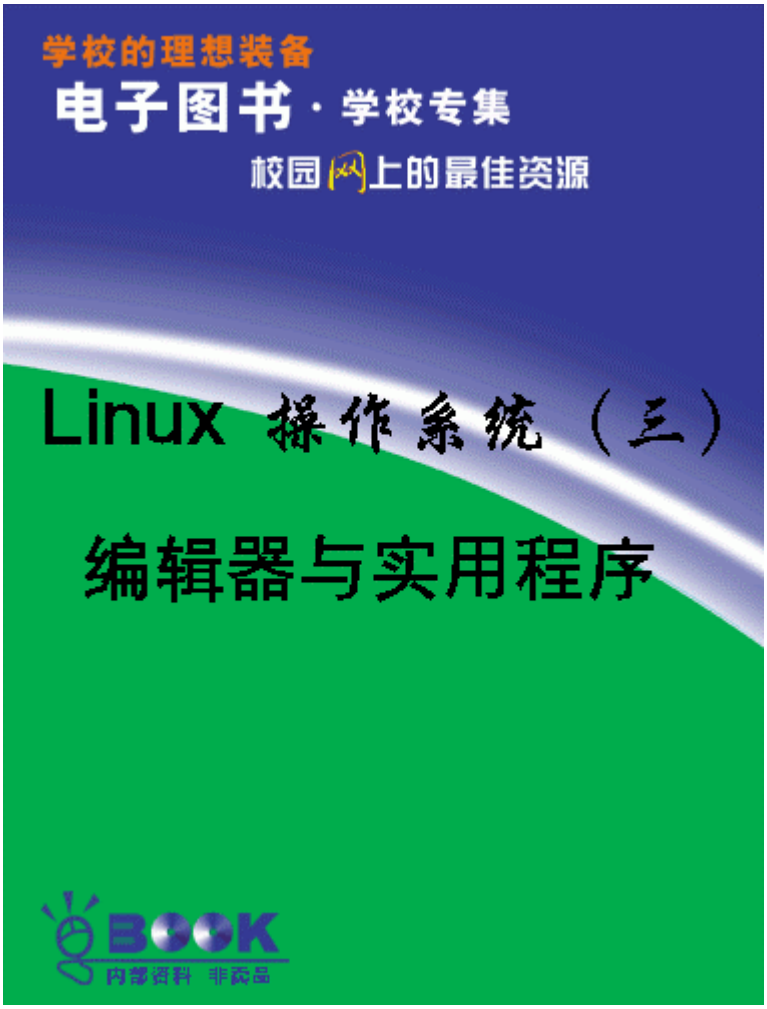

学校的理想装备
电子图书·学校专集
校园网上的最佳资源
Linux 操作系统（三）
编辑器与实用程序
BOOK
内部资料 非卖品

返回总目录

第五部分编辑器与实用程序

第 17 章 Vi 编辑器

第 18 章 Emacs 编辑器

第六部分系统管理

第 19 章系 统 管 理

第 20 章网 络 管 理

第五部分编辑器与实用程序

第 17 章 Vi编辑器

任何一套 Linux 系统上可能有各种不同的编辑器供用户使用，但所有的系统都有两个标准的编辑器：Ed 编辑器及 Vi编辑器。Ed 及 Vi编辑器是 Unix 系统下的标准应用程序，它们是在 Unix 系统下最初开发的 Ex 编辑器的基础上发展而来的。Vi 是可视化（visual）的意思，Vi 编辑器目前仍然是 Linux 系统下使用最为广泛的编辑器之一，而行编辑器 Ed 目前已经很少使用。Ed 作为行编辑器，它一次仅能编辑并显示一行，因此，它使用起来非常困难且不方便。与 Ed 不同，Vi可一次在整个屏幕上显示数据，你可以编辑显示在屏幕上的任何数据。

当 Unix 系统最初被开发出来时的时候，Vi编辑器与当时其它各种编辑器相比是一个极大的进步。自从 Vi编辑器被开发出来以后，Unix 及 Linux 系统下许多功能更强大、使用更方便的编辑器，如 WordPerfect 等编辑器相继被开发出来。在你的 Linux 系统下可能有你更熟悉的文本编辑器或文字处理软件。如果

有，则你可以不必学习 Vi 编辑器。OpenLinux 系统有一个名为 Crisplite 的编辑器，它比 Vi 编辑器使用起来要容易的多。但是，由于 Vi 编辑器是所有 Unix 及 Linux 系统下标准的编辑器，了解 Vi 编辑器对你仍然是非常有帮助的。如果你不得不在另外一个系统下工作，而该系统有一个与你的系统不同的字处理软件，这时你可以使用 Vi 编辑器，因为对所有 Unix 及 Linux 系统的任何版本，Vi 编辑器是完全相同的。

由于其可视化界面，对初学者来说，Vi 编辑器相对于行编辑器 Ed 学起来要容易些。但 Vi 编辑器又是一个非常复杂的编辑器，它有大量复杂的命令，当初次尝试学习该编辑器时，初学者非常容易变的不知所措。因为 Vi 编辑器通常可以用有多种细微差别的方法来完成相同的操作。本章将首先重点介绍 Vi 编辑器的核心命令，这些核心命令用来完成一些基本的操作。如果你一旦掌握这些核心命令，你可以考虑去掌握一些更复杂的操作与命令。这些都将在本章中给予介绍。

17.1Vi 命令模式、输入模式及行编辑模式

Vi 编辑器用键盘来完成两类不同的操作：完成编辑命令及接收字符输入。当处于编辑命令模式时，一些特定的键用来完成删除操作，一些键用来执行修改操作，还有一些用来完成关标移动。当处于字符输入模式下时，这些键则表示要输入到正在编辑的文件中的字符。在 PC 机上，许多常见的编辑器将键盘

上的按键根据以上两种模式分把它们分为两类。其中，字母类按键用来输入字符，功能键及控制键用来完成特定的编辑命令，如删除文本、移动光标等。

这类 PC 机上的编辑器必须要在有扩展键的键盘上才能完成文本编辑，这些扩展键包括功能键及光标键。但是，Unix 系统的设计是独立于任何的特定的键盘的，任何类型的终端及 PC 机键盘都可以映射到 Unix 系统上。Unix 系统下的编辑器在设计的过程中假定终端键盘只具有字母按键、部分基本控制键及 ESC 键、ENTER 键等基本功能键。与 PC 机上的编辑器把键盘上按键分成字符键及功能键不同，Unix 系统下的编辑器有两个独立的操作模式：命令模式及输入模式。在命令模式下，键盘上所有的键入都作为命令来接收。在输入模式时，你从键盘上的输入就成为正文来接收。

当你改变编辑器的操作模式时，键盘上的按键的功能就会改变。当启动并进人 Vi 编辑器时，你处于命令模式下。此时，所有按键的键入都被作为命令来接收，按某键将执行某特定的命令，例如：键入 x 键将删除当前编辑文本中的字符；键入 l 键将把光标键右移一个字符。在命令模式下键入一些编辑命令如：键入 a 键或 i 键将进入输入模式，也就是说，键入 i 键将退出命令模式而进入输入模式。

一旦进入输入模式，键盘上键的功能将改变，此时，键盘上的每个键将代表一个将要键入到文本中的字符，键盘也就类似于一部打字机，当你键入某个键时，相应的字符就被添加到编辑的文本中去。例如，如果你键入 x 键，字符 x 将被添加到文本中去；键入 TAB 键将输入制表符到文本中去。唯一例外的是 ESC 键。如果你键入 ESC 键，Vi 编辑器将自动回到命令模式下，此时所有的

按键又成为编辑命令。但你可以键入某个输入编辑命令又进入输入模式下。当编辑文本的时候，你会发现自己不断地在命令模式与输入模式之间来回切换。

尽管在 Vi 编辑器的命令模式下能完成大多数的编辑操作，但是有些操作，如文件保存及全局替换是不能实现的。对于这类操作，你必须进入底行命令模式。你可以使用 Vi 的冒号命令（:）进入底行命令模式。:命令是一个特殊的 Vi 编辑器命令，它能够使你执行行命令操作。如果进入:命令，屏幕的底行将出现一编辑行，而光标将出现在该行的行首，此时，你处于底行命令模式下。在该模式下，你可以在该行上键入编辑命令，然后键入 ENTER 键，该编辑命令将被执行。进入该模式只是暂时性的，键入 ENTER 键后将退出该模式，然后自动进入 Vi 编辑器的命令模式下，光标也将到回屏幕上先前所在的位置。图 17-1 说明了 Vi 编辑器下的三种不同的操作模式。

命令模式及输入模式是两种独立的操作模式，加上刚才介绍的底行命令模式，因此在 Vi 编辑器下有三种不同的操作模式。底行命令模式只在一行上进行操作：你键入命令及其参数，然后键入 ENTER 键来执行该命令或退出该模式。但在 Vi 命令模式下，仅键入单个键来接收命令，通过键入单个键或一系列的键来完成一个编辑命令或一系列的编辑命令。Vi 编辑器的输入模式用来把字符输入到文本文件中去，除 ESC 键外的所有键都是有效键（此时，键入 ESC 键将返回到命令模式下）。

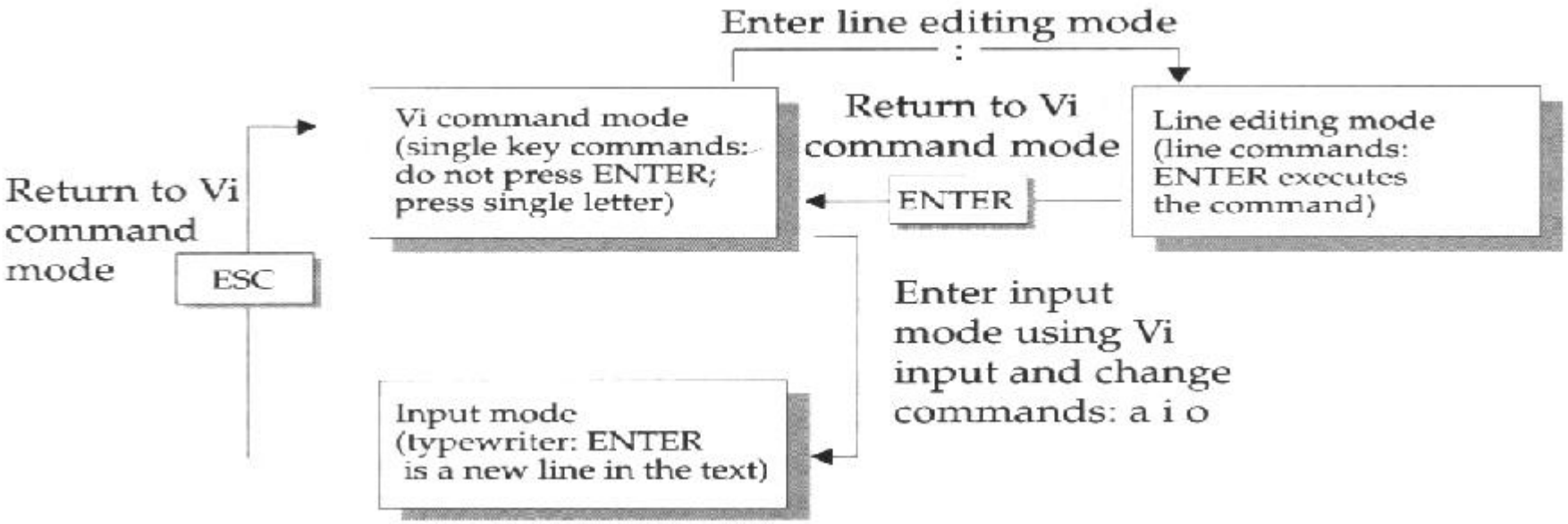


图 17-1Vi 编辑模式：输入模式、命令模式及底行命令模式

17.2 在 Vi编辑器下创建新文件、保存编辑文件及退出编辑的文件

在 Vi 编辑器下，你可以创建、保存、关闭及退出编辑的文件。彼此这些命令都完全不同。保存及退出一个文件的编辑需进入底行命令模式，而关闭一个编辑的文件是一个 Vi编辑命令。创建一个文件通常是在启动 Vi编辑器时在 Shell

命令行上指定该创建的文件名。

要编辑一个文件，只需在 Shell 命令行上键入 vi 及要编辑的文件名。如果该文件不存在，那么系统将创建该文件。事实上，指定一个不存在的文件名就是告诉 Vi 编辑器去创建该文件。下面的命令启动 Vi 编辑器，并编辑 booklist 文件，如果该文件不存在，编辑器将创建该文件。

```
$ vi booklist
```

执行上述 vi 命令后，你将键入 Vi 编辑器的命令模式。每个键入都将成为一个 Vi 编辑命令。而屏幕成为编辑的文本文件的一个显示窗口，即文本被显示到显示屏上。首先是文本的第一屏显示在屏幕上，而光标停留在左上角的位置上。当编辑一个新创建的文件时没有字符显示在屏幕上，而只是在屏幕的左边有一列符号～，该符号表示屏幕此部分不是文件的内容。

必须记住，当你最初进入 Vi 编辑器时，编辑器处于命令模式下，要输入字符，必须进入输入模式。在命令模式下，进入按键 a 表示要往文本文件中添加（append）文本，于是，编辑器进入输入模式下。此时，你可以象操作打字机那样去操作键盘，并输入文本至文件。如果你键入 ENTER 键，你将在新的一行上开始文本编辑。编辑完文本之后，你可以按 ESC 键退出输入模式而进入命令模式下。

一旦完成上述操作，需要退出 Vi 编辑器，你可以键入两个大写的字母 ZZ，即按注 SHIFT 键的同时击打字母键 Z 键两次。该命令将首先保存编辑的文本，然后退出 Vi 编辑器并返回到 Linux 系统的 Shell 下。

当你开始编辑一个文件时，Vi 编辑器将首先把该文件读入到工作缓冲区。

对文件的任何修改与添加都是对文本副本（在文件缓冲区中）的改变，磁盘上的原文件仍然保持不变。只有执行保存命令之后，当前工作缓冲区的内容才被保存并覆盖原文件。完成每次编辑操作之后，用 ZZ 命令可以在退出之前保存文件，即用工作缓冲区的内容覆盖原文件。

但是，你可能想在你的编辑过程中多次保存一个文件。要多次保存文件而不想退出编辑器，你可以进入 Vi 编辑器的底行命令模式下，用 w 命令来完成文件的保存。这与其它字处理软件的磁盘命令基本相同。在 Vi 编辑器中，要保存文件，首先键入:（冒号键）进入底行命令模式，再键入 w 键并按 ENTER 键。同时，系统自动回到 Vi 编辑器的命令模式下。

初学者通常要范的一个错误是：在输入模式下，要保存编辑的文件时常常忘记键入 ESC 键回到命令模式下。:命令（冒号键）是一个编辑命令，它仅仅在命令模式下才能执行。如果你在输入模式下键入:键（冒号键），编辑器将把它认为是字符:添加到文本中去。因此，在输入模式下，如果你要保存文件，你必须首先键入 ESC 键退出输入模式，进入到命令模式。此时，命令:w 将保存编辑的文本。必须指出的是，带文件名参数的:w 命令类似于其它字处理软件的另存为（Save AS）命令，即当前编辑的文件将被保存为你刚才输入的文件名。

同时，当编辑一个你未曾命名的文件时，你也要使用:w 命令。当你启动 Vi 编辑器时没有键入文件名参数，编辑器将默认为你是在编辑一个未命名（unnamed）的文件。下面的例子中，用户在 Shell 下启动 Vi 编辑器，并且没有指定编辑的文件名，此时，用户在编辑一个未命名（unnamed）文件。

```
$ vi
```

你可以在未命名（unnamed）文件中输入文本，执行编辑操作与命令，但此时并没有创建实际的文件。如果你正在编辑一个未命名（unnamed）文件，Vi 编辑器禁止使用 ZZ 命令退出当前的编辑工作，因此，你必须首先创建该编辑的文件。你可以使用带文件名参数的:w 命令来创建该文件，操作如下：首先必须回到命令模式下，然后键入:（冒号）键，屏幕的底部将出现一行，再键入 w 键，一个空格及创建的文件名，最后键入 ENTER 键。

:w booklist

键入 ENTER 键后，你将返回到命令模式下，光标也将停留在光标先前所在的位置。创建的新文件的文件名也将被显示在屏幕的底部。此后，你可以使用 ZZ 命令退出当前的编辑工作。需要指出的是，如果你指定的文件已经存在，文件创建与保存将失败。出现这种情况，你只需重复上述存在，键入另外一个文件名即可。

在 Vi 编辑器中，你可以使用:q 命令退出编辑器。与 ZZ 命令不同，在执行:q 命令之前，编辑器不执行任何保存文件的操作。但该命令有一个重要限制：如果你在上一次保存之后作了任何修改，:q 命令将失败，即它不会退出编辑器。但是，你可以在:q 命令之后键入!（惊叹号）来忽略该限制。命令:q!将结束文本编辑并退出编辑器，同时，它不保存上一次保存之后的任何修改。

有许多 Vi 编辑命令是与底行命令模式下的编辑命令相等同的，例如 Vi 编辑命令 ZZ。你可以在底行命令模式下使用 wq 命令来完成与 Vi 编辑命令 ZZ 相同的操作，即:wq 命令将在退出编辑器之前先保存编辑的文件。因此:wq 命令与 ZZ 命令是完全等同的。

17.3 在 Vi 编辑器中管理编辑模式

保存、退出及向文件中增加文本三个命令涉及到了 Vi 编辑器的三种模式。当启动进入 Vi 编辑器时，编辑器处于命令模式；键入 a 命令将进入输入模式，你可以在这种模式下输入文本，此时，键入 ESC 键将从输入模式回到命令模式。要保存或退出编辑的文件，你必须用:（冒号）命令进入底行命令模式。底行命令模式下的 w 和 q 将保存并退出编辑的文本文件。键入底行命令最后再键入 ENTER 键将执行该命令并回到 Vi 编辑器命令模式下（退出 Vi 编辑器命令除外）。

管理 Vi 编辑器编辑模式的困难在于你必须知道你所处在模式。在 Vi 编辑器的一个编辑过程中，键盘不断地在输入模式与命令模式下相互改变。同时，这两种模式是排他性的，不能同时存在。在输入模式下，不能执行任何命令模式下的命令，而在命令模式下，不能进行任何输入。没有任何提示告诉你是处于输入模式下还是处于命令模式下。如果你认为你处于输入模式下，而实际上你处于命令模式下，有时这种错误是破坏性的。有时，你处于命令模式下，而你认为处于输入模式下，此时你输入的文本实际上是执行一系列的编辑命令，这些命令会对编辑文件产生一些意想不到的操作。

如果你不知道自己所处在的模式，你可以键入 ESC 键来间接地判断出你所处的模式。在输入模式下，如果你键入 ESC 命令将退出输入模式而进入命令模式下。但是，在命令模式下键入 ESC 命令，你会听到扬声器的鸣笛声。如果你

键入 ESC 命令后扬声器鸣笛，则表明你处于命令模式下。但如果没有任何事情发生，则表明你刚从输入模式下返回到命令模式下。

一个常见的错误是：你通常在输入模式下试图结束文本文件的编辑。实际上，你只能在命令模式下才能结束一次编辑工作。必须记住：在输入模式下键入 ZZ 命令只是简单地把两个 Z 字符输入到编辑的文本中。两个 Z 字符出现在屏幕上后，Vi 编辑器仍然处于输入模式下，并等待用户的进一步输入。要结束一次编辑工作，你必须键入 ESC 键，退出输入模式而回到命令模式，再键入 ZZ 命令。一旦进入命令模式，键盘上的键将成为编辑命令，而两个大写的 ZZ 编辑命令将结束本次编辑工作，并保存编辑的文本。

17.4Vi编辑命令：常用操作命令

本节将介绍 Vi编辑器的一系列编辑命令，包括基本的光标移动、文本输入、查询及编辑操作。Vi 编辑器实用程序设计的目的是用来编辑文本的，它包括对文本行的操作、字符的操作、句子的操作甚至段落的操作。本节将仅仅介绍对单个字符及行的操作。对编辑文件中文本的其它操作（如：单词及句子的操作）将在本章后半部分的高级操作与命令中作详细介绍。

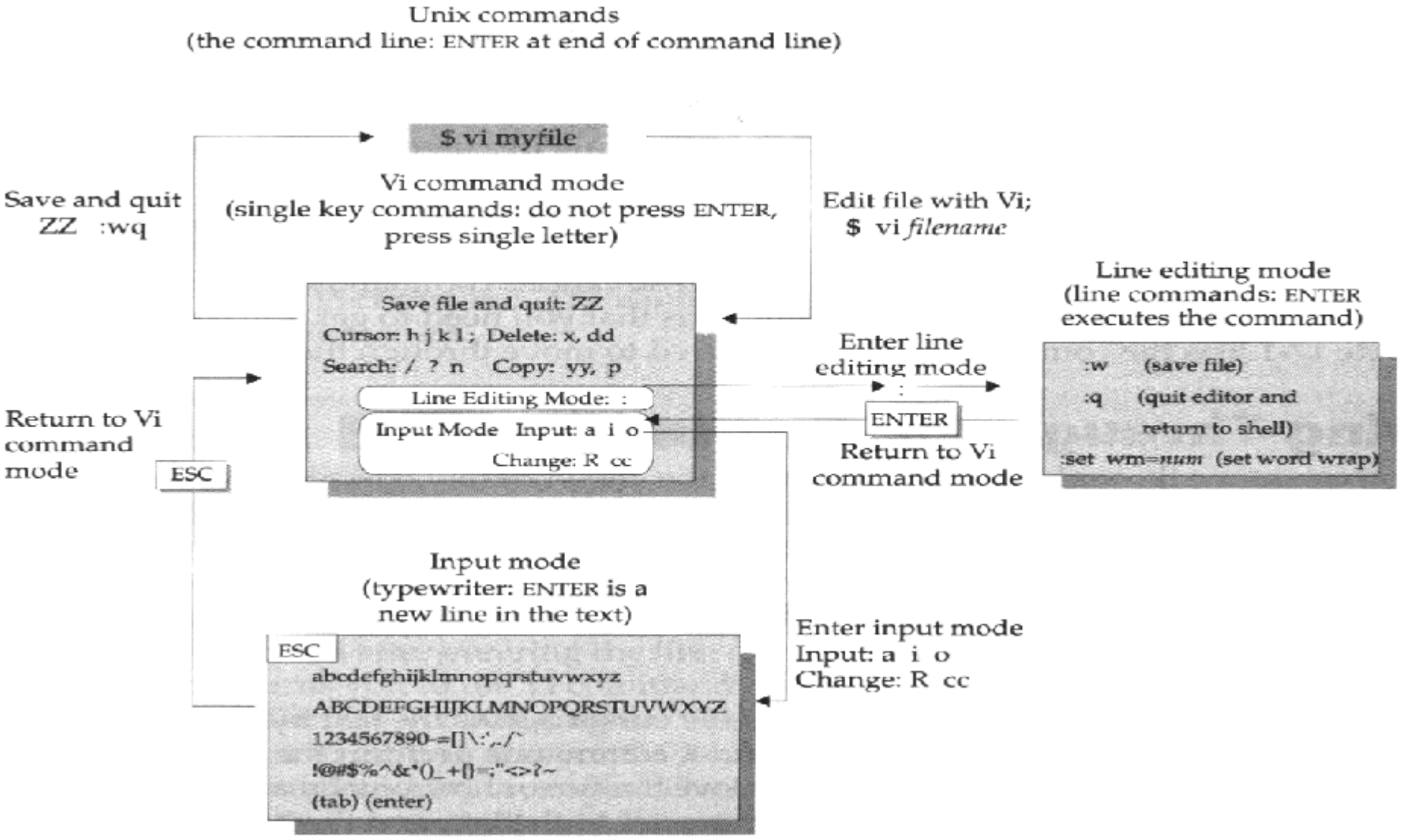


图 17-2Vi 编辑器常用命令要点小结

图 17-2 中列出了一组基本的 Vi 命令，这组命令对初学者开始学习 Vi 编辑

器时是非常重要的。表 17-1 列出了本节介绍的一些常见的命令。

表 17-1Vi编辑器常用命令

光标移动命令	作用、效果	命令	作用、效果
h	左移	x	删除一字符
l	右移	dd	删除一行
k	上移		
j	下移	改变	作用、效果
CTRL-F	下一屏	r	替换一字符
CTRL-B	前一屏	cc	改变一行
G	移动至指定行	R	覆盖改写字符
输入命令	作用、效果	移动命令	作用、效果
a	添加	p	插入删除/拷贝文本
I	插入	dd p	移动一行
O			
查询命令	作用、效果	拷贝命令	作用、效果
/	搜索模式	yy p	拷贝一行
?	搜索模式		
n	重复搜索		

17.4.1 在 Vi 编辑器编辑的文本中移动

你可以在 Vi 编辑器编辑的文本中移动，包括移动光标、滚屏及移动到指定的行等。Vi 编辑器光标的移动命令包括在屏幕上显示的文本里左、右、上、下移动。你也可以一次把文本向前、后整屏滚屏。有时你需移动到指定的行，你可以指定行号来完成这些操作。

光标的移动

在命令模式下，你可以用键盘上的移动命令键使光标在文本中来回移动。按键 h、j、k 及 l 命令执行基本的光标移动操作。其中，h 和 l 键用来在文本行上水平移动，且 h 键用来左移一个字符，l 键用来右移一个字符；j 及 k 键用来在文本行上上、下移动光标。j 键用来下移一行，k 键用来上移一行。需要指出的是，继续按光标命令键将重复光标的移动。例如始终按住 l 命令键将把光标移动至该行尾。

同时需要指出的是，为了方便地移动光标，这些键就放置在你右手各手指下面。在继续文本编辑的时候，你需要经常在文本行上或行与行之间移动光标。也有部分键盘使用一组光标方向键来代替 h、j、k 及 l 键，还有部分用 ENTER 键来代替 j 键来将光标下移一行。SPACEBAR 键（空格键）来代替 l 键去左移一个字符。

每个文本行从屏幕的左列开始，直至你键入 ENTER 键（回车键）来结束

一行。如果光标在该行的行尾，你键入 l 键来右移光标，扬声器将鸣笛而光标不再移动。如果光标在该行的第一个字符上，你试图键入 h 键来左移光标也将鸣笛。扬声器鸣笛表明光标已移动到文本行的边界上。

行尾不一定是屏幕窗口的右侧，而是任何你键入了 ENTER 键（回车键）的地方。当用 l 键在文本行上一直右移光标的时候，光标将停留在该文本行的行尾（即键入了回车键的地方）。

你只能在文件中真正存在字符（包括空格字符）的地方进行光标移动。在行尾及屏幕右边界之间的区域称为屏幕死区（dead space），它不是属于文件部分。如果光标放置在某行的行尾，而你将光标移动到下一行的时候，必须请注意：光标将自动移动到下一行的行尾。

屏幕此时好像你所编辑的文本文件的一个窗口，每次只是显示文件的一部分。你可以通过各种文本编辑命令来把文件在该窗口内前后移动。如果光标被放置在屏幕的首行或底行，你可以通过 j 及 k 键来逐行垂直滚动文本。j 键用来将光标在屏幕上下移一行，一旦光标达到屏幕的底行，按 j 键将使文本上滚一行，而先前屏幕顶行的文本将在屏幕上消失，新的文本将在底行上出现。当光标在屏幕的底行时，连续不断的键入 j 键将使文件内容不断向上滚屏（或者说屏幕窗口相对于文本不断下移）。类似的，键入 k 键也是如此，但文本或屏幕的移动刚好相反。

你也可以使用 CTRL-F 命令及 CTRL-B 命令来整屏移动文本，其中 CTRL-F 命令向前（即向文本尾部）整屏滚屏而 CTRL-B 命令将向后（即向文本首部）整屏滚屏。一旦移至文件尾部，你想移动至文件首部，可以通过不断键入 CTRL-B

命令来逐步整屏移动来实现。

行号：G

你可以使用 G 命令及行号来移动至特定的行。Vi 编辑器对编辑的文件中的每一行按顺序进行编号，G 命令（必须是大写的 G）将把光标移动到指定的行。你首先输入要移动的行号，然后键入 G 键，于是，光标将移动至指定的文本行。你也可以用 G 命令迅速地将光标移动至文件的尾部。如果在键入 G 键之前没有输入行号，那么，光标将直接移动至文件尾部。因此，对于大文件来说，G 命令是一种非常迅速、有效地定位至尾部的一种方法。

17.5 在 Vi 编辑器中修改文件：输入、删除及修改

在对文本文件进行编辑的时候经常涉及到文件某些方面的修改。例如，你可以在文件不同的地方添加更多的文本，可以任意删除已经存在的文本，也可以修改文件的任意部分。Vi 编辑器有一系列的控制命令，可以采取不同的方式用这些命令来输入字符、删除及修改文件任意的部分，如：字符、行、单词、句子及段落等。你也可以用 Vi 编辑器的编辑命令来取消（undo）你对编辑文本的任何改变，例如：你可以取消（undo）你的文本输入。取消（undo）命令为 u 命令，它将取消（undo）用户先前键入的一些 Vi 命令。当你修改了文件的时

候，必须记住该命令可以取消你对文件的修改。许多时候你会无意中对文件的内容进行了修改，这时，你可以非常方便地用 u 命令去取消这些修改。

输入

你已经知道，当启动 Vi 编辑器的时候，一个新的文件会被创建，而你此时处于 Vi 编辑器的命令模式下。要进行字符输入，你需要执行一个输入命令，例如 a 命令。该命令可以使你处于 Vi 编辑器的输入模式下。一旦你进入输入模式，键盘就类似于一部打字机，此时你键入的任何文本都将被输入到文件中去。

一旦进入输入模式，你可以输入任何字符及任意个你想输入的字符，直至你键入 ESC 命令。你也可以用空格键（SPACEBAR）来删除最近键入的字符。空格键的使用仅仅局限于输入模式下，它仅仅删除光标左边的字符。连续不断地键入空格键，你可以删除进入输入模式以后所键入的所有字符。

当你达到右边界的时候，许多 PC 式的文字处理软件将自动对输入的文字进行换行（软回车），但 Vi 编辑器不会。在 Vi 编辑器中，只有在输入模式下键入 ENTER 键（回车键）之后编辑器才会创建新的文本行。尽管当光标达到屏幕右边界的时候，Vi 编辑器不会自动插入新的行(尽管看起来好像增加了新的行)。如果文本行的长度超过了屏幕的宽度，文本行将自动绕行(wrapped around)，好像在屏幕上是两行一样。但编辑器把它们当作是同一行。如果需要，你可以设置 Vi 编辑器的选项来设置当编辑的文本达到指定的边界时自动插入新的行。这个边界称为文本换行边界（word-wrap margin），你可以在底行命令模式下键入命令:set wm=col来设置它，这里，col是右边界的列号（column）。

例如命令:set wm=70将设置文本换行边界为70列，当你输入的文本达到70列时，Vi编辑器将自动插入新的行。

有几个输入命令可以进入输入模式，而这每个命令进入输入模式后进行编辑时输入字符的位置与光标有关。最常见的输入命令是a命令（append）、i命令（insert）及o命令（open）。用a命令进入输入模式后，编辑器将在光标所在的字符后输入文本。用i命令进入输入模式后，编辑器将在光标所在的字符的前面输入文本。用o命令进入输入模式后，将在光标所在行下插入新的一行，且光标位于下一行的行首。

当你在文本行的行首或行尾添加字符时，你会发现i命令与a命令有很大的不同。你可以使用i命令在文本行行首添加字符，而使用a命令在行尾添加字符。要在文本行的行首添加文本，你必须把光标放置在该行的首字符上，用i命令将把插入的字符添加到该字符的前面。要在文本行的行尾添加文本，你必须把光标放置在该行的最后一个字符上，用a命令将把插入的字符添加到该字符的后面。请注意不要混淆这两个命令。如果光标被放置在文本行的最后一个字符上，此时你键入i命令，那么新增加的文本将被添加在最后一个字符之前。如果你想在文本行的首字符之前用a命令插入文本，类似的错误也会发生，即：如果光标被放置在文本行的首字符上，此时你键入a命令，那么新增加的文本将被添加在首字符之后。而不是在文本行的行首。

除了在指定行上增加文本之外，你也可以插入新的整行字符，此时，你需使用o命令。该命令可以认为是在两行之间插入新的文本行。插入的文本行不仅仅只局限于在新增加的行上输入文本，新插入的行只是进入输入模式进行文

本编辑的一个起始点。一旦进入输入模式下，你可以通过键入 ENTER 键来输入你希望的任何行文本。实际上，o 命令等同于：把光标放置在文本行的行尾，并键入 a 命令，然后再键入 ENTER 键，于是，a 命令使编辑器进入输入模式，而 ENTER 命令用来插入新的文本行。o 命令通常被用来在文件的末尾添加文本。把光标放置在文件的最后一行，键入 o 命令，编辑器将在文件的最后一行之后插入新的文本行。于是，你可以在输入模式下输入任何你想输入的文本。

删除

最简单的删除操作是单个字符及整行文本的删除。键入 x 键将删除单个字符。键入 x 键将删除光标所在位置上的任何字符。dd 命令将删除光标所在的行。

修改

你可以修改任何字符。首先，你可以用删除命令来删除该字符，然后，进入输入模式下输入新的字符。要修改整个单词，你必须重复地使用 x 命令来删除该单词，然后键入 i 或 a 命令进入输入模式下并键入新的单词。Vi 编辑器也有一组修改命令，用这些命令可以简化上述修改过程。这组命令可以自动地删除单词、文本行、句子甚至段落，然后进入输入模式下。这些命令都会在本章的高级命令里详细介绍。这里，我们只是用一些简单的操作来实现字符及文本行的修改。

cc 命令可以用来修改整行。首先，cc 命令将删除光标所在行上所有的文本，

然后进入输入模式下。此时，你可以输入新的字符，直至你键入 ESC 键回到命令模式下。dd 命令和 o 命令两命令的组合与 cc 命令的功能相同，其中，dd 命令用来删除一文本行，而 o 命令用来插入新的文本行。

你可以使用 r 命令来修改单个字符。r 命令删除光标所在位置上的任何字符，并等待用户输入新的字符，即光标位置上的字符将被你键入的任何字符所代替。r 命令与其它修改命令不一样，键入 r 命令后，编辑器不进入输入模式，在你键入替换的字符后，编辑器仍然处于命令模式下。

R 键是一个替换命令，键入该命令后进行输入时将覆盖光标位置所在的文本，直到你键入 ESC 键为止。键入 R 命令后，编辑器处于输入模式下，且当你在光标位置上每键入一个字符时，Vi 编辑器将同时删除光标位置上相应的字符。该命令似乎与其它文字处理软件中的覆盖模式类似，但实际上不是。与 cc 命令一样，在你进行任何文本编辑之前，编辑器实际上已退出命令模式而进入了输入模式下。请注意不要混淆 r 及 R 这两个命令。r 命令仅仅用来修改单个字符，而 R 命令将覆盖所有的字符，直至你进入 ESC 命令，退出输入模式为止。如果你想键入 r 命令时不小心键入了 R 命令，你可能无意中覆盖了原来的文本，此时，你可以键入 ESC 键回到命令模式下，并键入 u 命令来取消这次操作。

取消操作（undo）

在删除或修改文本的时候，你可能进行了意外地删除或修改。如果你希望取消这些意外操作，并回到原来的状态，可以使用 u 命令。u 命令可以使你恢

复到最近你所执行的编辑命令之前的状态。u 命令可以取消任何你对文件所作的修改，包括输入操作、删除命令及修改命令等。在学习 Vi 编辑器时，你可能要大量地使用该命令，因为你容易执行意外的操作或命令。如果意外地删除了一行，你可以键入 u 键来恢复删除的文本。如果你修改了一行，而又决定不想进行这一修改，你也可以使用 u 命令来取消这一修改。

重复因子

你可以在删除或修改命令之前键入一数字，以便一次进行多次修改或多行删除。此时，与 G 命令中的数字不一样，它们不代表行号，而是代表该命令的重复因子，即命令的重复次数。任何命令（G 命令除外）之前键入的数字表示该命令重复的次数。例如：3x 命令将删除三个字符而不是一个字符，2dd 将同时删除两行。

同时，请注意不要混淆数字的上述两种用法。例如，假设你想用 G 命令把光标移动到指定的行，但你在键入一个数字之后、键入 G 命令之前突然改变了主意，例如键入了 200 之后改变了主意，而用 x 命令代替了 G 命令，那么编辑器将删除 200 个字符。该重复因子也可以用在输入及修改命令之中。例如，如果你输入了 50 之后执行了 i 命令，而不是 G 命令，且输入了某些字符，那么，在键入 ESC 键停止输入并退出输入模式之时，Vi 编辑器将插入此次输入字符的 50 个拷贝到编辑的文件中去，也就是说，编辑器会立即重复 50 次你刚才的输入操作。如果在输入一数字后，你突然想改变了主意而去执行其它的编辑命令，你可以键入 ESC 键来取消刚才输入的数字。

尽管你非常小心，仍可能会出现错误。这时，你可以用 u 命令来取消上次执行的操作，从而更正由于上次的操作而导致的错误。一个重复的编辑命令会被看作是一个编辑操作，你可以键入一次 u 键来取消上述重复的操作。例如，你用 200x 命令重复删除了 200 个字符，键入 u 命令之后，删除 200 个字符这一操作将会立即取消。

行的拆分与合并

使用 Vi 编辑器对文本中行的拆分与合并过程要比 PC 机式的文字处理软件要复杂。对于 PC 机式的编辑器，你可以非常简单地键入 ENTER 键来对一行进行拆分，或者使用 BACKSPACE 键来合并行。在 Vi 编辑器中，对行的拆分需要键入一个输入命令。因此，首先要进入输入模式，此时你才能键入 ENTER 键。合并行时需要一个新的 Vi 编辑器中的 J 命令（必须是大写），而你不能使用 BACKSPACE 键来合并行。

要把一行拆分成两行，首先必须把光标放置在你所希望的拆分处，然后键入 i 或 a 命令（必须注意 i 命令将在光标所在的字符前插入字符，而 a 命令将在光标所在的字符后添加字符）。一旦进入输入模式，键入 ENTER 将即可。要回到命令模式下，只需键入 ESC 键。要合并两行，键入 J 命令即可，此时，光标所在行的下一行将被合并到光标所在的行的行尾上。在 Vi 编辑器里，你似乎从来没有键入 ENTER 键，实际上是因为 ENTER 键是不可见的。Vi 编辑器里的换行标识（不可见的）本身不会被任何删除命令所删除，但有些命令，如 dd 命令，可以把它与文本行上的其它字符一并删除。

17.5.1 在 Vi 编辑器中拷贝、移动及搜索文本

你可以把文件中的文本拷贝或移动至文件的另一个地方。尽管在 Vi 编辑器中可以拷贝或移动任意文本，包括单词及句子等，但本节只介绍文本行的移动与拷贝。你也可以在文件中搜索指定的单词与模式。在 Vi 编辑器中，你既可以向前也可以向后搜索，同时，你也可以重复搜索。

移动、拷贝文本行：dd、yy 及 p 命令

要移动文本，首先你必须删除你需要移动的文本。dd 命令将删除一行文本。在 dd 命令之前键入一数字 n 将删除从光标所在的行起的 n 行文本（包括光标所在的行）。这些删除的文本将被保存到一个临时缓冲区中。然后，移动光标至你将要插入的行上，最后键入 p 命令，编辑器将把刚才保存到临时缓冲区中的所有文本插入到光标所在的行之后。

你可以用 yy 命令来拷贝文本行。在 yy 命令之前键入一数字 n 将拷贝从光标所在的行起的 n 行文本（包括光标所在的行）。要拷贝这些文本行，首先把光标放置在要拷贝的文本行的首行上，然后键入要拷贝的文本行行数，再键入 yy 命令，该命令将把要拷贝的文本行保存到一个临时缓冲区中，最后把光标移动到要拷贝插入的行上键入 p 命令，这些被拷贝的文本行将被插入到光标所在的文本行之后。

搜索

在你进行文本编辑的时候，你可以需要搜索或定位特定的单词或单词的一部分。Vi 编辑器有向前或向后搜索指定模式的功能。斜扛/是 Vi 编辑器的模式搜索命令。键入斜扛/后将在屏幕的底部出现一行，同时一个斜扛会出现在底行的行首上，且光标被放置在斜扛之后。于是，你可以在光标位置键入你需要搜索的模式，键入 ENTER 键之后，编辑器将开始模式搜索。当执行斜扛命令的时候，编辑器将从光标所在的位置起向文件尾部搜索你键入的模式。斜扛命令是向前（forward）搜索命令，如果输入的模式被找到，光标将停留在找到的模式上。

问号?命令将向后（backward）搜索编辑的文本。键入问号?命令也将在屏幕的底部出现一行，再键入你需要搜索的模式，最后键入 ENTER 键，编辑器将从光标所在的位置起向文件首部搜索匹配的模式。

先键入斜扛/命令或问号?命令，再键入搜索模式，按 ENTER 键进行搜索之后，可以用 n 命令用来进行重复搜索。键入 n 命令将开始在文件中再进行一次模式的搜索。如果键入的是斜扛/命令，那么，将继续向文件的尾部方向向前（forward）进行的搜索；如果键入的是问号?命令，那么，将继续向文件的首部方向向后（backward）进行的搜索。

如果达到了文件的首部或者是尾部，光标将回到文件的尾部或首部继续搜索文件中匹配的模式。你可以重复键入 n 命令在文件中多次搜索指定的模式。

17.6 高级 Vi 编辑命令

到目前为止，我们已介绍了 Vi 编辑器的一些基本编辑命令，但是，Vi 编辑器实际上还有很多的命令。大部分小写字母命令都有与其对应的由大写字母组成的命令，它们执行相同的操作。例如，小写的命令 o 将在当前行与下一行之间插入新的行；与之相对应，大写的命令 O 将在当前行与上一行之间插入新的行。本节将详细介绍不同的屏幕光标移动命令及各种修改命令，例如：执行输入、修改及删除等操作。

你也可以根据各种搜索模式来搜索你要查询的文本，包括单词、句子及段落等。在前面的章节中，你只学习了怎样搜索单个字符及文本行。例如，l 命令把光标左移一个字符，而 j 命令将把光标下移一行。但是，你也可以用 w、)（右原括号）及}（右大括号）命令来在文本之间移动。w 命令将以单词为单位进行移动，右原括号将以句子为单位进行移动，而右大括号将以段落为单位进行移动。你也可以由这些 Vi 命令去限定另外一些 Vi 命令来执行文本操作。例如，你可以用 w 命令来限定修改命令 c，即用它们组合成新命令 cw，该命令用来修改一个单词；你可以用)命令来限定 d 命令，即用它们组合成新的命令 d)来删除一个句子。限定命令位于主命令之后。

17.6.1 高级光标移动命令

Vi 编辑器提供了许多命令，这些命令可以使用户在文件中任意移动。你可以在文本行上的以字符为单位在不同的字符之间进行移动，也可以以单词、句子甚至段落为单位进行移动。此外，你还可以在屏幕上的不同文本行之间进行移动，包括屏幕顶部、屏幕底部及屏幕中部等。你甚至可以采用搜索、定位的方法来在不同的文本行之间移动，例如行号、搜索模式或标识等。例如：你可以用特定的代码在某行作一标识，以后，你可以通过这个特定标识返回到该行。

所有这些移动命令都可以认为是对位置的搜索与定位。例如行号及搜索模式可以用来定位特定的文本行。一个单词或句子定位命令可以用来搜索特定的单词或句子。你还可以使用定位命令与修改命令的组合来修改字符、文本、句子、段落等。例如，用单词搜索定位命令及修改命令可以用来修改特定的单词；用模式搜索命令与行删除命令来删除包含匹配模式的文本行；用屏幕底行定位命令与行拷贝命令来把光标所在的文本行拷贝至当前屏幕上的最后一行上。请记住本节介绍的有关光标移动的高级命令。在下面的章节中，你可以使用本节介绍的光标定位命令及编辑修改命令来编辑你要修改的文本。

Vi 通过行号来组织编辑的文本，因此你可以通过行号、搜索模式及特定标识来搜索、定位。正如本章前几节中指出的那样，你可以使用 G 命令及其行号来定位。在 G 命令及其前面输入行号，可以把光标移动到指定的文本行。如果没有指定行号，缺省状态下光标将移动到文件的尾部。你也可以用 G 命令与修改或删除命令的来组合成新的命令。例如，dG 命令可以删除从当前行至文件结

尾的所有文本；cG 命令在删除从当前行至文件结尾之间所有文本的同时，Vi 编辑器将进入输入模式下。

模式搜索命令（由标识符斜扛/开始）将在文本中进行模式匹配，并把光标定位于与搜索模式匹配的文本行上。键入斜扛/后将在屏幕的底部出现一行，于是，你可以在底行上键入你需要搜索的模式，键入 ENTER 键之后，编辑器将开始搜索。如果搜索到该模式，光标将出现在搜索到的模式上。问号?命令也用来进行模式搜索，但它向文件的首部方向进行搜索，请记住：问号?命令将从光标所在的位置起向文件首部方向搜索键入的模式，而斜扛/命令将从光标所在的位置起向文件尾部方向搜索键入的模式。

n 命令重复先前的搜索，或者向前（forward）搜索，或者向后（backward）搜索。大写 N 命令也用来重复先前的搜索，但它与 n 命令的搜索方向相反。如果用户已使用了斜扛/命令进行向前（forward）搜索，那么，键入 N 命令将进行向后（backward）搜索；如果用户已使用了问号?命令进行向后（backward）搜索，那么，进入 N 命令将进行向前（forward）搜索。

搜索到的模式可以认为是一个文本的边界。删除命令和修改命令可以使用模式去搜索文本行。在删除命令后键入搜索模式将删除当前行及与搜索模式匹配的文本行之间（包括这两行）的所有文本。例如：d/hello/命令将删除当前行与包含字符串 hello 的文本行之间（包括这两行）的所有行，c/hello/命令将修改这些行。

Vi 编辑器有两个特殊的字符，这两个特殊字符用来在模式中代表单词的起始或结束。特殊字符\<代表单词的开始，而特殊字符\>代表单词的结束。例如，

模式/\<make\>/将仅仅搜索包含单词 make 的文本行，而模式/\<make/将搜索所有包含由 make 打头的单词的文本行，如包含单词 makeup 的文本行等。

%命令将搜索、定位相应的开始或结束圆括号（、）、括弧［、］及大括号{、}等。例如，如果光标位于起始圆括号（ 上，那么，键入%命令将向前（forward）搜索最近的一个回括号），无论该回括号位于文件的何处。如果光标位于回圆括号） 上，那么，键入%命令将向后（backward）搜索最近的一个起始圆括号（。同理，对于括弧及大括号也是如此。在进行程序设计的源代码编辑过程中，该命令是非常有用的，因为在源代码中常有大量的圆括号、括弧及大括号。

