tail -f pppd.log | sed 's/145\.37\.123\.26/QiaoXi/g'

sed的编辑命令有很多，这里的s命令是“替换（substitute）”，三个斜线分割的第一部分是正则表达式145\.37\.123\.26，第二部分是替换字符串QiaoXi，最后的g是global flag，这一特征字符，使得s命令在一行中遇到多个模式描述的字符串时，都替换为QiaoXi，否则，一行仅替换一次。

上述命令中，sed将IP地址转为一个名字。

（2）tail -f pppd.log | sed -f sed.script

其中sed.script文件内容如下：

s/145\.37\.123\.26/QiaoXi/g

s/102\.157\.23\.109/LiuYin/g

s/145\.37\.123\.57/DaTun/g

……

使用这样的方法，定义一张IP地址/名字的对照表，sed可以将一些程序的输出内容进行编辑替换，加工之后再显示出来。

2.12 tr:翻译字符

用法：tr *string1 string2*

在字符串*string1*中出现的输入字符被替换为字符串*string2*中的对应字符。

【例2-13】使用tr命令的例子。

（1）将大写UVX改写为小写uvx。

```
cat telnos | tr UVX uvx
```

（2）将小写字母改为大写字母。可以使用［ ］指定一个集合。

```
cat report | tr '[a-z]' '[A-Z]'
```

（3）也可以使用\加3个八进制数字（类似C语言中描述字符常数的方法）表示一个字符。下面的命令将%改为换行符：

```
cat file1 | tr % '\012'
```

（4）在1.3.5小节中介绍过UNIX和DOS文本文件的区别，下面的命令将按照二进制格式从DOS复制来的文件中多余的回车改为空格，回车的ASCII码是八进制的015。

```
cat myap.c | tr '\015' ' ' > myap1.c
```

这几个例子的命令中，注意不要漏掉必需的单引号。

2.13 cmp和diff:比较两个文件

*用法：*cmp *file1 file2*

用法：diff file1 file2

这两个命令用于比较两个文件是否相同。

cmp命令逐个字节比较两个文件是否完全相同。两个文件完全相同时，不给出任何提示；当两个文件不同时，打印出第一个不同之处。这个命令常用来判断两个文件的内容是否完全一致，无论是ASCII码文件还是二进制格式的程序或数据文件。

UNIX的diff命令可以逐行比较包括源程序文件在内的任意内容的文本格式文件。Windows的FC命令使用时经常需要/N选项，在列出文本文件行时打印行号。

命令*diff file1 file2*每发现两个文件中的一处不同，就列出一个如何将file1转化为file2的指令，这些指令有a(add)，c(change)和d(delete)，指令的格式见表2-5。指令用一个字母a，c或d表示，指令字母左边的行号是*file1*的行号，指令右面的行号是*file2*的行号。列出内容时，大于号后边的内容是需要在*file1*文件中增加的内容，小于号后边的内容是需要从*file1*中删除的内容。

表2-5 由命令diff *file1 file2*产生的文件转化指令

指　令	如何将文件*file1*转化为文件*file2*
*l1*a*l2*,*l3* > *file2*第*l2*~*l3*行内容	将*file2*的第*l2*~*l3*行追加到*file1*的第*l1*行之后
l1,*l2*c*l3*,*l4* < *file1*第*l1*~*l2*行内容 --- > *file2*第*l3*~*l4*行内容	将文件*file1*的第*l1*~*l2*行换成*file2*的第*l3*~*l4*行
l1,*l2*d*l3* < *file1*第*l1*~*l2*行内容	将文件*file1*的第*l1*~*l2*行删除以后，就与*file2*的第*l3*行以后内容相同

【例2-14】 比较两个不同版本的C语言源程序文件，找出文件的改动之处。

源程序文件f1.c被修改后的新版本源程序文件为f2.c，下面是UNIX下比较两个文本文件f1.c和f2.c的结果。

```
$ cmp f1.c f2.c
f1.c f2.c differ: char 69, line 3
$ diff f1.c f2.c
3,5d2
<    tmp->vm_mm = mm;
<    mm->map_count++;
<    tmp->vm_next = NULL;
260c257
<    i = (i+1) * 8 * sizeof(long);
---
>    i = i * 8 * sizeof(long);
```

```
528c525
<    p->swappable = 1;
---
>    p->swappable = 0;
548a546,547
>    retval = p->pid;
>    p->tgid = retval;
```