你可以用标识命令 m 标识某行，并可以在其它命令中用这一标识去搜索、定位该行。如果在 m 命令后键入一个字母，编辑器将会用该字母去标识本行。在标识字母之前键入单引号’，编辑器将搜索并定位于用该标识字母标识的文本行上。你可以在任何命令中用标识字母去代替行号。例如，mb 命令将用字母 b 去标识一行，命令’bG 将把光标移动至用字母 b 标识的行。单引号加标识字母本身也将把光标移动至该标识行，例如’b 命令。你也可以用标识字母去搜索标识行，并对标识行进行修改或删除，例如’bd 命令将删除当前行与用字母 b 标识的行之间的所有文本（包括标识行及当前行）。

单引号也用来在文件中搜索、定位先前光标所在的位置。你可以用两个单引号’’去回到光标先前所在的位置。从某种意义上来说，编辑器是用第二个单引号来标识光标先前所在的位置的。多次重复地键入两个单引号将会使光标在当前位置与先前位置之间来回移动。

17.7 底行编辑命令

Vi 编辑器的底行编辑命令与第 14 章中介绍并列出的 Sed 编辑器的命令基本相同。大多数编辑任务都可以用 Vi 命令简单方便地完成，但是，如果想用行号去搜索、定位文本，你需用底行编辑命令。例如，如果你想删除第 9 至 17 之间的文本行。在 Vi 命令模式下，你必须首先定位于第九行，然后使用合适的命令去删除这九行文本。但是，在底行命令模式下，你只需在删除命令 d 中指定要删除的行号范围就可以完成删除，例如，:9,17 d。底行命令模式下的行定位能力使你在进行全局操作时非常有用。你也可以用 1, $来指定整个文件中的文本行，其中字符$用来指定文件的最后一行。

前面已经指出，冒号:是一个编辑命令，它用来在屏幕的底行创建新的一行，并允许你在该行上键入底行编辑命令。键入底行编辑命令并执行该命令之后，光标将返回到先前所在的位置上，然后你可以继续进行你的编辑工作。Vi 编辑器的底行命令模式通常用来进行一些全局操作。在 Vi编辑器中，通常键入冒号:命令使编辑器进入底行命令模式下。

你可以通过搜索文件中所有的文本并对每一个与模式匹配的字符串进行替换操作来实现全局替换。字符$代表文件中最后一行，而 1,$将代表文件中所有的行。替换命令将用要替换的模式来代替一个与搜索模式相匹配的字符串，加入 g 以后，将对所有与搜索模式匹配的文本进行替换，即在替换命令中指定所有文本行将执行全局替换操作。该命令的使用格式如下：

: 1, $ s/psttern/text/g

下面的命令将把文件中所有搜索到的字符串 milk 用字符串 yogurt 代替。

: 1, $ /s/milk/yogurt/g

与 Sed 编辑器一样，所有的特殊字符在 Vi 编辑器底行命令模式下仍然起作用。在下面的例子中，每一行的第一个单词都将被用字符-代替。当在模式中使用特殊字符^的时候，它代表某行的首字符。本例中，括号内代表一系列可能的字符，[a-z] 表示任何匹配的小写字母，而 [a-z] *代表任何字母组成的字符串，其后是一个空格。该搜索模式表示文本行上第一个单词，该单词可以是任意一组字母组合成的，而替换命令中的替换文本是由字符-及空格组成的文本。

: 1, $ /s/^ [a-z] * /- /

17.8 Vi 编辑器中的选项；set 及 .exrc

Vi 编辑器有一组选项，这些选项用配置你的 Vi 编辑器。你可以用 set 命令来设置编辑选项。你也可以在 Vi 编辑器中的底行命令模式下设置选项。set 命令后加入选项名可以打开该选项，如果在选项名的前面加入字符串 no 将关闭该选项。例如，set number 命令将设置行选项，用来在屏幕上显示文本行的行号，而 set nonumber 将关闭行选项，即在屏幕上不显示文本行的行号。不带参数的 set 命令本身将列出用户已设置的所有选项。如果某个选项已经设置，在 set 命令后键入选项的名，系统将显示该选项的值。命令 set all 将显示所有设选项的

设置。

17.8.1Vi编辑器初始化文件.exrc

有时你希望在编辑所有的文件时都设置了特定的选项。与每次都需对每个你所编辑的文件手动设置选项不同，你可以用Shell变量EXINIT或初始化文件来自动设置你的编辑器选项。如果想用EXINIT变量来自动设置选项，你必须用set命令来设置你想设置的选项，且需把它们用单引号括起来，并赋值给变量EXINIT。任何时候启动Vi编辑器，保存在变量EXINIT中的set命令都被自动执行。在下面的例子中，用户给EXINIT变量赋值，该变量把编辑器设置成nu（带行号）及ic（搜索时忽略大小写）。

$ EXINIT='set nu ic '

尽管你可以在Shell命令下把set命令设置的选项赋值给变量EXINIT，但更多的是可以在登陆或Shell初始化文件中给EXINIT变量赋值。于是，你任何时候登陆，EXINIT变量都会被自动赋值。看下面的例子：在用户初始化文件.bash_profile中，被单引号括起来的set命令被赋值给变量EXINIT。

.bash_profile文件清单：

EXINIT=' set nu ic '

用这种方法设置的选项会在你每个编辑的文本中都起作用。但是，有时你希望这些指定的选项只对某部分特定的文件起作用，例如， 你只希望在编辑C源程序时显示行号，而仅在你编辑的普通文本文件中自动换行，这时，你可以

用编辑器初始化文件.exrc 对特定类型的文件定制特定的选项。.exrc 文件包含一些设置编辑器选项的命令。当 Vi 编辑器启动时，Shell 首先搜索当前的工作目录，如果该文件存在，将执行运行该文件。通常我们用 set 命令来为编辑器设置编辑选项。在下面的例子中，.exrc 文件包含两条 set 命令，用来设置 nu（带行号）及 ic（搜索时忽略大小写）选项。

.exrc 文件清单：

```
set nu
set ic
```

如果在当前工作目录下没有搜索到.exrc 文件，Shell 将会在用户主目录下搜索.exrc 文件。任何目录下都可以有单独的.exrc 文件，这样，你可以根据目录下文件的类型来定制你的 Vi 编辑器选项。例如，在存放程序源代码文件的目录下进行源代码编辑时，如果你希望显示代码的行号，可以在该目录下.的 exrc 文件中用 set 命令设置 number 选项。当在该目录下启动 Vi 编辑器时，该文件中的命令将被执行，从而自动设置 number 选项。在另外一个目录下进行文本编辑时，如果你希望自动换行，可以在该目录下.的 exrc 文件中用 set 命令设置 wordwrap 选项。当在该目录下启动 Vi 编辑器时，编辑的文本将自动换行。

有一些选项设置可以用来控制搜索操作。如果你设置 ignorecase 选项，在搜索过程中进行模式匹配的时候将忽略大小写。例如，执行搜索命令/There 时，字符串 there 及 There 都将被搜索到。你可以把 ignorecase 选项简写成 ic，命令 set ignorecase 及 set ic 将把忽略大小写选项打开，而 set unignorecase 及 set noic 命令将关闭大小写选项。

有几个选项可以使你用来控制文本的显示方式。你可以在文本行上显示行号、显示文本中的非打印字符、控制显示在屏幕上的文本行数及文本中 TAB 键的空格数等。你已经知道，使用 set number 命令可以使编辑器显示行号（也可以简写为 set nu），使用 set nonumber 命令来关闭显示行号选项。

你可以用 tabstop 选项来设置更改键入制表键（TAB 键）时制表键中包含的空格数。命令 set tabstop=5 用来把制表键（TAB 键）中的空格键设置为五个空格。tabstop 选项可以简写为 ts。set ts=3 将把 TAB 键中的空格键设置为三个空格。tabstop 选项并不真正改变文本中的空格数，只是改变文本在屏幕上的显示方式。尽管在编辑器编辑的过程中，上述 tabstop 选项被设成 5，但当打印出文本时，TAB 键仍然代表标准的八个字符。

有几个选项影响你输入文本时的方式。当你在输入模式下且在进行字符输入的时候，这些选项会起作用。在输入模式下，你可以在指定的边界上开始新的一行，也可以进行文本自动缩进。当你输入回括号）时，编辑器还可以自动检查你是否输入了起始括号（。在进行字符输入时，你还可以设置按 tab 键时输入的空格数。

wrapmargin 选项（可以缩写为 wm）用来决定文本的右边界。设置了 wrapmargin 选项后进行文本编辑的时候，如果输入的文本到达边界，编辑器会自动回车，插入新的文本行，并在新的文本行上继续进行文本的输入。命令 set wrapmargin=30 将把输入边界定为 30 个字符，当你在文本行上输入第 30 个字符的时候，当前文本行下会自动插入新的文本行，并在新插入的文本行上继续将要的输入。set wm=0 命令将取消右边界的设置。如果右边界没有设置，新的

文本行不会自动插入到文件中去，你必须键入 ENTER 键或在原文本行上继续将要的输入。

autoindent 选项（可缩写为 ai）可实现文本行的自动缩进。当你处于输入模式下，且正进行字符输入的时候，自动缩进选项会起作用。当你打开自动缩进选项时，且在行首键入了若干 TAB 键，那么，下一行将自动缩进相同的 TAB 键数。如果你在下一行的缩进处继续键入 TAB 键，该行的下一行将缩进与上一行相同的 TAB 键数（包括新键入的 TAB 键）。尽管自动缩进选项允许你增加缩进量（例如，在原缩进的基础上继续键入 TAB 键），但有时你需要在某些行上减少缩进量,此时,你只需键入 CTRL-D 键来取消一次缩进。键入一次 CTRL-D 将使光标左移一个 TAB 键，键入第二次 CTRL-D 将使继续使光标左移一个 TAB 键，依次类推。你可以使用 set noai命令来禁止编辑器缩进。

17.9Vi编辑器小结

Vi 编辑器是一个文本编辑器，用来编辑文本文件。Vi 编辑器有三种编辑模式：命令模式、输入模式及底行命令模式。在不同模式下，键盘的特性将改变。在命令模式下，每个键都是一个编辑命令；在输入模式下，键盘类似于一台打字机，每个键用于向文件中输入文本；在底行命令模式下，屏幕的底行将出现一行，你可以在该行上输入行编辑命令。

在 Shell 命令行上键入 Vi 命令，同时指定编辑的文件名就可以启动 Vi 编辑

器，并开始该文件的编辑。你可以用 w 命令保存文件，用 q!命令退出文件的编辑。这些命令必须处于底行命令模式下。键入:命令可以进入底行命令模式。ZZ 命令将保存编辑的文件、结束本次编辑并退出 Vi 编辑器。

当使用 Vi 编辑器的时候，你会在输入模式与命令模式之间不断地转换。在命令模式下，键入 a 命令及 i 命令可以直接进入输入模式；修改命令 cc 将首先删除文本，然后进入输入模式。尽管有很多方法进入输入模式，但只有一种方式退出该模式，即键入 ESC 键。一旦进入输入模式，键入 ESC 命令将退出输入模式。

Vi 是一个非常复杂的编辑器，它有大量的命令，通过这些命令用户可以操作、控制这一全屏幕编辑器。你可以以单词、句子及段落为单位进行光标移动。大多数小写字母命令都有与之相对应的大写字母命令，例如：命令 i 及命令 I 都用来在文本行上输入字符。本章的前半部分重点介绍了一组核心命令，这部分命令主要用来实现基本的编辑操作，而本章的后半部分集中介绍 Vi 编辑器的一些高级命令与高级特性。Vi 编辑器中的编辑命令可以编辑处理字符、文本行、单词、句子及段落等。这些修改、拷贝及删除等操作都可以用来操作文本（见表 17-2 至表 17-4）。你可以移动、拷贝及取消文本操作，也可以根据指定的模式向前或向后搜索文本。用来删除字符的命令是 x 命令，而用来删除文本行的命令是 dd 命令。有时无意中删除或修改了文件中的文本，你可以使用 u 命令取消向前的编辑操作。如果你意外地删除或修改了文本，你可以键入 u 键来取消这一意外操作。

Vi 编辑器还有大量的选项，这些选项可以用 set 命令来设置。这些选项可

以用来设置 Vi 编辑器的一些特征，如：是否显示行号，是否自动回车等。你可以在 Vi 编辑器中手动设置这些选项，也可以在登陆或 Shell 初始化文件中，给 EXINIT 变量赋值（其值为单引号括起来的 set 命令）。当你启动 Vi 编辑器的时候，这些指定的选项会被自动设置。你也可以创建编辑初始化文件.exrc，并把 set 命令放置到.exrc 文件中。任何时候你启动 Vi 编辑器，当前目录下.exrc 文件中的命令将被执行。

表 17-2Vi编辑器命令

按键	光标移动
h	左移光标一个字符
l	右移光标一个字符
k	光标上移一行
j	光标下移一行
w	光标右移（forward）一个单词
W	光标右移（forward）一个空格定界词
b	光标左移（backward）一个单词
B	光标左移（backward）一个空格定界词
e	光标移动至下一单词词尾
E	光标移动至下一空格定界词词尾
o	光标移动至行首
$	光标移动至行尾

续表

ENTER	光标移动至下一行行尾
-	光标移动至上一行行尾
(	光标移动至句首
)	光标移动至句尾
{	光标移动至段首
}	光标移动至段尾
CTRL-F	向下滚屏，并显示下一屏文本
CTRL-B	向上滚屏，并显示上一屏文本
CTRL-D	向下滚半屏
CTRL-U	向上滚半屏
G	移动光标至文本最后一行
NumG	移动光标至指定一行，例如，45G 将把光标移动至第 45
H	移动光标至屏幕顶部
M	移动光标至屏幕中间
L	移动光标至屏幕底部
“	移动光标至先前光标所在的位置
mmark	移动光标至标识文本处，标识（mark）可以是字母表中任何字符
’mark	移动至用 mark 标识的文本行

续表

输入	所有输入命令将使 Vi 编辑器进入输入模式，键入 ESC 命令退出输入模式
a	在光标后插入文本
A	在行尾插入文本
i	在光标前插入文本
I	在文本行的第一个非空白处字符前插入文本
o	在当前行之下插入文本行
O	在当前行之上插入文本行
删除	
x	删除光标处的一字符
X	删除光标处前一字符
D w	删除至单词的末尾
d b	删除至单词的开始
d W	删除空格定界单词
d B	删除至空格定界单词的开始
d d	删除光标所在行
D	删除到行尾
d 0	删除到行的开始
d	删除其后指定的文本
d)	删除至句的末尾

续表

d}	删除至段的末尾
dG	删除到工作缓冲区的末尾
dm	删除至标识所有文本（该命令后跟一标识）
dL	删除到屏幕最后一行
dH	删除到屏幕第一行
J	把光标下一行合并至当前行行尾，实际上是删除当前行上的分行字符（该字符不可见）
修改	除非重键入命令 r，所有修改命令都使用户在删除文本后进入输入模式
s	删除光标所在的字符，并使用户进入输入模式
cw	删除光标所在的单词，并使用户进入输入模式
cb	修改至单词词首
cW	修改空格定界单词
cB	修改至空格定界单词词首
cc	删除光标所在的字符，并使用户进入输入模式
c0	从光标处修改至行首（即第一个非空格字符）
C	修改至行尾，并使用户进入输入模式
c	修改其后指定的文本
c)	修改至句子结尾
c}	修改至段落结尾

续表

cG	修改至文件结尾
cm	修改至标识的所有文本（该命令后跟一标识）
cL	修改至屏幕底部
cH	修改至屏幕顶部
r	替换光标处的字符；键入r命令后，用户键入替换字符，但不必进入输入模式就能完成替换，用户仍然处于命令模式下
R	首先进入输入模式，然后用键入的字符覆盖原来的字符。似乎是处于命令模式下，而实际上处于输入模式下
移动	在移动文本的同时首先删除文本，然后把光标移动到想要插入的地方键入p命令（当该文本被删除时，它会被自动保存在特殊缓冲区之中）
P	插入删除或拷贝的字符至光标所在的行或字符之后
P	插入删除或拷贝的字符至光标所在的行或字符之前
dw P	删除一单词，移动光标至你指定的地方，键入P命令之后，被删除的单词将被插入至光标所在单词的与下一单词之间
dw p	删除一单词，移动光标至你指定的地方，键入p命令之后，被删除的单词将被插入至光标所在单词的与前一单词之间

续表

dd p	删除一文本行，移动光标至你指定的地方，键入 p 命令之后，被删除的文本行将被插入至光标所在的行与下一行之间
d p	首先删除其后指定的文本，然后把它移动光标指定的地方（键入 p 或 P）
d) p	移动至句子结尾
d} p	移动至段尾
dG p	移动至文件结尾
dm p	移动至标识处的所有文本（该命令后跟一标识）
dL p	移动至屏幕底部
dH p	移动至屏幕顶部
拷贝	拷贝命令意味着该命令与 p 命令最后在一起使用。在拷贝文本之前，用户把光标移动至需要插入文本的地方，然后键入 p 命令，于是被拷贝的文本将被插入到光标所在的字符或文本行之后
yw	拷贝光标位置处的单词，然后把单词插入到光标所在的位置上（键入 p 命令后将把单词插入到光标所在的单词之后）
yb	从光标位置处拷贝字符至单词词首
yW	拷贝光标位置处空格定界单词

续表

yB	从光标位置处拷贝字符至空格定界单词词首
yy 或 Y	说明：拷贝光标所在的文本行，然后把拷贝行拷贝至光标指定的行上（键入 p 命令将把拷贝行拷贝至光标所在的行之后）
y	拷贝其后指定的文本
y)	拷贝至句子结尾
y}	拷贝至段落结尾
yG	拷贝至文件结尾
ym	拷贝至标识处所有的文本（该命令后跟一标识）
yL	拷贝至屏幕底部
yH	拷贝至屏幕顶部
搜索	有两种搜索命令。执行搜索命令时将在屏幕的底行开辟一行，然后用户在该行上键入搜索的模式，按 ENTER 键后执行搜索（回车键）
/pattern	向前（forward）搜索文本中的模式
?pattern	向后（backward）搜索文本中的模式
n	重复先前的搜索（可以向前也可以向后）
N	与先前相反的方向搜索文本中的模式
/	向前（forward）重复先前的搜索
?	向后（backward）重复先前的搜索

续表

缓冲区（Buffers）	有 9 个数字缓冲区，有 26 个有名缓冲区。有名缓冲区由字母表中 a-z 的 26 个小写字母来标识。你可以用双引号来引用指定的缓冲区
“ buf-letter	引用一个指定的有名缓冲区，如 a、b、c 等
“ num	引用一个指定的数字缓冲区，该数字在 1-9 之间

表 17-3Vi 行编辑命令

文件操作		功能
W	Write	保存文件
r filename	Read	插入文件
q	Quit	退出编辑器
删除、移动、拷贝		
d	Delete	删除一行或多行
mNum	Move	移动一行或多行文本（首先删除一行或多行，然后在指定的 Num 行之后插入要移动的文本行，也就是刚才删除的文本）

续表

coNum	Copy	拷贝一行或多行来文本（首先拷贝它们，然后在指定的 Num 行之后插入要拷贝的文本行行）
行定位	**说明**	
Num	一行	用行号来定位一行
Num,Num	两行	定位用逗号,隔开的两行文本
Num-Num	一系列的行	定位用符号-隔开的多行文本
-Num	行偏移	定位从当前行向上偏移 Num 的文本行
+Num	行偏移	定位从当前行向下偏移 Num 的文本行
$	文本最后一行	符号$定位至文件最后一行

续表

/Pattern/	模式定位	用模式 pattern 定位一行（符号/表示向前（forward）搜索定位一行）
?Pattern?	模式定位	用模式 pattern 定位一行（符号?表示向后（backward）搜索定位一行）
g/Pattern/	模式定位（全文件）	一系列的行被搜索定位（所有包含指定模式的行都将被搜索到）
特殊字符	**说明**	
.	字符通配符	与任何可能的字符都匹配的字符
*	任意字符或字符串	与任意字符或字符串都匹配的字符
[]	字符集	与字符集中指定的任意字符相匹配的字符
^	行起始符	定位于行首
$	行结束符	定位于行尾

续表

/<	单词开始	定位于单词词首
>/	单词结束	定位于单词词尾

替换命令	说明
s/pattern/replacement/	用 replacement 替换当前行上的一个 pattern
s/pattern/replacemant/g	用 replacement 替换当前行上的所有 pattern
Num-Num s/pattern/replacement/	用 replacement 替换指定行之间的 pattern
1,$ s/pattern/replacement/g	用 replacement 替换文件中所有的 pattern

表 17-4Vi 搜索、显示、输入选项

搜索选项	缩写	缺省值	说明
ignorecase	ic	noic	搜索是忽略大小写
magic		magic	使特殊字符有效
wrapscan	ws	nows	搜索时可绕回至文件首

文本显示选项		
缩写	缺省值	说明

续表

number	nu	nonm	显示行号
list		nolist	用^I 及$标识显示出新的行
window		window-23	设置屏幕上显示的文本行行数
tabstop	ts	Ts-8	设置 tab 键显示的空格数
输入选项	**缩写**	**缺省值**	**说明**
wrapmargin	wm	wm-0	在进行文本输入、到达文本右边界的时候编辑器自动换行
autoindent	ai	noai	自动缩进，按 CTRL-D 返回至上一缩进处
shiftwidth	sw	sw-8	移动宽度
showmatch	sm	nosm	显示初始括号（、{、[及结束括号）、}、]

续表

beautify	bf	nobt	禁止输入控制字符

第 18 章 Emacs 编辑器

你可以象使用其它标准的文字处理器那样使用、操作 Emacs 编辑器。键盘上普通的按键用来输入字符，而编辑器的操作命令是通过键盘上的一些特殊的按键来实现的，例如：控制键 CTRL、ALT 键等。与 Vi 及 Ed 编辑器不同，Emacs 编辑器没有特定的输入模式与命令模式之分。如果正在输入文本，必要时，你也可以执行编辑命令，例如用 CTRL 键来移动光标、保存文件等。

这样组织 Emacs 编辑器的目的在于使该编辑器易于使用。但 Emacs 编辑器绝非一个简单的编辑器，因为该编辑器是一个复杂且非常灵活的编辑器，它有好几百个编辑命令。Emacs 编辑器也有一些特殊的特性，例如：多窗口特性。你可以在编辑文本时同时显示两个窗口。你也可以同时打开并对多个文件进行编辑与操作，并在屏幕上与之对应的编辑窗口内显示每个文本。

Linux 的 GNU 版本是由 Richard Stallman 开发的，你可以在你的 Caldera 光盘上找到 Emacs 编辑器。在大多数 Linux 系统上，该编辑器是一个标准的实用程序。但是，在安装 Linux 系统时，如果你选择快速安装，那么 Emacs 编辑器是不会安装到你的系统中去的。此时，如果你要使用 Emacs 编辑器，首先必须使用 glint 来安装 Emacs 编辑器软件包。必须注意：你必须以超级用户登陆，

并在光盘驱动器上放置了你的 Caldera 光盘。

Emacs 编辑器通过巧妙地控制、操作工作缓冲区来实现其强大、灵活的功能。Emacs 编辑器可以被认为是面向缓冲区(buffer_oriented)的编辑器。在任何编辑器中编辑文本时，该文件将首先被拷贝到工作缓冲区中，而所有的编辑操作都在工作缓冲区进行。许多编辑器在编辑文件时仅开辟一个工作缓冲区，因此，仅能打开一个文件，而 Emacs 编辑器同时可以开辟并管理多个工作缓冲区，因此允许你同时对多个文件进行编辑操作。你还可以用编辑缓冲区来保存、删除或拷贝文本，你甚至可以开辟你自己的缓冲区，并在这些缓冲区中保存文本，必要时可以把这些缓冲区内的文本保存到文件中去。

18.1 用 Emacs 编辑器创建文件

你在 Shell 命令下的 emacs 来启动 Emacs 编辑器。你可以在 emacs 命令后输入你想编辑的文件名，如果该文件名不存在，该文件将被创建。在下面的例子中，用户准备用 Emacs 编辑器来编辑名为 mydata 的文件。

```
$ emacs mydata
```

与 Vi 编辑器一样，Emacs 编辑器是一个全屏幕编辑器。当编辑的文件是新创建的文件时，除屏幕的底部两行外，屏幕是空白的，光标将位于屏幕的左上角。屏幕底行被称为显示域（echo area），它在 Emacs 编辑器中起着命令行的作用，它通常用来显示 Emacs 编辑器提示信息。该行之上是模式行（mode

line），它用来显示正在编辑的文本的状态信息，该行在屏幕上通常是反色（与屏幕）并高亮显示的。图 18-1 是 Emacs 编辑器在进行文本编辑时的不同显示域。

你可以在任何时候输入文本，因为你永远处于输入模式下一样。一些编辑命令，例如移动命令，通过 CTRL 键来实现。例如，你想右移光标，你可以键入 CTRL-f 命令；左移光标，可以键入 CTRL-b 命令；上移一行，可以使用 CTRL-p 命令；下移一行，可以使用 CTRL-n 命令。

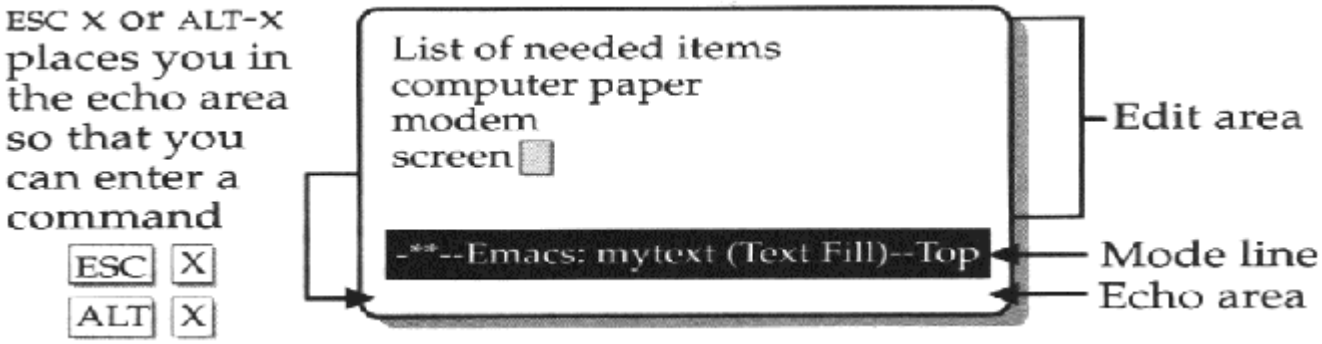


图 18-1Emacs 编辑器的编辑屏幕：编辑域、模式行及显示域

你可以在任何时候键入 CTRL-x CTRL-s 命令来保存你编辑的文件。许多 Emacs 编辑命令是由多个 CTRL 键与其它键的组合组成的命令。例如命令系列 CTRL-x CTRL-c 用来退出 Emacs 编辑器。如果你已经完成了一个文件的编辑，在用 CTRL-x CTRL-c 系列命令退出 Emacs 编辑器之前，你必须首先用命令系列 CTRL-x CTRL-s 来保存编辑的文件。

18.2Meta-keys、行命令及编辑模式

Emacs 编辑器的编辑操作与许多普通的文字处理器类似。Emacs 编辑器仅有一种编辑模式：输入模式。如果你键入任何字符键，键入的字符将被输入至编辑的文件中去，所有的字符键都被用来输入字符，而不是用来输入命令。与 CTRL 键及 ALT 等键相反，一个字符键可以被认为是任何你可以直接键入到文件中的字符。我们可以这么认为：字符键是可以在打字机上打印出的字符。

Emacs 编辑器编辑命令是 CTRL 键及 meta-key 键的组合。在本书介绍的 Emacs 编辑器版本中，一个 meta-key 可以是一个 ALT 键，或者是一个 ESC 键，在其它的系统中，可能是其它键。ALT 键命令系列与 CTRL 键命令系列的操作基本相同，即首先一直按住 ALT 键，再键入相应的按键，最后同时放开这两个键。但 ESC 键命令系列稍微有点不同：首先键入 ESC 键，然后放开该键，再键入相应的按键。ESC 键的使用频率要远比 ALT 键的使用频率要高，因为很多早期的键盘根本没有 ALT 键。因此，本章主要介绍 ESC 键等这些 meta-key 键，同时必须记住，这些 meta-key 键也可以用你的终端上的 ALT 键来代替。

CTRL 键及 meta-key 键只是 Emacs 编辑器中众多编辑命令的一部分。所有的命令都可以显示在编辑显示域（可以把 Emacs 编辑器中的显示域看作 Vi 编辑器中的底行命令行）。meta-key 命令 ALT-x 及 ESC-x 等将被放置在编辑器的显示域中。一旦进入编辑器显示域中，你可以在该行上键入任何命令及其参数，并键入 ENTER 键来执行该命令。表 18-1 列出了 Emacs 编辑器的一些

编辑命令。在模式行上将显示正在编辑的文本的状态信息。模式行由几个部分组成，其形式如下：

-ST-Emacs: BufferName (major minor) ------Place-

第一个域是 ST（status），它表示最近对文件进行修改以来，该修改的文件是否已经被保存过。如果该域是两个星号（**），表示该文本已经被修改过，且并没有保存；如果该域是两个连字符（--），表示该文本从最后一次保存到现在还没有被修改过；如果该域是两个百分号字符（%%），表示该文本是一只读文件，不能被修改。

BufferName 域表示该工作缓冲区的名字，这个工作缓冲区名将是目前编辑的文件名。而 Place 域表示你目前处于被编辑的文件名的何处。例如，如果 Place 域是 40%，那么，表示你目前（即光标）约处于文件的 40%处。在下面的例子中，模式行表示自最近一次修改以来，该文件还没有被保存过，工作缓冲区的名是 mytext，光标处于文件的顶行。

18.3-* *-Emacs:mytext (text fill)---- --Top--

major/minor 域表示在编辑文本的过程中的主要及辅助模式。Emacs 编辑器能自动识别几个常见的标准模式。最常见的模式是文本模式。不同类型的文件需要不同的编辑配置与选项。例如，对 C 程序来说，需具有自动缩进特性，因此，编辑 C 程序时有其特定的编辑模式。Emacs 编辑器还能自动识别其它的几种标准编辑模式，例如 nroff 及 Lisp 等。Emacs 编辑器通过编辑的文件名的控

制名来决定将使用的编辑模式。如果由于某些原因，Emacs 编辑器不能决定使用的编辑模式，该编辑器将使用基本的编辑模式，该编辑模式不具有某些特殊的特性。

Emacs 编辑器还有几种辅助模式：fill、overwrite 及 abbrev。fill 模式通常是缺省的模式。该模式将在编辑文本时自动换行；overwrite 模式允许你覆盖文本；而 abbrev 模式允许你在输入文本的时候使用缩写。

18.4Emacs 编辑器的编辑命令

Emacs 编辑器的编辑命令能执行许多操作，其它编辑器基本上也有类似的编辑操作。Emacs 编辑器中所有的编辑命令基本上都通过 CTRL 键、ALT 键或 ESC 键组成的命令系列来完成与实现。所有这些编辑命令也都有对应的命令名，你可以在显示域中输入这些命令。本节将介绍许多常见的实用命令。

18.4.1 移动命令

与 Vi 编辑器一样，Emacs 编辑器有一组基本的光标移动命令。CTRL-f 命令用来将光标前移（forward）一个字符，而 CTRL-b 命令用来后移（backward）一个字符。CTRL-f 命令及 CTRL-b 命令分别也可以看作将光标右移及左移一个字符。但是，Vi 编辑器的光标右移与左移命令仅局限于一行上，而 Emacs 编

辑器的前移命令与后移命令却不是如此。Emacs 编辑器把一个文件看作一种流式字符串，而不是一系列的文本行。向后（backward）移动光标命令将使光标沿流式文本左移一个字符（例如在当前行的行首上将光标左移一个字符时，光标将回到上一行的行尾上）。向前（forward）移动光标命令也是如此。

也有一组编辑命令能使用户在文件本中以行为单位移动光标或整屏移动光标。CTRL-n 命令及 CTRL-p 命令分别用来下移（down）及上移（up）光标一行。CTRL-n 命令将把光标移动到下一行上，如果此时光标位于屏幕最后一行，屏幕将下滚一行，并使当前行的下一行显示在屏幕上。CTRL-p 命令将把光标移动到上一行，如果此时光标位于屏幕最顶行，屏幕将上滚一行，并使当前行的前一行显示在屏幕上。CTRL-v 命令及 CTRL-z 命令将整屏滚动文本。CTRL-v 命令将使文本向前（forward）滚屏并显示下一屏文本，而 CTRL-z 命令将使文本向后（backward）滚屏并显示上一屏文本。

你也可以以单词、段落等为计量单位来移动光标。meta-key 键命令 ESC f 及 ESC b 用来以单词为单位前、后移动光标。ESC] 命令可以把光标移动至下一段落的段首，而 ESC [命令将把光标移动至前一段落段首。这些命令可以使你在编辑文本时向任意方向快速移动光标。还有一些移动命令可以使光标移动至特定的行上。例如，CTRL-a 命令将把光标移动至行首，而 CTRL-e 命令将把光标移动至行尾；CRTL-l 命令将把光标移动至屏幕的中央；ESC <命令将把光标移动至文件的第一行，而 ESC >命令将把光标移动至文件的结尾。

你可以在编辑命令前输入 Emacs 编辑器的重复命令来重复执行一个命令，此重复命令是 ESC num，这里 num 是你将要重复执行的命令的运行次数。例

如，要右移光标 5 次，首先输入重复命令及重复的次数，然后键入 CTRL-f 命令：

ESC 5 Ctrl-f

你也可以用相同的方式来使用重复命令完成重复输入：首先键入 ESC 命令，再键入命令重复执行的次数，最后键入输入的内容。例如，ESC 3 T 命令将往文本中输入三个 T 字符。

18.4.2 删除命令

删除文本意味着永久地删除文件中的字符。有两类基本的删除操作：一类是删除光标之后的字符，另一类是删除光标之后的字符。CTRL-d 命令将删除光标之后的字符，而 DEL 键将删除光标之前的字符，它与 BACKSPACE 键的功能相同。

18.4.3 删除缓冲区及移动文本

在 Emacs 编辑器中，delete 文本与 kill 文本是有区别的。delete 文本将永久地将文本从文件中删除掉，而 kill 文本只是将文本从文件缓冲区中删除，但编辑器会将其拷贝到 kill 缓冲区中保存，必要时可以将其恢复。Emacs 编辑器中 kill 文本与 Vi 编辑器中的删除操作非常类似。被 kill 掉的文本被保存在一组有编辑器创建的 kill 缓冲区中。当你 kill 文本的时候，每一个 kill 缓冲区将依次

保存被 kill 掉的文本。kill 缓冲区是一个循环链表，当所有的 kill 缓冲区都保存了文本时，最先保存了文本的 kill 缓冲区将被下一个要删除的文本覆盖掉。

在任何时候，你都可以把 kill 缓冲区中的文本插入到文本中去。从这种意义上说，kill 命令意味着对文件中的文本进行移动操作。你首先 kill 它们，于是该文本会在文件中删除，但是，你可以把光标移动至任意地方，并插入已经 kill 掉的文本。

你也可以以不同的计量单位来 kill 文本，例如，以单词及行为单位来 kill 文本。ESC DEL 命令将删除光标所在单词的前一个单词，而 ESC d 命令将删除光标所在单词的后一个单词。你也可以在删除命令之前键入 ESC num 命令来重复执行删除操作，以连续删除几个单词。例如 ESC 3 ESC d 将连续删除三个单词。

CTRL-k 命令将删除从光标所在位置至行尾的所有字符。要删除整行字符，必须首先用 CTRL-a 命令将光标放置在行首，然后键入 CTRL-d 命令。然而，此时 CTRL-d 命令并不删除该行，该行依然存在，只是该行将成为没有任何字符的空行。要删除字符的同时删除该空行，需要键入删除命令：CTRL-k CTRL-k 命令，其中，第二个 CTRL-k 命令将删除空行。实际上，要删除空行就是要删除新行标识字符（键入 ENTER 键时）。与删除单词一样，如果你想同时删除多行，你可以在 CTRL-k 命令之前输入重复命令。例如，ESC 10 CTRL-k CTRL-k 命令将删除 10 行文本。

一旦你 kill 了文本（同时将被保存至 kill 缓冲区），你可以使用 CTRL-y 命令把它们插入到文本中去。CTRL-y 命令也通常被称之为 yand 命令。在 kill 缓

冲区中的任何文本都将被插入带编辑的文本中去。如果 kill 缓冲区保存了单词，CTRL-y 命令将插入单词，如果 kill 缓冲区保存了一行或多行文本，CTRL-y 命令将插入一行或多行文本。

移动文本涉及到两个过程：首先用删除命令将文本删除至 kill 缓冲区，然后使用 yank 命令 CTRL-y 来插入文本。在下面的例子中，一系列的命令将当前行上移五行。

Ctrl-k Ctrl-k
Esc 5 Ctrl-p
Ctrl-y

如果你意外地删除了多行或将文本插入到了不想插入的地方，你可以用 CTRL-x u 命令取消这一意外操作。Emacs 编辑器甚至可以让你使用 ESC x 命令取消自开始这一次编辑以来所有的修改。

18.4.4 字符块：点(point)与标识(mark)

你可以创建一个与光标位置相关的字符块，并选择该字符块。一个字符块可以是光标所在的位置与任意先前标识的字符之间的所有文本。你可以使用 CTRL-@ 命令来标识一个字符（该标识的字符就是你要选择的字符块结束字符），然后把光标移动至文件的任何地方，于是被标识的字符与光标所在的位置之间的字符可以称之为字符块。在 Emacs 编辑器中，此光标的位置常称之为点(point)，而被标识的字符称为一个标识(mark)。

在 Emacs 编辑器中，有一部分命令用来对字符块进行操作。例如，你可以拷贝或删除字符块：CTRL-w 命令将删除选择的字符块，并把它保存在删除缓冲区中；ESC w 命令将把字符块拷贝至 kill 缓冲区中去；你可以使用 CTRL-w 命令删除一个字符块，而用 CTRL-y 命令把删除的字符块插入到编辑的文件中，即用这两个命令来实现字符块的移动。

Emacs 编辑器在编辑的文件中并不显示标识。如果你忘记了标识的位置，你可以使用 CTRL-xCTRL-x 命令来定位一个标识。键入 CTRL-x CTRL-x 命令，编辑器将把光标定位在先前的标识字符上，继续键入 CTRL-x CTRL-x 命令，光标将移动到光标先前所在的位置上。

你也可以用计量单位来定义字符块。你可以用段标识命令 ESC h 来选择光标所在的段为字符块。此时，point 是光标所在段的段首，而标识是光标所在段的段尾。CTRL-x CTRL-p 命令将选择光标所在的页为字符块。可以用整个编辑缓冲区标识命令 CTRL-x h 来选择编辑缓冲区，并把该缓冲区作为一个字符块。下面是一些有关字符块操作的一些命令：

Esc h Ctrl-w	删除光标所在的段落
Esc h Esc w	拷贝光标所在的段落
Ctrl-x Ctrl-p Esc w	删除光标所在的页
Ctrl-x h Esc g	调整整个文本

18.4.5 模式搜索命令

你可以键入 CTRL-s 命令来完成模式搜索。CTRL-s 命令将把光标放置在编辑器的显示域上，此时，你可以在显示域上输入搜索的模式（pattern）。一旦你键入字符，Emacs 编辑器就开始搜索，当你继续输入字符时，Emacs 编辑器将继续搜索与正在输入的模式匹配的字符串。例如，如果你想键入搜索模式 preface，一旦你键入字符 p，光标将移动至文件中与模式 p 匹配的字符处。继续键入字符 r，光标将移动到与模式 pr 匹配的字符串处。要结束模式的输入，你可以键入 ESC 键。下面是在文件中向前（forward）进行模式搜索的基本格式：

Ctrl-s pattern

Crtl-r 命令将在文件中向后（backward）进行模式搜索。这两个命令都不会自动绕自行首成行尾。多次执行 CTRL-s 命令后，光标将停留在文件的结尾，而 CTRL-r 命令将停留在文件的首行。Emacs 将保存最后一次的搜索模式。直接键入 CTRL-s 或 CTRL-r 命令而不键入搜索的模式，编辑器将用前一次的搜索模式进行搜索。

正则表达式搜索

Emacs 编辑器允许你使用在 Vi、Ex 及 Ed 等编辑器中使用任何正则表达式及特殊字符。要在搜索时使用正则表达式，你可以在 CTRL-s 或 CTRL-r 搜索

命令之前键入 ESC 键，即 ESC CTRL-s 或 ESC CTRL-r 命令允许你在搜索字符串中使用正则表达式。

18.4.6 替换操作：全局替换命令与查询替换命令

在 Emacs 编辑器中，你可以使用全局替换命令或查询替换命令操作来实现文本的替换。由于全局替换操作主要执行全局代替，因此我们很少使用该操作。而查询替换操作允许你执行单个的替换。事实上，当执行查询替换操作时，如果编辑器搜索到与模式匹配的字符串，编辑器将询问你是替换该字符串还是不执行替换操作而直接进行下一次搜索与模式匹配，或者退出当前的查询替换操作。这样，在你执行替换的时候，查询替换比全局替换具有更多的控制权。在进行正则表达式搜索时，这两个操作都有与之对应的特殊命令。

全局替换：Replace-string 命令

你可以直接在编辑器的显示域中使用 replace-string 命令来执行全局替换操作，且在键入 replace-string 命令后，你不必键入任何键。其操作过程如下：首先键入 ESC x 命令进入编辑器的显示域，然后键入 replace-string 命令，编辑器立即提示你输入搜索的模式及将要替换的字符串。该命令与 Ex 编辑器中的全局替换命令功能相同。Repalce-string 命令不能实现正则表达式的替换，如果你要在替换字符串中使用正则表达式，你必须使用 replace-regexp 命令。

查询替换命令

ESC %命令是用来执行查询替换操作的。它首先搜索到与模式匹配的字符串，然后在必要时替换该字符串。该命令与 Ex 编辑器中的替换命令的功能相同。要执行查询替换命令，首先必须键入 ESC %命令，然后键入将要被替换到的模式，并键入 ENTER 键，此时，再键入将要替换的字符串并按 ENTER 键。完成上述操作之后，与被替换的模式匹配的第一个字符串被搜索到，同时出现几个选项，每个选项都有与之对应的按键。例如，如果你键入 y 键，搜索到的字符串将被替换字符串所替换，同时，光标将位于已被替换掉的字符串上；如果你键入 n 键，将取消搜索到的字符串的替换操作，同时，光标将位于搜索到的字符窜上。其格式如下：

ESC % pattern ENTER
replacement-text ENTER
Query-replace options

查询替换命令的选项见下表：

y 或者 Spacebar 键	替换搜索到的字符串
n 或者 Del 键	取消搜索到的字符串的替换
^	回到前一个搜索到的字符串
!	替换所有没有替换的与模式匹配的字符串
.	替换后退出
ESC 键	退出本次查询搜索

下面的命令系列将用 yogurt 字符串替换文本中的模式 milk，然后继续下一次搜索。用户可以键入 ESC 键来退出查询搜索操作。

ESC % milk ENTER
yogurt ENTER
y

与其它搜索命令一样，查询搜索命令不允许在搜索模式中使用特殊字符。但如果你想使用特殊字符，你可以使用 query-replace-regexp 命令，要运行该命令，你必须首先键入 ESC x 命令进入编辑器的显示域中。

18.5 在 Emacs 编辑器中使用多窗口

Emacs 编辑器中的多窗口允许你在窗口内浏览同一文件的不同部分或同时浏览多个文件。一个窗口命令通常是在 CTRL-x 命令之后指定一个数字，例如 CTRL-x 2 将在编辑器中重开一个窗口，CTRL-x 0 将关闭当前窗口。Emacs 编辑器中的窗口命令见表 18-2。

如果你同时打开了多个窗口，那么，光标所在的窗口称为当前窗口，也称为活动窗口。任何编辑命令与编辑操作都是针对于当前活动窗口的。你可以用 CTRL-x o 及 CTRL-x p 命令切换到另一个窗口。CTRL-x o 将按窗口打开的次序在个窗口之间继续切换，而 CTRL-x p 命令将切换到前一窗口。你可以用 CTRL-x 0 命令来关闭一个窗口，用 CTRL-x 1 命令来关闭除当前窗口之外的所

有窗口。

当你打开一个新的窗口的时候，你可以把该窗口放置在当前窗口旁或当前窗口下。你可以用 CTRL-x 5 命令把当前窗口沿水平方向分割成左右两个并排的窗口，而用 CTRL-x 2 命令把当前窗口沿垂直方向分割成上下两个窗口。

一旦你打开一个窗口，你可以使用 CTRL-x ^及 CTRL-x }命令调整窗口的大小，其中，CTRL-x ^命令将使窗口沿垂直方向增加窗口的高度，CTRL-x }命令将使窗口沿水平方向增加窗口的宽度。

在文件中移动文本或字符块的时候，窗口是非常有用的。例如，首先给正在编辑的文件重新打开一个窗口，此时，新的窗口将显示与原窗口一样的文本，在新窗口中，把光标移动到文件的其它地方，并删除文件中要删除的字符块至 kill 缓冲区，再切换到另一个窗口，就把刚才删除的文本插入到当前窗口中光标所在的位置上。

18.6 缓冲区与文件

正如前面所提到的那样，当你用任何编辑器进行文本编辑的时候，编辑器首先读入文件的内容至内存缓冲区中。一个内存缓冲区实际上是一段用来保存编辑文件的内存，你也可以把它理解为一个字符串数组。然后你的任何编辑与修改操作都是在这一缓冲区中进行的，必要时，你可以把缓冲区中的内容保存到磁盘上。

Emacs 编辑器的编辑操作也是如此，即 Emacs 编辑器的编辑与修改操作都是在缓冲区中进行的。你可以使用编辑缓冲区来保存文件或用作其它用途。你甚至可以创建增加的缓冲区，并在缓冲区中保存字符串，如果需要，你也可以把你创建的缓冲区的内容保存到文件中去。而在 Vi 编辑器中，你只能创建一个文件编辑缓冲区。

18.6.1 文件缓冲区

当你打开一个文件时，用来保存文件内容的缓冲区将被创建。你可以仅仅只创建一个文件缓冲区，进行必要的编辑之后再把编辑的内容保存到文件中去。在任何时候，该缓冲区将与其编辑的文本之间建立联系，因此该缓冲区称为文件缓冲区。命令系列 CTRL-x CTRL-f filename 将打开一个文件并为其创建一个文件缓冲区；命令系列 CTRL-x CTRL-s 将保存文件缓冲区的内容至文件；而命令系列 CTRL-x CTRL-c 将退出编辑的文件。文件缓冲区的有关命令在表 18-3 中列出。

你可以同时打开多个文件，每个文件都有其自己的缓冲区。要在屏幕上同时显示多个文件（即文件缓冲区），你可以用前一节介绍的窗口命令来实现。CTRL-x 2 命令首先创建一个新的窗口，然后键入命令系列 CTRL-x CTRL-f filename，以便在新创建的窗口上显示文件缓冲区内容。命令 CTRL-x 4 f filename 也将执行相同的操作：创建一个新的窗口，并把文件缓冲区的内容显示在窗口内。Emacs 编辑器有一个特殊的实用命令称为列目录命令（dired

utility）。该命令将交互式地列出你当前目录下所有文件的文件名，并允许你选择相应的文件，以便对该文件进行编辑、保存甚至删除等操作。要进入列目录实用命令，你可以键入 CTRL-x d 命令。键入该命令后，你当前目录下的一系列文件将被显示在屏幕上。这一系列文件将组成一个文件名列表，该列表如同一个菜单，而当前目录中的每一个文件名就是该菜单的一个菜单项，你可以在各个菜单项之间来回移动，并可以执行各种操作。

列目录实用命令（dired utility）有它自己的一组命令，这一组命令用来在不同的文件名列表项之间移动或选择文件名列表项。n 命令用来下移一个文件列表项，而 p 命令用来上移一个文件名列表项，e 命令用来打开当前文件，以便对该文件进行编辑与修改。如果被选中的文件名是一个目录，则列目录实用命令将使用户进入该目录，并显示该目录中的所有文件。字符串..表示当前目录的父目录，选择该项将回到当前目录的父目录下，并同时显示其父目录中的文件。

你也可以保存或删除文件名列表中的文件。移动至一个文件名列表项上，键入 s 命令标识该文件将被保存；移动至一个文件名列表项上，键入 d 命令标识该文件将被删除（实际上该文件并不会立即被删除，直到你退出列目录实用命令后才会删除标识了的文件）。请记住：你可以用 u 命令来取消文件的删除或保存标识。你可以用 CTRL-x b 命令可以退出列目录实用命令，并回到你先前编辑的窗口（即先前编辑的文件缓冲区）。

18.6.2 独立缓冲区（Unattached Buffers）

使用命令 CTRL-x b，你可以创建一个与任何文件都不相关联的缓冲区，也称为独立缓冲区。当你键入 CTRL-x b 命令，Emacs 编辑器将立即提示你输入所创建的缓冲区的名字，如果该缓冲区并没被创建过，则 Emacs 编辑器将创建这个新的缓冲区。你也可以使用 CTRL-x b 命令转换到指定的缓冲区。键入 CTRL-x b 命令，并在提示符后键入缓冲区名，则你输入的缓冲区名中的内容将被显示在当前窗口中。因此，通过命令 CTRL-x b，你可以在不同的缓冲区之间来回切换。命令 CTRL-x CTRL-b 将列出所有缓冲区的名字。

如果你想在新的窗口上创建一个新的缓冲区，你首先必须创建一个新的窗口，并切换至该窗口，再使用 CTRL-x b 命令来创建一个新的缓冲区。你也可以使用 CTRL-x 4b 命令来同时创建一个新的窗口与新的缓冲区。如果你键入该命令，一个新的窗口将被创建，同时，编辑器将提示你输入创建的缓冲区的名字。

图 18-2 显示了三个窗口，每个窗口用来编辑一个缓冲区。在这三个缓冲区中，只有一个文件缓冲区 mytext，其它两个缓冲区 topics 及 preface 都是独立缓冲区。当在 mytext 缓冲区进行编辑的时候，你可以切换到其它窗口，例如 topics 缓冲区窗口或 preface 缓冲区窗口，并在这些窗口中增加新的文本。当你退出 Emacs 编辑器之后，在独立缓冲区中保存的内容将丢失，但是，用户可以把独立缓冲区中的内容拷贝到文件缓冲区中，或者把独立缓冲区中的内容直接保存到新的文件中去。

每个缓冲区不必有其独立的窗口，你同样可以用 CTRL-x b 命令来在不同的编辑缓冲区之间进行切换。例如：CTRL-x b topics 将使用户切换到 topics 编辑缓冲区。此外，编辑器根据缓冲区创建的顺序给予编号，不带任何参数的 CTRL-x b 命令将按顺序在缓冲区之间进行切换。

与前面介绍的列目录命令一样，用一个特殊的实用命令可以管理缓冲区。命令 buffer-menu 是一个交互式实用命令，该实用命令将显示所有的缓冲区的名字。与列目录命令一样，你可以修改、删除及保存缓冲区。你可以首先用 ESC-x 命令进入编辑器的显示域，然后键入 buffer-menu 命令。

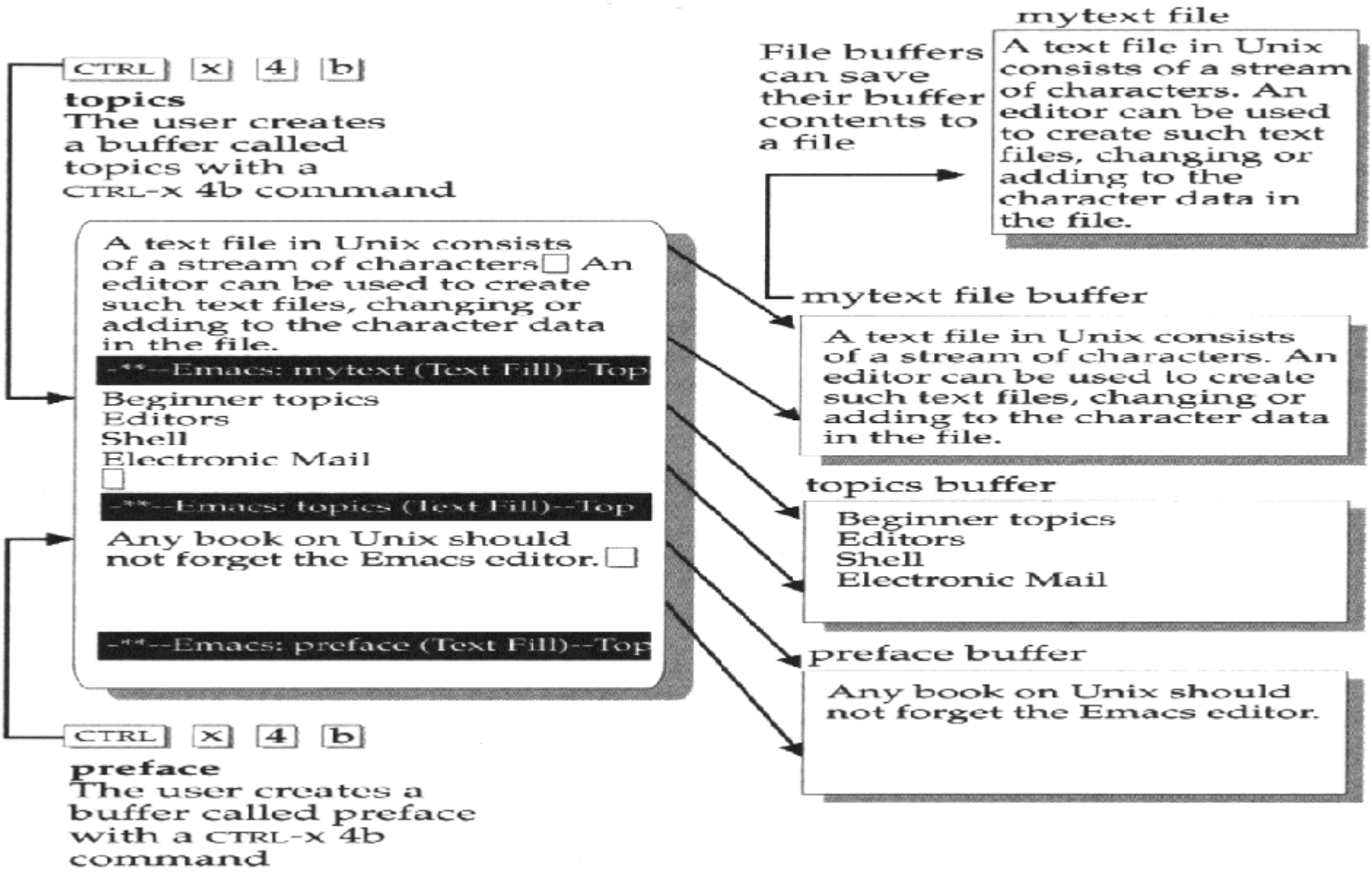

CTRL x 4 b
topics
The user creates a buffer called topics with a CTRL-x 4b command
A text file in Unix consists of a stream of characters. An editor can be used to create such text files, changing or adding to the character data in the file.
-**--Emacs: mytext (Text Fill)--Top
Beginner topics
Editors
Shell
Electronic Mail
-**--Emacs: topics (Text Fill)--Top
Any book on Unix should not forget the Emacs editor.
-**--Emacs: preface (Text Fill)--Top
CTRL x 4 b
preface
The user creates a buffer called preface with a CTRL-x 4b command
File buffers can save their buffer contents to a file
mytext file
A text file in Unix consists of a stream of characters. An editor can be used to create such text files, changing or adding to the character data in the file.
mytext file buffer
A text file in Unix consists of a stream of characters. An editor can be used to create such text files, changing or adding to the character data in the file.
topics buffer
Beginner topics
Editors
Shell
Electronic Mail
preface buffer
Any book on Unix should not forget the Emacs editor.

图 18-2

18.7 编辑器帮助

Emacs 编辑器提供几个实用帮助程序，例如在线帮助（online help）与帮助指南（tutorial）。你可以键入 CTRL-h 系列命令来访问实用帮助文件。例如，键入 CTRL-h 命令之后再键入一次 CTRL-h 命令将列出所有可能的帮助选项，其中有一个特殊的帮助就是 tutorial。键入 CTRL-h t 命令将进入在线帮助指南（online tutorial），该帮助可以提供一些特殊的 Emacs 编辑器教程。表 18-4 列出了 Emacs 编辑器的部分帮助命令。

18.8XEmacs 编辑器

XEmacs 编辑器是图形用户界面（GUI）下一个功能更完整、齐全的 Emacs 编辑器，也是一个 Internet 应用程序。你可以通过 XEmacs 编辑器的主快捷按钮来访问 Internet，包括：Web 浏览实用程序、mail 实用程序及读取 news 实用程序。你可以用以下三种方式启动 XEmacs 编辑器：1)在 Goodstuff 工具条上双击 XEmacs 编辑器的图标来启动 XEmacs 编辑器；2)从 fvwm workshop(为一个应用软件与开发环境）的菜单上启动 XEmacs 编辑器；3)直接在 Xterm 窗口的命令提示符上键入 xemacs 命令来启动 XEmacs 编辑器。在 XEmacs 编辑器的主窗口的上部有一组基本的快捷键图标，这些图标用来执行基本的编辑操

作及运行 Internet 应用程序。XEmacs 编辑器基本的 GUI 编辑操作：用可以用双击鼠标来选择文本；通过鼠标拖动等操作来完成字符块的删除、拷贝与粘贴；通过滚动条的移动来滚动文本。XEmacs 编辑器的底部是模式行及显示域，这部分与标准的 Emacs 编辑器完全相同。你也可以象在标准的 Emacs 编辑器的显示域一样在 XEmacs 编辑器的显示域上输入键盘命令。从 XEmacs 编辑器窗口顶部的菜单上你可以执行大多数的文件、编辑与缓冲区操作。File（文件）菜单用来管理缓冲区、窗口及 Frames，Edit（编辑）菜单用来处理编辑操作，如删除、拷贝与粘贴，此外还执行字符串搜索与替换等操作。Option 菜单允许你设置编辑选项与打印选项。在 Apps（应用程序）菜单上，你可以访问其它应用程序，例如 mail，newsread 及 Web 浏览器等。Buffer（缓冲区）菜单将列出你的缓冲区。Help（帮助）菜单将列出几个帮助文件，包括用户常问的问题（FAQs）及相关网页等。Xemacs 编辑器的菜单对于一些更复杂的 Emacs 操作是非常有用的，例如为一个文本开辟多个窗口、管理文件缓冲区及其它类型的缓冲区。当你通过一个菜单项来执行一个操作的时候，相应的键盘操作命令会显示在 XEmacs 编辑器的显示域上。图 18-3 显示的是一个 XEmacs 窗口。

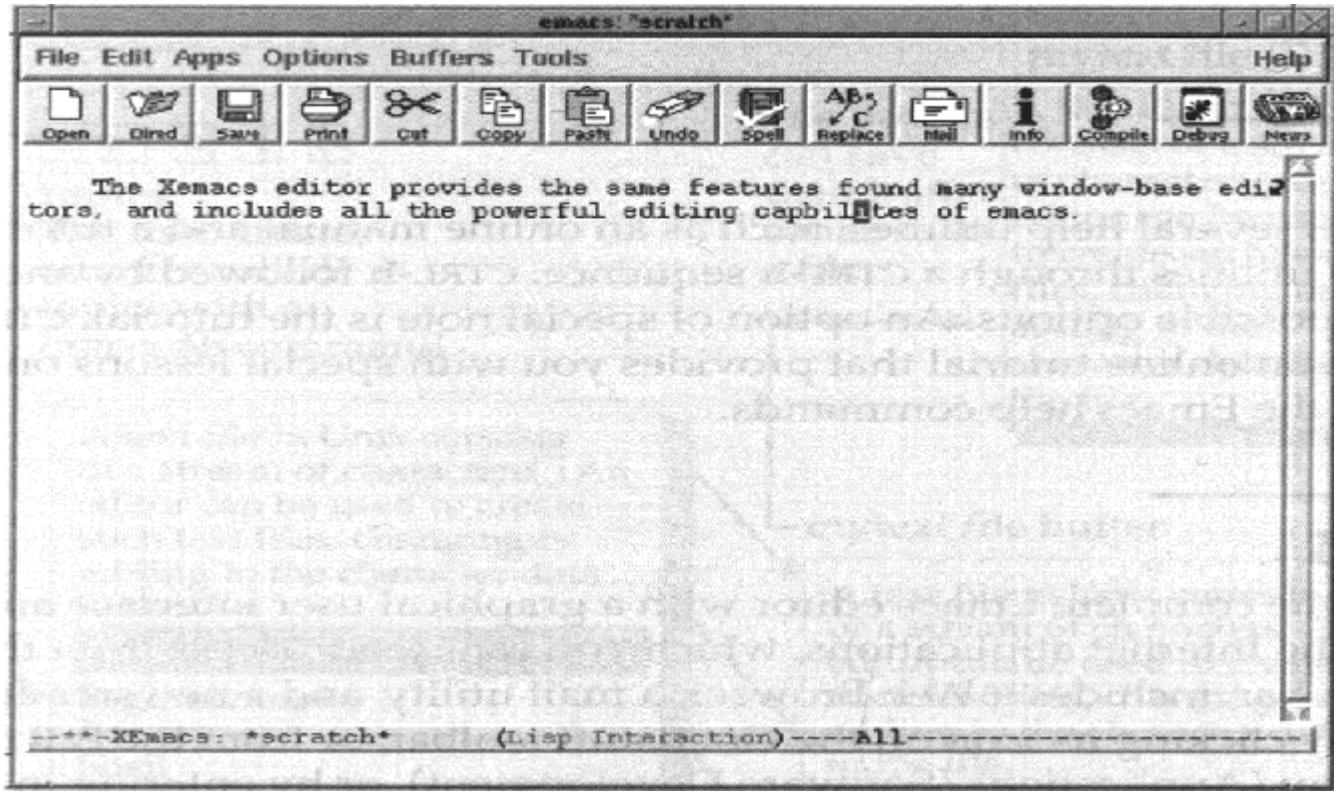


图 18-3Emacs 编辑器

单击 Mail **图标将开辟另一个窗口，在该窗口中将显示收到的** E-mail，**当然你也可以用它来编辑文件并把编辑的文本发给其它用户。你可以单击窗口上的** compose、reply **及** print **等图标来编辑、回复及打印邮件。要显示一个** mail **的内容，你可以单击其** Header，**然后键入** SPACEBAR **键（空格键）。你可以在** Folder **菜单下选择** Display Summary **项来显示每个邮件的** Header。**但你在编辑一个** E-mail **的时候，你可以使用** XEmacs **编辑器的所有编辑命令及所有特性，包括拼写检查及字符串搜索与替换。当开辟一个新的窗口时，编辑器将提示你**

输入收件人的 E-mail 地址及 E-mail 的主题（subject）。该窗口也具有 XEmacs 编辑器的所有编辑图标按钮与菜单。当你完成了编辑的 E-mail 之后，你可以选择 Mail 菜单的 Send and Exit 菜单项来发出刚才编辑的 E-mail。

单击 News 图标按钮也将开辟一个新的窗口，该窗口将显示 XEmacs 的 newsreader（列出你从新闻组中的收到的新闻）。如果你使用 ISP，那么，你需把 NNTPSERVER 变量设置成你的新闻组服务器。一个简单的启动 XEmacs 编辑器中 Web 浏览器的方法是在 Help 菜单上选择 XEmacs WWW Page 项。该 Web 浏览器支持标准的 Web 浏览器 b 浏览器特性，如 bookmarks 等。但是，你也可以根据你的喜好选择合适的浏览器。在 Option 菜单的 Open Uæith 菜单项上，你可以选择你的系统中已经安装了的浏览器，如 Netscape 及 Lynx 等。

18.9Emcas 编辑器小结

Emacs 编辑器是一个面向缓冲区(buffer-oriented)的文本编辑器，你可以象使用其它常见的文字处理软件一样使用这个编辑器。键盘上的所有的键都是输入键。所有的命令都要使用 CTRL 键或 meta-keys 键。一个 meta-key 键可以是 ALT 键或 ESC 键。Emacs 编辑器编辑的文本文件是面向字符的，而不是面向文本行的。你可以以字符、单词、行或段落为单位在文本中移动光标。

Emacs 编辑器在 delete 与 remove 文本之间有较大的区别。delete 是永久地从文件中删除文本，而 remove 则首先将文本删除到缓冲区中去，必要时你

可以访问这些缓冲区。这些缓冲区通常称为 kill 缓冲区，而这些 remove 文本的命令也称为 kill 命令。在 Emacs 编辑器中，常有一些术语如 kill 一个单词或 kill 一行，它们指的是将一个单词或文本行删除至 kill 缓冲区。必要时，你可以把 kill 缓冲区的内容插入到文件中去。你可以通过以下方式来移动文本：首先把文本删除至 kill 缓冲区，然后把它输入到文件中适当的地方。

Emacs 编辑器允许你在屏幕上同时打开多个窗口，在这些窗口中，你可以显示多个不同的文件，也可以显示同一文件。你也可以沿水平或垂直方向把一个窗口分割成多个窗口，这些窗口可以用来显示文件的不同部分，并可以把一个窗口内的文本删除至 kill 缓冲区，然后切换并插入到另一个窗口的文本中完成字符块的移动。

你可以同时打开多个文件，每个文件都有其自己的缓冲区。你也可以创建自己的缓冲区，这些缓冲区不依赖于任何的文件。你可以在各个缓冲区之间方便地切换，以便对缓冲区进行编辑。你甚至可以访问 kill 缓冲区，并对它们进行编辑。buffer-menu 实用命令提供一个菜单界面，该菜单界面会列出所有的缓冲区，你可以非常方便地选择要编辑的缓冲区。

你也可以同时在各自的窗口中显示不同的缓冲区。用来保存文件内容的缓冲区称为文件缓冲区。当你在屏幕上显示多个文件缓冲区的时候，实际上是在屏幕上同时显示多个文件的内容。列目录命令实用命令提供给用户一个菜单界面，该菜单将列出你当前目录下的所有文件。你可以在该菜单界面上打开、保存或删除指定的文件。

表 18-1Emacs 编辑器编辑命令

光标移动命令	功能
CTRL-b	左移光标一个字符，即向后（backward）一个字符
CTRL-f	右移光标一个字符，即向后（forward）一个字符
CTRL-n	下移光标一行，即移至下一行（Next）
CTRL-p	上移光标一行，即移至上一行（previous）
CTRL-v	光标向前（forward）滚屏
CTRL-z	光标向后（backward）滚屏
CTRL-l	光标移动至屏幕中央
ESC f	光标前移（forward）一个单词
ESC b	光标后移（backward）一个单词
ESC]	光标移至下一段
ESC [	光标移至上一段
CTRL-a	光标移动至行首（行上第一个非空格字符）
CTRL-e	光标移动至行尾（行尾最后一个非空格字符）

续表

ESC <	光标移动至缓冲区头部（通常是文件头部）
ESC >	光标移动至缓冲区尾部（通常是文件尾部）
ESC num	重复执行其后的命令 num 次
ESC x	移动至显示域，以便输入一个命令
删除（deletion）命令	**功能**
DEL	删除光标前的字符
CTRL-d	删除光标后的字符
删除（kill）与 Yanks 命令	**功能**
CTRL-k	删除（kill，下同）至行尾
CTRL-k CTRL-k	删除至行尾，并同时删除行尾的换行字符
ESC d	删除光标后的单词
ESC DEL	删除光标前的单词
ESC k	删除至句子结尾
CTRL-w	删除一个字符块
CTRL-y	插入缓冲区内容至文件
CTRL-x u	取消前一次命令
搜索与替换命令	**功能**

续表

CTRL-s	向前（forward）搜索文件中与模式匹配的字符串
CTRL-r	向后（backward）搜索文件中与模式匹配的字符串
ESC CTRL-s	向前（forward）搜索文件中与正则表达式匹配的字符串
ESC CTRL-r	向后（backward）搜索文件中与正在表达式匹配的字符串
replace string	执行全局替换
replace regexp	对正则表达式执行全局替换
query-replace-regexp	查询正则表达式，然后再执行替换操作
ESC % pattern ENTER replacement ENTER	查询与模式匹配的字符串，然后再执行替换操作
	SPACEBAR 先替换，再移动到下一匹配的字符串
	DEL 不替换，直接移动至下一匹配的字符串
	ESC 退出搜索-替换操作

续表

	. 替换后退出
	!替换所有没有替换的匹配字符串
	^移动至前一替换处
字符块操作命令	**功能与效果**
CTRL-@ 或	标识一个字符块
CTRL-SPACEBAR	
CTRL-x CTRL-x	交换光标位置与标识位置
ESC H	把一段落标识为字符块
CTRL-x CTRL-p	把一页面标识为字符块
CTRL-x h	把整个文件缓冲区标识为一个字符块
	文本格式
auto-fill-mode	设置覆盖模式选项
ESC num CTRL-x f	设置右边界
ESC q	调整一段落
ESC q	调整一字符块

表 18-2Emacs 窗口命令

命令	**功能**
CTRL-x 2	垂直方向分割一个窗口
CTRL-x 5	水平方向分割一个窗口
CTRL-x o	选择另外一个窗口

续表

CTRL-x p	选择前一个窗口
ESC CTRL-v	滚动当前窗口
CTRL-x 0	关闭当前窗口
CTRL-x 1	关闭初当前窗口外的所有窗口
CTRL-x ^	增加窗口的高度
CTRL-x }	增加窗口的宽度

表 18-3Emacs 文件缓冲区与缓冲区命令

文件缓冲区命令	功能
CTRL-x CTRL-f	打开并读取一个文件到缓冲区
CTRL-x CTRL-s	把缓冲区的内容保存到一个文件中去
CTRL-x CTRL-c	退出编辑器
CTRL-x CTRL-v	关闭当前的文件，并读取一个新的文件到当前窗口
CTRL-x I	把文件的内容插入到一个缓冲区
CTRL-x CTRL-q	打开一个只读文件（你不能修改它）
CTRL-x d	进入列目录缓冲区。该缓冲区列出了当前目录下的所有文件与子目录，你可以移动到其它的文件或目录下，也可以显示其它目录中的文件。此外，你还可以选择并打开一个文件。 n 移动到下一文件或目录 p 移动到前一文件或前一目录

续表

	e 如果当前光标在一目录上，将进入该目录下。如果当前光标在一文件上，则打开该文件。
	s 标识一个文件，表明将保存该文件
	d 标识一个文件，表明将删除该文件
	u 取消删除文件标识
	x 执行标识的文件
缓冲区命令	**功能**
CTRL-x b	切换到另一缓冲区。编辑器将提示你输入要切换的缓冲区的名字（要创建一个新的缓冲区，输入新的缓冲区的名字即可）。
CTRL-x k	删除一个缓冲区（kill)
CTRL-x CTRL-b	显示所有的缓冲区列表
ESC x buffer-menu	从缓冲区列表中选择一个缓冲区
	d 或 k 标识一个将要删除的缓冲区
	u 取消一个缓冲区的标识
	s 标识一个将要保存的缓冲区
	x 执行标识的缓冲区

18-4Emacs 编辑器帮助命令

帮助命令	**功能**
CTRL-h CTRL-h	列出所有的帮助选项

续表

CTRL-h l	访问 Emacs 帮助手册（manual）
CTRL-h t	运行 Emacs 编辑器的帮助指南 tutorial
CTRL-h b	显示它们代表的按键与命令

第六部分系统管理

第 19 章系 统 管 理

作为 Unix 的一个版本，Linux 用于同时为多个用户服务。作为一种操作系统，Linux 为用户与计算机之间提供了一个接口，通过其存储介质，例如硬盘和磁带。每个用户有其对应的 shell，通过 shell 用户与操作系统打交道。但是，你可以通过多种方式配置该操作系统本身。你可以增加用户帐号、打印机或终端。在第 7 章你已经掌握了如何增加新的文件系统。这些都是系统管理的重要部分。执行这些操作的人被称为系统管理员，也称为超级用户。从这个意义上讲，与 Linux 的交互包括两类：普通用户和超级用户。本章将仅讲述系统管理的基本操作。你将会学习系统状态，管理用户帐号，以及配置打印机和终端的有关内容。本章的目的是作为复杂的系统管理的一个入门。

19.1 系统维护：超级用户

为了执行系统管理的操作，你必须首先具有超级用户的口令。由于超级用户具有修改系统上任何配置和文件的权限，超级用户的口令一定要保密好，仅仅让管理系统的人知晓。有了正确的口令，你就能够以系统管理员的身份登录到系统上，并能够以各种方式配置和维护系统，你可以启动系统，关闭系统，也可以改变系统的运行模式，例如单用户模式。你也可以增加或删除用户帐号，增加或删除文件系统，备份和恢复文件数据，甚至指定系统的名称。在表 19-1 中列出了你可用于系统维护而使用的命令。

当你以超级用户的身份登录到系统中，你就被放到一个 shell 下面，通过这个 shell，你执行管理 Linux 的命令。该 shell 的提示符是#。下面的例子是你以超级用户的身份登录到系统中。输入的口令当然不会被显示出来。

```
login:root
password:#
```

作为超级用户，你可以使用 passwd 命令来改变超级用户的口令，也可以改变系统上任何用户的口令。

```
# passwd root
New password:
Re-enter new password:
#
```

当以普通用户的身份登录到系统时，也许需要转换成超级用户的身份。通常，可以首先退出该用户帐号，然后的超级用户身份重新登录。但是，通过 su 命令，你可以直接转换到超级用户的身份。CTRL-d 将退回到你原来的用户帐号状态。下面的例子，用户已经登录到系统。su 命令使用户转换到超级用户状态。

```
$ pwd
／home／chris
$su
login: root
password:
# pwd
／
# ^D
$
```

19.1.1 超级用户的桌面

OpenLinux 为你提供了几个 Redhat 系统管理工具，通过这些工具将简化许多系统管理的工作。这些工具只有在你的超级用户桌面上才能够使用。首先，你以 root 用户登录，并输入正确的口令。一旦登录到系统，你就可以通过 startx 命令启动时的超级用户桌面。与普通用户的桌面不同，超级用户桌面显示了大

量的用于系统配置工具的图标。表 19-1 中列出了这些工具，图 19-1 显示了超级用户桌面的图标。

usercfg 工具用于管理用户帐号和工作组--增加、修改或删除用户帐号(在本章后面将详细讲述)。fstool 工具用于配置文件系统--加载，御载仅格式化文件系统(见第 7 章)。timetool工具用于设置系统时间和日期(在本章将详述)。printtool工具由于配置系统的打印机(在本章将详述)。最后，glint工具用于从你的 Caldera CD-ROM 中安装软件包，也可以用于删除软件包。除了使用上面这些工具外，你还可以通过 Lisa 工具完成上面的所有工作。

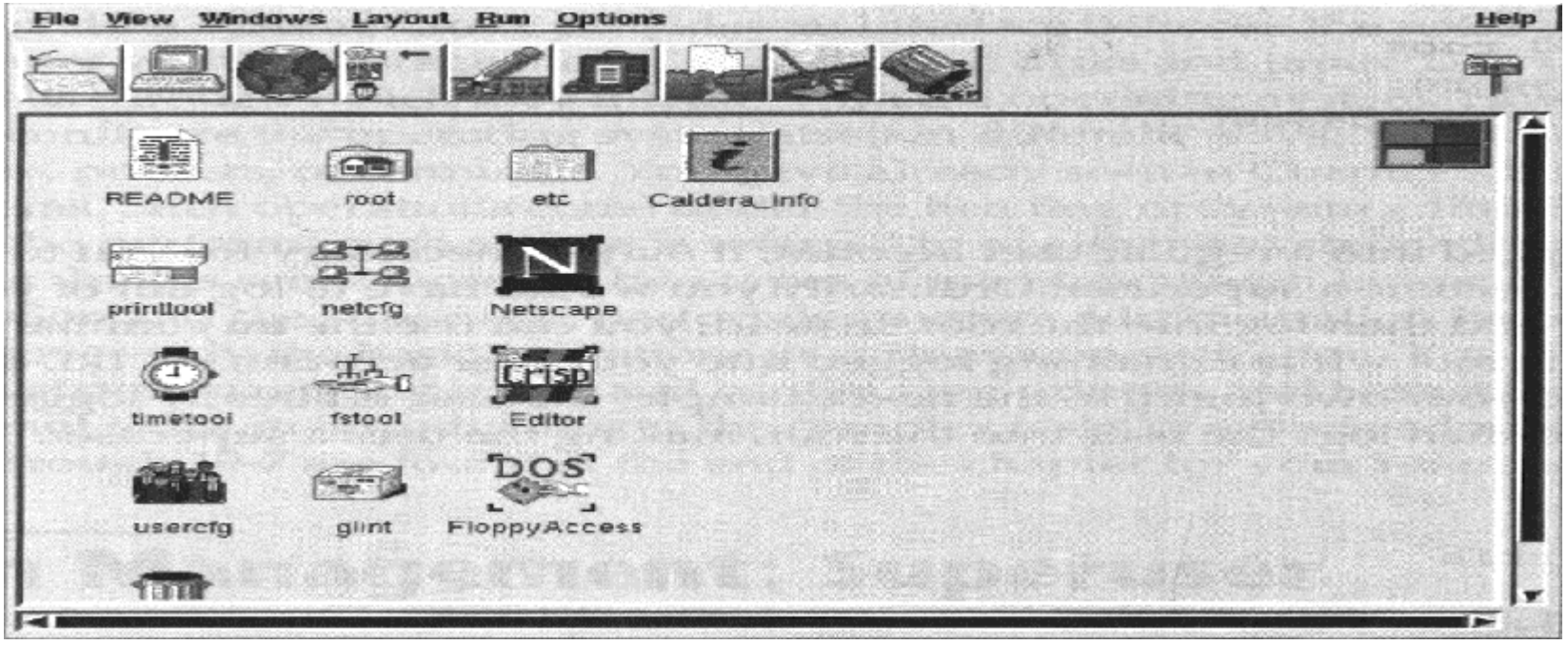


图 19-1 超级用户桌面:usercfg,netcfg,timetool,printtool fstool,glint

19.1.2 系统时间和日期

通过超级用户桌面上时间和日期的配置工具，你可以轻松地设置系统的时间和日期。该工具的图标为 timetool。回想一下你在安装系统时设置时间和日期的情形。你不必再那么配置。如果你设置的时间不正确或时区有误，你就可以利用这个工具改变系统时间，而不必重装系统。

双击 timetool 图标以打开时间设置窗口。你可以改变时间中的任何部分(见图 19-2)。例如，移动鼠标指针到小时，之后单击。小时的部分将被选中，你可以通过时间和日期显示下面的两个三角形来增加或减少时间或日期的内容。如果你选中了小时，之后单击向上的三角形将把小时设为前一个小时。

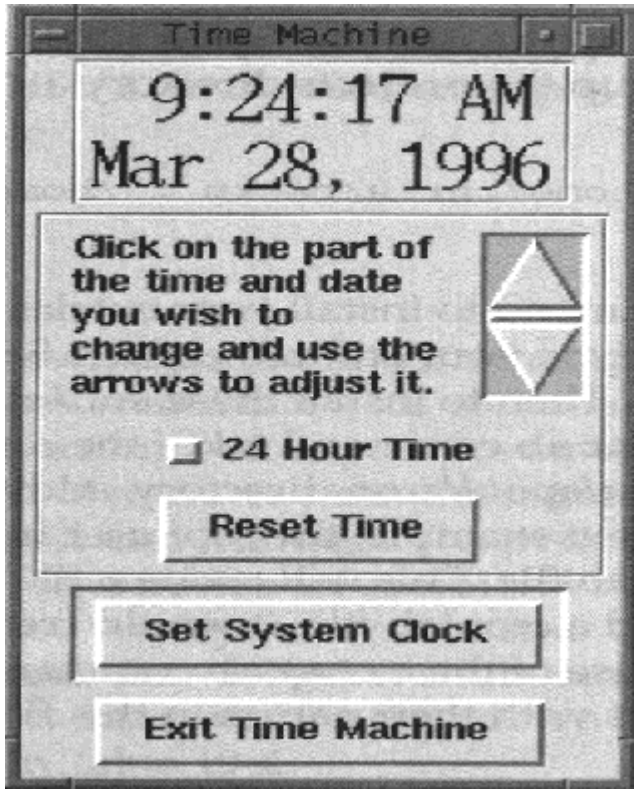


图 19-2 时间配置实用程序

一旦已经设置了新的时间和日期，单击窗口底部的 Set System Clock 按钮。之后单击 Exit Time Machine 按钮的退出时间设置窗口。

你也可以在你的命令行上通过 date 命令来设置系统的日期和时间。在 date 命令的参数中，你需要列出月份，日期，时间和年份(中间设有分隔符)。在下面的例子中，日期被设置为 1996 年 9 月 18 日下午 2：30(09 表示 9 月份，18 表示日期，1430 表示时间，96 表示年份)：

```
# date 0918143096
```

Sat Sept 18 14:30:00 EDT 1996

19.1.3 调度任务：crontab

虽然它不是一个系统文件，但你会发现 crontab 文件在维护系统时是很有用的，crontab 文件中列出在一定的时间完成指定的任务。cron 守护进程时刻在检查用户的 crontab 文件以查看是否该执行这些任何。任何用户都可以创建自己的 crontab 文件。超级用户可以做一个 crontab 文件来维护系统，例如每星期或每个月定时化完成文件备份。

crontab 项有六个域：前五个域用于指定的务执行的时间，最后一个域用于指定要执行的任务。第一个域指定分钟(0-59)，第二个域指定小时(0-23) ，第三个域指定日期(0-31)，第四个域指定月份(1-12)，第五个域指定星期几(0-6)，0 表示星期日。在每个时间域中，你可以定义一个范围，一些值，或用等号(*)表示所有的值。例如，在第五个域中 1-5 表示周一到周五。在小时域中，8，12，17 表示 8 点，12 点和 17 点。在月份域中，*表示每个月。下面的例子将在每个工作日的下午 2 点备份 projects 目录。

0 2 1-5 * * tar cf ／home／chris／backp／home／chris／projects

通过命令 crontab 来向 crontab 文件中安装任务记录。首先你要创建一个文本文件，并输入你的 crontab 记录。以任何名称保存该文件，例如 mycroafile。之后安装该记录，输入 crontab 的应该文件文件名称。crontab 命令接受该文本文件的内容。并在目录／usr／spool／cron 下创建一个 crontab 文件。下面的

例子中，超级用户将 mycronfile 的内容安装为 crontab 文件。它将创建一个文件，该文件是／usr／spool／cron／root。如果 justin 用户创建了一个 crontab 文件，该文件将是／usr／spool／cron／justin，通过文件／etc／cron.allow 你可以控制哪些用户可的使用 crontab 命令。只有在该文件中列出的用户才能创建他自己的 crontab 文件。

```
# crontab mycronfile
```

决不要直接编辑 crontab 文件。而应该通过 crontab -e 命令改变，这个命令将以标准文本编辑器，例如 vi，打开／usr／spool／cron 目录下的 crontab 文件。crontab -l 命令将显示你的 crontab 文件的内容，crontab -r 命令将删除该 crontab 文件。

19.1.4 系统状态：init 和 shutdown

你的 Linux 系统有几个状态，编号从 0 到 6，还有一种单用户状态，用字母 s 或 S 表示。当启动系统时，进入的是缺省状态，通过 init 命令，你可以转入其他的状态，例如，0 号状态表示关闭电源状态。命令 init 0 将关闭系统。状态 6 中断系统并重新启动。其它的状态反映你将如何使用系统。状态 1 是管理状态，只有超级用户可以使用系统，状态 s 是单用户状态，仅允许系统上有一个用户使用。状态 2 是部分的多用户状态，允许多用户使用，但没有远程文件共享。状态 3 是缺省的状态，是完全多用户状态，允许远程文件共享。通过编辑文件／etc／inittab 可以改变系统的缺省状态。在表 19-3 中列出了这些状态。

无论你启动时进入的是什么状态，你都可以通过 init 命令转换到另一个状态。如果你的系统缺省状态是 2，启动后进入的是状态 2，但你可以改变到，比如说状态 3，通过命令 init 3，下面的命令将转换到单用户状态。

```
# init s
```

虽然你可以通过 init 0。命令关闭系统，但也可以通过 shutdown 命令来完成。该命令有一个时间参数，在关闭系统之前给系统上登录的所有用户提示一条警告信息。你可以指定一个精确的时间，也可以指定从现在开始的一个时间段。精确的时间格式是 hh:mm，表示小时和分钟。时间段由＋和分钟数来表示。shutdown 命令有几个参数，通过这些参数，你可以指定如何关闭系统。-h 选项仅仅关闭系统，而-r 选项关闭系统并重新启动，在下面的例子中，系统在十分钟之后关闭。在表 19-4 中列出了 shutdown 命令的选项。

```
# shutdown -h ＋10
```

要立刻关闭系统，可以使用＋0 或 now。下面的例子与 CTRL_ALT_DEL 方式的关闭系统效果是相同的，它立刻关闭系统，并重新启动。

```
# shutdown -r now
```

通过 shutdown 命令，你可以包含一条警告消息发送给当前登录在系统上的所有用户。如果你不指定-h 选项或-r 选项，shutdown 命令将关闭多用户模式并切换到管理单用户模式，实际上，系统状态由 3(多用户状态)转变到 1(管理单用户状态)。只有 root 用户是活动的，允许超级用户完成任何系统维护的操作，而不受其他用户的干扰。

```
# shutdown -h ＋5 System needs a rest
```

通过命令 runlevel，你可以查看当前的运行状态。下面的例子中，系统处于状态 3。

```
# runlevel
N 3
```

19.1.5 系统初始化文件：／etc／rc.d

每次启动系统，都会读／etc／rc.d 目录下系统初始化文件中的启动命令。这些初始化文件是根据不同的任务而组织的。有些存放在／etc／rc.d 目录下，而有些则存放在子目录 init.d 下，你不必改变这些文件。配置工具，例如 netcfg 和 fstool 会改变这些文件的内容。系统初始化文件的组织随着 Linux 发行版本的不同而有所区别。下面讲述的是 Redhat 系统。在表 19-2 中列出了一些目录／etc／rc.d 下的系统初始化文件。

／etc／rc.d／rc.sysinit 文件包含了初始化系统的命令，包括加载文件系统。／etc／rc.d／rc.local 是最后要执行的初始化文件，在这里，你可以放入你自己的命令。如果你打开这个文件，你会发现每次系统启动时显示的信息，在这里，你可以改变该消息内容。

／etc／rc.d／init.d 目录用于启动和停止特定的守护进程。网络和打印机守护进程是在这里被启动的。你还会发现字体服务器和 Web 服务器守护进程。这些文件完成双重功能，在系统启动时启动守护进程，并在系统关闭时停止这些守护进程。skeleton 文件是一个例子脚本文件，它显示了如何编写系统初始化

脚本文件，它使用在文件 functions 中定义的函数。这些文件一般都是自动设置好的，你不必改变它们。

当你关闭系统时，文件 halt(它包含关闭系统的命令)，被调用。init.d 下的文件用于停止守护进程，之后文件系统被御载。在 Redhat 的当前版本，文件 halt 位于 init.d 目录下，其他版本的 Linux 系统，该文件可能被称作 rc.halt，并存放在／etc／rc.d 目录下。

19.2 管理用户帐号

作为超级用户，你可以管理系统上的所有用户。你可以增加或删除用户，增加或删除用户组，你还可以配置用于设置所有用户 shell 的系统初始化文件。

通过命令 who，你可以查看当前系统上用户的信息，-u 选项将显示当前登录的所有用户，该命令显示用户名，登录端口，登录时间和日期，非活动的时间长度(如果用户当前还是活动的)，以及登录 shell 的进程 ID。例如：

```
# who -u
root       console      Oct 12 10:34        .       1219
valerie     tty1        Oct 12 22:18       10       1492
```

19.2.1 通过 usercfg 增加和删除用户

你可以增加任意多个用户，也可以删除这些用户。通过 usercfg 工具，你可以配置用户帐号(也可以使用 Lisa 工具)。

双击该图标，将显示用户配置窗口。该窗口，称为 Usercfg，列出系统上所有的用户帐号。

要增加一个用户帐号，单击 Add 按钮，它将打开一个新的窗口，里面有帐号的有关信息(见图 19-4)。在 Name 域，输入新的用户的登录名称。Name 不是用户的真实姓名，而是该帐号的名称，例如 root 或 richlp。登录名必须小于或等于 8 个字符，并且不能包含空格符。

输入登录名称后，其它的域将自动添入缺省的值，除了 Full Name 域之外。你可以为该帐号输入口令。用户日后可以通过命令 passwd 改变他自己的口令，系统会产生一个缺省的口令，但你可以在此改变它。其他的域允许你选择用户的登录 shell，用户的主目录及所属用户组。你可以不必改变这些缺省值。当所有这些域都设定之后，单击窗口底部的 Add 按钮，系统会提示你是否要设置该用户的主目录。选择 OK。新的用户帐号将被显示在 Usercfg 窗口中。

要删除某个用户帐号，首先单击该用户帐号，再单击 Usercfg 窗口底部的

Remove 按钮，系统将提示你确认要真正删除。选择 YES。这时会提示你是否要删除用户的主目录和电子邮件信息。仍旧选择 YES，删除用户帐号并不会自动删除该用户的所有文件。

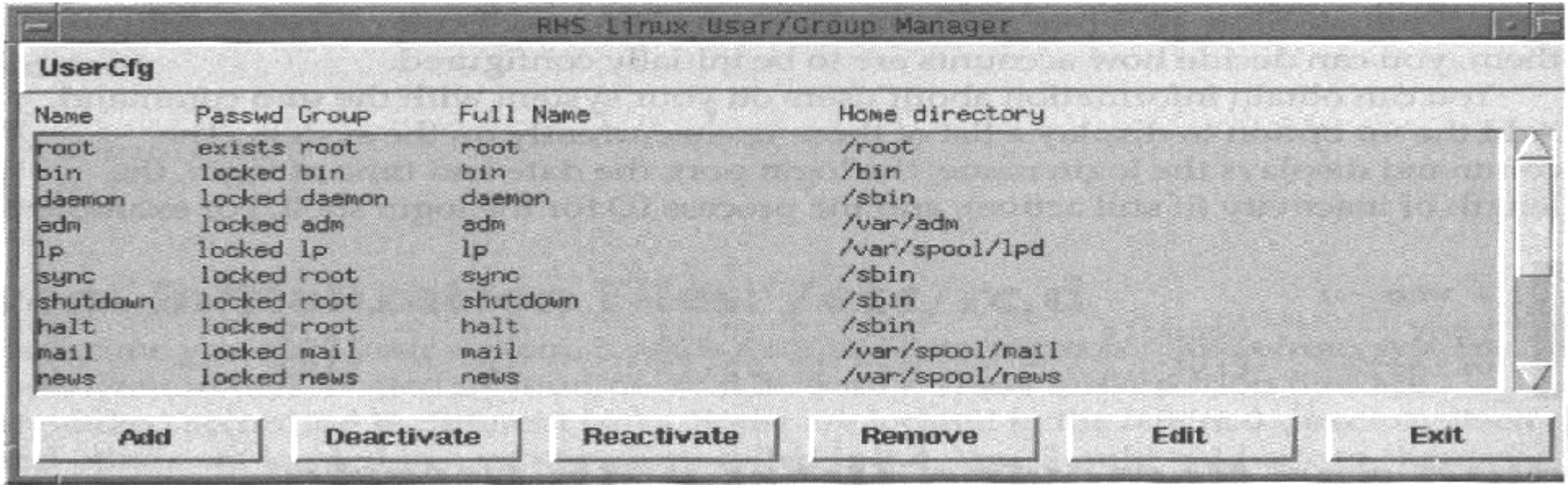


图 19-3 用户配置实用程序

有时，你可能要锁住某个用户帐号，禁止其访问系统及其文件。为了锁住用户，首先在 Usercfg 窗口单击该用户名，之后单击 Deactivate 按钮。你可以选择压缩该帐号的文件，这将节省磁盘空间(系统将提示是否压缩对应用户的主目录)。选择 YES，以压缩的用户的文件。

要解除加在某个用户上的锁，首先在 Usercfg 窗口中单击该用户名，之后单击 Reactivate 按钮。如果在加锁对该用户的文件被压缩，此时将被解压缩。

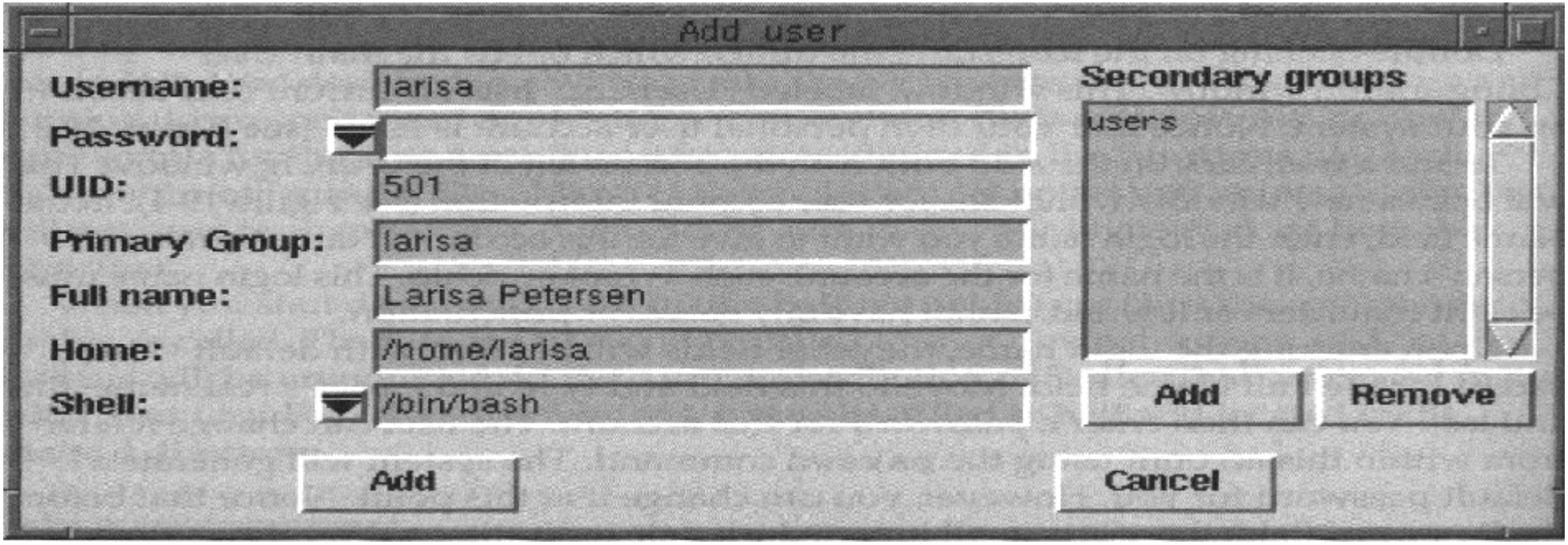


图 19-4 添加新用户

像 usercfg 这样的用户帐号管理工具，使用一些缺省文件和目录来设置新的帐号。有一些路径名，usercfg 用来定位这些缺省文件或了解在哪里创建用户帐号目录。例如，／etc／skel 包含了新的帐号的初始化文件，新增用户的主目录被放到／home 目录下，你可以改变这些路径名，通过在窗口左上角的 Usercfg 菜单中 set paths 命令。将会打开一个窗口，列出所有路径名的项，下面是一些路径名：

／home 用户主目录的位置

／mail 用户电子邮件目录的位置

／etc／skel 包含登录 shell 的缺省初始化文件，例如 bash_profile 和 cshrc。

／etc／shell 包含登录 shell，例如 BASH 和 TCSH

19.2.2／etc／passwd 文件

当增加用户帐号时，该帐号的记录项就被写入文件／etc／passad，该文件通常被称为口令文件。该文件中每个记录项占用一行，用冒号分隔各个域，这些域是：

username 用户登户名称
password 用户的加密后的口令
user id 系统分配的唯一的用户数字
group id 用户所属的用户组的数字
comment 用户的附加信息，例如用户全名
home directors 用户的主目录
login shell 用户登录后使用的 shell,缺省的 shell 是／bin／bash

下面是／etc／passwd 文件的记录项的例子。用户 mark 在口令域是*，这意味着该用户的口令还没有被创建。对该项，必须用命令 passwd 来创建一个口令。还应该注意到用户的 ID，在该系统中是以 500 作为基数，每个用户逐一加一。

richp:YOTPd3Pyy9hAc:500:500:Caldera Desktop
User:／home／richp:／bin／bash
mark:*:501:501:Caldera Desktop User:／home／mark:／bin／bash

／etc／passwd 文件是一个文本文件，你可以用任意的文本编辑器进行编辑。你可以修改各个域的值，甚至还可以增加新的用户帐号。唯一的不能直接修改的是口令域，它必须被加密。要改变这个口令域，你应该使用命令 passwd。

虽然可以直接修改／etc／passwd 文件，通常 usercfg 工具更简单更安全可靠，它不仅修改／etc／passwd 文件内容，而且为用户创建主目录和电子邮件目录，并可以在用户的主目录下创建初始化文件。

19.2.3 管理用户环境：／etc／skel

每当用户登录到系统，有两个脚本配置文件被执行。有一个系统配置脚本文件，对每个用户那里一样的，每个用户在其主目录下有一个.bash_profile 文件。系统配置脚本文件位于／etc 目录下，名称为 profile。作为超级用户，你可以编辑该文件，并加入任何命令，以便在每个用户登录到系统时自动执行。例如，你可以定义一个缺省的路径环境变量。

当你第一次增加用户时，你必须为用户提供一个.bash_profile 文件。命令 useradd 会自动完成这个操作--通过在／etc／skel 目录下搜寻一个.bash_profile 文件，并拷贝到用户的新的主目录下，／etc／skel 目录下包含.bash_profile 的样本初始化文件，或者如果你使用 C-shell 对应的是.login 文件和.logout 文件。该目录下也包含 BASH 和 c.shell 的.bshrc 和.shrc，文件。

作为超级用户，你可以任意配置／etc／skel 目录下的 profile 文件。通常这些文件中包含的是系统变量设置及路径变量的设置，命令提示符，电子邮件路

径名及终端的缺省定义，简而言之，就是 PATH，TERM，MAIL 和 PS1 变量的定义。用户有了自己的.bash_profile 文件。就能够随意定义自己的环境。

19.2.4 用命令 adduser 增加用户帐号

adduser 命令是在命令行上输入的，而且非常简单。有许多脚本的 adduser。Redhat 的 adduser 源自 Debian Linux。它的执行与其它脚本--例如 Slackware 有些区别。它的参数是新增的用户名，之后它使用缺省值创建新的用户帐号。通过命令 passwd 可以为新增的用户设置口令。该 adduser 命令是存放于／usr／sbin 目录下的一个 shell 脚本文件。如果你对 shell 编程熟悉，可以通过修改这个文件来改变其缺省值。

```
# adduser robert
Looking for first available UID...503
Looking for first available GID...503
Adding login: robert...done.
Creating home directory: ／home／robert...done.
Creating mailbox: ／var／spool／mail／robert...done.
Don ’t forget to set the password.
# passwd robert
Changing password for robert
Enter an empty password to quit.
```

```
New passwrod(? for help):
New password(again):
Password changed for robert
#
```

19.2.5 用 useradd,usermod 和 userdel 增加和删除用户

其它发行版本的 Linux 可能通过命令的 useradd,usermod 和 userdel 来管理用户帐号，这些命令都在命令行上接收选项的输入。如果没有指定选项，则使用预先定义的缺省值。useradd 命令等待你输入选项，例如用户名。之后，它将使用缺省值创建该用户帐号。

```
# useradd chris
```

useradd 有一些预先定义的缺省值，包括用户组名，用户 ID，用户主目录，shell 目录及用户登录 shell。缺省的用户组名是 other，这意味着用户不属于任何用户组。使用-D 选项的 useradd 命令显示所有的缺省值。useradd 命令对每个值都有其缺省的预定义值，表 19-5 列出了所有 useradd 可以使用的选项。在创建用户帐号时，你可以使用特定的值来代替这些缺省值。增加了新的用户后，需要为其设置口令，否则用户是无法登录的，下面的例子中，chris 用户帐号的用户组名被设置成为 introl，用户 ID 被设置为 578。

```
# useradd chris -g introl -u 578
```

命令 usermod 允许你改变用户帐号的属性，你可以改变用户的主目录或用

户 ID 号，甚至可以改变帐号的用户名。

如果要从系统中删除用户，可以使用命令 userdel。在下面的例子中，用户 chris 将从系统中删除。

```
# userdel -r chris
```

19.2.6 增加和删除用户组

Linux 的 Redhat 发行版本通过 userfg 工具管理用户组。其它的 Linux 系统可能使用 groupadd，groupmod 和 groupdel 命令管理用户组，在文件／etc／group 中列出了所有的用户组，每行一个记录，记录的各个域之间用冒号分隔，用户组的域包括：

group name 用户组名称；必须唯一

password 通常是一个星号，表示任何用户可以加入该用户组；口令可用于接到用户组的访问

group id 系统标识用户组的数字

users 属于该用户组的所有用户

下面是／etc／group 文件的一部分内容。该用户组称为 engines，没有口令，用户组 ID 是 100 ，用户包括 chris，robert,valerie 和 aleina。

```
engines::100:chris,robert,valerie,aleina
```

与／etc／passwod 文件一样，你可以用文本编辑器直接编辑／etc／group 文件。

通过 usercfg 管理用户组

通过超级用户桌面上的 usercfg 工具，你可以轻松地增加，删除和改变用户组。要使用工具 usercfg,首先打开 Usercfg 窗口左上角的 Usercfg 菜单。该菜单有几个选项，其中有一个 Edit Groups。还有一个 Edit Users，要管理用户组，要选择 Edit Groups 项。Group 窗口将被打开，并列出所有系统上的用户组(见图 19-5)。每个记录项有三个域：用户组名称，用户组 ID，和属于该用户组的用户的项目。在该窗口底部有三个按钮：要增加一个用户组，就单击 Add 按钮；要删除或修改一个用户组，就首先单击表中的对应的用户组，之后单击 Remove 按钮以删除该用户组，或单击 Edit 按钮以编辑该用户组。

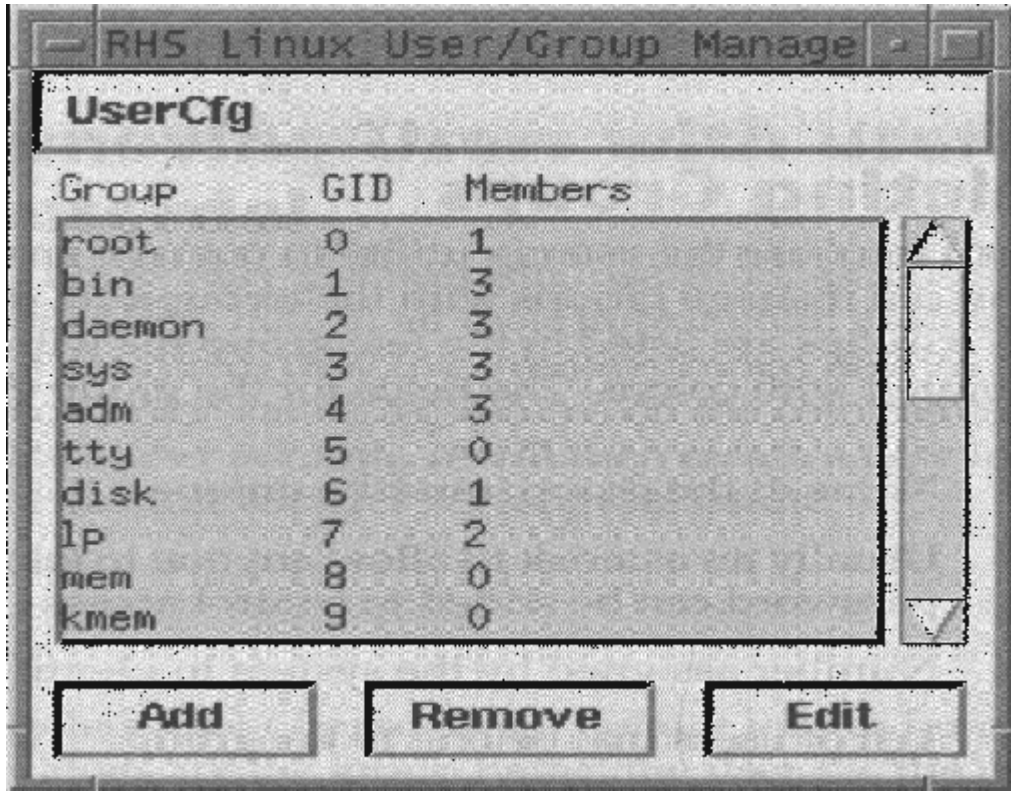


图 19-5UserCfg 组窗口

选择 Add 按钮后，将打开另一个窗口，提示你输入用户组的各个域内容(见表 19-6)。在 Name 域输入用户组名称，在该窗口下部显示的是属于该用户组的用户名称，你可以单击 Add 按钮或 Remove 按钮来增加属于该用户组的用户或从该用户组中删除某个用户。

设置好上面的内容后，单击该窗口左下侧的 Add 按钮，该窗口将关闭，并且对应的用户组将被加到系统中。

Edit 操作与 Add 操作非常类似。你可以修改用户组名和用户组 ID，增加或

删除属于该用户组的用户。

通过 groupadd,groupmod 和 groupdel 管理用户组

通过 groupadd 命令，可以增加一个新的用户组。当增加新的用户组时，系统会将用户组名放入／etc／group 文件中，并为其分配一个用户组 ID 号。命令 groupadd 仅仅创建用户组，其中的用户要分别加入。下面的例子中，groupadd 命令将创建 engines 用户组。

```
# groupadd engines
```

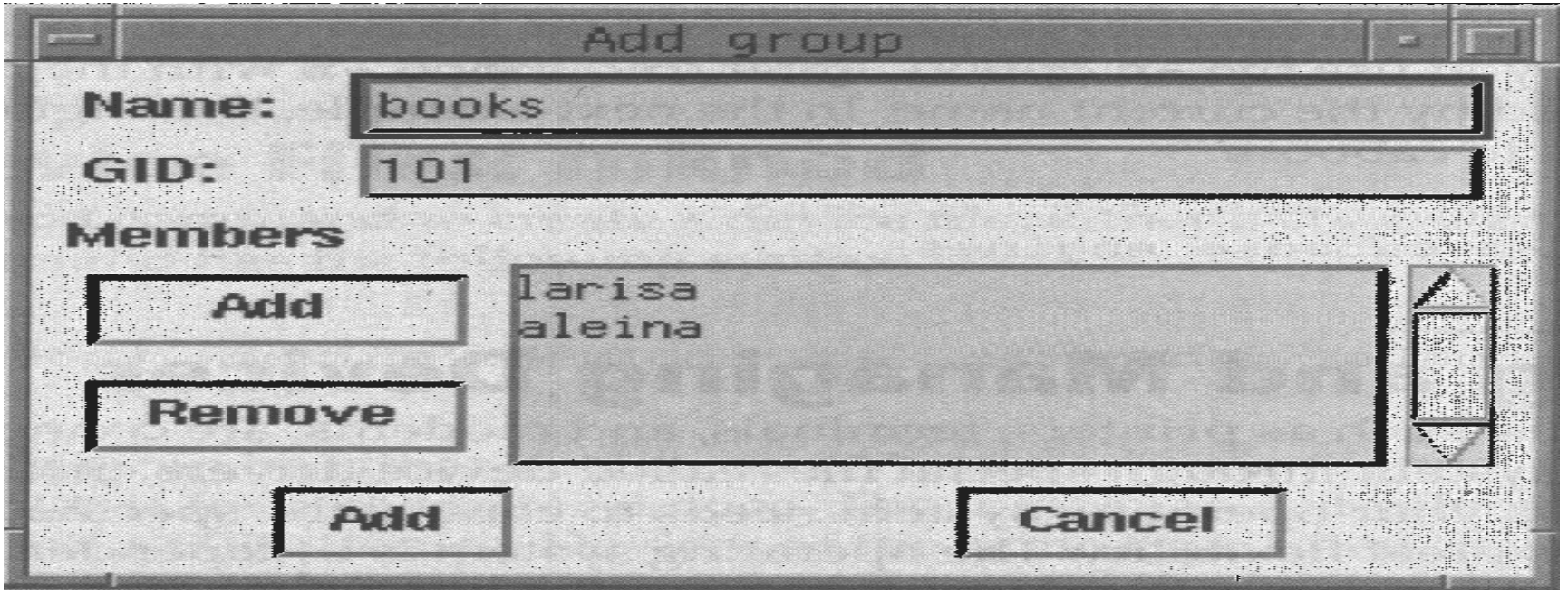


图 19-6 添加新组窗口

通过 groupdel 命令，可以删除一个用户组。下面的例子中，engines 用户

组将从系统中删除。

```
# groupdel engines
```

通过 groupmod 命令，可以改变用户组的名称和用户组 ID。在 groupmod -g 后输入新的 ID 号和新的用户组名称。要改变用户组名称，可以使用-n 选项。下面的例子中，engines 用户组的名称被改变为 caboose。

```
# groupmod -n caboose engines
```

19.3 安装和管理设备

所有的外设，例如打印机，终端，和调制解调器，都是通过被称为设备驱动程序的特殊文件连接到 Linux 操作系统的。这些文件内包含了操作系统在控制特定设备时的需要的所有信息，这种设计带来了很大的灵活性。操作系统本身是与管理特定设备无关的；所有的规定都是由这些设备文件处理的。操作系统仅仅是通知设备要完成什么任务，而设备文件告诉它怎样去完成，如果你要更换设备，那么只要改动设备文件就够了，而无需要改变个系统。

设备文件的名称可以反映出该设备要完成的任务。打印机设备文件的 LP 打头，代表 line print。由于系统中可能连接了多部打印机，特定的打印机设备文件通过在 LP 后面的两个或更多的字母来区分，例如 lp0,lp1,lp2。终端设备文件也是同样的道理，它们以 tty 开头。代表 teletype，通过后面的字母来区分特定的设备，例如 tty0,tty1,ttyso，等等。

在 Linux 系统中，共有两类设备：块设备和字符设备。块设备，例如一个硬盘，每次以块为单位传输数据。字符设备，例如打印机或调制解调器，每次传输一个字符数据，或传输连续的数据流，而不是一独立的块。字符设备文件类型是 c。而块设备文件是 b。 下面的例子中，lp0(打印机)是一个字符设备文件，而 hdal(硬盘)是一个块设备文件。

```
# ls -1 hdal lp0
brw-rw----1 rootdisk3,1 Sep 7 1994 hdal
crw-r-----1 rootdaemon6,0 Dec 31 1979 lp0
```

安装和维护设备，例如打印机和终端是系统管理的一个重要部分，需要装新的打印机，你必须首先的 root 身份登录到系统。你可以更改关键目录--例如／dev 目录--下的文件。

设备文件前缀后面的数字是该设备的唯一标识符。因此不能由自己创建，而只能由操作系统创建。

19.3.1 创建设备文件：mknod

通过 mknod 命令，可以创建设备文件，无论块设备文件还是字符设备文件。mknod 命令的语法是：

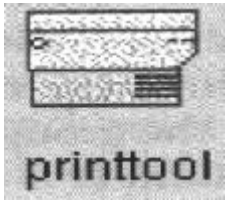
```
mknod options device device-type major-num minor-num
```

设备类型可以是 b,c,p 或 u。前面已经提过，b 表示块设备，c 表示字符设备。u 表示非缓冲字符设备，p 表示 FIFO 设备，同一类型的设备文件通常具有

相同的名称，例如，以太网卡都是 eth，同一类型的不同设备通过名称后面的数字来区别。这个数字有两部分内容，主号码和次号码。更进一步，不同设备文件可能具有相同的主设备号，但次设备号一定不能相同。这种主／次设备号的方法用来处理几个设备依靠一个更大的设备的情况，例如几个调制解调器被连接到同一个 I／O 卡上，它们将具有同一个主设备号来指向该卡，但每个调制解调的必须有唯一的而且互不相同的次设备号。块设备和字符设备都需要主设备号和次设备号(b,c 和 p)，但 FIFO 设备不需要。

19.3.2 安装和管理打印机

Linux 系统可以方便地访问连接到个人机上的打印机，设置打印机的界面是很简单的，无非是决定要使用的设备文件和将打印机配置记录放入 printcap 文件，此外，可能还要设置打印机过滤器。超级用户桌面上的 PrintTool 工具就是管理打印机上的工具(下面是该工具的图标)。



仅通过 PrintTool，你就能在 Linux 系统上安装打印机。下面是具体的步骤：

1.在 PrintTool 窗口选择 Add 按钮，这将打开一个 Edit 窗口，其中列出几个配置打印机区域。

2.在 Name 域，输入打印机名称。每个名称之间用｜分隔，你应该将 lp 作为其中一个名称，lpr 命令后不接具体的打印机名称将使用名称为 lp 的打印机。

3.在 Spool Directory 域中，缺省的值已经显示出来。你可以根据需要改变它的值。

4.在 Printer Device 域中，通常输入／dev／lp1。这是使用 AT 总线的计算机上的并行打印机设备。如果你使用的是串行打印机，则必须使用其它的设备文件。

5.在 Input Filter 域中，单击 Select 按钮将显示一个窗口，该窗口内有三个域，每个域都有对应的 Select 按钮，通过这个按钮可以选择对应域的可能的选项。Printer Type 域的 Select 按钮将列出你可以选择使用的打印机类型。Resolution 域上 Select 按钮将列出系统支持的几种分辨率。PaperSize 的 Select 按钮将列出页面的大小选项，例如信件大小等。

6.配置好之后，单击 OK 按钮关闭该窗口，Edit 窗口也是一样。在 PrintTool 窗口将显示所有的系统上装好的打印机，见图 19-7。

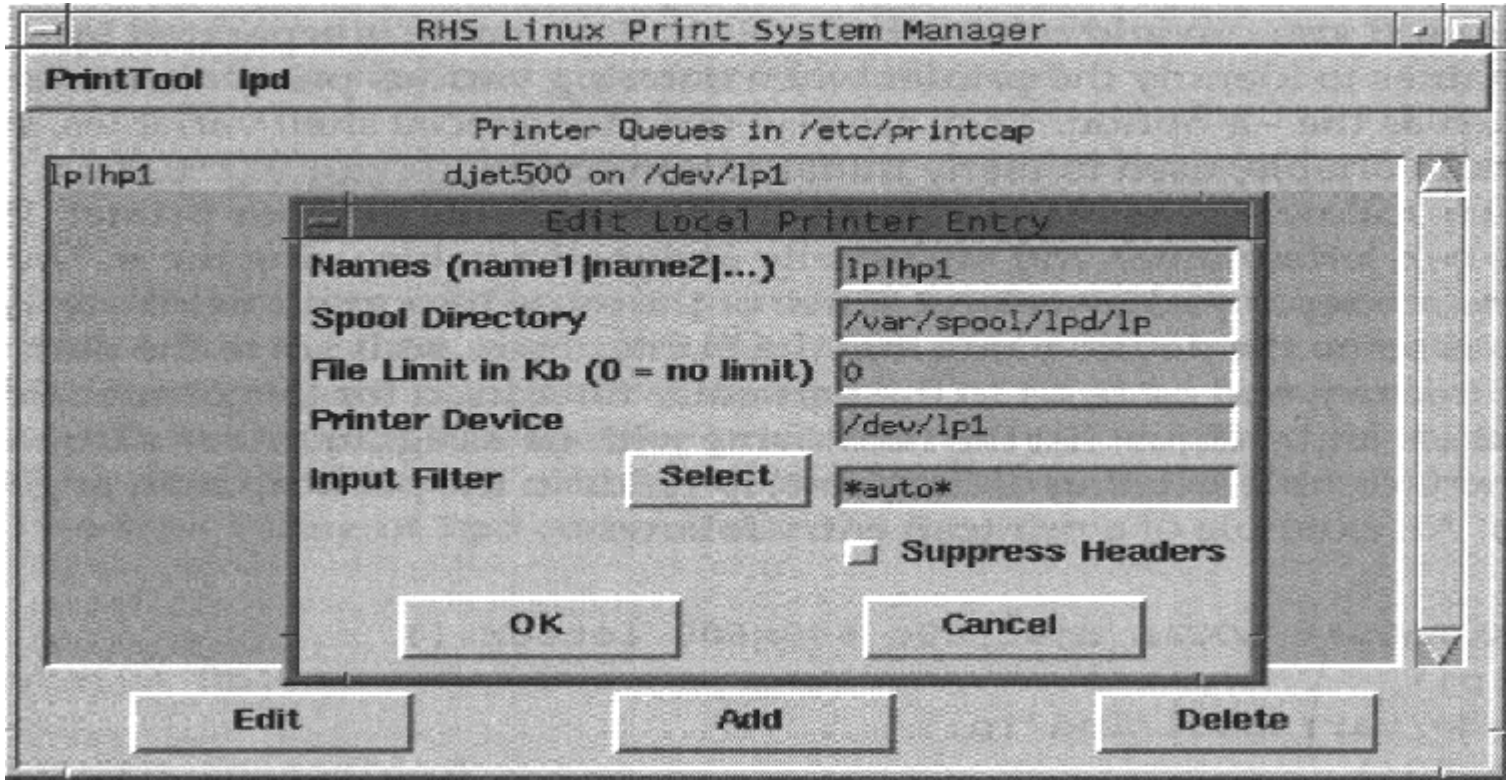


图 19-7 利用 Printtool 安装新打印机

7.在 PrintTool 的菜单中选择 Quit 的退出 PrintTool。现在就能够使用打印机了。

有关打印机的安装更详尽的信息，可参照位于／usr／doc／HowTo 目录下的 Printing-HowTo。

Redhat 系统在安装后自动创建三个并行打印机名称：lpo,lp1 和 lp2(现今大多数系统使用 lp1)这些名称中的数字与计算机中的并行口相对应。lpo 与 LPT1 并行口相对应，地址是 oxo3bc；lp1 与 LPT2 并行口相对应，地址是 ox0378，

lp2 与 LPT3 相对应，地址是 ox0278。如果你不了解你的并行口地址，可通过 DOS 下的 msd.exe 命令找到答案。lpo 连接的是 XT 总线，而 LP1 连接是 AT 总线。

```
# mknod -m 620 ／dev／lp0 c 6 0
# chown root.daemon ／dev／lp0
```

mknod 命令可以创建打印机设备。打印机设备是字符设备，必须归 root 用户和 daemon 所有。打印机设备的权限应该是所有者可读可写，用户组 620 可读(见第 7 章)。主设备号是 6，次设备号被设为打印接连接的端口号，例如 0 对应 LPT1，1 对应 LPT2，创建设备后，要使用 chown 命令将其所有者设为 root.daemon，下面的例子中，创建了在第一个并行口上的并行打印机，／dev／lpo。-m 选项指定其读写权限，620。该设备文件是一个字符设备，因为参数中的设备类型是 c。主设备号是 6，次设备号是 0，如果要创建／dev／lp2 设备，它的主设备号仍是 6，但次设备号应该是 2。

一定要确保为你新建的设备创建了一个缓冲池目录，如果还没有创建，你一定为其创建一个缓冲池目录。

在你的系统打印一个文件时，它使用一个被称为缓冲池目录的特殊目录。打印任务就是一个要被打印的文件。当你要把一个文件进行打印时，该文件的一份拷贝就被放到对应打印机的缓冲池目录中。缓冲池目录所处的位置是通过文件／etc／printcap 中对应打印机的记录项来设定的，针对每一个打印任务，在缓冲池目录中要创建两个文件，一个文件的名字以 df 打头，它的内容就是要被打印的文件的一份拷贝，这是一个数据文件；另一个文件以 lf 打头，它是该

打印任务的控制文件。它包含有关该打印任务的信息，例如，该打印任务属于哪个用户帐号。

文件／etc／printcap 中包含了连接到系统上的每个打印机的定义配置说明。printcap 记录项包含的信息有，该打印机对应的缓冲池目录所有放的位置，该打印机使用的打印机端口所对应的设备名称，等等。记录项的第一个域包含了该打印机的所有可能的名称，这些名称你可以自己创建，你也可以根据需要添加其它的名称。每个名称之间用符号｜来分隔。通过这些名称来标识打印机，另外在特定 shell 变量中也使用这些名称，例如变量 PRINTER，这个变量在许多初始化文件中都会用到。

```
##PRINTTOOL## LOCAL djject500c 600x600 letter ()
hp1｜lp:＼
:sd=／var／spool／lpd／lp:＼
:mx#0:＼
:lp=／dev／lp1:＼
:if=／var／spool／lpd／lp／filter:
```

名称域后面的各个域设置打印置的有关属性。这些域包括两个字符的名称，可用=来为这个名称符值。这些赋值语句之间用冒号来分隔。最为重要的域有三个 lp,sd 和 of。lp 于设置打印机使用的设备文件；sd 用于设置打印机使用的缓冲池目录存放的位置；of 用于设置打印机使用的过滤器。通过命令 man 8 printcap 可以找到 printcap 中各个记录及各个域的详细说明。

不必要直接在／etc／printcap 文件中做修改，可以使用 Redhat 系统提供

的 printtool 工具来自动为你设置／etc／printcap 文件，这个工具可以在超级用户桌面上找到。

系统对打印的处理是由一个称为 lpd 的守护进程负责的。lpd 进程是时刻都在运行的，它等待打印任务的到来，并对相应的打印任务进行处理。lpd 从一个打印队列中获取打印任务，你可以通过命令 lpq 来列出打印队列中存放的内容。lpr 命令会将一个打印任务放入打印队列中，之后 lpd 守护进程会得到它，并打印出来，在第 5 章已提到过，lpr 命令的参数是要被打印的文件的文件名。lpr 命令的-P 选项可以指定特定的打印机，而不是缺省的打印机，在下面的例子中，用户首先打印文件 preface，之后，用户使用 cat 命令显示文件 intro 和 digest 并用道与 lpr 命令连接起来，执行的结果是在打印机上输出这两个文件的内容。最后，用户使用名称为 hpl 的打印机来打印文件 report。

```
$ lpr preface
$ cat intro digest | lpr
$ lpr -Php1 report
```

你也可以直接通过重定向来把某个文件内容送给打印机设备文件。这样做就不会向打印队列中存放任何数据。这时，打印操作就变成立即执行方式。但是，这样会使你的系统一直被占用，直到文件结束打印为止。下面的例子中，使用了这种方法直接打印文件 report。

```
$ cat report > ／dev／lp1
```

要想管理打印任务，输入命令 lpc，并按回车符。这时会出现提示 LPC，你可以在此处输入 LPC 命令来管理打印机和打印任务命令 status 后接打印机名称

会显示打印机是否准备就绪，该打印机上有多少个打印任务，等等。stop 命令和 start 命令可以停止和启动一个打印机。在表 19-6 中列出了 lpc 命令的使用说明。

```
# lpc
lpc>status hp1
hp1 | lp1:
queuing is enabled
printing is enabled
1 entry in spool area
```

通过命令 lpq 和 lprm 可以管理打印队列。lpq 命令列出当前打印队列上的所有打印任务。通过选项-P 可以指定某个特定的打印机。如果你指定一个用户名称，会列出对应用户的打印任务。-l 选项会列出每个打印任务更详尽的信息。如果要得到某个特定打印任务的信息，只要在命令 lpq 后面加上该任务的 ID 号码。

lprm 命令有一个特殊的参数-，它表示该用户的所有打印任务。例如，要想删除你的所有打印任务，只要输入 lprm-即可。如果你是以超级用户身份登录到系统的，那么命令 lprm 一会将所有打印机上的所有打印任务删除。

不要用 lprm 命令终止一个已经开始打印的任务，而可以使用 kill 命令来完成。通过命令 ps-ax command，来显示进程号，再用 kill 命令终止该进程。

19.3.3 安装和管理终端及调制解调器

对于象 Linux 这样的多用户操作系统，同时登录到系统上的用户可能有多个，每个用户都必须具有他自己的终端，用户是通过这个终端来访问 Linux 系统的。你的 PC 的显示器充当一个特殊的终端，称为主控制台，但你可以通过串行口或多用户卡来增加其它的终端。这些其他的终端可以是独立的终端，也可以是运行终端仿真程序的 PC。对终端的安装，最详尽的信息是 Term-HowTo 文件，它存放在／user／doc／HowTo 下。在此处，仅讲述简单的安装信息。

对每一个加入的终端，必须要创建一个字符设备文件。就像前面讲的打印机安装一样，通过命令 mknool 来创建终端设备。终端文件的读写权限是 660。终端设备文件是字符类型的设备文件，它的主设备号是 4，次设备号一般从 64 开始，机器上的串行口被称为 COM1。直到 COM4，它们与终端设备文件／dev／llys0 到／dev／llys3 相对应。要注意的是，这些串行设备可能有的已被其它输入设备--例如鼠标或调制解调器--占用。如果你已经装了一台串行打印机，那么它也会占用一个串行口。在创建终端设备之前，首先要确认它与哪个串行口相连。如果已经装了多用户卡，那么就会有很多可供选择的串行口，一旦创建完终端设备文件，需要将这个设备文件的所有者属性设置为 root.tty。在下面的例子中，用户为 COM3 串行口创建了一个终端设备，／dev／ttys2。

```
# mknod -m 660 ／dev／ttys2 c 4 66
# chown root.tty ／dev／ttys2
```

终端设备是通过 getty 程序和一些配置文件来管理的。当系统启动时，它从

inittab 文件中读取一系列的连接的终端，之后为每个终端执行／etc／getty 程序。getty 程序的作用是为你的 Linux 系统和特定的终端之间建立起通信。它从文件／etc／gettydefs 中获取终端的有关参数，例如波特率，登录提示符及一些专用指令。

Format: <speed># <init flags> # <final flags> #<login>0

string>#<next-speed>

38400 fixed baud Dumb Terminal entry

DT38400# B38400 CS8 CLOCAL # B38400 SANE -ISTRIP CLOCAL #0@ S login:

#DT38400

文件／etc／inittab 包含的指令决定了如何管理终端设备。／etc／inittab 文件中每一行包括四个部分。ID 运行级别，动作和进程。终端设备通过 ID 号标识，第一个设备为 1。终端对应的运行级别一般是 1，动作通常是 respawn，这表示连续运行该进程，对应的进程部分是／etc／getty，后面接的参数表示波特率和终端设备文件。

文件／etc／ttys 把终端类型与某个特定的设备联系起来。它包含两列，第一列表示终端类型，第二列为终端设备名称。终端类型是一个在 termcap 文件中定义的词汇，例如，vt100 代表 VT100 终端，tvi920 代表 TeleVideo 920c。console 代表系统主控制台。终端类型可通过 shell 的 TERM 环境变量来设定。

／etc／termcap 文件包含了对不同终端类型的规定，这里是一些终端的类型，用户通过这些类型的终端登录到系统上，系统上的／etc／termcap 文件中

已经包含了对大多数终端的定义，该文件中的记录项包含通过｜分隔的可用于某个终端的名称，以及一系列参数规定，每个规定用冒号结束。在这里可以找到某个特定终端类型的名称。你可以使用命令 more 来显示系统上的／etc／termcap 文件，之后用搜索命令，／，来定位某个终端类型。

对每个终端类型都可以有许多选项，要改变这些选项，可以使用命令 stty，而不是直接改变配置文件。不加选项的 stty 命令会显示出当前终端的设置情况。

tvi925｜ 925｜ televideo model 925:＼

:hs:xn:am:bs:co#80:li#24:cm=＼E=%＋ %＋

: cl=＼E*:cd=＼Ey:ce=＼Et:is=＼El＼E:＼

:al= ＼ EE:dl= ＼ ER:im=:ei=:ic= ＼ EQ:dc= ＼ EW:if= ／ usr／ lib ／ tabset／ stdcrt:＼

:ho=^^:nd=^L:bt= ＼ ET:pt:so= ＼ EG4:se= ＼ EG0:sg#1:us= ＼ EG8:ue= ＼ EG0:ug#1:＼

:up=^K:do=^V:kb=^H:ku=^K:kd=^V:kl=^H:kr=^L:kh=^^:ma=^V^J^L:＼

:k1=^A@ ＼ r:k2=^AA ＼ r:k3=^AB ＼ r:k4=^AC ＼ r:k5=^AD ＼ r:k6=^AE ＼ r:k7=^AF＼r:＼

:k8=^AG＼r:k9=^AH＼r0=^AI＼r:k0=ic,dc,al,dl,cl,ce,cd,bt:＼

:ts=＼Ef:fs=^M ＼Eg:ds=＼Eh:sr=＼Ej:

当一个用户登录到系统上，最好通过命令 tset 来初始化终端设备。通常，命令 tset 被放在用户的.bash_profile 文件中，这样，每当用户登录到系统就会自动执行这个命令。通过命令 tset 可以设置终端类型以及终端设备所需的其它

选项。通常放在.bash_profile 文件中的 tset 命令带有-mdialup 选项，它会提示用户输入一个终端类型。这个类型有个缺省的类型值，用户可以按回车键来选项缺省的类型值，提示信息会像这种形式，TERM=(vt100)?

```
eval ‘tset -s -Q -m dialup:?vt00 ’
```

19.4LILO

通过配置文件／etc／lilo.conf 和命令 lilo 可以配置 Linux 加载器(LILO)。如果你读过文件／etc／lilo.conf 的内容，就会发现它被分成不同的部分，每个部分是由 LILO 可以启动的操作系统。如果你的 Linux 系统是与 DOS 系统共享的，则会在文件／etc／lilo.conf 中看到两个部分，其中一个是配置 Linux 的，另一个是配置 DOS 系统的。每个部分指定了对应操作系统存放的磁盘分区，此外还定义了对应的卷标，这是在 LILO 提示符中输入的要启动对应操作系统的名称。

你可以根据需要来通过文本编辑器直接修改文件／etc／lilo.conf。改变这个文件后，必须执行 lilo 命令来使它生效。

```
／etc／lilo.conf
#general section
boot=／dev／had
install=／boot／message
message=／boot／message
```

```
prompt
# wait 20 seconds(200 10ths)for user to select the entry to load
timeout=200
#default entry
default=dos
image=／vmlinuz.gen
label=linux
root=／dev／hda3
read-only
other=／dev／hdal
label=dos
table=／dev／had
loader=／toot／chain.b
```

除非通过 default 项来指定，否则 LILO 启动的缺省操作系统总是在 lilo.conf 文件中第一个列出的那个部分。因为 Linux 是在 lilo.conf 文件中第一个列出的部分，所以启动的缺省操作系统是 Linux。如果希望启动的缺省操作系统是 DOS 系统，可使用带有-D 选项的 lilo 命令来重置缺省值，之后编辑 lilo.conf 文件，并为 default 项赋成 dos。一定不要忘记在编辑 lilo.conf 文件后执行 lilo 命令。下一定启动系统的时候，在 LILO 提示符处可以直接按下回车键来启动 DOS 系统，而不必输入 dos。

LILO 具有很多的选项，可以在命令行上使用这些选项，也可以通过设置

lilo.conf文件。在表 19-7 中列出了 LILO 的选项的详细内容。

19.5 总结：系统管理

你可能需要执行基本的系统管理操作，例如增加或删除用户帐号，向系统的文件结构中增加文件系统，设置不同的设备，例如打印机和终端。要具有完成完成这些操作的权限，必须以超级用户的身份登录到系统。root 的登录赋予你全权的控制，允许你改变系统的任何部分。

作为超级用户，你可以管理系统上的其它用户，增加用户帐号和修改用户口令，也可以删除用户帐号，你可以创建用户组或删除用户组，也可以向用户组增加或减少用户。你还可以增加系统上的设备。例如打印机，调制解调器，终端。为了增加新的设备，你需要使用命令 mknod 来为它创建一个设备文件。除此之外，还需要在各种配置文件中作相应的修改，这取决于所加的设备。

超级用户桌面上包含许多用于完成系统管理任务的实用工具。usercfg 工具可以使你轻松地增加、删除并修改用户帐号和用户组。timetool 工具允许你设置系统的日期和时间。printtool 工具能自动地配置打印机，并自动地修改／etc／printcap 文件。netcfg 程序帮助你配置网络接口(在第 20 章将讨论)。fstool 程序帮助你管理文件系统，加载和御载文件系统(见第 7 章)。

表 19-1 系统管理命令

超级用户桌面工具	描述
usercfg	管理用户帐号及工作组，允许新增、删除或修改用户及工作组
timetool	设置系统日期和时间
printtool	配置打印机，新增打印机设备
netcfg	配置网络接口(见第 20 章)
fstool	配置文件系统，允许加载、卸载、新增、删除或格式化文件系统(见第 7 章)
基本系统管理	
su root	从普通用户身份切换至超级用户；超级用户返回到普通用户可通过 CTRL-d
passwd login-name	为用户帐号设置口令
crontab options file-name	以 file-name 作为参数，创建时间精灵记录项；其中选项包括 -e 编辑 crontab 文件 -l 显示 crontab 文件内容 -r 删除 crontab 文件
init state\	改变系统状态(见表 9-3)

续表

lilo options config-file	配置 Linux Loader(LILO)
shutdown options time	关闭系统，类似于 CTRL-ALT-DEL
date	设置系统日期及时间
mknod	创建设备
adduser username	新增一个用户帐户，通过命令 passwd 为新增的用户设置口令
打印机管理	
lpr options file-list	打印一个文件；你要打印的文件放入打印缓冲池中
	-P printer 在指定的打印机上打印文件
	-l 显示打印队列内容
lpq options	显示打印队列内容
lprm	删除某个打印队列
lpc	管理打印机
终端管理	
stty	为终端设备设置不同的选项
tset options	用某个终端类型初始化终端设备

表 19-2 系统管理配置文件

文件	描述
／etc／inittab	设置系统缺省状态及设置终端连接
／etc／passwd	含有用户口令及登录配置情况
／etc／group	包含工作组信息
／etc／fstab	系统启动时自动加载的文件系统
／etc／lilo.conf	系统的 LILO 配置文件
／etc／printcap	包含所有打印机说明
／etc／termcap	包含所有终端说明
／etc／gettydefs	包含终端配置信息
／etc／skel	包含拷贝给用户的类似 bash-profile 文件的目录
／etc／ttys	终端类型列表
系统初始化文件	
／etc／rc.d	包含系统启动，停止文件的目录
／etc／rc.d／rc.sysinit	系统的初始化文件
／etc／rc.d／rc.local	用户自己命令的初始化文件；在该文件中可以加入自己的启动命令；它是最后执行的初始化文件
／etc／rc.d／init.d	含有系统启动，停止文件的目录
／etc／rc.d／init.d／halt	每次关闭系统时执行的文件

续表

／etc／rc.d／init.d／lpd.init	启动和停止 lpd 守护进程的初始化文件
／etc／rc.d／init.d／inet	系统网络接口的初始化文件
／etc／rc.d／init.d／httpd.init	启动、关闭 Web 服务器的初始化文件
／etc／rc.d／init.d／cfs.init	启动、关闭 Caldera 字体服务器的初始化文件
／etc／rc.d／init.d／ipx	启动、关闭 NetWare IPX 守护进程的初始化文件

表 19-3Redhat's 的系统状态

状态	描述
init state	改变系统的状态
系统状态	
0	关机
1	维护式单用户状态，禁止非超级用户登录
2	多用户状态，不支持 NFS
3	完全的多用户状态，支持 NFS

续表

4	未使用
5	仅 X11 使用
6	重新启动
s 或 S	单用户状态，仅允许同时在系统上有一个用户存在

表 19-4 关机选项

选项描述	
shutdown [-rkhncft] time [warning-message]	在指定时间后关机，向系统上用户发出警告信息
参数	
time	可以有两种格式--绝对时间和相对时间。绝对时间的格式为 hh:mm,hh 为小时(1 位或 2 位数字)，mm 为分钟(2 位数字)，相对时间的格式为-m，m 为等待的分钟数
选项	
-t sec	通知 init 等待 sec 秒
-k	不真正的关闭系统；仅仅向系统上的用户发送警告信息

续表

-r	关闭系统后重新启动
-h	停止系统
-n	不通过 init 来关闭系统
-f	快速重新启动
-c	取消刚才的关闭系统动作

表 19-5 用户及组管理命令(对 Redhat's 系统使用 usercfg 或 adduser 命令)

命令	描述
useradd username options	新增用户帐号
usermod username options	修改帐号属性
userdel -r username	删除用户帐号
useradd,usermod 选项	
-u userid	设置用户的 ID；缺省的是当前 ID 最大值加 1
-g group	设置用户组
-d dir	设置用户的主目录
-s shell	设置用户的登录 shell
-c str	为用户设置注释

续表

-k skl-dir	设置初始化框架目录，该目录下的文件，如 prfile 令拷贝到用户的主目录下，缺省的是／etc／skel
-D	显示所有缺省的设置情况
组管理命令	
groupadd	创建一个用户组
groupdel	删除一个用户组
groupmod options	修改一个用户组
-g	改变组 ID
-n	改变组名称

表 19-6lpc 命令

命令	操作
help [command...	] 显示每个命令的帮助信息
abort printers	立即终止一个活动的打印缓冲池
clean printers	删除所有不能打印的临时文件，数据文件和控制文件
disable printer	关闭某个打印队列
down printer message	关闭某个打印队列

续表

enable printers	打开某个打印机
quit or exit	从 lpc 中退出
restart printers	开始一个新的打印守护进程
start printers	打开一个打印机
status printers	显示某个打印机状态只打印队列情况
stop printers	停止一个打印队列
topq printer [jobmum...] [user...]	在打印队列顶部按顺序放入打印作业
up printers	打开某个打印机

表 19-7LILO 选项及 libo.conf 文件

命令行选项	lilo.conf选项	描述
-b bootdev	boot=bootdev	启动设备
-c	compact	加快启动速度
-d dsec	delay=dsec	启动时等待用户输入内容的额定等待时间
-D label	default=label	使用 label 指定的系统核心来启动系统

续表

-i bootsector	install=bootsector	用于新的启动扇区的文件
-f file	disktab=file	磁盘属性参数文件
-1	linear	创建线性扇区地址，而不是扇区／头／柱面地址
-m mapfile	map=mapfile	使用指定的映像文件
-p fix	fix-table	修改损坏的分区表
-P ignore	ignore-table	忽略损坏的分区表
-s file	backup=file	启动分区的替换文件
-S file	force-backup=file	允许覆盖已存的保存文件
-v	verbose=level	增加冗长性
-u		除去 LILO
-v		显示版本号
-t		仅作为测试使用
-I		显示标鉴和核心的路径名称

第 20 章网 络 管 理

Linux 系统通过 TCP／IP 协议与网络连接。这与 Internet 所使用的协议是相同的，与许多局域网使用的协议也是相同的。在第 10 章中已经介绍了 TCP／IP 协议，它是一个健壮的协议集合，用于连接不同操作系统及硬件的系统，这个协议集合始创于七十年代，它是 DARPA 项目的一个特殊部分，用于各大写和研究中心之间进行通信。这个协议最早是在 Unix 系统上开发的，大部分工作是在 University of California,Berkeley P 完成的。Linux 作为 Unix 的一个版本，从这些早期的工作中得到了帮助。

TCP／IP 协议实际上是由许多协议组成的，其中两个协议用于完成特定的任务。两个最基本的协议是传输控制协议(TCP)，它处理发送和接收数据，以及网际网协议(IP)，它处理传输数据。其它的协议提供许多网络服务。域名服务(DNS)提供了地址解析功能。文件传输协议(ftp)提供了文件传输的功能。网络文件系统(NFS)提供了访问远程文件系统的功能。表 20-1 中列出了不同的 TCP／IP 协议。

管理和配置 Linux 系统上的 TCP／IP 网络并不是非常复杂的工作。系统上存在着许多配置文件，通过这些配置文件来设置和维护网络。表 20-2 中列出了

所有这些配置文件。这些文件的大部分可以通过像 netcfg 那样的工具来管理。也可以使用更专业化的程序，例如：netstat 命令、ifconfig 命令，和 route 命令。表 20-3 中列出了常用的网络配置命令。

如果在安装 Linux 系统的时候选择了网络功能并且没有设置错，则应该能够正常工作。如果你的 Linux 系统已经连接到了一个网络，比如以太网，那么你不需要读本章的内容，除非你想理解 Linux 是如何设置网络连接的。如果你通过调制解调器连接到 Internet 服务供应商(ISP)那里，那么本章的 SLIP 和 PPP 部分是很有用处的，它会告诉你如何在 Linux 系统上与 ISP 进行连接。

20.1TCP／IP 网络地址

在第 10 章已经讲过，TCP／IP 地址被分成由点号分隔的四个部分，这被称为 IP 地址。这个地址的一部分是用于网络地址，另一个分用于标识某一个网络内的一个特定的主机。通常，IP 地址的网络部分占用前三段，主机部分占用第四段。这两个部分加起来形成唯一的地址，用它来标识 TCP／IP 网络上的一台计算机。例如，在地址 199.35.209.72 中，网络部分是 199.35.209，而主机部分是 72。该主机是这个网络的一部分，这个网络有它自己的地址：1995.35.209.0。

主机的 IP 地址仅是所需的用于连接网络的一种地址，此外，还需要网络地址，广播地址，网关地址(如果有网关的话)，域名服务器地址，以及子网掩码。

在安装 Liunx 系统的时候，都要求输入所有这些信息，如果你输入了相应的值，就会自动地存放到相应的配置文件中，此外，通过 netcfg 工具设置的值也会自动地被放入相应的配置文件(表 20-2 中列出了这些网络配置文件)。

20.1.1 网络地址

通过主机地址可以计算出它的相应的网络地址，它是主机地址的网络部分，即将主机部分设置为 0。因此，主机地址为 199.35.209.72 的网络地址为 199.35.209.0。

系统是通过子网掩码来从主机地址中获取网络地址的。在位一级上将子网掩码和主机地址进行与操作并将主机部分设为 0，这就得到了主机地址对应的网络地址。

20.1.2 广播地址

广播地址允许一个系统一次地向网络中所有系统发送消息。当网络地址一样，通过主机地址可以计算出对应的广播地址，它就是将主机地址中的主机部分设为 255。因此，主机地址为 199.35.209.72 相应的广播地址是 199.35.209.255。

20.1.3 网关地址

有些网络指定一台计算机作为与其它网络相联的网关。与其它网络相联的所有数据的发送和接收都会经过这台网关计算机。如果你所使用的网络是这种类型，那么你需要提供网关地址。如果你的网络上没有网关，或者你使用的是独立的系统，或通过拨号与 ISP 相连，那么就不需要网关地址。

通常，网关地址与普通主机地址的网络地址部分是相同的，只是主机部分的值为 1。主机地址为 199.35.209.72 对应的网关地址可能是 199.35.209.1。但这仅仅是一种约定，要想确认你的网关地址，必须询问网络管理员。

20.1.4 域名服务器地址

许多网络，包括 Internet，都有域名服务器来把主机名方式的地址转换成 IP 地址。你必须知道你的网络所使用的域名服务器的 IP 地址。从系统管理员那里可以得到这个地址(经常有多个地址)。即使你使用的是 ISP，也必须知道 ISP 为 Internet 执行的域名服务器的 IP 地址。

20.1.5 子网掩码

子网掩码用于获取网络地址，子网掩码通过主机地址作为模板而获取。你的主机地址的网络部分都设置成 255，而主机部分设置成 0，那么这就是你的

子网掩码。因此，主机地址是 199.35.209.72 对应的子网掩码为 255.255.255。网络部分，199.35.209 设置成 255.255.255，而主机部分，72，被设置为 0。之后，系统会根据子网掩码的值来从主机地址计算出网络地址。

20.2TCP／IP 配置文件

在／etc 目录下有一系列文件，在下面已经列出，用于配置和管理 TCP／IP 网络，这些网络包括：主机名和域名，IP 地址，及接口选项。如果你在安装 Linux 系统时已经配置了网络，就可以从这些文件中得到上述的信息。超级用户桌面的 netcfg 工具和命令行上的 netconfig 程序都会自动地设置这些文件。

文件	功能
／etc／hosts	将主机名与 IP 地址联系起来
／etc／networks	将域名与网络地址联系起来
／etc／rc.d／init.d／inet	包含系统启动时配置网络的命令
／etc／HOSTNAME	包含系统的主机名
／etc／host.conf	解析选项
／etc／resolv.conf	包含一系列域名服务器

20.2.1 标识主机名：／etc／hosts

在TCP／IP网络中，主机是通过IP地址来寻址。由于IP地址难于记忆，通常使用的是域名地址。针对每一个IP地址，都有一个域名与其相对应。当你通过域名来访问某台主机，系统会将域名转换成对应的IP地址。这个IP地址才能被用来定位某台主机。

最开始的时候，每台主机都要维护域名和IP地址的对应关系，这样对应关系至今仍然存在于／etc／hosts文件中。当你使用域名的时候，系统会在hosts文件中查寻对应的IP地址。系统管理员有责任来维护／etc／hosts文件的内容。随着Internet的迅猛发展，以及大量庞大网络的产生，就产生了域名服务器，它的作用与／etc／hosts一样，也是将域名地址转变成对应的IP地址。但是，／etc／hosts文件仍旧存在于系统中，并且通常用于域名的解析。在向域名服务器发送服务请求之前，系统总要检查hosts文件来将域名地址转换成IP地址。

hosts文件的格式是：IP地址，后面是对应的域名，中间用空格分隔，后面还可以为主机名加上别名。在每一项记录的最后，可以加入注释内容，注释内容是以#符合开头的一段内容。在hosts文件中总可以找到localhost的一项，它对应的IP地址为127.0.0.1。Localhost是用于标识本地主机的特殊地址。地址127.0.0.1是每个系统用于该目的而保留的一个地址，它标识被称为回送的设备。

／etc／hosts

127.0.0.1turtle.trek.comlocalhost

199.35.209.72turtle.trek.com
204.32.168.56pangol.train.com
202.211.234.1rose.berkeley.edu

20.3 网络名称：／etc／networks

／net／networks 文件中包含的是域名与网络域名的对应关系，而不是某个具体的主机名。随着网络类型的不同，可能使用 IP 地址中的一个，两个或三个数字部分。此外，还要定义 localhost 的网络地址 127.0.0。

该文件中，网络域名后面接的是 IP 地址。要记起 IP 地址是由网络地址部分和主机地址部分共同组成的。网络地址部分会在 networks 文件中找到。

／etc／networks
loopback 127.0.0.0
trek.com 199.35.209.0

20.3.1 网络初始化文件：／etc／rc.d／init.d／inet

／etc／rc.d／init.d／inet 文件中包含有配置网络的启动命令。这个文件的大部分内容都在你使用 netcfg 命令或在安装 Linux 系统时自动建好的，例如，

可以在这个文件中找到 ifconfig 和 route 命令。此外，还可以找到主机名，网络地址，以及其它必备的地址，除非很熟悉 Linux 的 shell 编程，否则不要直接修改这个文件的内容。如果你使用的是其它发行版本的 Linux 系统，这个初始化文件可能是／etc／rc.d／rc.inet1 或／etc／rc.inet1。

20.3.2／etc／HOSTNAME

／etc／HOSTNAME 文件中包含了系统的主机名称。要改变主机名，可以修改这个文件的内容。netcfg 工具允许你更改主机名，并将新的主机名放入／etc／HOSTNAME 文件中。你可以使用命令 hostname 来显示系统的主机名。

```
$ hostname
turtle.trek.com
```

20.4 网络接口和路由：ifconfig 和 route

与网络的连接要通过一个特殊的硬件设备接口，例如以太网网卡或调制解调器。通过这个接口的数据会传送给你所连接的网络。ifconfig 命令用于配置网络接口，而 route 命令会为这个接配置路由信息。netcfg 工具可以完成 ifconfig 和 route 所能完成的工作。如果你使用 netcfg 工具，就不再需要使用 ifconfig 命令和 route 命令。如果你使用的是其它版本的 Linux 系统，那么 netconfig 工

具与 netcfg 的作用是一样的。但是，你还是可以直接使用 ifconfig 命令和 route 命令来直接配置网络。

每次系统启动的时候，必须建立起网络接口以及它的路由信息，你可以将 ifconfig 和 route 命令放入／etc／rc.d／init.d／inet 文件中，这种会在系统启动时自动配置，因为／etc／rc.d／init.d／inet 文件是在系统启动时自动执行的。如果你使用 netcfg 工具,它会自动地把 ifconfig 命令和 route 命令放入／etc／rc.d／init.d／inet 文件中。

20.4.1Netcfg 和 Lisa

创建网络接口的最简单的方法是使用下面的配置工具：Lisa 或者 netcfg。要使用 Lisa 工具，只需在命令提示符处输入 lisa，在这个工具中可以配置域名服务器地址，定义或修改主机名，等等。

也可以使用超级用户桌面上的 netcfg 工具来配置网络接口。只要以 root 用户身份登录到系统，并执行 startx 命令来启动桌面。你会找到一个称为 netcfg 的图标，双击这个图标，它会弹出一个配置网络接口的窗口。在图 20-1 中已经显示出，通过 netcfg 工具，可以增加或修改网络的配置信息。

在 netcfg 界面中显示了网络接口，域名服务器和主机信息，每一项都有其独立的区域，并有其独立的命令按钮。接口区域内列出当前系统上所有的接口。可以通过这个区域下面的按钮来增加、配置、激活或取消某个接口。如果选择增加接口，会弹出另外一个窗口，列出所需要的信息，在此处，你可以输入该

接口的名称和 IP 地址。关闭这个窗口后，会在接口区域中看到刚才新增加的接口。

域名服务器区域弹出当前所有的域名服务器地址。你可以通过按钮来增加或删除某个域名服务器。你所做的修改会自动地被放入文件／etc／resolv.conf 中。窗口的底部列出的是你可以通过主机连接的其它主机的信息。对这些信息的改动会自动存放到／etc／hosts 文件中。

此外，你还可以更改主机名。在 netcfg 菜单中选择 hostname 项，会提示你输入新的主机名。你所做的修改会自动地放到／etc／hostname 文件中，并替换原来的内容。

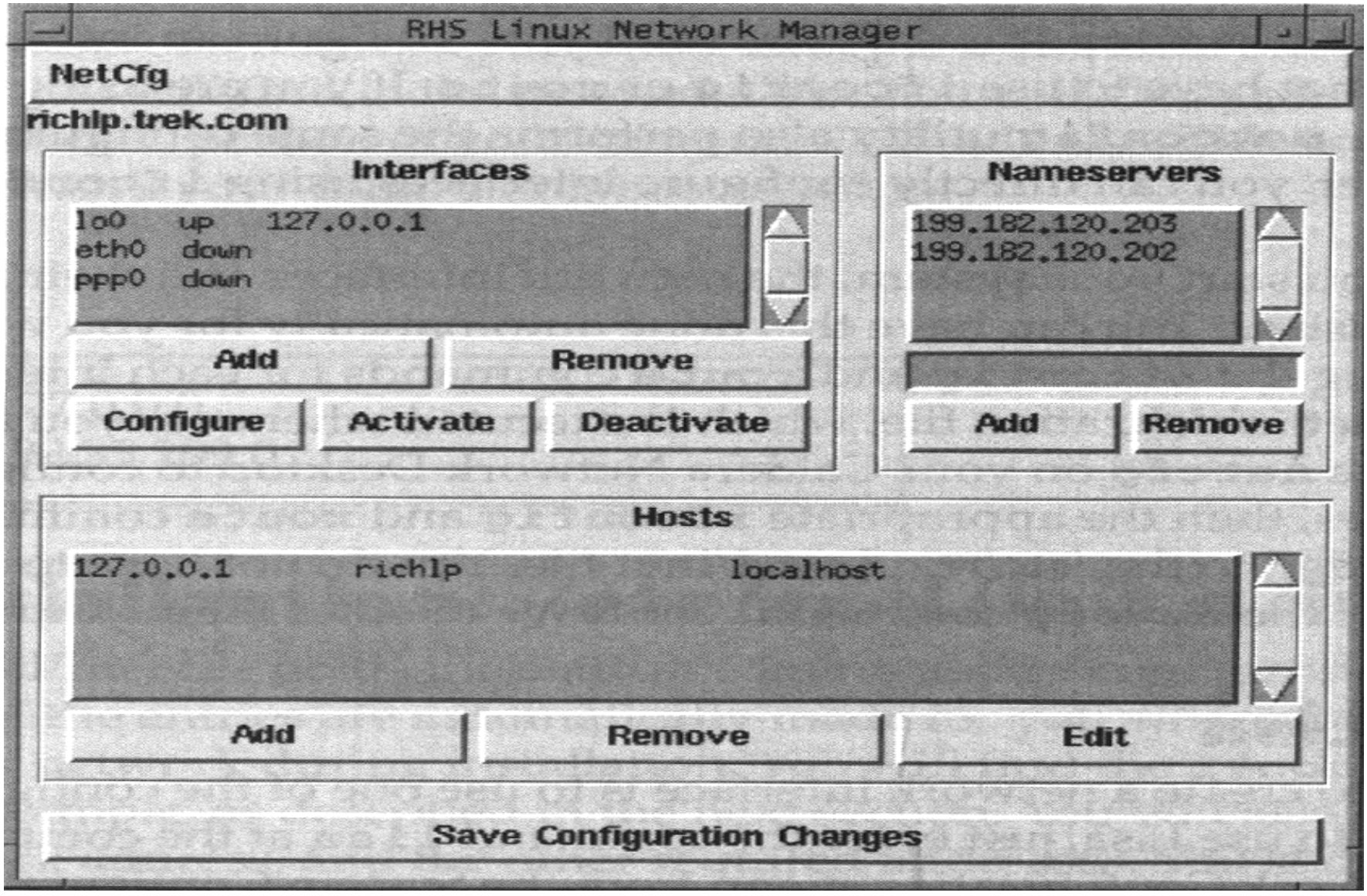


图 20-1netcfg 工具

在做完修改之后，单击窗口底部的 Save Configuration Changes 按钮，以

使这些改变生效。这些改变会自动被加入到相应的网络配置文件中。

20.4.2ifconfig 命令

ifconfig 命令后面接的参数是接口名称，IP 地址的及其它的选项。ifconfig 命令之后以该 IP 地址定义这个网络接口的地址，系统就会知道对应的接口以及这个接口的 IP 地址。此外，你还可以指定这个 IP 地址是一个主机地址还是一个网络地址。ifconfig 命令的语法是：

ifconfig interface -host_net_flag address options

其中 host_net_flag 可以是-net 或 host，来指定 IP 地址类型：是主机地址还是网络地址。缺省的情况使用-host。ifconfig 命令可以带有几个选项，它设置接口的不同特性，例如最大传输单元(mtu)或广播地址等等。up 和 down 选项用于激活或取消这个接口。在下面的例子中，ifconfig 命令用于配置一个以太网接口。

ifconfig eth0 204.32.168.56

对这种简单的配置，ifconfig 会自动地创建一个标准的广播地址和子网掩码。标准的广播地址是网络地址加入值为 255 的主机地址。标准的子网掩码是 255.255.255.0。但是，如果你连接的网络是一个特殊的网，则需要在使用 ifconfig 命令时指定这些值。指定广播地址的选项是 broadcast;指定子网掩码的选项是 netmask。表 20-4 中列出了 ifconfig 命令的所有选项，在下面的例子中，ifconfig 命令包含了子网掩码和广播地址。

ifconfig eth0 204.32.168.56 broadcast 204.128.244.127
netmask 255.255.255.0

点对点接口，例如并行 IP(PLIP)，串行线 IP(SLIP)以及点对点协议(PPP)，需要指定 pointtopoint 选项。PLIP 接口名称的 plip 开头，后接一个数字；例如 plip0 是第一个 PLIP 接口。SLIP 接口使用 slip0，PPP 接口从 ppp0 开始。点对点接口通常用于连接两台主机，例如通过调制解调器连接的两台计算机，在指定 pointtopoint 选项的时候，需要指定主机的 IP 地址。在本章的后面，你将学习如何通过 SLIP 和 PPP 接口拨号并与 ISP 进行连接。

在下面的例子中，PLIP 接口被配置为连接 IP 地址为 199.35.209.72 的主机，另一台主机的 IP 地址为 204.166.254.14。

ifconfig plip0 199.35.209.72 pointopoint 204.166.254.14

你还可以通过 ifconfig 命令来配置回送设备。回送设备的名称为 lo，它的 IP 地址是固定的 127.0.0.1。下面是一个配置回送接口的例子：

ifconfig lo 127.0.0.1

ifconfig 命令对显示接口的状态也是很有帮助的。如果输入 ifconfig 命令，后面接对应接口名称，会显示这个接口的有关信息。

要检查是否回送接口已被配置，只需在命令 ifconfig 后面接回送设备名称，lo。

```
# ifconfig lo   Link encap:Local Loopback
                inet addr:127.0.0.1 Bcast:127.255.255.255 Mask:255.0.0.0
              UP BROADCAST LOOPBACK RUNNING MTU:2000 Metric:1
```

RX packets:0 errors:0 dropped:0 overruns:0
TX packets:12 errors:0 dropped:0 overruns:0

20.4.3 路由

数据包要到达目标地址要经过一定的路由。在大型的网络中，数据包要到达目标地址主机中间要经过许多的计算机。路由决定了这个过程的开始以及要到达目标主机中间哪个计算机要进行数据包的转发，在小型的网络中，路由信息可能是静态的；也就是说，从一个系统到另一个系统的路由是固定的。但是在大型的网络以及 Internet 上，路由信息是动态的。你的系统知道第一次把数据包发送给哪台计算机，而那台计算机从那里接受该数据包，之后再发送给下一台主机，对动态路由而言，你的系统不需要知道很多。而静态路由可能会非常复杂，因为你必须清楚所有的网络连接。

你的路由信息会列在路由表文件／proc／net／route 中，要显示路由表内容，直接在命令行上输入没有任何选项的 route 命令。

```
# route
Kernel routing table
DestinationGatewayGenmaskFlagsMetricRefUseIface
loopback*255.0.0.0 U 0 0 12 lo
pangol.train.com*255.255.255.0 U 0 0 0 eth0
```

路由表中的每一项都是由几个域构成的，提供了例如路由目标和接口类型

等信息。在下面的表中列出了各个域的说明。

域	说明
Destination	路由目标的 IP 地址
Gateway	路由使用的网关的 IP 地址或主机名；*表示没有网关
Genmask	路由使用的子网掩码
Flags	路由的类型：U=up,H=host,G=gateway,D=dynamic,M=modified
Metric	路由长度
Ref	取决这个路由的路由数目
Window	AX.25 网络的 TCP 窗口
Use	使用的次数
Iface	该路由使用的接口类型

在路由表中至少应该有回送接口的一项记录，如果没有，则必须用 route 命令加上。在使用某个接口之前，必须将该接口的 IP 地址加到路由表中。加入地址要使用带有 add 选项的 route 命令。

route add address

在下面的例子中，将回送接口的 IP 地址加入路由表。

route add 127.0.0.1

add 参数有几个选项，在 route 命令的联机手册中有详细的说明。如果你在增加一个特定的静态路由，需要使用这些选项来指定一些特性，例如子网掩码，网关，接口设备或目标地址。但是如果接口已经通过 ifconfig 命令启动，那么 ifconfig 命令能够产生大部分信息。例如，要为一个已经由 ifconfig 命令设置过

的以太网连接配置路由，只需输入-net 选项和目标的 IP 地址。ifconfig 命令通过 IP 地址来定位接口，并用这个信息来建立路由。下面的例子是一个以太网接口的路由。

```
# route add -net 204.32.168.0
```

如果你的系统连接到网络，应该至少有一项缺省路由在路由表中。当其它的路由无法将数据包发送给目标地址时，就会使用这个缺省路由。缺省路由的关键字是 default。

你也可以将路由表中的某项路由信息删除，通过 route 命令带有 del 参数以及该路由的 IP 地址，例如：

```
# route del -net 204.32.168.0
```

20.5 监视网络：ping 和 netstat

通过命令 ping，你可以检查你的主机是否可以与网络上其它主机进行连接。ping 命令会向目标主机发送一个请求，要等待响应。之后目标主机发回响应，并显示到屏幕上。ping 命令会不断地发送请求，直到你用停止命令 Ctrl-c 来停止它。如果 ping 命令不能访问一台主机，会显示一条信息指明该主机是不可到达的。如果 ping 命令失败，则说明你的网络工作得不正常。可能是由于某个网络接口的问题，配置有问题，或者是由于物理连接的问题。使用 ping 命令的方法是输入 ping，后面接的是目标主机名称。

$ ping pang01.train.com

netstat 程序提供了有关系统网络的及时信息，以及网络统计数据和路由信息。netstat命令有几个选项，可以得到不同的信息(见表 20-5)。

```
# netstat
Active Internet connections
ProtoRecv-QSend-QLocal AddressForeign Address(State)User
tcp00turtle.trek.com:01pangol.train.com.:ftpESTABLISHEDdylan
Active UNIX domain sockets
ProtoRefCntFlagsTypeStatePath
unix1 [ ACC ] SOCK_STREAMLISTENING／dev／printer
unix2 [ ] SOCK_STREAMCONNECTED／dev／log
unix1 [ ACC ] SOCK_STREAMLISTENING／dev／nwapi
unix2 [ ] SOCK_STREAMCONNECTED／dev／log
unix2 [ ] SOCK_STREAMCONNECTED
unix1 [ ACC ] SOCK_STREAMLISTENING／dev／log
```

不带选项的 netstat命令会显示系统上的所有网络连接。首先是活动的 TCP 连接，之后是活动的域套接字。域套接字包含配置系统间通信的过程。在下面的表中，引出了六个域。

域	注明
Proto	用于连接的协议 TCP，UDP

Recv-Q　系统接收了但还没有使用的字节数
Send-Q　系统发送给远程主机的但还未确认接收的字节数
Local Address　本地主机名和端口号
Foreign Address　远程主机名和端口号；端口号可以是连接类型，例如 telnet 或者 ftp
(state)　与远程主机连接的状态
ESTABLISHED,连接已经建立
SYN_-SENT，尝试创建连接
SYN_REC，正在创建连接
FIN_WAIT1,关闭连接
CLOSE，连接已被关闭
LISTEN，监听远程的连接请求
UNKNOWN，未知状态
域套接字
Proto　套接字的协议，通常是 unix
RefCnt　当前在套接字中进程的数目
标记
Type　访问的套接字的类型
State　套接字的状态
FREE，套接字没有被使用
LISTEING，等待连接

UNCONNECTED，当前没有连接
CONNECTING，尝试创建连接
CONNECTED，当前连接已经建立
DISCONNECTED，关闭一个连接
Path 进程用的访问套接字的路径名

通过常有-r 选项的 netstat 命令可以显示路由表信息，带有-i 选项的 netstat 命令会显示不同的网络接口的使用情况。下面的表中解释各个信息的含义。

```
# netstat -i
Kernel Interface table
Iface   MTU    Met   RX-OK  RX-ERRRX-DRPRX-OVR TX-OKTX-ERR TX-DRP TX-OVR  Flags
Lo 2000   0      0      0      0      0      58   0      0        0    BLRU
```

MTU	一次传输的最大字节数
RX-OK	数据包被接收没有错误
RX-ERR	数据包被接收并出现错误
RX-DRP	数据包被丢掉
RX-OVR	数据包过多错误
TX-OK	数据包被发送并没产生错误
TX-ERR	数据包被发送并产生了错误
TX-DRP	数据包在传输时被丢掉
TX-OVR	数据包在传输时由于过载而丢掉

标记 接口字母含义：

A，接收多宿目地址的数据包
B，接收广播
D，调试状态打开
L，回送接口
M，杂项模式
N，数据包未加跟踪
O，地址解析协议被关闭
P，点对点接口
R，接口正在运行
U，接口被激活

20.6 域名服务(DNS)

连接到 TCP／IP 网络--例如 Internet--上的百台计算机都是由它的 IP 地址标识的。IP 地址是一套四个数字，用以指定网络地址以及该网络内部主机的地址，IP 地址不容易记忆，因此每个 IP 地址的域名版本也可以用于标识的一台主机。在第 10 章已经指出，域名由两部分组成，主机名和域，主机名是主机的专有名称，域用于标识该主机属于的网络。美国使用的域通常扩展名说明了主机的类型。例如，.edu 用于教育部分，而.com 用于商业用途。国际域通常扩展名用于指定所处国家，例如，.du 代表德国，.au 代表澳大利亚。主机名，域以及扩展名共同组成了一个名称，通过这个名称可以找到一台计算机。而域又可以再分成子域。

网络上的计算机仅能够通过 IP 地址标识，即使它有自己的域名。你可以通过域名来访问网络上的一台计算机，但幕后要在一个数据库中查询以将域名转化为对应的 IP 地址。之后，网络使用这个 IP 地址，而不是域名，来访问对应的计算机。在大型 TCP／IP 网络(例如 Internet)出现之前，每个机器上维护一个 IP 地址和对应域名的信息是可行的，而且方便的。每当访问一个域名，就查找这个文件，从而找到对应的 IP 地址。

随着网络的增大，上述方法变得不太实际，尤其是在 Internet 上。为了提供将域名转换为 IP 地址的服务，产生出域名的数据库并存放到服务器上。为了找到一个域名的 IP 地址，局域名服务器发送一个请求，域名服务器之后会查询

对应这个域名的IP地址，可将其返回。在大型的网络中，可能有多个域名服务器来覆盖网络的不同部分。如果一个域名服务器找不到对应域名的IP地址，它会将服务请求发送给另一个域名服务器。

域名服务器是由解析器查询的，解析器是专门用于从域名服务器中获取IP地址的程序。要想在你的系统上使用域名服务器，必须配置你自己的解析器。你本地的解析器是通过文件／etc／host.conf和／etc／resolv.conf来配置的。

20.6.1host.conf

该文件列出了解析器的选项(在下面的表中已经列出)。每个选项也能有多个域，中间用空格或制表符分隔。在一行的行首如果是#，则表示该行的内容是注释。选项告诉解析器使用何种服务。该文件中各行的顺序是很重要的。解析器会由上至下读取该文件的内容。host.conf文件被存放在／etc目录下。

Order	指定域名解析方法的顺序
Hosts	在本地／etc／host 文件中找
Bind	查询 DNS 域名服务器
Nis	通过网络信息服务协议获取地址
Alerm	检查访问你的系统的远程主机的地址通过 on 或 off 选项将该功能打开或关闭
Nospoof	确认访问你的系统的远程主机地址
Trim	对检查你本地的主机文件，删除域名并仅检查主机名；允许你仅使用主机名，而不是 host.domain.ext 名称
Multi	允许一个主机有多个 IP 地址；通过 on 或 off 选项将其打开或者关闭。

在下面 host.conf 的例子文件中，order 选项指示解析器首先查找本地／etc／hosts 文件，如果失败，查询域名服务器。该系统不含有多个 IP 地址。

```
／etc／host.conf
# host.confile
# Lookup names in host file and then check DNS
order bind host
# There are no multiple addresses
multi off
```

20.6.2 /etc/resolv.conf

在resolv.conf文件中指定域名服务器的地址。在resolv.conf文件中可以有三种记录项，每一项由关键字打头：domain,nameserver和search。对domain项，列出的是系统的域名。如果愿意，可以加入search记录项。search项给出一个主机名后提供一些要尝试的域。

在search项的后面，可以加入域名服务器记录项。你的系统所要访问的域名服务器，都要加在这个记录中。输入nameserver和域名服务器对应的IP地址。通常，网络中有一个主域名服务器和几个从域名服务器。应该首先查询主域名服务器，因此，必须将它放在nameserver项的第一个。

下面是一个resolv.conf的样本文件，该主机的域名berkeley.edu。该域中的域名服务器的IP地址已经列在nameserver记录项中。

20.6.3 配置自己的域名服务器：named

如果你在管理一个网络并需要为其配置域名服务器，可以将Linux系统充当域名服务器的角色，前提是必须启动named守护进程。named守护进程随系统的启动而启动，并等待域名的查询。它使用几个配置文件来处理查询请求。named.boot文件指定该服务器支持的域以及工作文件所在的目录。named.hosts文件包含这个域的数据信息，它是由一些资源记录构成的，这些资源记录列出了该域中有关不同主机的信息。这些记录使用非常 格式，代码放

在适当的域中，named.rev 文件将 IP 地址映射成主机名。named.ca 文件为域名服务器设置高速缓存。配置自己的域名服务器的过程和可能会很复杂的。可以参照 named 的联机手册,How-To 文档等信息。

20.7SLIP 和 PPP

除了使用硬件网络连接，例如以太网，你还可以通过与电话线连接的调制解调器进行连接。有两个协议可以在电话线上传输 IP 数据，它们是串行线 Internet 协议(SLIP)和点对点协议(PPP)，SLIP 是更旧的协议，而 PPP 是新的协议。当前许多高速连接，ISP 们都使用的是 PPPP。SLIP 和 PPPP 协议专门用于使用户通过调制解调器和电话与 Internet 进行连接。通常是连接到 ISP，而 ISP 通过它本身的系统连接到 Internet。

设置 SLIP 或 PPPP 是一个复杂的过程。更详尽的信息可以参照 PPP-HowTo 和 Net -2 -HowTo 文档，存放在／user／doc／HowTo 目录下。在／usr／doc／HTML 目录下还有 Web 页面格式的文档。

20.7.1 准备连接到 SLIP 或 PPP

要创建一个 SLIP 或 PPP 连接，首先必须启动 TCP／IP 网络并已经配置了回送接口。Internet 服务提供商通常会维护域名服务器。你应该知道这些域名

服务器的地址，并加入到／etc／resolv.conf 文件中去。如果没有这一步工作，作输入的任何一个 Internet地址都不会被承认。

20.7.2PPP

点对点协议(PPP)是更新的更广泛功能更强大的协议，并很快流行起来。它提供更稳定的连接，并能支持许多种类的网络协议，而不再是 Internet 协议。PPP 自动完成其大部分功能，与 SLIP 不同，它不必针对每一个步骤都要求有一些命令，PPP 能够自动地确定远程的 IP 地址，静态的或动态的。

通过 pppd 程序来设置 PPP 连接。pppdd 能设置你的连接，设置 MTU 限制，获取 IP 地址。但是，与 dip 程序不同，pppd 不创建初始的连接。它不通过调制解调器进行拨号，也不提供登录和口令信息。要使用 pppd，首先建立与远程主机的连接，可以通过 chat程序来建立连接，它有自己的选项和格式。chat 程序首先建立连接，之后 pppd 对这个连接进行配置。但是，不必先调用 chat，之后再调用 pppd。pppd 被设计成有一个参数，该参数是一个程序，用于建立连接的程序--在这种情况下，这个程序就是 chat。

在命令行上是否为 pppd 程序包含一套 IP 地址及是否使用 noipdefalt 选项可以区分是动态 IP 地址还是静态 IP 地址，如果包含了一套 IP 地址，那么 pppd 就会认为你使用的是静态连接，而且这些 IP 地址是远程和本地的地址，如果未指定任何地址作为参数，那么 pppd 认为存在一个缺省的远程地址和本地地址。缺省的本地地址是你的系统的 IP 地址，它是在文件／etc／hostname 中指定的，缺省的远程主机地址是在文件／etc／hostname 中指定的，缺省的远程主机地

址是在文件／etc／hosts 中指定的。pppd 程序会认为你在使用动态地址，并在建立连接的时候进行查找。为了使 pppd 程序认为在使用动态地址，要使用 noipdefault 选项，同时也不能指定任何地址，noipdefault 选项指示 pppd 不要使用缺省地址，若是没有指定到动态地址。

本地静态地址和远程地址是相邻的，中间用冒号分隔。本地地址放在前面。下面的例子中，指定本地地址为 199.35.209.72，而远程地址为 163.179.4.22。

199.35.209.72:163.179.4.22

如果你使用的是动态远程地址，而使用静态的本地地址，可以先指定本地地址，之后是冒号。pppd 会使用你的本地地址，并接收远程地址。

199.35.209.72:

由于在／etc／hostname 文件中指定的本地地址是你的缺省的本地地址，所以不需要在命令行上输入。只需输入没有地址的 pppd 命令。pppd 会使用你的主机名地址作为你的本地地址，并从远程主机上获取远程地址。

大部分 Internet 服务供应商，如果使用动态 IP 地址，会为你提供本地和远地地址。在这种情况下，你不必输入任何地址，同时必须指定 noipdefault 选项。noipdefault 选项会防止使用 hostname 地址作为缺省的本地地址。没有指定任何地址的情况下，pppd 会从远程主机获取地址。

chat 脚本

使用 chat 的最佳方式是启动一个 chat 脚本。为了创建连接，chat 必须指定所有的连接信息：电话号码，登录提示及用户 ID，口令提示和口令，及所有

连接字符中。可以在 chat 命令的后面输入这些字符串，但这将使命令非常的长而且会变得很复杂，可以创建一个文件，该文件中包含 chat 所需的信息，之后用-f 选项使用这个文件作为 chat 命令的参数。这个文件就称作 chat 脚本。

chat 脚本文件是由一行组成的，在这一行中是由多个部分组成，每个部分包含连接信息的一部分。每个部分是等待--响应对方式的字符串。第一个字符串是你期望要接收的信息，第二个字符串是你要发送的信息。如果期望什么都不接收，只需使用空字符串，。每个等待--响应对完成登录过程的特定任务。你可以初始化调制解调器。大部分用户可以使用缺省的字符串信息。之后，拨对应的号码来创建连接。之后，可以检查连接是否已经建好。在登录提示上输入登录用户名称，最后在口令提示上发送口令信息。

如果要初始化调制解调器，需要在脚本文件中加入对应的等待--响应字符串信息。首先什么都等待，因此等待字符串是空字符串，。在响应字符串中指定调制解调器初始化所需的代码。在初始化字符串中输入代码时，将从&开始的代码退出，对 chat 来说，&是中断符号，它会终止过程。

"" AT＼F2V1L0

下一个等待--响应字符串会拨动对应的电话号码。如果在它的前面有初始化字符串，那么它的等待字符串是 OK，这是从你的调制解调器发来的 OK 响应，表示在初始化调制解调器的过程中没有出现问题。但是如果没有进行调制解调器的初始化工作，那么等待字符串应该是空字符串，，在这两种情况下，响应字符串是调制解调器用于拨号的调制解调器命令，ATDT，后面是对应的电话号码。

OK ATDT5558888

通常会接收到一个连接字符串，提示你必须连接到远程系统。这会随着系统的不同而有所不同。甚至可能没有连接字符串信息。在许多系统上，连接字符串是 CONNECT 这个词；在其它的系统上，是波特率或者速度。在这个例子中，用户接收到的是速度。双连接字符串的响应通常是空的，有时也可能是一个分行符。在表 20-6 中列出了 chat 脚本中的特殊字符。在下面的例子中，响应是一个分行符。

57600 ＼n

在连接字符串之后，远程系统通常会发送登录提示，通常是 login，后面接一个冒号，你只需要后面的几个字符，ogini：不要把冒号漏掉。在响应信息中，你发送你的用户 ID。取决于你的 ISP，你可能还要在用户 ID 后面加入回车符，如 mulogin＼n。有些 ISP 可能还要求你加入其它的字符。

ogin: mylogin

在登录信息之后，会等待口令提示。同样，只需要后面的几个字符，word:。在响应信息中，发送你的口令。如果必要，还应在口令后面加入＼n 作为回车符。

word: mypass

所有这些内容都要放入一行。

＼＼"" AT＼&F2V1L0 OK ATDT8888888 CONNECT ＼n ogin:mylogin word: mypass

如果你没有进行调制解调器的初始化，它看起来可能略有不同。

"" ATDT8888888 CONNECT ＼n ogin: mylogin word: mypass

如果你的 ISP 不需要你输入回车符，可以将回车键去掉，就像下面这样：

"" ATDT8888888 ogin: mylogin word: mypass

在 chat 脚本中，可以将等待--响应对放到不同的行中，每行是一个等待--响应对。这个文件的扩展名要设为.chat。在下面的例子中，mucon.chat 脚本就是将等待--响应对放入不同的行中。

mycon.chat

""AT＼&F2V1L0

OKATDT4448888

CONNECT＼n

ogin:mylogin

word:mypass

之后，可以用含有-f 选项的 chat 命令来调用这个脚本。chat 程序会使用脚本中的信息来初始化你的调制解调器，拨入远程主机，并使用你的用户 ID 和口令登录到远程主机上。

chat -f mycon.chat

如果在连接的过程中出现了问题，许多远程系统会发送出错信息。你可以利用特殊的等待字符串，ABORT，后面接一个关键字来发现这种错误消息，如果收到对应关键字对应的出错消息，chat 会取消连接过程。如果你仅仅使用命令行，可以输入异常退出字符串。在一个 chat 脚本中，异常退出字符串可以放在文件的开始处。下面的例子将等待一个 NO CARRIER 或 BUSY 响应--在登

记提示之前。

mycon.chat

ABORT'NO CARRIER'

ABORTBUSY

""AT＼&F2V1L0

OKATDT5558888

CONNECT＼n

ogin:mylogin

word:mypass

你需要在 pppd 命令后接入 chat 操作。整个 chat 操作会被放入一个引用中，并在 pppd 命令行上的输入。chat 程序会使用 chat 脚本中提供的信息初始化调制解调器，拨号远程主机，并通过你的用户 ID 和口令登录远程主机，chat 将创建这个连接，而 pppd 会为其作配置。

点对点协议守护进程(pppd)有几个选项。它的标准语法是

pppd options serial-device-name speed local:remote-addresses ppp-options

serial-device-name 是你的调制解调器的设备文件名，有可能是／dev／cua 后面接一个数字，通常是 0 到 3，取决于调制解调器使用的端口。端口 1 是 cua0，端口 2 是 cua1，等等。speed 是波特率。对一个 14.4 调制解调器来说，它的值是 14400。对 vi28 调制解调器来说，它是 38400，或者是 56700。查询 ISP 有关你的调制解调器所支持的最大速度。

选项用于指定例如 MTU 大小的及是否接收动态 IP 地址等特性。connect 选项指示 pppd 创建一个连接，它的参数是一个 Linux 命令，这个命令完成实际的连接的工作--通常在 chat 命令，你输入 pppd，之后是 connect 选项，以 R 含有-f chat 脚本选项的 chat 命令。chat 命令和它的-f 选项要放入引号中的与其它的 pppd 选项相区别。在下面的例子中，用户启用 chat 命令选项的 pppd，并使用 mycon.chat 脚本文件。调制解调器被连接到端口中，／dev／cua1，速度是 57600 波特率。

pppd connect'chat -fmycon.chat'／dev／cua157600

要与 ISP 断开连接，只需使用含有 disconnect 选项的 pppd 命令。必须使用 chat 来使调制解调器挂断，要完成这一步，必须发送一个调制解调器命令，例如 HO。将 chat 命令放入 chat 文件中可能会更方便，例如：

#pppd disconnect 'chat -f turnoff.chat'

turnoff.chat

--＼d＋＋＋＼d＼c OK

ATH0 OK

要使断开连接更容易，可以将 pppd 命令放入 shell 脚本，就像为创建连接那样。

PPP 选项

pppd 程序含有许多选项，有关该程序的更常用的选项在表 20-7 中已经列出，或者查看 pppd 的联机手册的获取更加详尽的帮助信息。例如，mru 选项

的功能就是设置最大接收单元的大小，pppd 通知远程的系统在发送数据包的时候每个数据包的最大长度不能超过这个选项指定的长度大小。缺省的长度是 1500,对于速度较慢的调制解调器应该使用的大小是 296 或者 542。defaultroute 选项通知 pppd 程序将 pppp 连接设置成为缺省的路由。如果找不到地址并且使用了 noipdefault 选项，就会使 pppd 程序查找并使用从远程 ISP 处得到的动态 IP 地址。如果你的 ISP 支持动态 IP 地址，那么必须要指定这个选项。crtscts 选项使用硬件流控制，modem 选项使用调制解调器控制线。在命令行上可以列出选项的内容，就像下面的例子这样。

pppd connect 'chat -f mycon.chat' ／dev／cua1 57600 mru 1500 defaultroute noipdefault crtscts modem

在下面的命令中，提供的地址指示 pppd 程序使用这些地址创建一个静态连接。此处的本地地址是 199.35.209.72，而远程地址为 163.179.4.22。

pppd connect 'chat -f mycon.chat' ／dev／cua1 57600 199.35.209.72:163.179.4.22 mru 1500 defaultroute crtscts modem

这样可能会形成一个非常长而且非常复杂的命令行，这取决于你要使用多少种选项。作为更外一种方式，pppd 程序允许在文件／etc／ppp／options 或者在文件.ppprc 文件中输入需要使用的选项内容。pppd 程序会自动地读取这两个文件，并使用对应的选项。你可以根据自己的需要使用任意多的选项，在这个文件中每个选项要占用一行。#开头的行表示该行的内容是注释，即不起作用。文件／etc／ppp／options 中包含了为 pppd 程序作的系统缺省选项。你以超级用户的身份创建这个文件，它是 pppd 程序被调用时要读取的第一个文件。每

个用户可以在他的主目录下创建属于他个人的.ppprc 文件。这个文件是由某个具体的用户指定的，pppd 程序会在读取／etc／ppp／options 文件之后读取用户主目录下的.ppprc 文件。在下面的例子中列出了这些文件的简要内容。

```
／etc／ppp／options
#／etc／ppp／options-*-sh-*-general options for pppd
crtscts
defaultroute
modem
mru 542
asyncmap 0
netmask 255.255.255.0
noipdefault
```

为了使这个过程变得更加方便，Redhat 系统在／etc／ppp 目录下加入了一个文件称为 options.tpl。它是一个选项模板文件。它已经列出了含有缺省值的所有选项，并对每个选项作了详尽的注释说明。但是，所有这些选项都以#符号开头的，也就是说，这些选项都是不起作用的。你不必从头开始创建一个选项文件，而可以将文件 options.tpl 拷贝成 options 文件。之后使用文本编辑器(例如 vi 命令)，将一些#符号删掉，就可以方便快速地创建自己需要的选项文件了，例如，为了使用 noipdefault 选项，通过文本编辑器来找到这一行，并删除行首的#符号。要去掉这种选项功能，只需在行首插入#符号即可。做好了选项文件之后，你就可以启动 pppd 程序，而在该命令后面不再需要加入选项作为参数

了。

由于在文件／etc／ppp／options中已经指定了要使用的选项，因此在pppd程序的时候只需加入chat调动，后面接调解调器对应的设备文件以及调制解调器的速度。

pppd connect 'chat -f mycon.chat' ／dev／cua1 57600

通过将pppd的调用放入一个shell文件中，可以更进一步地简化pppd程序的使用。你应该还记得，shell脚本文件就是一个普通的文本文件，它可以用任何文本编辑器来创建和修改。在shell文件中，你输入命令调用，并加入所有要在命令行上使用的参数。一定要记住，在pppd命令前要加上exec命令。exec命令会在你的命令行shell中运行pppd程序,而不是在该shell脚本文件的shell中运行pppd程序。之后，通过命令chmod 755脚本文件名来使该脚本成为可运行的，现在，要执行pppd操作，只要直接在命令行上敲入该脚本文件名，之后按回车键，在下面的例子中，pppd连接操作被放在一个称作pppcon的脚本文件中，用户只要敲入pppcon就会启动pppd程序。

pppcon

exec pppd connect 'chat-f mycon.chat' ／dev／cual 57600

$ pppcon

现在就应该可以与远程系统建立pppd连接了。在操作过程中，有许多处都可能出现错误。你可能是没有使用正确的连接字符串，或者调制解调器可能初始化过程中有错误，pppd程序会为每一个步骤作记录，并将记录存放到目志文件／var／log／messages中。你可以通过命令tail显示这些信息。在成功的pppd

连接中，你会在这个日志文件中看到本地的 IP 地址和远程的 IP 地址。

$ tail ／var／log／messages

要获取变化的／var／log／message 文件的内容，在使用 tail 命令的时候加上-f 选项，就你下面显示的那样，接下 CTRL-C 会终止这个操作。

$ tail -f／ var／ log／ messages

Mar 23 20:01:03 richlp pppd［208］: Connected...

Mar 23 20:01:04 richlp pppd［208］: Using interface ppp0

Mar 23 20:01:04 richlp kernel:ppp:channel ppp0 mtu=1500,mru=1500

Mar 23 20:01:04 richlp kernel:ppp:channel ppp0 open

Mar 23 20:01:04 richlp pppd［208］: Connect: ppp0 ／dev／cua3

Mar 23 20:01:09 richlp pppd［208］: local IP address 204.32.168.68

Mar 23 20:01:09 richlp pppd［208］: remote IP address 163.179.4.23

Linux 的 Redhat 发行版本为你提供了一个登录 PPP 脚本的例子文件，称作 ppp-on，这个文件存放在／usr／sbin 目录下。你可以使用一个文本编辑器来设置正确的值，例如电话号码和用户密码。在设置 chat 命令和 pppd 程序的选项的时候，也一定要谨慎。如果已经在／etc／ppp／options 文件中设置了选项，就可以在命令行中不使用这些已经设置好的选项，就可以在命令行中不使用这些已经设置好的选项。另外，也要注意为调制解调器设置正确的速度和设备名称。一旦准备工作做好了，剩下的只需输入 ppp-on 命令来创建 PPP 连接。如果在断开连接的时候遇到了麻烦，可以尝试命令 ppp-off。它是一个 shell 脚本命令文件，存放在／usr／sbin 目录下，它直接地杀掉 PPP 进程。

此外，也可以使用 dip 命令来启动 pppd 程序从而创建一个 PPP 连接。当你在 dip 脚本中设置成为 PPP 模式的时候，你实际上是在执行 pppd 程序。在这种情况下，就不再需要一个 chat 脚本来创建这个连接，因为 dip 已经完成了这部分工作了。但是，还需要在 pppd 调用之前设置几个选项。这些选项是不能在 dip 脚本文件中指定的。相反，必须在／etc／ppp／options 文件中设置这些选项。

提供从其它系统来的 PPP 接入连接

跟 SLIP 一样，你可以将你的系统配置成为一个 PPP 服务器，允许远程系统拨号进入你的系统，并创建 PPP 连接。你只需创建一个特殊帐号和一个脚本文件来的选项-detach 和选项 silent 来启动 pppd。这个脚本文件通常被称为 ppplogin，下面是这个文件的一个例子：

```
／etc／ppp／ppplogin
exec pppd-detach silent modem crtscts
```

-detach 选项使得 pppd 不会与正在连接的线断开。silent 选项 pppd 等待远程系统的连接请求。modem 选项会监视调制解调器线，而 crtscts 选项会使用硬件流控制。

这个特殊的用户帐号名是 PPP。／etc／passwd 文件中 PPP 用户的有关信息应该是像下面这样：

```
ppp:*: 501:300:PPP Account:／tmp:etc／ppp／ppplogin
```

PPP 安全：CHAP

为了确保 PPP 连接的安全性，你必须使用额外的协议。已经有两种针对 PPP 的协议：口令认协议(PAP)和响应握手认识协议(CHAP)。

两者相比较而言，CHAP 是更为安全的协议。它使用一个加密响应系统，它需要连接的双方不断地为对方作认识。加密的密钥被存放在文件／etc／ppp／chap-secrets。为了在 PPP 连接中使用 CHAP，在启动 pppd 程序的时候，要加入 auth 选项。此外，还要将远程主机的有关信息输入到文件／etc／ppp／chap-secrets 中。下面是文件／etc／ppp／secrets 的一部分。文件／etc／ppp／pap-secrets 中 PAP 对应的记录项与这个文件的格式是相同的。

etc／ppp／chap-secrets

```
pango1.train.com        turtle.trek.com         "my new hat"
*                       turtle.trek.com         "confirmed tickets"
turtle.trek.com         pango1.train.com        "trek on again"
```

一个 CHAP 密钥记录项由四个域组成：客户端的主机名，服务器的主机名，密钥和一系列可能的 IP 地址。对某个特定的要与你的系统进行连接的计算机，你可以指定它必须提供相应的密钥。如果不指定特定的计算机，可以使用符号*来表示所有的计算机。任何知道密钥的计算机都能够与你的系统进行连接。在下面的例子中的第一项，服务器是用户本身的系统，turtle.trek.com，它允许主机 pango1.train.com 在提供正确的密钥的情况下连接到系统上。在第二个记录

项中，任何系统只要知道密钥 confirmed tickets 就能够连接到 turtle.trek.com。

pango1.train.comturtle.trek.com"my new hat"

*turtle.trek.com"confirmed tickets"

你也必须为你要访问的远程系统创建相应的记录项。在那种情况下，远程系统就是 PPP 服务器，而你的系统就成为客户端。在下面的例子中，turtle 能够连接到 pango1，对应的密钥是 trek on again。

turtle.trek.com pango1.train.com"trek on again"

20.7.3SLIP 和 CSLIP:dip

存在两种类型的 SLIP 连接，标准的被称为 SLIP，此外还有更新的压缩 SLIP，被称作为 CSLIP。一定要清楚 Internet 服务供应商为你提供的是哪种类型的连接。你一定要在连接模式中选择一种或另一种作为协议。除了连接模式不同之外，它们的其它方面都是一样的。

通过 dip 程序来管理和设置 SLIP 连接。如果有合适的信息，dip 就会为你创建到 ISP 的连接。之后，你就可以使用所有的 Internet 服务了，例如 Netscape 或者 ftp。

dip 程序的执行类似于一个解释器。在一个称为 dip 脚本的文件中，你指定一些命令，这些命令将登录到 ISP，并创建连接。dip 程序之后读取这个文件中的命令，分步地执行这些命令。例如，命令 dial 会通过你的调制解调器拨号给你的 ISP。dip 脚本的文件扩展名是.dip，一旦创建好了 dip 脚本，就可以用 dip 命令调用这个文件。也可以使用-v 选项来在每个命令被处理时都显示对应的结

果。下面的内容显示了启动 dip 程序的最基本的语法。

```
$ dip -v scriptfile.dip
```

表 20-8 列出了 dip 脚本中可以使用的各种命令。这里几个比较重要的命令是 port,speed,dial,get,modem 和 default。dip 还使用一些特殊变量来包含连接信息，例如$rmtip，它存放远程系统的 IP 地址。

根据各个 Internet 服务供应商的登录过程的不同，脚本文件的内容也有所不同，各个 ISP 的登录提示符、电话号码和连接字符串都可能不同，但是有两种非常重要的不同的格式，这取决于你的 ISP 提供的是静态 IP 地址还是动态 IP 地址。在设置 SLIP 连接之前，要首先确定你的 ISP 提供的是何种 IP 地址。

如果通过 dip 脚本文件进行连接时有困难，可以以交互的方式运行 dip 程序，对应的选项参数是-t。在这种方式下，dip 程序产生自己的 shell，这个 shell 的提示符是 DIP>。在这个提示符下，每次输入一个 dip 命令。如果另外再加上-v 选项，dip 会显示每个动作的详细描述。对于登录过程，还可以使用 term 命令。这会进入到终端模式，这种模式下 ISP 会直接提示你输入用户标识符和口令。一旦获得了连接，CTRL-] 会返回到 dip 的 shell 提示符下。之后可以继续设置路由和模式。term 模式下，会准确地看到远程 ISP 发送给你的登录提示和口令提示。一旦完成了连接过程，当然会更易于使用 dip 脚本。

```
# dip -t -v
DIP>
```

如果你需要剪切 dip 连接，可以使用带有-k 参项的 dip 命令。此外还可以使用-l 选项来剪切某个特定的连接。dip 的-k 选项会剪切最近一次的连接。如果你

成功地创建了连接，但由于某些原因未能登录或创建线式模式，那么-k 选项是很有帮助的。这个连接将保持激活状态，直到用-k 选项的 dip 命令剪切这个连接。

```
$ dip -k
```

dip 脚本文件中的静态 IP 地址

静态 IP 地址是 ISP 提供给你的需要保存的 IP 地址。你的计算机正是由这个地址来标识的。在通过 SLIP 连接到 ISP 的远程计算机时，你是通过这个静态 IP 地址来标识你的系统的,在通过静态 IP 地址创建的连接中,要为变量 Plocal 赋值为这个静态 IP 地址，并为变量$remote 赋值为 ISP 的远程主机的 IP 地址。你可以使用 get 命令来为变量赋值。在每次 dip 脚本的开始处进行这些变量的赋值。dip 程序会从这些变量中查找各自的 IP 地址。一旦这些 IP 地址被设置好，你就可以继续登录过程的命令。在表 20-9 中列出了 dip 程序承认的各种变量。

```
get $local static-IP-address
get $remote remote-IP-address
```

dip 脚本

下面是一个使用静态地址的 dip 脚本文件的例子，这个例子可以在 dip 命令的联机手册中找到。这个脚本文件的开始部分是一系列设置调制解调器的命令。首先，要判断调制解调器对应的设备。COM1、COM2、COM3、和 COM4 端

口对应的设备文件分别是／dev／cua0,／dev／cua1,／dev／cua2 和／dev／cua3。注意设备文件是从 0 开始的，因此 COM4 对应于设备文件／dev／cua3。调制解调器的速度被设置为 S7600，这是标准的 V128 调制解调器的设置。init 命令指定调制解调器的初始化字符串。在此处要输入针对调制解调器特定的代码。reset 命令接收这个字符串并通过这个字符串来初始化调制解调器。

```
    #For a static IP address assign your system's IP address to the variable
$local
    get $local 199.35.209.72
    #For a static IP address assign the remote system's IP address to the
variable
    $remote
    get $remote 163.179.4.22
    #Set the netmask to 255.255.255.0
    netmask 255.255.255.0
    #Set port and speed.
    port cua3
    speed 57600
    init AT&F2V1M0
    reset
    # The Standard errlvl values:
    #0-OK
```

```
#1-CONNECT
#2-ERROR
#3-BUSY
#4-NO CARRIER
#5-NO DIALTONEwait OK 2
if $errlvl !=0 goto modem_trouble
dial 555-8888
if $errlvl !=1 goto modem_trouble
#Connection made,now login to the system.
wait ogin:20
if $errlvl!=0 goto error
send mylogin\n
wait ord:20
if $errlvl!=0 goto error
send mylogin\n
#W e are now logged in.
wait CONNECT 15
if $errlvl!=0 goto error
#Set up the SLIP operating parameters.
get $mtu 296
#Sets up SLIP connection as default route.
```

```
default
mode CSLIP
print You are now connected to $locip as $rmtip
goto exit
modem_trouble:
print Trouble occurred with the modem...
error:
print CONNECT FAILED to $remote
quit 1
exit:
exit
```

下一步，静态的 IP 地址和远程主机的 IP 地址就分别赋值给变量$local 和$remote。实际的登录过程在此处开始。dial 命令拨指定的号码。如果拨号过程失败，会返回错误信息给$errlvl 变量。对这个变量的值进行检查的工作放在 dial 命令之后，从而实际出现错误时的处理。wait 命令等待连接字符串，这部分内容可能会随 ISP 的不同而有所不同。对有的 ISP，可能是单词 CONNECT；有的可能是波特率，例如 ST600，wait 会一直等待远程主机发送过来的字符串的出现。在连接字符串后面的数字是要等待的时间，单位是秒。如果超过这个时间，wait 命令会退出，而不是无限地继续等待。同样，再做一次变量$errlvl 的检查，以查看是否出现了错误。一定要确保知道这个连接字符串，如果不知道，则询问 ISP，或者尝试使用终端模式来交互地进行登录并查看远程主机发送给

你的内容。作为对连接字符串的响应，你可以发送一个回车。回车符的代码是＼n。对有些远程主机，你还要使用 DOS 版本的回车符，它是由＼r＼n 两个字符来表示的。

下一个 wait 命令会等待远程主机发送给你登录提示符。你只需要提示符的后面部分，而不是全部的单词。通常，这个登录提示符是 login:，你只要指定 ogin: 就行了。在接收登录提示符之后，send 命令会发送用户的登录名称给远程主机。必须在登录名称的后面加上回车符，例如：mylogin＼n。有些 ISP 要求给出完整的登录名称。下一个 wait 命令会等待发送来的口令提示符，它通常是 password，而你只需要后面的几个字符，word:。之后，你通过 send 命令发送你的口令，在口令后面不要忘记加上回车符，＼n。

print 命令可以显示出远程主机的地址。现在就可以配置连接，设置路由及连接类型。default 命令会使当前这个连接成为你的缺省路由。mode 命令会判断连接的类型，这取决于你的 ISP 为你提供的连接类型。你可能使用标准的 SLIP 连接，它用 CSLIP 来表示，也可能使用压缩的 SLIP 连接，它用 CSLIP 来表示。在 mode 命令中使用适当的类型，来创建 TCP／IP 连接。

dip 的动态 IP 地址

动态 IP 地址是你的 ISP 在你连接时为你提供的一个 IP 地址。ISP 保存一个 IP 地址池，在用户进行连接的时候为用户分配。这意味着，在你每次进行连接的时候，这个 IP 地址都可能是不同的，你不可能事先就知道你的 IP 地址。作为连接过程的一部分，你的 ISP 会发送给你这个 IP 地址，之后你可以找到这

个地址，并使用这个地址作为你的本地的 IP 地址。

在你已经与 ISP 建立连接并且已经登录，ISP 会向你的系统发送你的 IP 地址。这时必须获取这个地址，并将变量中 local 赋值与这个地址，带有 remote 选项的 get 命令会为从远程主机得来的变量赋值，在本例子中则为你要连接的 ISP 的远程主机。在你发送口令给远程主机之后，而在设置缺省路由和模式之前，需要输入下面的 get 命令，首先，获取本地主机的 IP 地址，之后是远程系统的 IP 地址。

```
get $local remote
get $local remote
```

在这些命令的后面还可以加入过时数字，以防止远程系统没有发送的情况。下面的例子显示了在 dip 脚本文件中远程和本地 get 命令应处的位置。

```
# We are now logged in.
wait CONNECT 15
if $errlv1 !=0 goto error
# get the dynamically provided IP address for the remote system
get $remote remote
# get the dynamically provided IP address assigned for the local
system,your system
get $local remote
# Set up the SLIP operating parameters.
get $mtu 1500
```

```
# Sets up SLIP connection as default route.
default
```

通过 Netcfg 程序，你可以更方便更简单地配置 dip 脚本。netcfg 窗口的左上角框中显示了系统上所有的网络接口。对 SLIP 连接，你可以增加一个新的接口，并选择 SLIP 类型。有三种选择：Ethernet,SLIP 和 PPP。需要为接口命名，SLIP 接口名称是 sl 后接一个数字，由 0 开始，例如 sl0。可以通过 dip 程序来配置 PPP 连接，但通过在前面讲述的 pppd 程序来配置会更可靠。

命名接口之后，就会出现一个窗口，里面列出了 dip 脚本要使用的各种值，包括电话号码、IP 地址、口令和登录用户名称。在各个域中输入相应的值，之后按下 OK 按钮，这些值被一个专门的 dip 脚本使用，这个脚本文件是／etc／dip-script。激活 SLIP 接口的时候，会自动执行这个文件。如果有必要，你还可以编辑文件／etc／dip-script。你还可以拷贝，修改这个文件。

文件／etc／dip-script 是专门用于静态 IP 地址的。如果你的 ISP 提供动态 IP 地址，必须修改文件／etc／dip-script。必须删除这个文件顶部的两个 get local 和 get remote 项，并输入 get $local 和 get $remote，这两个命令要放在 get mtu 命令之前。

```
wait @@@connect-string@@@ 30
if $errlv1 !=0 goto error
loggedin:
# Lines for dynamic addressing
get $local remote
```

```
get $remote remote
get $mtu @@@mtu@@@
default
done:
print CONNECTED to $remote with address $rmtip
```

提供 SLIP 接入服务：diplogin

通过 Linux 操作系统，你不仅可以通过 SLIP 与远程主机进行连接，而且远程主机还可以通过 SLIP 与你的系统进行连接。其它的主机可以拨号进入你的系统，并创建一个 SLIP 连接。如果为远程主机上的用户在本地创建一个帐号，则该用户就能够拨号进入 SLIP 连接并以对应的帐号登录。这种远程拨号 SLIP 连接可以通过带有-i 选项的 dip 程序来管理，这种选项将使 dip 进入拨入模式，从而接收进入的连接。这种方式的 dip 启动，也可以通过 diplogin 程序来完成，它是 dip 文件的一个符号链接。在拨入模式下，dip 会提示远程用户输入用户 ID 和口令，从而创建 SLIP 连接。

首先，需要在本地主机为远程用户创建一个用户帐号，可以通过 Lisa 实用工具，usercfg 或 adduser 命令创建用户帐号，如果想使系统更加安全，可以将用户的主目录设置为一个特殊的你更容易控制的目录，例如／tem。在下面的内容是／etc／passwd 文件中用户 robert 的记录项，该用户的口令是 starg，主目录是／tmp。

```
robert:starq:204:12:UUNET:／tmp:／usr／sbin／diplogin
```

在该记录项中最后一个域将指定用户的登录 shell，在本例子中，robert 用户的 shell 是 diplogin，它是 dip 程序的符号链接。dip 将在／etc／diphosts 文件中查找用户名，从而获取登录仅配置信息。

一旦创建了用户帐号，下一步就要配置它的 SLIP 连接。这些配置信息存放在／etc／diphosts 文件中，在该文件中，每个记录项由 7 个域组成，各个域之间用冒号分隔。第一个域是用户 ID，下一个是口令。第三个域是主机名或 IP 地址(远程主机的地址)，后面一个域是本地主机的主机名或者 IP 地址，第五个域是网络掩码，第六个域是注释的信息域，最后一个域是该用户的连接参数，例如协议(CSLIP 或者 SLIP)以及 MTU 值。

如果指定了第二个辅助口令，diplogin 程序会在标准用户登录过程前提示输入该口令。

20.8 总结：网络管理

TCP／IP 网络是通过一组工具来配置和管理的，例如 netcfg、ifconfig、route 和 netstat。ifconfig 工具在超级用户状态下运行，允许你完全地配置系统上的网络接口，新增网络接口，以及修改网络接口配置信息。相应的修改会自动放到对应的配置文件记录项中。在表 20-2 中列出了各种网络配置文件。ifconfig 命令和 route 命令是较低层的程序，要有效地使用它们需要你对网络配置有效深的了解。netstat 命令为你提供了有关当前网络连接状态的各种信息。

如果系统上没有直接连接网络上的硬件，例如以太网网卡，而你要通过调制解调器拨入到远程系统上，则需要配置 SLIP 或 PPP 连接。如果你在使用 Internet 服务供应商提供的服务，则必须配置这种连接。通过 dip 程序和 dip 脚本文件，可以容易方便地设置 SLIP 连接。PPP 连接使用 chat 脚本和一个选项。PPP 是一个新的协议，目前正在越来越广泛地应用。

表 20-1TCP／IP 协议

协议	描述
传输	
TCP	传输控制协议；系统之间进行直接通信
UDP	用户数据报协议
路由	
ICMP	Internet 控制报文协议；IP 协议使用的状态报文
IP	Internet 协议；传输数据
RIP	路由信息协议；决定路由
OSPF	开放式首次最短路径，决定路由
网络地址	
ARP	地址解析协议；决定系统唯一的 IP 地址
DNS	域名服务；将主机名解析为 IP 地址
RARP	反向地址解析协议；决定系统的物理地址
用户服务	
FTP	文件传输协议；在不同主机间通过 TCP 进行文件的传输
TFTP	缩小文件传输协议；通过 UDP 在不同系统间传输文件
TELNET	远程登录
SMTP	简单邮件传输协议；在不同主机之间传输电子邮件
网关	

续表

EGP	外部网关协议；为外部网络提供路由信息
GGP	网关到网关协议；在因特网网关之间提供路由信息
IGP	内部网关协议
网络服务	
NFS	网络文件系统；允许加载远程系统上的文件系统
NIS	网络信息服务；在网络在维护用户帐号信息
RPC	远程过程调用；允许程序调用远程主机上提供的服务函数
BOOTP	启动协议；通过网络上提供启动信息服务的主机启动本地主机
SNMP	简单网络管理协议；提供 TCP／IP 配置上的状态报文

表 20-2TCP／IP 配置地址与文件

地址	描述
主机地址	系统的 IP 地址；由此地址来唯一地标识一台主机
网络地址	网络的 IP 地址(IP 地址中网络部分)
广播地址	一次性向网络中所有主机发送信息所对应的 IP 地址
网关地址	网络中网关对应的 IP 地址
域名服务器地址	

续表

网络掩码	
文件	
／etc／hosts	将 IP 地址与主机地址相联系
／etc／networks	将域名与网络地址相联系
／etc／rc.d／init.d／inet	启动时用于配置以太网接口的初始化文件
／etc／host.conf	解析地址选项
／etc／resolver.conf	列出域名服务器的 IP 地址
／etc／protocols	列出系统中存在的协议
／etc／services	列出系统中提供的服务
／etc／hostnome	主机的主机名称

表 20-3 网络管理程序

程序	描述
netcfg	超级用户桌面上的用于配置网络接口的程序
ifconfig	配置网络接口
route	配置路由信息
ping	检查是否可以与另一台主机进行通信
netstat	检查当前网络状态
hostname	显示主机名称
dip	为远程连接建立 SLIP 连接
pppd	创建调制解调器的 PPP 连接

表 20-4ifconfig 选项

选项	描述
interface	接口的名称；通常位于／dev 目录下，例如／dev／eth0
aftype	用于解析协议地址的地址家族；缺省的是 inet
up	激活某个网络接口
down	关闭某个网络接口
-arp	打开或关闭 ARP；在前面加-号表示关闭
-trailers	打开或关闭以太网帧中的跟踪位
-allmulti	打开或关闭混合模式；该选项用于网络监听
metric n	路由成本(目前还不支持)
mtu n	每次传输的最大字节数
dstaddr address	点对点连接的目标 IP 地址
netmask address	IP 网络掩码
broadcast address	广播地址
point-to-point address	接口的点对点模式
hw	设置硬件地址

表 20-5netstat选项

选项	描述
-a	显示所有 Internet 套接字的信息
-I	显示所有网络设备的统计数字
-c	不断地显示当前网络状态，直到该程序被中断
-n	显示 IP 地址
-o	显示定时的状态，过期时间
-r	显示核心路由表信息
-t	显示 TCP 套接字的信息
-u	显示 UDP 套接字的信息
-v	显示版本信息
-w	显示原始套接字信息
-x	显示 Unix 域套接字信息

表 20-6chat选项及特殊字符

选项	描述
-f floname	在 chat 脚本中执行由 filename 指定的文件-l
lockfile	通过 lockfile 创建 UUCP 风格的文件
-t num	设置为 num 秒的时间期限
-v	所有输出到／log／messages 文件中的 chat 动作的描述；

特殊字符

续表

BREAK	向调制解调器发送中断
''	发送空字符串
＼b	删除符
＼c	在响应字符串后不输出换行符
＼d	使 chat 等待一秒钟
＼k	发送中断
＼n	发送换行符
＼N	发送空字符
＼p	暂停 1／10 秒
＼q	字符串不输出到 syslog 文件中
＼r	发送或等待一个换行符
＼s	发送或等待一个空格符
＼t	发送或等待一个制表符
＼＼	发送或等待一个＼符号
＼num	指定八进制的字符
^C	指定一个控制字符

表 20-7pppd 选项

选项	描述
设备名	使用指定的设备；如果设备名没有在前面出现／dev，pppd 会为其自动加上

续表

speed num	设置调制解调器速度(波特率)
asgncamp map	设置异步字符映射
auth	设置远程主机认证
connect 连接选项	设置连接选项，在 Linux 中，此处一般是 chat，它完成正的连接
crtscts	使用硬件流控制
xonxof	使用软件流控制
defaultroute	设置缺省路由
disconnect Linux-command	在 pppd 后运行指定的命令
escape c,c,...	使指定的字符在传输时被忽略
file filename	从指定的文件中读取 pppd 选项
lock	在串行设备上使用 UUCP 风格的加锁
mru num	设置最大接收单元
netmask mask	指定 PPP 网络接口掩码
noipdefault	在 ISP 提供动态 IP 地址时使用。
Passive	使 pppd 等待有效的连接，而不是在不能立即创建连接时退出来
Silent	pppd 等待远程主机创建连接

表 20-8dip 选项及命令

选项	描述
-v	冗长的模式；显示所有动作的描述
-t	测试模式；将使 dip 处于交互的状态下，对应的 shell 提示符是 DIP >
-p mode	设置模式，线协议为 CSLIP 或 SLIP
-a	提示用户输入用户名和口令
-l	充当拨入服务器
-k	杀掉 dip 进程
-l tly-line	指定某条线路被杀掉(需要与 -k 选项配合使用)
-m mtu	设置最大传输单元(MTU)，缺省值是 296
chatkey keyboard code	增加关键字和错误级别代码
config args	直接配置 SLIP 接口
databito bits	连接时使用的位数(缺省值是 8)
default	设置缺省路由
dial telephone number	指定远程电话号码
echo on／off	打开或关闭回显；如果打开回显，则 dip 会将所有发送和接收的数据显示出来
flush	删掉调制解调器中未读的响应
get $var value	设置变量 $var 的值为 value
get $var	设置变量 $var 的值为从连接中接收的下一个值

续表

get $var ask	提示用户输入变量的值
goto label	在 dip 脚本中跳转到 label 行
help	列出 dip 命令
if $var operator number	测试变量的值；数字 number 必须是一个整数
init string	为调制解调器初始化字符串
model SLIP／CSLIP	设置连接使用的协议模式并将 dip 放入守护进程的运行状态
modem type	设置调制解调器类型：HAYES
netmask	为 dip 使用的路由设置网络掩码
parity E／o／N	设置奇偶校验为偶／奇／无
password	提示用户输入口令
print	显示信息
port dev	设置 dip 将使用的端口
quit	退出 dip 程序
reset	向调制解调器发送 init 字符串
send text	向远程主机发送 text 字符串
sleep number	睡眠 number 秒
speed number	为连接设置波特率：2400.9600.38400.56700
stopbit bits	设置停止位的个数
timeout number	设置过时秒数

续表

term	将 dip 置为终端仿真模式
wait word numberdip	等待指定秒数内出现指定的字符串接收

表 20-9dip 变量

变量	描述
$loca1	本地的主机名
$locip	本地主机的 IP 地址
$remote	远程系统的主机名
$rmtip	远程系统的 IP 地址
$mtu	最大传输单元
$modem	使用的调制解调器类型(只读的)
$portdip	使用的串行设备名称(只读的)
$speed	串行设备的传输速度(只读的)
$errlvl	包含上一个命令返回的结果码；通过它可以测试命令运行的结果；0 表示成功(只读的)

第 21 章配置 X-windows 系统

X-windows 设计得十分灵活，用户可以通过各种方式对它进行配置。它可以在目前所见的大多数显示卡，甚至在图形加速卡上运行。X-windows 不拘泥于任何特定的桌面界面。它提供一套底层的图形操作集，由那些用户界面应用程序（诸如窗口管理器和文件管理器）使用它。窗口管理器利用这些操作来构建控制窗口的饰件，诸如滚动条，尺寸调整按钮，以及关闭按钮。不同的窗口管理器能够构建出不同外形的饰件，从而提供不同的界面。所有的这些都能在 X-windows 上运行。有许多不同的窗口管理器供用户选择。在系统中，每个用户可以运行一种不同的窗口管理器，但所有的用户都使用同一个 X-windows 底层图形操作集。用户甚至能在没有任何窗口管理器和文件管理器的情况下运行 X-windows 程序。

为了运行 X-windows,必须安装与系统显示卡匹配的 XFree86 服务器，并提供有关当前系统的显示器，鼠标以及键盘的配置信息。这些信息存放于 ／etc ／XF86Config 这个配置文件中。文件使用的技术信息最好由一个 X-windows 配置程序生成，如 XF86Setup。在用户安装系统的过程中进行 X-windows 配置时（见第 2 章），这个文件会自动生成。

用户可以通过.xinitrc 和／usr／X11R6／lib／X11／xinit／xinitrc 这两个配置文件来配置自己的 X-windows 界面，通过这两个文件中用户可以选择并且启动窗口管理器，文件管理器，以及初始 X-windows 应用程序（在 OpenLinux 系统里，目录 usr／X11R6／lib／X11／xinit／是目录／etc／X11／xinit／ 的一个连接，文件实际存放在目录／etc／X11／xinit／里。）还有一组特殊的 X-windows 命令，用户可以用它们来配置根窗口，载入字体，或者配置 X-windows 资源：如设定窗口边界颜色。也可以从在线资源下载 X-windows 实用程序，那些作为 Linux 镜象站点的在线资源通常在他们的 ／pub／Linux／X11 目录下。如果必须编译一个 X-windows 应用程序，用户可能必须采取特殊的步骤并且安装相应的支持软件包。除了可以配置 X-windows，用户也可以配置窗口管理器。窗口管理器 fvwm 有它自己的一套配置文件，可以用它们来增加任务栏按钮或增加菜单项，甚至建立键盘映射。

21.1XFree86 服务器

XFree86 服务器支持的显示卡和显示器范围很广。有支持单显,VGA ,SVGA 的服务器，有为图形加速卡设计的一系列服务器，每种芯片组对应一种服务器。表 21-1 列出了现有的 XFree86 服务器。如需要更加详细的信息，可查看位于目录 ／usr／X11R6／lib／X11／doc 的 XFree86 文档。在那儿能找到对应于各种服务器的文件以及各服务器所支持的显示卡类型。文件 AccelCards 列出

了所有目前被支持的硬件，包括芯片组；文件 Monitors 列出了显示器配置信息。也可以查看不同驱动程序的 man 帮助页。各驱动程序及相应的 man 帮助页在此列出：

显示卡 man 帮助页

图形加速卡 XF86_Accel

单色显示卡 XF86_Mono

VGA 显示卡 XF86_VGA16

SVGA 显示卡 XF86_SVGA

安装系统时所指定的 XFee86 服务器会自动安装。用户必须使用 Lisa,glint, 或者 rpm 来显式地安装其它服务器。他们在 CD-ROM 的目录 OpenLinux／install／xbasis1 里。每个包含 XFree 服务器的包以 XFree86"为名字的开头。用户必须安装与当前的图形卡匹配的包。大多数标准显示卡能在 SVGA 服务器-XF86_VGA 工作。如果用户正在使用简单的单色或 VGA 显示卡，只要安装一般的 XF86_Mono 或 XF86_VGA 服务器。但是，如果装有图形加速卡，必须首先确定它使用的是什么芯片组。查看随卡买回来的手册或文档。如果不能确定芯片的类型，可以使用一个叫作 Superprobe 的实用程序，它会分析当前的加速卡，确定所需信息。首先，看一下 Superprobe 的 man 帮助页以便在启动它之前确定正确使用的参数。一个特定服务器的名字通常包含它所支持的芯片组的名字。例如，如果正在使用的图形卡上有 S3 芯片组，应使用 XFree86_S3 包。 如果使用 ViRGE 芯片（Diamond Stealth 3D 2000）,应使用 S3V 服务器。在某一些情况下，不同品牌的图形卡使用同一种芯片。如果用户在 CD-ROM 中

没有找到所需的服务器包，可以查看任一 Linux 镜象站点（通常在目录 ／pub ／Linux／X11／servers 下面）寻找更新的版本。

rpm-iXFree86_S3-3.2-1.i386.rpm

21.2／etc／XF86Config 文件

一旦安装了 XFree86 服务器，下一步就该配置 X-windows 界面了。这会涉及到配置文件 XF86Config 的生成。它位于／etc 目录下，包含了有关当前图形卡、显示器、键盘以及鼠标的所有详细说明。用户可以直接创建这个文件，但最好使用诸如 XF86Setup 或 XF86Config 之类的实用配置程序。用这些实用程序配置时,只需回答有关硬件的一些问题,然后程序会生成相应的 XF86Config 文件。

用户手头上应该有关于当前系统硬件的详细信息。对于显示器，用户必须知道它的水平和垂直同步扫描频率范围以及带宽。对于图形卡，应该知道它的芯片型号，有时甚至还应知道时钟频率。对于鼠标，应该知道它是 Microsoft 兼容系列还是其它品牌，诸如 Logitech。同时还应知道鼠标所连接的端口号。

XF86 Setup 提供给用户的是一个全屏幕界面，用户可以非常轻松地为鼠标、键盘、图形卡和显示器选择各项特性。第 2 章中详细介绍了 XF86Setup。但是如果在用 XF86Setup 进行配置时碰到问题，用户可以使用 XF86Config 。它的界面是行模式提示符形式，可以敲入字符作为回应或者输入一个菜单选项。

每一步都有提示。

文件／etc／XF86Config 分为如下所示的几个节。用户可以在 XF86Config 的 man 帮助文档中找到关于所有这些节和对他们各项的详细讨论。所有这些都是 XF86Setup 程序建立的。例如，显示器配置屏生成／etc／XF86Config 文件中的显示器节，鼠标配置屏生成鼠标节，其它类推。文件里每一个节以关键词 Section 开头，紧接着是引号括起来的这一节的名字。每一个节以 EndSection 结束。注释行都以#开头。每一个节的各项都以数据说明开头，然后是一系列取值。例如，在列出 rgb 色彩数据的 File 节，以数据说明 RgbPath 开头，然后是 rgb 色彩数据文件的路径名。

Files	存放字体文件和 rgb 色彩数据的路径
Module	动态模块载入
ServerFlags	杂项
Keyboard	键盘配置情况
Pointer	鼠标配置情况
Monitor	显示器配置情况(设置水平和垂直扫描频率)
Device	显示卡配置情况
Screen	屏幕显示配置情况，如虚拟屏幕、显示颜色、屏幕大小等

尽管用户可以用标准的文本编辑器直接编辑这个文件，但使用诸如 XF86Setup 之类配置程序进行修改总是最好的。文件中的大多数节用户可能从来不会去动它。但是用户可能不得不对 Screen 节进行一些修改，这一节在文件的最后。为了这样做，可以编辑文件并增加或修改 Screen 节的各项。正如

第 2 章中指出，为了配置虚拟屏幕显示和确定所支持的颜色数，必须对 Screen 节进行一些修改。由于 Screen 节是用户最有可能修改的节，所以尽管它出现在文件最后，但我们还是最先讨论它。

21.2.1Screen 节

Screen 节首先是 Driver 项，它指定了驱动程序的名字。这里有五种驱动程序，每一种对应于一种 XFree86 服务器：Accel,Mono,SVGA,VGA2,以及 VGA16。驱动程序 Accel 用于诸如 S3_XFree86 之类的所有加速型 X 服务器。Mono 用于 XF86_Mono 服务器支持的非 VGA 单色驱动。VGA2 和 VGA16 用于 VGA 服务器，SVGA 用于 XF86_SVGA 服务器。如果正在使用 XFree86_SVGA 服务器，Driver 项将是 svga"。如果正在使用任意一种加速型服务器，这一项将是"Accel".诸如 XF86Setup 之类的配置程序将为以上这些的每一种情况生成 Screen 节。如果有一块加速卡并且能为它安装服务器，那么 X-Windows 将使用 Accel 屏幕节。那就是用户要进行修改的地方。如果正在使用 SVGA 服务器，将使用 SVGA 屏幕节。

在 Driver 项之后是 Device 和 Monitor 项，它们分别指定用户正在使用显示器和显示卡。在这些对应的节中，Identifier 项给出的名字用于引用那些部件。具有 Identifier 项 Nec3v 的显示器在 Screen 节中将会有 Monitor Nec3v 项。

```
Section "Screen"
Driver"Accel"
```

Device"Primary card"
Monitor "Primary Monitor"
DefaultColorDepth16
SubSection"Display"
Depth8
Modes"1152x864" "1024x768""800x600""640x480"
"640x400""480x300""400x300" "320x240" "320x200"
Virtual800 600
EndSubSection
SubSection"Display"

Screen 节下有 Display 子节，指定所支持的颜色深度。前面各部分是配置硬件的，而 Display 子节则配置一些显示特征，例如显示颜色数，虚拟屏幕尺寸。子节中主要有三项：Depth,Modes,以及 Virtual.。Depth 项指明屏幕分辨率：8，16，24。不管当前的 X 服务器支持什么显示卡，用户可以加入 DefaultColorDepth 项为服务器设定缺省颜色深度：8 表示 256 色，16 表示 32K 色，24 表示 16M 色。Modes 项指定在所给分辨率下允许的显示模式。Virtual 项指定虚拟屏幕的尺寸。用户能够拥有一个比正常显示面积大的虚拟屏幕。当移动鼠标至显示屏幕边缘时，它将向那部分隐藏的屏幕移动。这样就能够拥有一个比显示器物理尺寸大得多的工作屏幕。通常将虚拟屏幕设定为 1024 x 768，相当于 17 英寸显示器的尺寸。也可以设为 1152 x 864 ，相当于 21 英寸显示器的尺寸。如果想取消虚拟屏幕，可以把 Virtual 项设为 800 x 600，使虚

拟显示尺寸和屏幕物理尺寸大小相等。

Virtual 1024 76817 英寸虚拟屏幕

Virtual 1152 86421 英寸虚拟屏幕

Virtual800 60015 英寸屏幕（取消虚拟屏幕）

可以安全地改变这一节中的任何一特性。事实上，为了改变虚拟屏幕尺寸，不得不修改这一节。但是，除非用户对自己正在做的事很有把握，不要动 XF86Config 文件中的其它节。

21.2.2Files,Modules,ServerFlags,以及 Keyboard 节

Files 节列出了 Xfree86 所需资源的所在目录。这些资源大多数是系统能够提供的字体。一个字体项以数据说明 FontPath 开头，然后紧接着是那种字体的字体文件路径名。下面是这些项的一部分：

RgbPath"／usr／X11R6／lib／X11／rgb"'

FontPath"／usr／X11R6／lib／X11／fonts／misc:unscaled"

Module 节指定了一些动态加载的模块。Linux 的 ELF 系统支持动态模块。Load 项将加载一个模块。详细情况请查看 XF86Config 的 man 帮助页。

可以为 XFree86 服务器设置一些标志。用户可以在 XF86Config 的 man 帮助页中找到所有的这些标志。例如，标志 NoTrapSignals 允许清除内核，以便调试程序。标志 DontZap 禁止用 CTRL-ALT-BACKSPACE 组合键关闭服务器。DontZoom 标志禁止图形模式之间的切换。

Keyboard 节决定的键盘类型并且设定布局，模型，以及所用协议。例如，下面这一项就设置了布局。这一节有很多选项。如想得到全部清单，可参考 XF86Config 的 man 帮助页。

XkbLayout"us"

21.2.3Pointer 节

Point 节配置鼠标和所有其它定位设备。这一节只有几个项，其中一些用于定制某些特定类型的鼠标。Protocol 项指定鼠标所使用的协议，诸如 Microsoft 或 Logitech。Device 项指定了鼠标设备文件的路径名。下面的例子显示了一个标准的 Pointer 节，它配置的是一个波特率为 1200 的 Microsoft 鼠标。设备文件为 ／dev／mouse。

```
Section  "Pointer"
 Protocol     "Microsoft"
 Device       "／dev／mouse"
 BaudRate      1200
EndSection
```

以下列出了 Pointer 节各项：

Protocol	鼠标协议（运行 man XF86Config 可得到完整的清单）
Device	设备文件路径，比如说 ／dev／mouse 或 ／

	dev／cua0
BaudRate	串行鼠标的波特率
Emulate3Buttons	允许二键鼠标通过同时按左右键来模拟三键鼠标
ChordMiddle	配置某些 Logitech 三键鼠标
ClearDTR 和 ClearRTS	清除 TDR 和 RTS 行，只对 Mouse Systems 鼠标有效
SampleRate	设置抽样率（logitech）

21.2.4Monitor 节

在用户的系统上使用的每一台显示器都应该有一个 Monitor 节。正如第 2 章中指出，垂直和水平扫描频率必须精确，否则可能会损坏显示器。Monitor 节开始的几项用来标识显示器，比如销售商、型号等。HorizSync 和 VerRefresh 项指定了垂直和水平扫描频率。大多数显示器支持多种分辨率。Monitor 节中的各 ModeLine 项指定了那些分辨率。每一项 ModeLine 对应一种分辨率。每项有五个值，分辨率名称，时钟频率，然后是两组值（每组四个值），一组是水平定时参数，一组是垂直定时参数，以标志位结束。标志位指定显示模式的各种特征，比如 Interlace 表明显示模式是隔行扫描的，而＋hsync 和＋vsync 则是用来选择信号的极性。

Modeline"name"dotclockhorizontal-frepvertical-frepflags

如下是 ModeLine 的一部分。最好不要改动 Monitor 节，而完全由 XF86Setup

配置程序生成各项。

Modeline "800 x 600" 50.0 800 856976104060063764366 6 + hsync + vsync

这儿列出 Monitor 节中通常使用的一些项。

Identifier 显示器标识名

VendorName 生产厂家

ModelName 样式和型号

HorizSync 水平扫描频率；可以是一个范围或一系列值

VerRefresh 垂直刷新频率；可以是一个范围或一系列值

Gamma Gamma 校正

ModeLine 通过指定时钟频率、水平定时参数、垂直定时参数来确定相应的分辨率

21.2.5Device 节

Device 节指定了系统显示卡。Device 节的开头几项用来标识显示卡，比如 VendorName(销售商)、BoardName（显示卡名称）、Chipset(芯片组)。VideoRam 项指明了显存大小。Clocks 项列出了一组系统的时钟值。这一节中可以设定很多不同的项，比如，如果显示卡上有 Ramdac 芯片，可以用 Ramdac 项指定 Ramdac 芯片的类型；如果允许访问主缓冲区，MemBase 项指出了主缓冲区的基地址。详细清单和描述可参考 XF86Config 的 man 帮助页。尽管用

户可以安全地修改 VideoRam 项--例如，增加了显示卡的内存--，但修改 Clocks 项却不太安全。如果设错了时钟值，很容易损坏显示器。所以还是由 XF86Setup 或其它 XFree86 配置程序来生成时钟值为好。如果没有时钟值，服务器会自动确定它们。使用新的显示卡时可能会出现这种情况。

21.3X-Windows 和窗口管理器

正如第 3 章中指出，用户可以放弃命令行界面而使用 X-Windows 窗口管理器和文件管理器，这样通过使用窗口、按钮和菜单用户就能够与 Linux 系统进行交互。窗口管理器提供了基本的窗口管理操作例如打开、关闭窗口，改变窗口尺寸等。文件管理器则允许用户使用图标和菜单来管理和运行程序。此外，使用文件管理器还能够拷贝、移动或删除文件，也能够分别为不同的目录开不同的窗口。

为 Linux 设计的窗口管理器有好几种。其中较为流行的 Linux 管理器有：Free Virtual Window Manager(fvwm),Xview(olwm),twm,fvwm95(fvwm95) , LessTif(mwm),AfterStep(afterstep)以及 Motif(mwm)。除了 Motif 之外，其它几种都是免费的。Xview 是 Sum System 公司公司的 OpenLook 界面的 Linux 版。fvwm95 是 fvwm 的一个变体，它提供包含任务栏的 Windows 95 界面。LessTif 是一个免费的 Motif 克隆，它能够提供 Motif 界面，并能运行 Motif 应用程序。twm 是一个稍微旧一点的窗口管理器，它提供基本的开窗功能。

在 OpenLinux 上已经安装了 fvwm,twm 和 Xview。其它的窗口管理器可以从相应的互联网站点或 Redhat contrib 目录下载(可以参考第 3 章的表 3-2 记录的 Linux 资源页)。只要在相应的网址上有较新版本的窗口管理器，用户就可以下载并安装它。

和大多数 Linux 系统一样，目前 OpenLinux 系统也使用 fvwm。有了窗口管理器，用户可以用窗口替代命令行。在窗口中进行的各种操作经过解释后送到 Linux 系统执行。窗口管理器对底层的 X-Windows 系统进行操作，后者实际上提供各项基本的窗口操作，允许用户打开、移动和关闭窗口，以及显示菜单和选择图标。fvwm 和 Xview 分别以自己的方式管理这些操作，提供各自独特的界面。这样设计的好处是用户可以用不同的窗口管理器操作同一个 Linux 系统。从这层意义上来说，Linux 并不局限于一种图形用户界面(GUI)。在同一个 Linux 系统上，一个用户可能正在使用 fvwm 窗口管理器，另一个可能正在使用 Xview 窗口管理器，而第三个用户则在使用 fvwm95，所有用户都在同时使用系统。

fvwm 是 GNU 公众许可软件--用户免费拥有它，如果只使用 fvwm，用户仍然能够运行任何 X-Windows 程序。正如第 4 章中指出，fvwm 有它自己的工作位置菜单和任务栏。用户也可以从 Xterm 终端窗口运行任何 X-Windows 程序。从 fvwm 启动一个 Xterm 窗口时，只需要输入 X-Windows 应用程序的名字，然后按回车键，X-Windows 应用程序将启动自己的窗口。最好通过在命令后面加符号&的方法以后台进程方式启动一个 X-Windows 应用程序。系统将为用户使用的 X-Windows 应用程序开一个单独的窗口。用户可以以这种方式运行 Caldera CD-ROM 上的任何 X-Windows 软件。X-Windows 应用程序的名字经常是以字

母 x 开头的，但是也有例外，比如 Netscape 和 Mosaic。用户可以从 Linux 的 ftp 站点下载很多各种各样的 X-Windows 软件。它们通常在目录 X11 下，比如／pub／Linux／X11。

一个 X-Windows 应用程序只能在特定的窗口管理器下运行的情况是很少见的。用户可以在 fvwm,Xview,fvwm95,或任何其它窗口管理器下运行同一个 X 应用程序，例如 Netscape。甚至在没有任何窗口管理器的情况下，也可以运行 X-Windows 应用程序。用户仍然能够打开一个非常简单的 Xterm 窗口，然后从那儿启动 X 应用程序。X-Windows 应用程序运行所需要的只是 X-Windows。窗口没有任何诸如改变其尺寸或图标化之类的控制工具，甚至没有关闭框。关闭它的唯一方式是输入退出命令。这种情况下运行的 X-Windows 程序有它自己的菜单和按钮，但它是在一个简陋的固定的窗口中运行。

21.4fvwm 配置文件

fvwm 窗口管理器启动时将执行自己的配置文件。这些文件将完成一些任务，诸如在 fvwm 任务栏上显示按钮，在工作位置菜单里建立菜单项，决定启动什么初始程序（如果有的话）。这些配置文件在目录／usr／X11R6／lib／X11／fvwm 下面。每个文件的名字以单词 system 开头。

基本配置文件是 system.fvwmrc。这个文件依次调用其它文件，如 system.fvwmdesk,system.fvwmrc.goodstuff。用户可以修改这些配置文件，按

照自己的喜好配置 fvwm 窗口管理器。例如，如果想往任务栏添加按钮，可以往文件 system.fvwmrc.goodstuff 里添加项目。表 21-2 列出了这些 fvwm 配置文件。

对文件 system.fvwmrc 的修改将缺省设置系统上的所有用户的配置。当然，每个用户可以在自己的用户主目录下建立个人的配置文件.fvwmrc，它只对创建它的用户起作用。每次用户执行 startx 命令时，这个文件将被用来配置 fvwm。

21.4.1 任务栏：system.fvwmrc.goodstuff

要往任务栏添加按钮，可以在文件 system.fvwmrc.goodstuff 中建立新的项目。项目以关键词*GoodStuff 开头。紧接着是用户想在按钮的图像下显示的文本。然后输入用户想显示的图像文件名。图像是 pixmap 型，文件扩展名是.xpm。用户能够在目录／usr／openwin／lib／pixmaps 和／usr／share／data／pixmaps 下找到图像文件。图像文件名后面是一个 fvwm 操作。尽管这个操作通常是 Exec 命令，但它实际上可以是任何 fvwm 命令。Exec 将执行一个 Linux 命令或程序。Exec 命令后面是一个可以作为菜单项的名字的字符串。对按钮条来说，它只是一个空串。字符串后面是 Xlaunch 命令和程序名字。用来 Xlaunch 执行程序。下面的例子是为 Crisplite 编辑器加一个按钮。启动 Crisplite 编辑器的命令是 mcr。

```
*GoodStuff Crisplite dtp.xpm Exec "" xlauch mcr
```

如果想从任务栏移去一个项目，可以删除文件中对应与此项目的那一行，

也可以在那一行开头插入#符号作为注释。其它支持任务栏的窗口管理器也有类似的配置文件。

21.4.2 工作位置菜单：system.fvwmrc.menu

文件 system.fwmrc.menu 保存着工作位置菜单的配置信息。工作位置主菜单在该文件最后定义，子菜单则在主菜单前面定义。工作位置菜单和其它主要子菜单都用命令 PopupSMenu 定义。每一项占一行，每一项经常对应定义一个菜单，Popup 项指定了这种定义。Popup 项后面跟着的是将显示在菜单上的项目名称，然后是相关菜单的名字。菜单名称中前面有&的字母在菜单显示时带有下划线，并作为快捷键的关键字母。按下那个字母将选择次菜单项。部分菜单定义显示如下。

```
PopupSMennu('Workplace',' 
Title"Workplace"
Nop" "
Popup"&Help on Linux" Help
Nop" "
Popup"&Shells"Shells
)
```

很多子菜单用 PopupMenu 命令定义。紧跟着 PopupMenu 的是一串项目名称和命令，它们之间以逗号隔开，所有这些字符串被一对括号括起来。第一项

通常是子菜单名称；然后是子菜单名称，它后面是该子菜单执行的命令；紧接着又是一对子菜单名称和命令。可以用符号&来指定快捷键关键字母。子菜单Text and Publishing"就是这样一个 Popup 菜单，这里是部分菜单定义。

```
PopupMenu('Text and Publishing','MenuItem',
'L&yX(WYSIWIN TeX)','lyx',
'X&wpick(Screen shot)','xwpick HOME／xwpick.gif',
'X&fig(Vector graphic)', 'xfig'
)
```

如要往菜单里添加一个菜单项，可以先找到该菜单的定义部分，然后插入一空行，输入想添加的菜单名称和命令。例如，要在 Text and Publishing 菜单中插入对应 Crisplite 编辑器的菜单项，用户可以插入以下这行。字母 c 前面的符号表示建立了关键字母为 c 的快捷键。别忘了加上逗号。

```
"Crisplite",m&cr,
```

用户千万不要修改文件 system.fvwmrc.menu.prep。这个文件是由 fvwm 生成的，它实际上创建了工作位置菜单。这个文件是基于文件.menu 提供的信息生成的。

安装在目录／usr／X11R6／bin 下的 X-Windows 程序会自动加载到 Application 菜单下的 Other Applications 子菜单上。因此，用户能够通过工作位置菜单执行任何 X-Windows 程序。往菜单文件中添加一条项目将把该项目对应的菜单项放在一个特定子菜单中，而不是在 Other Applications 菜单中。

21.4.3 图标风格与设备映射：.styles 和 .bindings

文件 system.fvwmrc.styles 存放着用于指定特定程序的窗口风格的项目。例如，下面显示的指定页面调度程序风格的 StaysOnTop 使页面调度程序始终位于所有窗口前面。Style 项也被用来指定程序窗口图标化时所使用的图像文件。用户能够改变这些项目，以便显示不同的 pixmap 图象，或者为新的程序添加新项目。

```
Style "FvwmPager"StaysOnTop
Style "xgraph" Icon  graphs.xpm
```

文件 system.fvwmrc.bindings 存放着所有鼠标按键和键盘的映射。鼠标按键和键盘对工作位置菜单的操作都在这儿定义。按下组合键 ALT-F1 将显示工作位置菜单，单击鼠标左键（键 1）得到相同的效果。下面这一项就是定义了组合键 ALT-F1 与工作位置 菜单之间的映射。Context 域是指组合键的适用窗口；字母 A 代表任何窗口，字母 R 代表根窗口。Modifi 域指任何需要同时按下的按键 ALT 键。字母 M 代表元键（ALT 键），字母 S 代表 SHIFT 键。

```
# KeyContextModifiFunction
Key F1AMPopup "Workplace"
```

21.4.4 初始启动应用程序：system.fvwmdesk

在 OpenLinux 系统里命令 startx 首先启动 X-Windows，然后是 fvwm 窗口

管理器，接着是 Caldera Desktop Looking Glass。Caldera Desktop 的自动启动实际上是由 fvwm 的配置脚本文件完成的。这些配置文件可以用来指定那些在 fvwm 启动时被自动启动的程序。

通常，startx 命令通过配置文件／etc／X11／init／xinitrc 调用 xinit 命令。这个 xinitrc 文件执行基本的 X-Windows 配置操作并且最后启动 fvwm 窗口管理器。fvwm 窗口管理器有它自己的启动文件，它位于目录／usr／X11R6／lib／X11／fvwm 下。system.wmrc 文件是基本配置文件，它执行一个叫作 StartupFunction 的命令，文件 system.fvwmdesk 定义了这个命令。在这个文件中，StartupFunction 被定义为执行命令 lg，该命令启动 Caldera Desktop(lg 代表 Caldera Desktop 的真正名字--Looking Glass)。如果用户不想 Caldera Desktop 随着 startx 命令自动启动，只需在有 lg 命令的那一行开头插入符号#，使这一行成为注释行。如果用户想让 fvwm 自动启动其它的文件管理器或者桌面，可以把命令插入到文件 system.fvwmdesk。用户能够很轻松地把文件 system.fvwmrc 里的整个 StartupFunction 注释掉，如下所示：

#Function"l"StartupFunction

如果从 fvwm 移去了 Caldera Desktop 的自动启动功能，用户仍然能够使用工作位置菜单来启动它。在 Desktop 子菜单中,用户将能找到对应 Looking Glass（Caldera Desktop）的菜单项。选择这一菜单项将启动 Caldera Desktop。用户当然也可以启动别的文件管理器或桌面。

21.5X-Windows 命令行参数

用户能够以.xinitrc 脚本文件的方式，或者以 Xterm 窗口里的命令行方式启动任何 X-Windows 应用程序。大多数 X-Windows 应用程序支持一组标准的 X-Windows 参数，这些参数用来配置应用程序所使用的窗口和显示。用户可以设定窗口栏的颜色，确定窗口的标题名，指定文本颜色和字体，也可以把窗口定位于屏幕的特定位置。表 21-3 列出了这些 X-Windows 参数。它们在 X 的 man 帮助页中有详细说明（运行 man X）。

-geometry 是一个经常使用的参数。它还需要另外的参数，通过它用户能够指定应用程序的窗口在屏幕上的的位置。在下面的例子里，X-Windows 应用程序 xclock 在带有 geometry 参数的情况下被启动。那些数字（最多可以有四个）用来确定窗口位置。＋0＋0 表示左上角。用户启动 X-Windows 窗口时，将能看到左上角显示的时钟。-0-0 表示右上角。

& xclock-geometry＋0＋0 &

用户可以通过-title 选项设定应用程序窗口的标题名。注意当标题名不止一个单词时，要用引号把它们括起来。-fn 参数用来设定字体，-fg 设定文本和图表颜色， -bg 设定背景颜色。下面这个例子启动了一个标题名为 My New Window"的窗口。它的文本和图表颜色是绿色，背景颜色是灰色。字体是 Helvetica。

$ xterm-title"My New Window"-fggreen-bggray-fn

／usr／fonts／helvetica&

21.6X-Windows 配置文件

X-Windows 使用几个配置文件以及 X-Windows 命令来配置 X-Windows 系统。其中一些配置文件属于系统文件，不能修改。但是，每个用户可以有一个自己的一组配置文件，诸如.xinitrc,.Xresources,使用这些文件可以建立个人风格的 X-Windows 界面。当用 startx 命令启动 X-Windows 时，这些配置文件被自动读取并执行。在这些配置文件里，用户能够执行 X-Windows 命令来配置系统。通过诸如 xset ,setroot 之类的指令，用户能够增加字体或者控制主窗口的显示。表 21-3 提供了一些 X-Windows 配置文件和命令。用户可以通过使用 xdypinfo 得到当前使用的 X-Windows 系统配置的完整描述。Xman 帮助页有关于 X-Windows 命令和配置文件的详细介绍。

$manX

X-Windows 是由命令 xinit 启动的。但是用户无需直接执行这个命令，可以通过经常用来启动 X-Windows 的 startx 命令执行它。这两个命令都在目录／usr／X11R6／bin 下，和许多其它的基于 X-Windows 的程序存放在一起。startx 命令是一个执行 xinit 命令的 shell 脚本文件。xinit 命令按顺序首先查找位于用户主目录的一个 X-Windows 初始化脚本文件.xinitrc。如果在用户主目录里

没有脚本文件.xinitrc，xinit 将使用／usr／X11R6／lib／X11／xinit／xinitrc 作为其初始化脚本文件。.xinitrc 和／usr／X11R6／lib／X11／xinit／xinitrc 里都含有配置 X-Windows 服务器的命令，它们执行诸如启动窗口管理器之类的 X-Windows 初始化命令。可以把／usr／X11R6／lib／X11／xinit／xinitrc 脚本文件看作缺省脚本文件。在 OpenLinux 系统里面，／usr／X11R6／lib／X11／xinit 目录实际上是与目录／etc／X11／xinit 的一个符号连接，缺省初始化文件都存放在／etc／X11／xinit 目录里面。

OpenLinux 系统并不会自动在任何用户主目录下创建任何.xinitrc 脚本文件。需要.xinitrc 脚本文件的用户需要亲自创建它。每一个用户可以在自己的用户主目录下创建个人风格化的.xinitrc 脚本文件，以便按照自己的意愿来配置和启动 X-Windows。这对主用户同样适用。在某个用户创建自己的.xinitrc 脚本文件前，缺省使用／usr／X11R6／lib／X11／xinit／xinitrc 脚本文件。用户可以通过查看这个脚本文件来了解 X-Windows 是如何启动的。X-Windows 的某些配置操作必需在文件.xinitrc 中完成。对于想要创建自己的.xinitrc 脚本文件的用户，最好首先把脚本文件／usr／X11R6／lib／X11／xinit／xinitrc 拷贝到自己的用户主目录下，命名为.xinitrc。然后，用户可以随心所欲地修改这个脚本文件（注意：系统的 xinitrc 脚本文件在名字前面不带.号，而用户自己建立的主目录下的.xinitrc 文件名字前面带.号。）用户如需要更多的信息，可参考 xinit 和 startx 的 man 帮助页。

除了 xinit 命令之外，其它命令，如 xrdb,xmodmap，也能够用来配置 X-Windows 的界面。在资源文件.Xresources 中列出了 X-Windows 的一些图形配

置。每个用户可以在自己的用户主目录下拥有定制的.Xresources 文件，用它来配置特定的 X-Windows。.Xresources 文件包括一些配置特定程序的项目，比如某个工具箱的颜色。.Xresources 文件也存在一个适用整个系统的版本：／usr／X11R6／lib／X11／xinit／.Xresources （注意：与／usr／X11R6／lib／X11／xinit／xinitrc 不同，在／usr／X11R6／lib／X11／xinit／.Xresources 文件名中，Xresources 前面带有.号。）与文件.Xresources 一样，文件.Xdefaults 也包括一些用来配置资源的项目。在用户系统上运行的程序对文件.Xdefaults 有操作权限，但在其它系统上运行的程序无此权限。

OpenLinux 系统安装后只有一个系统.Xresources 文件。但用户可以在自己的主目录下创建自己的.Xresources 文件，并且往此文件中添加一些资源项目。用户也可以拷贝文件／usr／X11R6／lib／X11／xinit／.Xresources 到自己的主目录下，然后编辑或添加自己的项目。配置操作是由命令 xrdb 执行的，它会读取系统的.Xresources 文件和用户主目录下的任何.Xresources 文件和.Xdefaults。xrdb 通常在脚本文件／usr／X11R6／lib／X11／xinit／xinitrc 中被执行。如果用户在主目录下创建了自己的.xinitrc 脚本文件，务必在文件最后运行 xrdb 命令，使它读取.Xresources 文件或／usr／X11R6／lib／X11／xinit／xinitrc 文件（最好两者都读）。要做到这一点，用户只需简单地用系统的.xinitrc 脚本文件的一个拷贝作为自己的.xinitrc 脚本文件，然后按照自己的喜好修改它。关于资源的细节，请看 xrdb 的 man 帮助页。当然也可以在 X 的 man 帮助页中找到关于 Xresources 以及其它 X-Windows 命令的详细讨论。

Xresources 文件的每一条项目是由一个分配给资源、类、或一个应用程序

资源的值。通常他们是工具箱的资源或一个应用程序中工具箱的类的资源。这些资源的指定一般是三个元素组成，元素之间由.号隔开：应用程序、应用程序中的一个对象、资源。整个指定行以一个冒号以及随后紧跟的值结束。例如，假设想在 oclock 应用程序中把时针的颜色改为蓝色。应用程序是 oclock，对象是 clock，资源是 hour： oclock..clock.hour。这一项如下所示：

xclock.clock.hour:blue

对象元素实际上是一系列各等级的对象，由此确定一个特定对象。在 oclock 这个例子中只有一个对象，但在很多应用程序中，对象分级是非常复杂的。它将需要列出冗长的一串对象才能确定用户想指定的那个。为了避免这种复杂用户可以使用*号来直接引用想要指定定的对象，具体做法是用*号来替代.号。下面的例子是把 oclock 的分针和时针设置为绿色。

oclock*hour :green

oclock*minute: green

用户也可以使用*号把一个值赋于对象的所有的类。许多单个的资源被划分为一些类。用户可以使用类的名字来引用这个类中的所有资源。类名以大写字母开头。例如，在 Xterm 应用程序中，背景和 pointer 颜色都是 Background 类的一部分。引用 Xterm*Background 将改变 Xterm 窗口中所有这些资源。但是，任何个别的特定引用总是优于一般化的引用。

用户也可以使用*号来改变所有应用程序的对象中的资源值。在这种情况下，可以在资源名前置一个*号。例如，要把每一个应用程序中的所有对象的前景颜色改为红色，可以输入：

*foreground: red

如果只想改变所有应用程序中滚动条的前景颜色，可以输入如下：

*scrollbar*foreground:blue

命令 showrgb 将列出系统提供的各种颜色。用户可以使用描述名或十六进制的数字形式。值也可以是字体、位图以及 pixmap。用户可以改变某个对象中显示的字体，或图形应用程序中的背景或边框图象。各个应用程序的资源各不相同。应用程序可能支持各种各样的对象及其它们的资源。如果想了解一个应用程序所能支持的资源以及这些资源所能接受的值，可以查看此应用程序的帮助页和文档资料。某些资源的取值是逻辑量，用它们可以关闭或启动一些特性。其它一些资源则能够指定某些选项。一些应用程序有自己的一套资源缺省值，它们被自动置于系统的.Xresources 或 .Xdefaults 文件之中。

文件／usr／X11R6／X11／xinit／.Xmodmap 存放着诸如鼠标和键盘之类的系统输入设备的配置信息（例如，用户可以建立键的映射，如 BACKSPACE, 或者调换鼠标左右键）。每一个用户可以在自己的用户主目录下创建自己的定制.Xmodmap 文件，用它来配置系统的输入设备。如果用户是通过自己的终端和 Linux 系统相连接，这种定制的文件会对用户很有帮助。.Xmodmap 文件由 xmodmap 命令读取，它完成配置操作。xmodmap 命令将首先在用户主目录下寻找 Xmodmap 文件然后读这个文件。如果在用户主目录下没有.Xmodmap 文件，xmodmap 命令将读文件／usr／X11R6／lib／X11／xinit／.Xmodmap。xmodmap 命令通常在文件／usr／X11R6／lib／X11／xinit／xinitrc 内被执行。如果用户在自己的主目录下有自己的.xinitrc 脚本文件，系统不是在用户自己

的.Xmodmap 文件里就是在文件／usr／X11R6／lib／X11／xinit／.Xmodmap 里执行 xmodmap 命令。详细情况请参考 xmodmap 的帮助页。

通常，在.xinitrc 脚本文件里有一些 X-Windows 命令，比如 xset 和 xsetroot, 用它们来配置 X-Windows 对话的 各种不同特性。 Xset 命令设置一些不同的选项，例如启动屏幕保护，设定响铃和扬声器的音量。也可以用 Xset 来加载字体。详细情况请参考 xset 的帮助页。Xset 在选项 b 和参数 on 或 off 存在的情况下，将打开或关闭扬声器。下面的例子中，打开了扬声器。

Xsetbon

如要设置屏幕保护,可以执行带选项-s 的 xset。用户可以通过参数 on 或 off 来启动或关闭屏幕保护。作为参数输入的两个数值指定长度和间隔（单位：秒）。长度是屏幕保护程序在激活前等待的秒数。间隔是指屏幕保护程序在产生新的图案前等待的时间。

xsetroot 命令使用户能够设置主窗口的特性（设置颜色或显示位图图案--用户甚至能够自己设计的光标）。表 21-3 列出了不同的 xsetroot 选项。要了解各选项和详细情况请参考 xsetroot 的帮助页。下面的 xsetroot 命令通过使用-solid 选项把主窗口的背景设置为蓝色。

Xsetroot-solidblue

文件 ／usr／X11R6／lib／X11／xinit／xinitrc 里还包含有一些命令和变量，它们用来配置 Looking Glass 桌面，Xview 窗口管理器和 fvwm 窗口管理器。

在.xinitrc 脚本文件里，窗口管理器的启动往往是最后一条命令。无论.xinitrc

脚本文件的最后一条命令是什么，在执行完这条命令后 X-Windows 将会退出。通过把启动窗口管理器的命令放在最后，在退出窗口管理器时将关闭 X-Windows 进程。用户想初始启动的任何其它程序都应放在窗口管理器命令前面。如果想在任何时候执行 startx 时自动启动 Netscape 或 Xfm 文件管理器，用户必须把启动这些应用程序的命令放在窗口管理器命令前面。在应用程序后面应紧跟一个&号，使它们能在后台运行。下面的例子在 fvwm 启动时，自动启动 Xfm 文件管理器和 Netscape。

```
xfm&
netscape&
exec fvwm
```

下面是.xinitrc 文件的一个简单例子。它被设计成在启动 fvwm 窗口管理器的同时启动 Xfm 文件管理器。变量 OPENWINHOME 和 HELPPATH 为需要 Xview 库的应用程序指出了这些库的存放目录。如果.xinitrc 文件在用户主目录中，它会指示 xinit 启动 fvwm 的同时启动 Xfm（注意文件.xinitrc 名字前的.。当 fvwm 启动时，任务栏和页面调度程序将和 Xfm 窗口一起显示。在屏幕的任何位置点鼠标左键都会弹出工作位置菜单。

```
.xinitrc
#!／bin／sh
#$Xconsortium: xinitrc.cpp,v 1.4 91／08／22 11:41:34 rws Exp $
                userresources=$HOME／.Xresources
                usermodmap=$HOME／.Xmodmap
```

```
sysresources=／usr／X11R6／lib／X11／xinit／.Xresources
sysmodmap=／usr／X11R6／lib／X11／xinit／.Xmodmap
export    OPENWINHOME=／usr／openwin
export     HELPPATH=$ OPENWINHOME／help
#merge   in   defaults and keymaps
if [  -f $sysresources]   ; then
 xrdb -merge $sysresources
fi
if [-f $sysmodmap] ; then
 xmodmap   $sysmodmap
fi
if [  -f $userresources ] ; then
 xrdb -merge   $userresources
fi
if [-f $usermodmap] ; then
 xmodmap    $usermodmap
fi
#start some   nice       programs
xfm &
exec fvwm
```

在目录／etc／X11／xinit **下有一个名为** xinitrc.before.fvwm **的仅有简单框**

架的.xinitrc 文件样本。它没有对应 fvwm,Xview 以及 Looking Glass(Caldera) 桌面的配置命令。但是，这是一个仅包含有运行 X-Windows 所必须的基本配置操作的好例子。该脚本文件将打开三个 Xterm 窗口并且运行 twm 窗口管理器。如果需要，用户可以把这个脚本文件拷贝到自己的用户主目录，作为自己的.xinitrc 文件（如果用户现在有自己的.xinitrc 文件,务必对它进行备份）。然后就可以按照自己的意愿添加配置操作和命令。例如，可以移去一些 xterm 命令，启动另一个不同的窗口管理器（添加需要的配置变量），或者启动一个应用程序，如 Netscape。如果不想启动 xclock，用户可以在它的命令前加一个#号，把这个命令注释掉。

21.7 字体

X-Windows 的字体文件存放在目录／usr／X11R6／.lib／X11／fonts 下面。X-Windows 的字体文件由命令 xfs 来加载。xfs 读取配置文件／etc／X11／fs／config，这个文件在项目 catalogue 下列出了字体目录。在 Xman 帮助页里有关于字体的详细讨论。为了安装字体集，把相应的字体文件放在一个用户能够把它的路径添加到 catalogue 项的目录下面，这样字体就可以自动安装。用户也可以用带＋fp 选项的 xset 命令单独地安装某种特定字体。用户系统里的字体由字体路径指定。字体路径是一些文件名，每一个文件存放一种字体。文件名包括它们的完整路径。下面是文件／etc／X11／config 中 catalogue 项的

一个例子。这是一串以逗号隔开的目录清单。X-Windows 首先在这些目录中寻找字体文件。

Catalogue =

／usr／X11R6／lib／X11／fonts／misc,／usr／X11R6／lib／X11／fonts／Speedo／,／usr

／X11R6／lib／X11／fonts／Typel／,／usr／X11R6／lib／X11／fonts／75dpi／,／usr／X11

R6／lib／X11／fonts／100dpi／

在能够使用新安装的字体之前，用户必须首先使用 mkfontdir 命令对它们进行索引。从含有新的字体的目录里，输入 mkfontdir 命令。用户也可以把目录路径作为 mkfontdir 命令的一个参数。在对字体索引完以后，用户可以使用带有选项 fp rehash 的 xset 命令装入这些字体。为了能够自动装载这些字体，把目录以完整路径名的形式添加到 xfs 配置文件的 catalogue 项中去。下面的例子显示了如何安装一种新的字体并加载它：

```
$ cp newfont.pcf~／myfonts
$ mkfontdir ~／myfonts
$ xset fp rehash
```

在一个字体目录里，有几个特殊文件保存着关于字体的信息 。fonts.dir 文件列出了它所在目录下的所有字体。此外，用户可以通过建立一个 fonts.alias 文件给一个字体文件取其它别的名字。字体文件名字一般很长，很复杂。

Fonts.scale 文件存放着可伸缩字体的名字。详细情况请参考 xfs 和 mkfontdir 的帮助页。

用带有＋fp 和-fp 选项的 xset，用户能够个别地添加或者去掉特定的字体。带有参数 rehash 的 fp 将加载字体。在 default 参数的情况下，将恢复缺省字体集。＋fp 向字体路径添加一种字体。对于自己的字体，用户可以把他们放在任何目录下并且指定它们的名字（包括他们的完整路径名）。以下的例子把在目录／usr／local／fonts 下的 myfont 字体添加到字体路径。然后带有参数 rehash 的 fp 选项装入字体。

xset ＋fp ／usr／local／fonts／myfont

xset fprehash

要移去这种字体，须运行 xset -fp ／usr／home／myfont 然后是命令 xset fp rehash。如果想把系统恢复到缺省字体集，输入如下：

xsetfpdefault

xsetfprehash

用 xlsfonts 命令，用户列出目前安装在系统上的所有字体。如想显示已安装的字体以查看它的外观，使用 xselfonts 命令。用户可以浏览系统的字体，选择几种自己喜欢的。

21.8 编译 X-Windows 应用程序

为了编译 X-Windows 应用程序，用户必须确信安装了 Xfree86 开发包和其它可能要用到的开发包。这些包包含了头文件和 X-Windows 程序使用的库。这类包的名字含有词 devel，例如，Xfree86-devel-3.1.2-2.i386.rpm。还有，许多 X-Windows 应用程序可能需要特殊的共享库。

例如，有些应用程序可能需要 xforms 库或 qt 库。这些库中的某些部分可能必须从在线站点得到，但大多数都可以从 RPM 形式的 Redhat contrib 目录得到。

许多 X-Windows 应用程序需要生成一个为配置系统而创建的 Makefile 文件。如果有应用程序源代码，这个任务可以由 Imakefile 完成。安装在系统上的 xmkmf 命令能够 take an Imakefile 并且生成合适的 Makefile 文件。一旦有了 Makefile 文件，就可以用 make 命令编译应用程序。xmkmf 命令实际上调用程序 imake 从 Imakefile 生成 Makefile 文件。但是用户不用直接使用 imake。关于 xmkmf 和 make 的详细情况可查看相应的 man 帮助页（例子见第 7 章）。最近许多程序使用配置脚本文件生成 Makefile。在那种情况下，用户只需运行./configure 命令，而不是 xmkmf。

21.9 小结：配置 X-Windows

Xfree86 服务器对较多种类的硬件提供支持，但配置它们是很富有挑战性的一项任务。如果用户已经正确地安装了 XFree86 服务器（见表 21-1），可以通过运行位于目录／usr／X11R6／bin 的 XF86setup 程序配置它。也有其它的配置实用工具，如 xf86config,Xconfigurator。所有这些实用工具会问一些关于当前系统硬件的问题然后创建一个／etc／XF86Config 文件。当 X-Windows 启动时，它读取文件／etc／XF86Config 里的配置信息。

每个用户也可以通过使用在自己的用户主目录下的.xinitrc 文件，创建自己的启动窗口管理器和桌面的命令。用户可以把启动窗口管理器、桌面以及任何其它想要的 X 应用程序的命令放到这个文件中。也可以只指定一个窗口管理器，或者一个窗口管理器和一个桌面。最好以拷贝／usr／X11R6／lib／X11／xinit／xinitrc 文件的方式创建.xinitrc 文件。

fvwm 窗口管理器有自己的一套配置文件。用户可以用它们向任务栏添加按钮，或向工作位置菜单添加菜单项，或指定想初始启动的程序。

表 21-1XFree86 服务器

服务器	类型
XFree86_SVGA	彩色 SVGA 服务器。包括大部分显示卡的驱动程序

续表

XFree86_VGA16	16色SVGA和VGA非加速型服务器
XFree86_Mono	单色非加速型服务器
XFree86_S3	S3加速型服务器
XFree86_S3V	S3ViRGE和ViRGE／VX加速型服务器（许多3D显示卡）
XFree86_I128	Number 9 Imagine 128加速型服务器
XFree86_8514	8514／A加速型服务器
XFree86_Mach8	ATI Mach8加速型服务器
XFree86_Mach32	ATI Mach32芯片加速型服务器
XFree86_P9000	Weitek加速型服务器
XFree86_W32	ET4000／W32加速型服务器
XFree86_agx	IIT AGX加速型服务器
XFree86_Mach64	ATI Mach64芯片加速型服务器

表21-2fvwm配置文件和键盘映射

配置文件	描述
system.fvwmdesk	桌面启动
system.fvwmrc	主要配置文件
system.fvwmrc.bindings	鼠标和键盘映射
system.fvwmrc.config	配置变量
system.fvwmrc.functions	函数定义

续表

system.fvwmrc.goodstuff	任务栏项目
system.fvwmrc.menus	菜单项
system.fvwmrc.menus.exclude	专用菜单项
system.fvwmrc.menus.prepfvwm	生成菜单
system.fvwmrc.modules	组件
system.fvwmrc.options	选项
system.fvwmrc.styles	程序窗口风格
功能键键盘映射	
ALT-F1	工作位置菜单
ALT-SHIFT-H	帮助菜单
ALT-SHIFT-S	Shells 菜单
ALT-SHIFT-A	应用程序菜单
ALT-SHIFT-G	游戏菜单
ALT-SHIFT-D	桌面菜单
ALT-SHIFT-T	终端菜单
ALT-SHIFT-C	配置菜单
ALT-SHIFT-Pproductivity	工具菜单
ALT-SHIFT-F	图形菜单
ALT-SHIFT-X	文本和发行菜单
ALT-SHIFT-V	软件开发菜单

续表

ALT-SHIFT-I	因特网联结菜单
ALT-SHIFT-R	娱乐菜单
ALT-SHIFT-Y	系统管理菜单

表 21-3X-Windows 命令、配置文件、参数

X-Windows 命令	解释	
Xterm	打开一个新的终端窗口	
Xset	设置 X-Windows 选项；	
	完整清单见 man 帮助页	
	-b	配置响铃
	-c	配置键的单击
	＋fp fontlist	添加字体
	-fp fontlist	移去字体
	led	打开或关闭键盘 LEDs
	m	配置鼠标
	p	设定象素颜色值
	s	设置屏幕保护
	q	列出当前设置
xsetroot	配置主窗口	

续表

	-cursor cursorfile maskfile	设置当鼠标不在任何窗口时所显示的 pointer 位图图象
	-bitmap filename	设置根窗口的位图图案
	-gray 1	把背景设置为灰色
	-fg color	设置前景位图的颜色
	-bg color	设置背景位图的颜色
	-solid color	设置背景颜色
	-name string	把根窗口的名字设为一字符串
xmodmap	配置输入设备；读取.Xmodmap 文件	
	-pk	显示当前的键盘映射图
	-e expression	设置键的映射
	keycode NUMBER-KEYSYMNAME	把键设置为指定的键符号
	keysym KEYSYMNAME-KEYSYMNAME	设置某键的功能和指定键相同
	pointer-NUMBER	设置鼠标键代码

续表

xrdb	配置 X-Windows 资源；读取.Xresources 文件
xdm	X-Windows 显示管理器；运行系统 XFree86 服务器；通常被.xinitrc 调用
startx	执行 xinit 启动 X-Windows 并且指示 xinit 读取文件.Xclients
xfs config-file	X-Windows 字体服务器
mkfontdir font-directory	对新的字体进行索引，使它们能被字体服务器存取
xlsfonts	列出当前系统的字体
xfontsel	显示已安装的字体
xdpyinfo	列出关于 X-Windows 配置的详细信息
xinit	启动 X-Windows，首先读系统的.xinitrc 文件；当从 startx 启动时，它也读取用户的.Xclients 文件；xinit 并不被直接调用，而是通过 startxxmkmf 使用应用程序的 Imakefile 来为 X-Windows 应用程序创建一个 Makefile；启动 imake 生成 Makefile(不必直接启动 imake)配置文件配置文件
.Xmodmap	用户的 X-Windows 输入设备配置文件
.Xresources	用户的 X-Windows 资源配置文件
.Xdefaults	用户的 X-Windows 资源配置文件

续表

.xinitrc	如果这一文件存在，用户的 X-Windows 配置文件，由 xinit 自动读取
.Xclients	用户的 X-Windows 配置文件（在 Redhat 和其它 Linux 销售版本里使用）
／usr／X11R6／lib／X11／xinit／xinitrc	系统 X-Windows 初始化文件；由 xinit 自动读取
／usr／X11R6／lib／X11／xinit／.Xresources	系统 X-Windows 资源文件；由 .xinitrc 读取
／usr／X11R6／lib／X11／xinit／.Xmodmap	系统 X-Windows 输入设备文件；由.xinitrc 读取
／usr／lib／X11／rgb.txt	X-Windows 颜色;每一项有四个域；头三个是代表红色、绿色、蓝色的数字；最后一个域是颜色的名字
／usr／X11R6／lib／X11／xinit／xinitrc.before.fvwm	一个 X-Windows 的.xinitrc 样本文件，里面没有 fvwm 和 Caldera 桌面（在 OpenLinux 系统里）
X-Windows	应用程序配置参数

续表

-bw num	框架边界的宽度（单位：象素点）
-bd color	边界颜色
-fg color	前景颜色（对文本和图表而言）
-bg color	背景眼色
-display display-name	显示正在运行的客户；显示主机名组成的名字，显示器编号，屏幕编号（见 XMan 帮助页）
-fn font	文本显示所使用的字体
-geometry offsets	X-Windows 应用程序窗口在屏幕上的位置；offset（偏移量）是相对显示屏幕而言
-iconic	以图标的形式启动应用程序，而不是以窗口形式
-rv	对换背景和前景的颜色
-title string	窗口标题栏上的标题
-name string	应用程序的名字
xrm resource-string	指定资源值桌面，窗口和文件管理器桌面、窗口和文件管理器
lg	Caldera 桌面
fvwm	fvwm 窗口管理器
olvwm	Xview 窗口管理器（OpenLook）
fvwm95	fvwm95(Windows95) 窗口管理器
qvwm	qvwm(Windows95) 窗口管理器

续表

mwm	LessTif 窗口管理器
xfm	Xfm 文件管理器
aftersetp	AfterStep 窗口管理器
mlvwm	Macintosh 窗口管理器
mfm	Motif 窗口管理器

第 22 章排版工具：TeX，LaTeX 和 Ghostscript

从最初开始，Unix 就有文档处理工具，使用这些工具用户可以进行复杂的文字处理。用它们可以制作在线帮助文档，即运行 man 命令后显示的帮助信息。用户能够十分轻松地使用这些工具为自己开发的任何工程或者软件创建在线帮助文档。这些工具的运作类似过滤器，读入输入文件，然后生成输出文件，输出文件通常就是将要打印的内容。输入文件里存有一些特殊指令，这些指令指导过滤器如何格式化输出内容。Unix 的主要文档处理工具有：nroff，troff，tbl，eqn、以及 pic。Linux 里有所有这些工具的 GNU 版本，这些 Linux 工具的名字不是与 Unix 里相应工具的名字一样，就是在 Unix 工具名字前再加个 g’。GNU 版本的 troff,tbl,eqn,及 pic 的另一个名字分别为：gtroff,gtbl,geqn,及 gpic。nroff 用来进行基本的文本处理，troff 则可以用来作稍微复杂一点的排版工作。Tbl,eqn 和 pic 是 troff 的预处理工具，他们允许用户创建表格，显示数学公式以及绘制一些简单的图形。每种工具有自己的一组指令集，用户能够把这些指令和 troff 指令一起输入到输入文件中。GNU 版本的 troff 与 Unix 的 troff 兼容。用户可以使用相同的指令集运行这两种工具。但是，GNU 版本的 tbl,eqn 和 pic 的输出文件只能被 GNU 版本的 troff 处理，而不能被 Unix 版本的 troff 处理。

使用 nroff 和 troff 指令集进行文字处理可以变得非常复杂。为了简化操作，nroff 和 troff 有两个宏指令包：ms 和 mm。一条宏指令表示一组 nroff 指令。用户可以用宏指令代替指令。例如，为了新起一段，用户可以用一条宏指令来代替一组 nroff 指令。ms 宏指令包是为标准的文字处理操作设计的，而 mm 宏指令包进行一些诸如格式化信函、备忘录、列表之类的特殊操作。

与现在使用的文字处理程序比起来，Linux 中的文档处理工具看起来有点原始。用户直到输入文本和指令，然后对文本处理完之后，才能见到文档的格式。用户应该知道，当 70 年代开发这些工具时，很少有强大的文字处理器能够提供象 Unix 文档处理器提供的那样轻松的操作方法。即使在今天，nroff 和 troff 仍然是强大的文字处理工具。但是，他们确实缺少 WYSIWYG（所见即所得）文字处理器带来的轻松。现在，流行的文字处理器如 Corel WordPerfect 和 Microsft Word 在 Unix 和 Linux 系统上也实现了。在 Redhat's Applixware Office Suite，Caldera's StarOffice 和 Corel's WordPerfect7(www.redhat.com,www.caldera.com 和 www.sdcorp.com／wplinux.htm)软件包中收集了几种流行的商业文字处理器。

但是，nroff 和 troff 仍然是所有 Unix 系统以及 Linux 系统上的标准处理工具。用它们，用户能够轻松而快捷地处理简单文档资料。而且它们使用起来费用低又效率高。大多数工作是在简单的标准文本编辑器上完成的。然后可以把 nroff 和 troff 的任务放在后台运行，系统会抽出时间去处理它们。所有的 Unix 和 Linux 文档处理工具都包括一组复杂的指令集。

还有一个特殊的工具：groff,通过它用户可以启动其它工具。设计 groff 的目

的是帮助用户启动这些工具并且轻松地生成可打印的输出，尤其是 PostScript 输出。输出并不只是限于 PostScript。启动 groff 后，它将通过 gtroff 处理输出，然后取得 gtroff 的输出并为指定的设备处理 gtroff 的输出。缺省设备是 PostScript,ps。用户可以通过选项 -T 改变它，也可以通过设置配置变量 GROFF_TYPESETTER 来改变它。为了把输出送到打印机，须使用 -l 选项。下面的例子中，使用了宏 -ms 并打印文件 myfile。

groff -ms -l myfile

也可以通过管道输出到命令 lpr。

groff -ms myfile | lpr

如果需要将 PostScript 输出储存到文件中，就不能使用 -l 选项，而应将输出重定向到一个文件。

groff-ms myfile > myfileps

有了 groff，用户能够使用指定的选项，轻松地启动其它工具。正常情况下，当要使用 eqn 或 tbl 时，用户必须注意首先启动 eqn，然后是 tbl，最后是 troff，三者被连接在一个顺序管道中。但是，有了 groff 后，用户无须进行复杂的顺序管道操作，只要使用必要的选项，启动一次 groff 就行了。使用 -t 选项，groff 启动 gtbl 处理表格；使用 -e 选项，启动 geqn 处理器；使用 -p 选项，启动 pic 处理器。在本章，为了表明每一种操作如何与其它操作一起工作，所给的例子将单独启动每种工具。同时，对 groff 替换程序也将作些说明。下面的例子首先启动 geqn 和 gtbl。

Groff -e -t -l myfile

尽管 nroff 的输出能够在屏幕上显示，但 gtroff,geqn,gtbl,以及 gpic 的输出是为了被打印出来。当使用这些工具工作时，最好有一个 X-Windows 浏览器，这样每次要进行修改时不必把文档打印出来。当用 gtbl,geqn 或者 gpic 来格式化表格，数学式子，以及图表时，更是如此。groff 带有一个叫作 gxditview 的浏览器，当以选项-X 运行 groff 命令是它被启动。在用户的 X-Windows 桌面上将会开一个窗口来显示输出。单击窗口内部区域将打开一菜单，用于进行分页或退出浏览器操作。选项-X 使任何先前的后处理器的设置变为无效。在下一个例子中，用户在 groff 的 X-Windows 浏览器上显示了一个正在格式化数学式子和表格的文件的输出，而不是把输出打印出来。

groff -e -t -X myfile

为了使用 groff，首先使用任意一种标准文本编辑器创建一个输入文件。在输入文件里，用户输入 nroff 或 troff 指令以及文本。输入完成之后，把输入文件名作为参数启动 groff。groff 与其它过滤器不同，它读取嵌入在输入文件中的格式化命令。根据不同的格式化命令，groff 将格式化一个单词，一行，一段，或者是整个文本。每条指令是单独一行的，以圆点符号.开头。

输入文件中不分段这一事实使格式化操作更加复杂。输入文件中的一行可以被认为是一个基本格式化域。某些指令对一部分数行长的文本,几部分数行长的文本，甚至整个文本起作用。其它一些指令则用来设置整个文本的页面缩进，页眉、页脚。可以认为这些指令对文本中的所有行都起作用。

缺省情况下，nroff 在格式化输出中将填满各行。输入行通常在右页边之前以回车键结束。选项 fill 从下一个输入行取单词，然后用它们填满文本格式化版

本的当前行。从某种意义上说，fill 选项删除了输入行尾部的回车，而在这一行被填至右页边时插入它自己的回车。有了 fill 选项，输入行与输出行之间不在一一对应。这意味着用户可以在输入文件中输入任意多的回车。这与其它字处理器有很大不同，在这些字处理器里面，回车键被认为是一个格式化指令，它表明一个自然段的结束。用户能够组合几个格式化指令产生紧凑格式化效果。很多这种指令集都已被组合成宏命令。用户能够以相应的宏替代一组复杂的指令集。有两个宏包，ms(原稿)和 mm（备忘录）包。每个都是以 groff 选项的形式被启动。-ms 选项启动 ms 宏包。-mm 选项启动 mm 宏包。

例如，居中，增加下划线，以及缩排文本，输入硬回车和空行等操作只需要简单的指令。指令.ce 居中在它后面的输入行。用户可以使用.br 指令在输出中插入一个自己的回车。.br 指令将中断格式化，引起 nroff 起始新的一行。它使文本填充无效，并且强制在输出中插入回车。.sp 指令在输出中输入一空行。.ls 命令决定用户文档的行间隔。.ls 2 设置行间隔为 2，在实际效果上加倍了文本间隔。跟在这个指令后面的文本将加倍间隔，直至遇到另一个.ls 指令。

宏对复杂的格式化操作特别有用。一条宏命令代表一组 nroff 指令。例如，一个自然段可由指令序列.sp，.ti 指定，这两个指令分别引起新的一行和缩进。ms 宏.PP 代表这两个指令，它被用来格式化一个自然段。与规范的 nroff 指令一样，宏命令被单独放置在一行，前面紧接一个圆点符号.。宏命令都是大写的。这样就能够与小写的 nroff 指令区分开来。例如，宏命令.B，.I 和.R 设置当前字体类型。该类型一直有效，直至有另外的宏来改变它。.I 宏在文本中增加下划线。它代表意大利斜体字，某些打印机可能将文本置为斜体，而不是增加下划

线。宏命令.B 以黑体字显示文本。.R 则以正常字体类型显示文本。使用这个宏，用户可以把字体从宏.I 和 .B 变换回来。在下一个例子中，用户使用宏.B 以黑体字打印标题。然后，使用.I 宏在名字下加下划线。宏命令.R 把字体换回正常类型。在文本中,命令.I summer 只是值单词 summer"为斜体,而.B 置单词 beach 为黑体字。

```
.B
.ce
My Summer Vacation
.I
.ce
Larisa Petersen
.R
.PP
I spent my
.I summer
vacation
at the
.B beach
One day I went
swimming.
```

output

My Summer Vacation

Larisa Petersen

I spent my *summer* vacation at the **beach.** One day I went swimming.

22.1 排版

TeX 是用于对格式化文本进行排版的具有专业水准的实用工具。和 troff 一样，它读取一个带有标识符的标准字符文本文件，标识符指示 TeX 如何格式化文本。TeX 生成一个独立于设备的文件，因此它能够被转换成各种形式的输出，如一个 PostScript 文件或者 X-Windows 屏幕显示。和 troff 一样，TeX 有一些基本的低级命令，它们被组合成宏来完成更加复杂的任务，如格式化文本标题和段落。用户可以为任何一类文档设计自己的宏。TeX 被普遍地用于对含有复杂的数学式子的技术文档进行排版。而且它在任何 Unix 和 Linux 系统之间是可移植的。

TeX 的设计使得它很容易扩展。用户能够通过定义自己的宏来扩展 TeX 的标准宏指令集。利用这种扩展能力，可以为特定的任务定义宏指令包，从而开发一些基于 TeX 的程序。La TeX 就是这样一个程序，它提供了简化桌面出版任务的宏指令集。LaTeX 本身是可扩展的，能够加载和使用任何数量的宏指令包。另一个基于 TeX 的程序是 MuTeX，它使用户能对音乐进行编辑。BibTeX

则是用来生成目录的。用户还能够获取打印机驱动程序以及详细的文档资料。这些工具的大多数已经被集成到软件包 TeTeX 中。TeTeX 包括 TeX,LaTex,dvi 驱动程序,几个流行的宏指令包，支持程序，以及详细的文档资料。TeTeX 是为 Unix 和 Linux 系统设计的，在 OpenLinux CD-ROM 中包括了 TeTeX。

用户可以从因特网站点得到包括了宏指令包和支持程序的最新 TeX 软件，那些因特网站点是 TeX 综合档案网络（CTAN）的一部分。CTAN 是由三个组织的成员共同维护的一个合作工程，这三个组织是：TeX 用户协会（TUG），TeX 德语用户协会（DANTE），以及英国 TeX 用户协会（UKTUG）。每一个 CTAN 因特网站点收集了大量的 TeX 程序，宏指令包，以及不断在升级和完善的工具。所有站点收集的内容是相同的。如需要站点各目录内容的信息，可以查看各目录的 README 文件。流行的 CTAN 站点有：

ftp.cdrom.com ／pub／tex／ctan

ftp.tex.ac.uk ／tex-archive

ftp.dante.de ／tex-archive

从各个 TeX 用户协会的因特网站点，如 www.tug.org，用户能够获取很多 TeX 因特网资源，诸如在线教程、Tex、LaTex 相关软件，以及发行的用户手册。用户也可以连接某些有用的万维网站点，如 Don Knuth's 主页。相应的 ftp 站点，如 ftp.tug.org,存放有完整的 CTAN 收藏品。TUG 用户协会还出版一种以 TeX 为主题的会刊，TUGboat。

用户的 TeTeX 软件包已经包括了丰富的文档资料。一旦安装了 TeTeX，就可以在目录／usr／TeX／texmf／doc 和它的子目录下找到这些文档资料。

TeXFAQ 存放在 help 子目录中。子目录 fontname 包括了关于 TeX 中的所有字体的文档资料。目录／usr／TeX／info 包括了有关 LaTeX,字体，以及其它主题的详细文档资料。这些文档资料是以一种特殊的格式：textinfo 文件形式出现的。可以使用 info 命令加主题名阅读这些文件。命令 info info 启动了关于 texinfo 的教程。下面的例子启动 LaTeX 和字体名的手册。

```
$  info  latex
$  info  fontname
```

用户也可以阅读 http:／／sunsite.unc.edu／LDP／HOWTO 站点的 Linux 文档工程提供的 TeTeX HOWTO 资料。

22.2TeX 文件

与 troff 一样，用户可以用任何标准文本编辑器，诸如 Vi,Emacs,或者 Crisplite，创建一个 TeX 文件。TeX 的文件名应有扩展名.tex。一旦用户有了 TeX 文件，就可以用命令 tex 处理它，如下所示：

```
$ tex myfile.tex
```

如果用户使用本章后面介绍的 LaTeX 宏指令包，就需要使用命令 latex。

```
$ latex myfile.tex
```

命令 tex 和 latex 都生成一个 dvi 文件，用户可以把这个文件转换成不同形式的输出。dvi 文件名的前缀和生成它的 TeX 文件的前缀相同，但扩展名变

为.dvi。处理 TeX 文件产生的任何消息、警告或错误都存放在扩展名为.log 的文件里面。处理 myfile.tex 文件过程中的任何错误可以在文件 myfile.log 中找到。

用户使用众多 dvi 驱动程序中的一种生成自己想要的输出类型。生成 PostScript 文件的 dvi 驱动程序是 dvips。它把 dvi 文件作为参数，然后文件输出到标准输出设备上。如果有一台 PostScript 打印机，用户可以直接把 dvips 输出送到打印机。否则，为了使用这台打印机，用户需要用 Ghostscript 来处理 PostScript 数据。下面各条指令先生成 PostScript 输出，然后把它送到打印机；第二个例子使用 Ghostscript(gs)。

```
$ dvips myfile.dvi | lpr
$ dvips myile.dvi| gs | lpr
```

为把输出送到一个文件，可以使用选项 -o 文件名 选项。下面的例子生成一个名为 myfile.ps 的 PostScript 文件。用户可以用 ghostview 命令在桌面上查看这样的文件，如下所示。

```
$ dvips -omyfile.ps myfile.dvi
$ ghostview myfile.ps
```

由于 nroff 允许用户输出到屏幕，而 troff 为了得到详细输出需打印出来，用户可以把 TeX 输出显示到一个 X-Windows 窗口中去。从用户的窗口管理器执行的 xdvi 命令将把一个 dvi 文件的格式化输出显示在一个桌面窗口中。这样做，用户在查看文件输出时不必把这个文件打印出来。

```
$ xdvi myfile.dvi &
```

xdvi 使用一些单字母命令来移动被显示文件。n 将把文件下翻一页，b 或 p 把文件上翻一页。g 页码数将移至特定的一页，q 将退出程序。u、d、l 和 r 键分别上移、下移、左移、右移三分之二窗口。c 将移动页面使把光标所在位置在屏幕正中。

22.3TeX 命令

与 troff 不同，TeX 可以识别字处理组件，如句、单词以及自然段。一个单词是一组字符，组与组之间被任意多空格分隔开。空格的数量将被忽略，TeX 只把它们当作一个空格。一个句子是任意以句号、问号、感叹号，或者冒号结尾的单词。段的结尾以一空行标示。重复的空行被认为是同一行。TeX 命令列举在表 22-1。

TeX 命令置于文本中，以一个特殊的字符开头。所用的字符取决于操作的类型。标准的 TeX 命令以一个反斜杠或转义字符为前缀。TeX 使用的特殊字符有：＼、{、}、#、$、%、-、^、&以及空格键。空格键分隔开各单词，%加在注释前面，而＼加在一个 TeX 命令前面。例如，所有的 TeX 文件需要以命令＼bye 结尾。这样将结束 TeX 的处理过程。如果用户需要这些特殊字符（包括空格键在内）成为文本的一部分，只需在它们前面加一个反斜杠（反斜杠本身除外）。＼%输出字符%，后跟空格键的＼输出空格。＼＼ 是一个特殊命令，用来插入一空行。为了输出反斜杠字符，用户必须使用命令＼backslash。TeX

中特殊字符列举在表 22-2。下面的例子是一个有两个自然段的 TeX 文件。用户可以在任意行上以自己的方式输入文本。TeX 将会把它格式化成合适的自然段，忽略多余的空格键和任何 TeX 注释。

```
myfile.tex
 hello there
\bye
hello there
```

TeX 还允许用户使命令只对指定的一组文本有效。可以用左括号和右括号以及命令\begingroup 和\endgroup 分组文本。单个 TeX 命令或几个命令就放在左括号右面。这些命令对整组文本适用。如果用户只想改变一个标题或几个单词的字体，而不想显式改变其余文本的字体，这种分组适用会很有帮助。下面的例子只是把标题的字体变为意大利斜体：

```
{ \it my title }
```

22.3.1 段落

命令\par 将新起一段。用户可以用它替代一空行。可以把\par 命令嵌在文本的任何地方，它们将在各嵌入点另起一段。\noindent 将取消一自然段的首行缩进。它和 nroff 的 LP 命令类似。用户能够用命令\parindent 缩进量 来控制段落缩进。正的缩进量引起缩进，负的缩进量引起伸出。

有了\break 命令,用户能够插入一空行或一空白页。在一自然段中，\break

插入一空行，而在段与段之间它插入一空白页。命令＼nobreak 使用户能够保持那些本来要分散在两页文字在一页之中。例如，要使一行文字在同一页或同一段中，用户可以在这一行前加命令＼nobreak。

自然段页边距用命令＼leftskip 和＼rightskip 指定。正值表示增加页边距，负值表示减少页边距。＼leftskip 1 in 增加左页边距一英寸。＼rightskip -.5 表示减少右页边距半英寸。

22.3.2 间距

TeX 里有几个水平方向和垂直方向的置空操作。用户可以按照字符宽度（em），英寸（in），象素点数目(pt)，或者毫米(mm)来度量间距。命令＼hskip 和＼hfill 对行文本进行操作，它们允许用户增加间距或指定间距尺寸，或者填满该行文本的空间。例如，＼hskip 1 in 在一行中插入一英寸长的空间。＼hskip 4 em 将插入相当于四个字符尺寸的空间。

如果在某一行上还有空间，＼hfill 命令能够填满这些空间。对字符较少的行，如果想左对齐或右对齐行文本，这种功能是很有用的。如果＼hfill 命令放在文本前面，将文本右对齐；如果＼hfill 命令放在文本后面，将文本左对齐。如果在文本前后都有 ＼hfill 命令，则将使文本居中。

竖直方向上的间距是由命令＼vfill 和＼vskip 控制的。指定了间距尺寸的＼vskip 命令将在输出中留出指定大小的空间。＼smallskip,＼medskip ,和＼bigskip 是＼vskip 的简易版本，它们不要求尺寸。＼vfill 命令将在它后面的自然段之间

分配有效空间。例如，要填充一份文档最后的半页空白，用户可以使用＼vfill命令把那部分空间分配到各个段落。

命令＼baslineskip 控制行与行之间的间距。可以用它做有效的双间距。命令＼parskip 控制段与段之间的间隔。用户能够根据自己的喜好增加或减少间距。

22.3.3 页眉和页脚

命令＼headline 和＼footline 控制页眉和页脚。这两个命令将一组 TeX 命令置于一对大括号中。命令＼pageno 是显示页码的有效工具。在文本右边或左边的＼hfill 命令分别将把文本放置在左边的或右边的页边距。下面的命令把页码和标题"Chapter One"放在页眉的右页边距。另外，在页脚的右页边距也有页码。

```
＼headline={＼hfil Chapter One}
＼footline={＼hfil ＼the＼pageno}
```

22.3.4 字体

TeX 命令有自己的内部字体。用户能够很容易地改变字体。字体一旦改变，当前字体将一直有效，直至指定另一种字体。内部字体中，＼rm 代表罗马字体，＼tt 打字机字体，＼bf 代表黑体，＼sl 代表斜体字，＼it 代表意大利斜体。

TeX 有很多字体，用户可以自己定义并且在文件中使用。用户可以重新定

义一种已经存在的字体定义，也可以创建一种新的字体定义。例如，假设要使用 Helvetica 黑体，而不想用标准的黑体，并且想用＼bf 定义引用它。用户可以重新定义＼bf 来引用 Helvetica 黑体字。＼font 命令允许用户重新定义一种字体或定义一种新的字体。

＼font＼font-definition=font-name

一种 Adobe PostScript Helvetica 黑体字的字体名是 phvb8r。如果想重新定义＼bf 来引用这种字体，需使用下面的字体定义：

＼font＼bf=phvb8r

如果想创建一种新的字体定义，并且通过它来引用一种字体，用户应为新的字体定义取一个自己的名字，也就是别的命令没有在使用的名字。下面的例子为 Adobe PostScript Avant Garde Bold 字体创建了名为 agbf 的字体定义：

＼font＼agbf=pagb8r

一个字体名字由下面几部分组成：提供者，字体总称，字体粗细，以及其它变化量。提供者通常是某个公司，如 Adobe 公司，苹果公司。提供者由单个字母表示。例如，p 表示一种 Adobe PostScript 字体。一种字体的总称通常是一个由两个字体组成的代码。ag 代表 Avant Garde,hv 代表 Helvetica。字体的粗细是指字体的类型，诸如黑体或者意大利斜体。它由一个字母表示，例如，b 代表黑体，i 代表意大利斜体，r 代表规则体。这些变化量与实现细节有关。许多 Adobe 字体的变化量是 8r，表示采用 8 位编码。字体名字的完整清单可以在／usr／TeX／texmf／docfontname 的.map 文件中找到。这里是一些字体名的例子：

phvr8rAdobe PostScript Helvetica Normal
phvb8rAdobe PostScript Helvetica Bold
pagb8rAdobe PostScript AvantGarde Bold

为了指定一个特定的键盘字符，可以使用命令＼char，后跟该字符的字符编码值。这对使用特殊字符集得字体很有帮助，比如一种符号字体或者 Zaph Dingbats。例如，如果用户想在 Zaph Dingbats 字体里显示检验符，可以用命令＼char及其键盘编码值来指定这个字符，在这里，使用命令＼char 52 .

下面的例子展示如何用 TeX 命令来格式化文本和定义字体。TeX 命令的使用由下列各部分组成：组命令和文本，大括号对{} 和＼begingroup ,＼endgroup 对。用空行与＼par 命令来起始自然段。输出显示在图 22-1 中，但是图中没有此部分的编号。

```
＼font＼cbf=pcrb
＼font＼hbf=phvb
＼headline{Chapter 21 ＼hfill}
＼footline={＼hfill Page ＼the＼pageno}
＼def＼geginmylines{ ＼begingroup ＼medskip ＼noindent ＼cbf  }
＼def＼endmylines{＼medskip ＼endgroup}
＼noindent  { ＼hbf TeX Files }
```

As with troff, you create a TeX file with any standard text editor such as Vi,Emacs,or Crisplite. A TeX file should hae the

extension ＼bf .tex ＼rm in its name. Once you have a TeX file you process it with the {＼cbf tex} {＼it filename} command as shown here.

＼medskip
＼noindent { ＼cbf ＼$ tex myfile.tex}
＼medskip

If your file used LaTeX macros as described in later sections then you need to use the {＼cbf latex} command

＼beginmylines
＼$ latex myfile.tex
＼endmylines
＼par Both the tex and latex commands will generate a dve file that you can then convert to different forms of output . The dvi file will hae the same prefix as the original TeX file but with a {＼bf .dvi } extension. The previous command generates a dvi file called { ＼bf myfile.dvi}.
＼bye

1 TeX Files

As with troff, you create a TeX file with any standard text editor such asVi,Emacs, or Crisplite. A TeX file should have the extension .tex in its

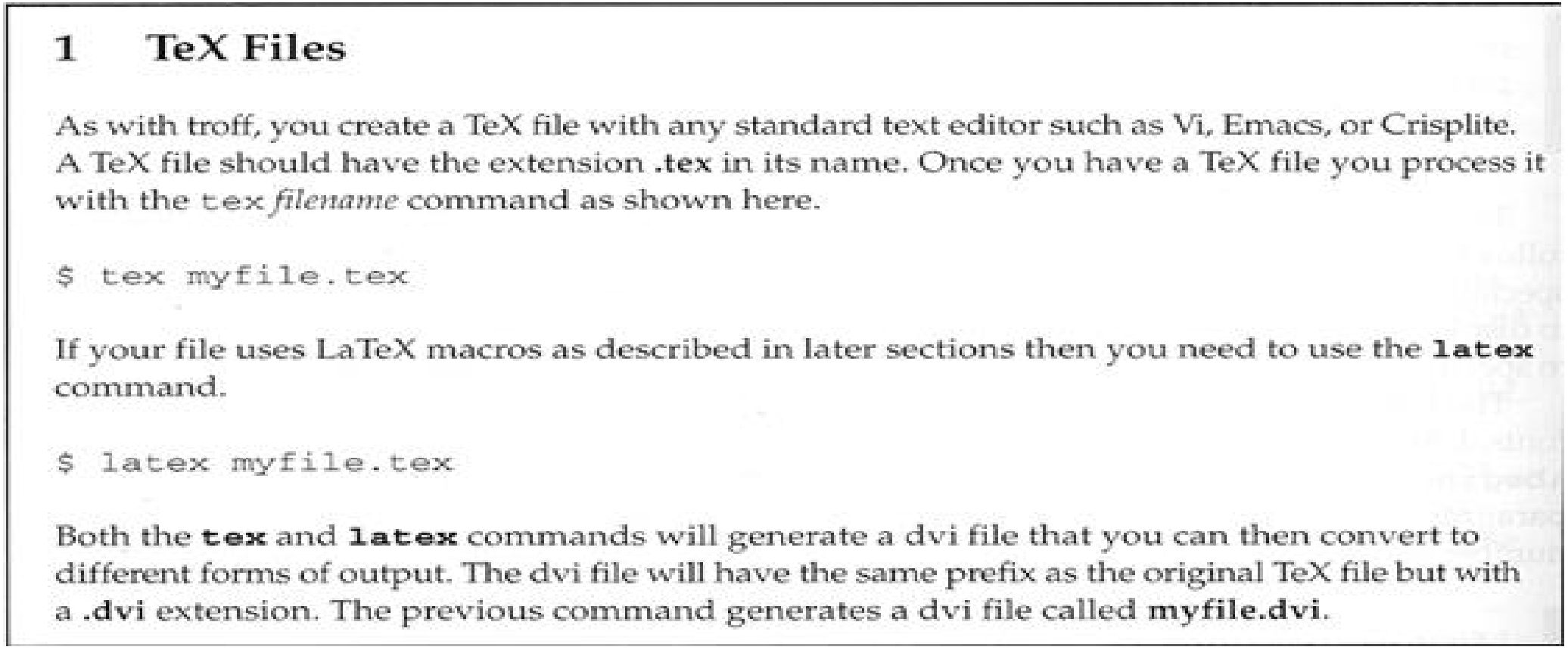
name.Once you have a TeX file you process it with the tex filename command as shown here.

```
$ tex myfile.tex
```

If your file uses LaTeX macros as described in later sections then you need to use the latex command

```
$ latex myfile.tex
```

Both the tex and latex commands will generate a dvi file that you can then contert to different forms of output. The dvi file will have the same prefix as the original TeX file but with a .dvi extension. The previous command generates a dvi file called myfile.dvi.



1 TeX Files

As with troff, you create a TeX file with any standard text editor such as Vi, Emacs, or Crisplite. A TeX file should have the extension **.tex** in its name. Once you have a TeX file you process it with the `tex` *filename* command as shown here.

```
$ tex myfile.tex
```

If your file uses LaTeX macros as described in later sections then you need to use the **latex** command.

```
$ latex myfile.tex
```

Both the **tex** and **latex** commands will generate a dvi file that you can then convert to different forms of output. The dvi file will have the same prefix as the original TeX file but with a **.dvi** extension. The previous command generates a dvi file called **myfile.dvi**.

图 22-1LaTeX 和 TeX 文本格式化输出结果

22.3.5 数学公式

和 eqn 一样，用户能够把数学公式嵌在文本之中，或者把它们放在自然段之间。符号$被用作起始和结束标志来指定数学式子。有特殊的 TeX 命令来格式化数学符号，诸如＼pi表示圆周率以及＼int表示积分（见表 22-3）。为了在

文本中嵌入一个式子，用户须用两个单重$把式子括起来。为了在自然段之间显示式子，用户须使用两个双重$（即$$与$$）把式子括起来。在下面的例子中，2＋3 将被放置在文本里面，而 x*y 将在空白处居中显示，将一行文字分开。下面的句子将生成例子中的输出：This is the $2＋3$ embedded formula.This is the display $$x*y$$ shown here.

This is the 2＋3 embedded formula. This is the display

x * y

shown here

下标和上标分别由命令^和_表示。下标命令的语法是 text_{subtext},其中的 subtext 部分将成为 text 的下标。a_{i}使 i 成为 a 的下标，ai。上标命令遵循同样的语法。text^{suptext}. x^{b}使 b 成为 x 的上标，x b。上标或下标可以是任何 Tex 和 LaTeX 命令，包括复杂的式子。

有许多关于显示数学符号的命令。其中较为普遍的两个是：＼sum 表示求和符号，＼int 表示积分符号。两个命令都需要两个参数，起始值以^为前缀，终值以_为前缀。

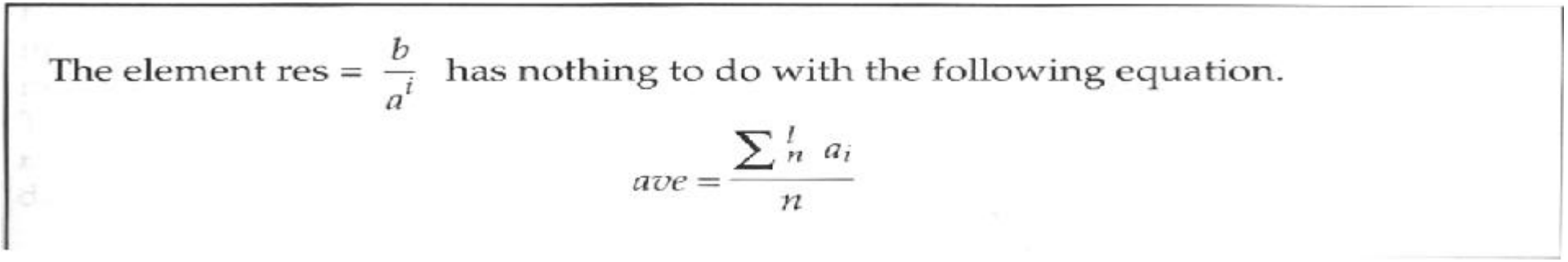


图 22-2 嵌在文本之中的以及单独的式子

下面例子的输出如图 22-2 所示。它展示了嵌在文本中的分数和单独一行显示的复杂式子。

```
\par The element $ res={b \atop \overline{a ^{i}}}$ has nothing to do with the following equation.
$$ave={\underline{ {\sum ^{1} _{n}} \; {a^ { l}}} \atop n} $$
\bye
```

宏指令经常被用户定制 TeX 操作。用户能够定义任何顺序的 TeX 指令集。用户可以使用命令\def来定义宏。格式如下：

```
\def macro-name {text}
```

宏操作是一种简单的文本替代。它以指定的文本代替宏的名字。这个文本可以是规则的文本，也可以是 TeX 命令。用户可以把宏作为一种简略长的标题或名字的手段。例如，下面的宏将把整篇文本中的"myed"以"The New Vi Editor"。

\def\myed{The New Vi Editor}

文本部分甚至能够包括对那部分文本起作用的 TeX 命令。

\def \myed{The \tt New \rm Vi Editor }

TeX 的宏能够带参数，允许用户在每次使用时指定不同的参数。用符号#定义参数,并且分配给每个参数一个数字。\agegp#1#2 定义了两个参数,宏 agegp 分别以#1 和#2 引用这两个参数。为了在宏的文本中引用参数，用户须用符号#和参数的编号。#1 引用第一个参数。

下面的例子定义了一个有两个参数的宏。

\def \agegp#1#2{This is a #1 movie for #2}

当使用带有参数的宏时，用户指定的每个参数必须分别被一对大括号括起来。一个带有参数的宏在使用时，每个参数的编号必须始终有不变。例如，宏 agegp 总是必须有两个参数，不能多也不能少。

\agegp{terrible}{teenagers}

\agegp{great}{adults}

这些宏的输出是：

This is a terrible movie for teenagers

The is a great movie for adults.

22.4LaTeX

LaTeX 是提供简单易用的格式化功能的一组宏指令。用户能够不必详细了

解复杂的 TeX 命令集，而使用相应的 LaTeX 宏命令。在很多方面 LaTeX 类似于 nroff 和 troff 的宏指令包 mm，使用户能够生成不同类型的文档。LaTeX 宏指令使用户把注意力集中到文本的总体布局排版。最新的 LaTeX 版本是 LaTeX2e，被收集在 OpenLinux 的 TeTeX 包中。LaTeX3 目前还在开发之中。

LaTeX 的文档资料可以在目录／usr／TeX／texmf／doc／latex 中找到。子目录 general 包括了 LaTeX 使用手册的 dvi 版本。用户可以使用 dvips 生成 PostScript 版本，然后用 Ghostview 阅读。不同包的文档在不同的子目录里面，比如，generic,latex 和 tools.fonts 包括各种当前系统字体的清单。大多数文档只是以 dvi 文件的形式出现。阅读它们的最简单方法是首先用带选项 -ofilename 的 dvips 命令生成 PostScript 文件，然后用 Ghostview 显示它们（用户也可以使用 xdvi，比较费时间）。下面的一组命令生成一个 PostScript 文件，然后使用 Ghostview 阅读它们。

```
Dvips -omyman.ps manual.dvi
Ghostview myman.ps
```

LaTeX 有三种基本操作模式：段落，数学，以及 LR（从坐到右）。在段落模式下，LaTeX 自动填充段落，使文本对齐于页边距。在数学模式下，LaTeX 创建数学式子并进行显示。在 LR 模式下，LaTeX 运行起来象一个画图程序，它可以画图和重复文本。用来创建表格的＼tabular 环境在段落模式下不工作，在用户输入表格数据的行末尾，它要求显式的行间隔。一行间隔可以以输入两个反斜杠得到。

22.4.1 文档类

有了 LaTeX，用户能够格式化不同类型的文档。LaTeX 定义了几种文档类型，称之为类。论文类用于刊物论文、书籍类用于全集之类的文档资料、报道类用于长篇报道。为了创建一篇 LaTeX 文档，用户首先必须定义将使用的＼documentclass 宏的文档类。宏＼documentclass 把类的的名字作为参数。用于 LaTeX 文档的文本被组织在一个序言和一个文本体中。文本体由宏＼begin{document}和＼end{document}定义。用户把自己文本的文本体放在这两个宏之间。任何在宏 begin{document}以前输入的文本都被当作序言。下面的例子定义了一个含有论文类的文档，并且建立了文档的文本体：

```
＼documentclass{article}
＼begin{document}
This is my text
＼end{document}
```

一旦创建了 LaTeX 文件，就可以使用 latex 命令，用 LaTeX 宏进行 TeX 处理。dvi 文件就用 tex 命令生成。然后，用户就可以使用 TeX 的转换命令生成输出。

```
$ latex myfile.tex
```

用户可以使用四种标准的文档类型：论文类，信函类，报道类，以及书籍类。每一种特定的类都有自己的定制命令。例如，命令＼closing 在信函类中被用于指定未签名的致敬词，如 Sincerely".论文类原来是为刊物论文设计的，现

用于篇幅较短的文档。书籍类用于包含有单独章节的大型文档，这一类的文档甚至能够被分为几部分。报道类用于中等篇幅的文档资料，它可能含有好几个章节，但是篇幅不如书籍类来得大。信函类有两种可选格式，商务和个人。它能组织一封信件，把地址和日期置于适当的位置。此外还有一种特殊的名为幻灯片的文档类，用于准备幻灯片。不同文档类及它们的操作列举在表 22-4 中。

为了设置整个文档的总体布局特征，用户须为宏＼documentclass 指定选项。选项被包于括号中并位于＼documentclass 名字后面。用户能够指定诸如缺省字体尺寸，页面数，纸张大小，公式的位置之类的特征。下面的例子将输出在双面纸(twoside)上并且设置字体尺寸为 12 个象素点（12pt）.多重选项由逗号分隔开。

```
＼documentclass［twoside,12pt］{article}
```

22.4.2 包

LaTeX 本身是可扩展的。用户能够创建新的宏定义并且把它们当作命令在 LaTeX 文档中使用。这种可扩展性使得用户能够为专门的任务开发自己的 LaTeX 宏指令包。例如，graphicx 包是为引入 epsf 文件定义的宏，而旋转包允许用户在页面上旋转文本。用户可以在安装 TeTeX 时或从许多因特网站点找到很多类似的包。目前能得到的包都在子目录 texmf／tex 下。它们的扩展名是.sty。文档资料在／doc 子目录。

要把一个包装入文档，用户须在开始段用＼usepackage 命令。下面的例子

将 makeidx 包添加到文档中，makeidx 包使用户能够建立索引：

＼usepackage{makeidx}

用户也可以使用＼import 命令把任何文件的内容插入道自己的 LaTeX 文件中去。＼import 用于这一类文件：它包含有用户为自己的文档定义的宏。用户可以定义自己的宏，把它们放在一个文件里，然后想要在文档使用它们时，引入这个文件。

＼import myfile.tex

22.4.3 页面格式化

＼pagestyle 宏能够设置页面的布局。页眉和页脚有四种标准的格式，分别由＼pagestyle 宏的不同参数指定：empty,plain ,headings,以及 myheadings。empty 的输出没有页眉和页脚,plain 将只输出页脚，而且页脚只是由一个位置居中等页码组成。eadings 则在每一页面放置一个栏外标题，页面的格式则是由文档类型决定。myheadings 使用户能够使用标记命令为奇数页或偶数页或者就单面页指定自己的标题。

对 myheadings 文档类型，用户能够使用标记命令设置标题文本。命令＼markboth{left-head}{right-head}用于有奇数页标题和偶数标题之分的文档。为生成奇／偶数页，用户的文本类型必须有选项 twoside；例如，＼documentstyle［twoside］{article}。＼markright{rithf-head}只用于单面文档中的单面标题。

还有一些命令是用于设置一些比较特殊的特征的。用户能够用＼twocolumn

命令建立双栏文档，并且可以用命令＼onecolumn 换回到单栏文档。＼pagenumbering{num-style}命令允许用户设置页面编码风格。可能的风格有：阿拉伯字体，罗马字体，以及 alph（拉丁字体）。宏＼thispagestyle 使用户能够设置当前页的页面布局。

22.4.4 字体

LaTeX 支持一套标准的字体定义，详细列于表 22-5。它也有字体尺寸命令，使用户能够非常容易的改变字体尺寸。用户能够选择诸如＼tiny,＼small,及＼large 之类的类型尺寸，＼LARGE 则适用于所有的大写字母。字体缺省尺寸是＼normalsize。

新字体的定义应在文本体之前的开始段。使用命令＼font＼ def-name=fontname。下面的例子为 PostScript Helvetica 字体创建了定义并且命名为 helf.于是能够以字体名＼helf 引用该字体。

```
＼documentclass{article}
＼font＼helf=phvr
＼begin{document}
＼helf This text is in helvetica,＼bf but this is in normal bold face.
＼end {document}
```

用户也可以用命令＼font 重新定义任何标准字体。用户可以为＼it 和＼bf 定义一种不同的字体。

22.4.5 分段

LaTeX 对于建立用户文档的不同组件特别有用，诸如建立某一部分的报头，标题，以及一个目录表格。论文类使用宏＼section,＼subsection,＼subsubsection,＼paragraph,＼subparagraph,以及＼appendix。除了＼appendix，它们都把一个报头名作为他们的参数。

＼section{ C compiler}
＼subsection{Libraries}

报道和书籍类有另外两个宏：＼part 和＼chapter。＼chapter 定义了一个章节标题，而＼part定义了书籍类或报道类中的一组章节。

这些分段宏被用来生成目录表格。宏＼tableofcontents 将生成目录表格。如果用户不想让一个特定的报头显示在目录表格中，可以在宏的名字后面置一个星号*,将此报头去掉。命令＼section*{computers}将不会显示在目录表格中。表 22-6 列出了分段宏指令。

为了新建一个标题，用户可以在文本体的一开始使用＼maketitle。但是，标题信息放在开始段中，在文本体前面。命令＼title{My title}建立了标题"My title"。放在这里的另一信息是作者，由宏＼author 指定。下面的一组宏指令设立了标题，然后创建它。

＼title{Hockey}
＼author{Christopher}
＼begin{document}

\maketitle

22.4.6 脚注与交叉引用

LaTeX 还支持脚注和交叉引用。用户创建脚注所使用的命令取决脚注是在外部段落模式还是内部段落模式下引用文本。外部段落模式是正常文本的段落模式，而内部段落模式是用于被包含在某些其它环境下的文本，如在图形框或一个表格单元中。表 22-6 列举了脚注和交叉引用的命令。

对于外部段落模式，用户使用命令\footnote [number] {text}。一个编了号的脚注文本位于页面底部。使用可选择的参数，数字，用户能够为自己的脚注指定数码。对于在内部段落模式中的脚注文本，用户可以使用宏命令\footnotemark 创建文本中的引用。然后，在外部段落模式下，用户使用命令\footnotetext [number] {text}来建立位于页面底部的脚注文本。

用户使用宏\label 和\ref 来建立交叉引用。命令 label 把一个关键词作为它的参数，有了这个关键词就能够引用文本中对应的文字。关键词可以是用户自己取的任何名字。然后使用带有该关键词的宏命令\ref 引用文本中的对应文字。\ref 将会生成对应于文本中被引用的文字的段编号。\pageref 则能够引用页码。

```
\label{myplace}
\ref{myplace}
\pagere{myplace}
```

22.4.7 环境

LaTeX 有一套环境，在这套环境里能够执行不同的任务，有些是用它们自己的一组命令。可以在 tabular 环境中使用 tabular 环境创建表格。图像环境将为页面中的一幅图定位和编号，并且允许用户使用命令＼caption 为它输入插图说明。表 22-7 列出了几个经常使用的 LaTeX 环境。

环境以宏＼begin{env-name}开始，以相应的＼end{env-name}宏结束。有些环境有专门的命令供用户使用，诸如＼circle 用于在图形环境中绘制圆。其它的环境，诸如 tabular 环境，也寻找特殊格式化文本。还有其它一些格式化任何被包括在内的文本。例如，后跟一行文本然后是宏＼end{quote}的命令＼begin{quote}将缩排首行并引用任何包含在内的文本。

某些特定的环境在段落模式下起作用，而其它的在 LR 或者数学模式下起作用。那些在 LR 模式下起作用的环境在每行末尾需用命令＼＼linebreak 生成行间隔。有如果没有，LR 模式将把环境里的所有行读成一行，而不是分开的几行。tabular,picture 和 eqnarray 都是 LR 模式环境。当创建表格和图时，用户用户必须在每一行的末尾输入＼＼。其它 LR 模式环境有：center,flushright,以及 flushleft。这些环境被设计成对单独的一行起作用，分别居中文本,或者将行文本左对齐或右对齐。每一行必须以行间隔＼＼结束。

```
＼befin{center}
This is centered text ＼＼
And so is this ＼＼
```

＼end{center}

为了在段落模式下进行居中或者填充文本操作，用户将要在环境或主要文本中使用命令＼centering,＼raggedleft,以及 raggedright。＼raggedright 将使各自然段左对齐，而右边未对齐。＼raggedleft 则将使各自然段右对齐。这些命令都是对整个起作用。表 22-6 列出了用于格式化自然段得 LaTeX 命令。

下面的例子说明了用于格式化文本的 LaTeX 环境的使用。字体的定义以及根据不同文本而对字体的分组与它们在 TeX 中的相应操作有很多相同之处。命令＼documentstyle 中的选项 twoside 允许用户使用＼markboth 设置偶数页和奇数页。输出与图 22-1 所示一样。＼documentstyle［twoside］{article}

＼font＼cbf=pcrb

＼font＼crf=pcrr

＼pagestyle{myheadings}

＼ markboth{＼ Chapter 21 ＼ hfill}{＼ hfill Linux: The Complete Reference＼}＼begin{document}

As with troff,you create a TeX file with any standard text editor such as Vi,Emacs,or Crisplite A TeX file should have the extension ＼bf .tex ＼rm in its name.

Once you have a TeX file you process it with the ＼cbf tex ＼it filename ＼rm command as shown here

```
\begin{flushleft}
\crf \$ tex myfile.tex \\
end{flushleft}
```

If your file uses LaTeX macros as described in later sections the you need to use
the {\cbf latex} command.

```
\begin {flushleft}
\crf \$ latex myfile.tex\\
\end{flushleft}
```

Both the {\cbf tex} and {\cbf latex } commands will generate a dvi file that you can then convert to different forms of output. The dvi file will have the same prefix as the original TeX file but with a {\bf .dvi extension. The previous command generates a dvi file called { \bf myfile.dvi}.

```
\end{document}
```

22.4.8 计数器

LaTeX 为那些需要进行对象编号的环境保留了计数器变量，如图象，公式，以及表格环境。计数器的名字和环境的名字一样。此外，分段命令也需要计数

器，以便对各部分进行计数，如章节，子节，以及自然段（见表 22-6）。注脚和页面也需要计数器。用户可以使用命令＼value{count-name}存取某一计数器的值。

命令＼addcounter{name}{increment}将使命令中 name 指定的计数器的值加一，而＼addcounter{name}{value}命令将重新设置 name 指定的计数器的值。命令＼newcounter{name}新建一个名为 name 的计数器。用户能够使用命令 ＼alph,＼roman,以及＼arabic 来改变计数器的显示，使它分别以拉丁文，罗马数字，或是阿拉伯数字显示。

22.4.9 列表

LaTeX 有几个列表环境，用来创建编号表，简报表，或者标号表。enumerate 环境创建编号表，描述标号表，以及项目简报表。一个表的各项可由命令＼list 建立。在不同的环境下，表有不同的选项。例如，在 description 环境下，表把各项的标签作为选项，即描述名。下面的例子生成了一个项目编号表。

```
＼begin{enumerate}
＼item { Clean house}
＼item{Mow lawn}
＼item{Get gas}
＼end{enumerate}
```

以下的 description 环境创建了一个项目描述列表。每一项的标签以括号括

起来。标签将显示在项目左边，右对齐且是黑体。

```
\begin{description}
\item [Aleina] {Book on drawing with pens and paints.}
\item [Larisa] {A new library card for all the books that she wants to read over again.}
\item [Cecelia] {A new word processor.}
\end{description}
```

环境 itemize 创建一个简报表。下面的例子中，在表中等每一项前面放置一个项目符号：

```
\begin{itemize}
\item{milk}
\item{yogurt}
\item{vegetables}
\end{itemize}
```

有了 list{label}{spacing}环境，用户就能够对列表的打印进行更多的控制。list 环境需要两个参数，一个标签以及一个间距参数。标签参数通常是一组决定列表如何被标识的 LaTeX 命令。间距参数指的是标签与其对应的项之间的间距大小。用户可以使用这个命令建立一些标签是图像或者图片的列表，而不是标准的项目符号，单词，数字。label 参数可以是能够创建图形的 LaTeX 图像或图片命令。在下面的例子中，列表标签是一个使用 Zaph Dingbats 字体的选中图形。首先，命令\zdf 定义了 Zaph Dingbats 字体(pzdr)。用\char 命令引用

该字体中的一个特定字符。＼zdf＼char 52 命令 就是检验号图形。这个命令用在命令＼list 的 label参数，使检验号成为列表项的标签。见图 22-3

```
＼documentclass{article}
＼newfont{＼zdf}{pzdr}
＼begin{document}
＼begin{list}{＼zdf＼char 52}{Gift List}
＼item{Book on drqwing with pens and paints.}
＼item{ A new library card}
＼item{A new word processor.}
＼end{list}
＼end {document}
```

有个名为 dingbats 的包定义了 dingbat 环境，用户能够轻松地使用命令引用和使用 Zaph Dingbat字符。在目录 doc／tools 有相关文档资料。

22.4.10 表格

有两个标准的 LaTeX 环境应用于表格：table 和 tabular。tabular 环境创建一个表格，而 tabel 环境将在页面上定位这一表格，并且对它编号和标识。还有另一个名为 supertabular的外部环境可用于篇幅超过一页的表格。

tabel 环境允许用户定位表格并且对表格进行标识。它有一选项，使用户能够控制表格的位置(见表 22-6)。选项 h 将定位表格于定义它的文本处，选项 t

将定位表格于页面顶部，选项 b 将定位表格于页面底部，选项 p 将其自身或与其它对象一起定位于页面。在 table 环境中，用户可以使用命令＼caption 创建表格标题。表格的编号自动生成。表格本身通常是由 tabular 环境创建的一个表格。在这种情况下，tabular 环境及制表命令被放置于 table 环境。但是，制作表格可以和加载一副外部 epsf 图片一样轻松(见本章后面的图形环境部分)。

```
＼begin{table}{placement}
The table
＼caption{table title}
＼end{table}
```

有了 tabular 环境，用户就能够创建自己的表格。它有一个强制参数用于列对齐。参数 column-align 参数指定行对齐的方式：左对齐，右对齐，或者居中。类似 gtbl，一个单字母代码对应一种方式。l 代表左对齐，r 代表右对齐，c 代表居中对齐。字符 | 将在列之间放置一行。命令{| l|r|r|c|}将左对齐第一列，右对齐接下去的两列，然后居中最后一列。每一列之间都有竖直线，在每一行的最后有一条外边界（框的边界）。tabular's position 选项允许用户在竖直方向上使表格照顶部的行或底部的行对齐（缺省是按中间行对齐）。

```
＼begin{tabular} [position] {column-align}
entry & entry ...& entry ＼＼
＼end{tabular}
```

用户把数据输入到表格的每一行。每一行必须以一个行间隔符＼＼结束。行中的每一项由符号&分隔开，把一行分成不同的域，每一个域对应于一列。

符号&需以空格与其它字符分开。下面的例子显示了一个共有三列的表格的一行数据。

War and Peace & Tolstoy & 15.75

命令\cline 和\hline 沿着表格的行跨列画线。\cline 将跨指定的数目的栏画一条线，而\hline 跨表格的整个宽度绘制一条线。两个\hline 命令用在一起将绘制一条双划线。\vline 在一行里绘制一条垂直线。如果想在行之间绘制一条线，用户可以在行间隔之后，在每一行的末尾放置一\hline 命令。命令\multicolumn{numcols}{align}{text}在表格中建立一个跨多栏的单元格，numcols 是所跨越的栏数，align 则是以 l,r,c 表示的对齐方式。下面的例子创建了一个四行三列的表格，都是左对齐。创建了一个跨越所有栏的标题并且使用\multicomumn 命令居中标题。也用\multicomumn 命令为每一栏创建单独的标题。

```
\documentclass {article}
\begin{document}
\begin{tabular}{|l|l|r|}
\hline
\multicolumn {3}{|c|}{Book List}\\
|multicolumn{1}{|c}{Title} &\multicolumn{1}{c}{Author}&
  [c|] {Price} \\  \hline \hline
    War and Peace & Tolstoy & 15.75  \\\\\hline
```

```
Christmas Carol & Dickens  &  3.50 \\\hline
Iliad & Homer & 10.25  \\\hline
Raven & Poe & 2.50  \\\hline
\end{tabular}
\end{document}
```

Book List

Title	Author	Price
War and Peace	Tolstoy	15.75
Christmas Carol	Dickens	3.50
Iliad	Homer	10.25
Raven	Poe	2.50

Book List

Title	Author	Price
War and Peace	Tolstoy	15.75
Christmas Carol	Dickens	3.50
Iliad	Homer	10.25
Raven	Poe	2.50

图 22-3A LaTeX 表格

tabular 环境的一个缺点是它不支持篇幅超过一面的表格。对于这种表格，用户可以使用 supertabular 包。这个包定义了 supertabular 环境，它拥有 tabular 环境所有的命令，但是增加了用于跨页表格的命令。标题和分标题能够在每页的顶部重新显示。详见 doc／latex／styles 目录下的文档资料。

22.4.11 数学公式：数学模式

LaTeX 当然支持 TeX 所支持的数学环境，但它也支持其它几种数学环境。LaTeX 既支持用于内嵌于文本的数学式子的单个$符号设置，也支持用于单独占一行的数学式子的双$符号设置。LaTeX 把这些定义为 math 和 displaymath 环境。用户能够轻松地使用＼begin(math)命令和 ＼begin(displaymath)命令来操作这类式子。在这些环境中也可以添加一些特征，例如等式的编号，将各等

式组织入数组。

除了 math 和 displaymath 环境，LaTeX 又增加了几种环境，用于添加一些其它特性，诸如对等式的编号和把等式组织入数组（如环境 equation,array,和 eqnarray）.环境 equation 将把等式放置于居中位置并且置等式编号于式子右边。

```
\begin{equntion}
math formula
\end{equation}
```

环境 array 是一个用于数学式子的 tabular 环境。和在 tabular 环境中一样，匹配的式子按列，按行排列。每一整行是由符号&分开的多栏行。每一行必须以一个\\结束。array 有一个用于指定每一栏的对齐方式的参数，就跟 tabular 一样（l 代表左对齐，r 代表右对齐，c 代表居中）。

```
\begin{array} {justification}
entry & entry ... & entry \\
\end {array}
```

环境 eqnarray 运作起来类似环境 array，但前者用式子编号标识每一行。这对根据相应的编号在垂直方向上列出数学式子这一操作很有帮助。

```
\begin{eqnarray}
formula \\
formula \\
\end {eqnarray}
```

LaTeX 还增加了几个用于格式化公式的新命令。命令＼frac{num}{dem}创建了一个分数，num 表示分子，dem 表示分母。下面的例子创建了分数 3／5。分子和分母也可以是任何复杂的数学式子。

＼frac{3}{5}

有许多命令用于显示数学符号。这里只对其中一些经常使用的符号描述了一下。在表 22-8 中有完整的命令清单。有两个经常使用的命令，＼sum 用于求和符号，＼int 用于积分符号。两个命令都需要两个参数，起始值以^为前缀，终值以_为前缀。

n ＼sum ^{1}_{ a_{i}

命令＼overbrace{text}和＼underbrace{text}分别把文本放置在水平大括号的上面和下面。命令＼overline{text}和＼underline{text}分别。把文本放置在水平线段的上面和下面 LaTeX 还支持不同形式的椭圆。＼ldots 产生一个正常椭圆。＼cdots 在一条线段的中点处放置一椭圆，＼vdots 显示一个竖直放置的椭圆，而＼ddots 显示一个 45 度角斜置的椭圆。有许多代表数学函数的符号，比如平方根号。＼sqrt [root-value] {number}显示 number 的 root-value 次方根。root-value 是可选的，如果没有，root-value 就不会显示，代表开平方根。下面的例子显示了 16 的平方根和 9 的立方根。

＼sqrt{16}

＼sqrt{3}{9}

下一个例子使用 LaTeX 命令格式化与前一个部分 TeX 例子中一样的数学式子。输出同图 22-2 所示。equation 环境用来对式子进行编号。

```
\documentclass{article}
\pagestyle{empty}
\begin{document}
The element $ res = \frac{b} {a ^ {i}} $ has nothing to do with the following equation.
\begin{equation}
ave=\frac { \sum ^{1} _{n} \; {a _{i}}} { n}
\end {equation}
\end {document}
```

22.4.12 图形

有了 LaTeX，用户就能够在文本中显示图形。图形可以是图片，也可以是照片。有好几种方法能够生成 LaTeX 文档的图片。用户可以使用 drawing 环境来制作，在该环境里有绘制图元的命令，比如直线和圆，或者可以用 xfig 之类的绘图程序来创建图片，然后将它作为照片引入。

图像

用户可以使用图像环境来定位、编号和标识一副图片或图形。图片可以是 drawing 环境创建的一幅草图，或者是一引入的图形，比如 PostScript epsf 文

件。用户在图像环境中包含这些命令和它们的图片环境。用户能够把这些中的任何放置于＼begin{figure} ＼end{figure}对中。在这一对中，用户可以用宏＼caption为图片创建一个标题。下面的例子创建了一个简单的方框：

```
＼begin {figure}
picture
＼caption{mybox}
＼end{figure}
```

引入照片和草图：Encapsulated PostScript Files(epsf)

用户可以以 encapsulated PostScript 文件的形式把照片或草图引入自己的 TeX 文档。如果想引入的照片或草图事先不是 epsf 格式，用户就必须转换它。例如，有一 jpef 或 gif 格式的图片，用户必须首先把它转换成 epsf 文件.xv，OpenLinux 系统提供有转换程序，能够轻松地完成这一转换。打开 xv（在 Xterm 窗口中输入命令 xv &）后，按鼠标右键弹出命令菜单。选择 load 菜单项，装入图片。然后选择 save 菜单项，选择该菜单的第一项把文件类型转换为 PostScript。这个 epsf 文件有扩展名.ps。然后用户就可以把它引入 LaTeX 文档。为了把一个 encapsulated PostScript 文件引入 LaTeX 文档，用户必须首先包含一个包，这个包中定义了执行这些操作的命令。有好几个这样的包，每一个包定义不同的命令。其中包括在 TeTeX 包中的有：epsf,graphics,graphicx,以及 epsfig。epsf 包是最简单的，在开始段中使用 input 命令就可以装入它。这个包中定义了一个名为＼epsffile 的命令，用它用户就能够引入一个 epsf 文

件。下面的例子引入了一个名为 mypic.ps 的 eps 文件。

```
\documentclass{article}
\begin{document}
\input epsf
\epsffile{temp.ps}
\end{document}
```

其它包，诸如 graphicx，包括了一些功能更强大的命令。例如，有了 graphicx 包，用户能够引入压缩的 epsf 文件。graphicx 和 epsfig 命令由命令\usepackage 装入。graphicx 使用命令\includegraphics 引入 epsf 文件。在目录 doc/latex/graphics 下可找到有关 graphicx 的文档资料。下面的例子引入了一个 epsf 文件，同时装入 graphicx 包，还使用图像环境对引入的图形进行居中对齐，编号，标识。命令\caption 用来制作一个标签。

```
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\begin{figure}
\includegraphics{mypic.ps}
\caption{This is my picture}
\end{figure}
\end{document}
```

草图 环境

LaTeX 有一个标准的草图环境：picture。有了它用户能够生成很简单的图元，比如圆，直线。在用户的 OpenLinux 系统中还有一些功能更强大的包，比如 eepic,pictex,texdraw,pstricks,以及 xypic。在使用这些包之前，用户首先必须装入这些包。在目录 doc／generic 下存有关于 texdraw,pstricks，以及 xypic 的文档资料。绘图环境的命令和前一章中描述的 gpic 实用程序类似。用户也可以使用 gpic 来生成用来绘制图元的 TeX 命令。

对任何复杂的草图来说，最好使用单独的绘图程序（如 xfig）首先建立草图。Xfig 被用来生成适用于各种绘图包的 LaTeX 命令。用户可以选择 epic,eeepic,pictex,以及 picture。生成含有这些命令的一个 TeX 文件，然后用户可以把它插入到一个标准的 LaTeX 文件。用户需确信已用命令＼usepackage 装入了所需的绘图包。

或者，用户可以把草图文件转换成 encapsulated PostScript（epsf）文件。使用定义的 epsf 命令，TeX 文件能够引入任何 epsf(PosScript)文件。很多绘图程序允许用户以 epsf 类型保存文件。用户也可以使用程序 xv 转换这些文件。引入 epsf 文件的方法将在下一部分描述。

Picture 环境

有了 picture 环境，用户能够建立简单的线条，圆，以及矩形。Picture 需

要两组参数来指定位置。第一个是宽度和高度意义上的尺寸，第二个是它在页面上的位置，以 x 和 y 坐标系衡量。单位是毫米。Picture 在 LR 模式下运行，因此每一行必须以一行间隔＼＼结束。Picture 环境有自己的一组命令（见表 22-6）。＼circle 创建一个圆。下面的例子创建一个直径为 100 毫米的圆。

＼circle{100}

＼line 绘制一条指定长度和斜率的线段。＼vector 绘制一条带箭头的线段。这些命令把 x 和 y 值确定的斜率作为它们的第一个参数。这些值在-4 与＋4 之间。第二个参数是线段或向量的长度。＼makebox 建立一个指定宽度和高度的框。使用＼oval 命令，用户能够创建一个也是指定了宽度和高度的椭圆。＼framebox 命令在一图元周围放置一边框。＼makebox,＼framebox,以及＼oval 使用户能够置文本于一指定的象限中。用户可以把下面代码的两两组合作为这些命令的选项：t 代表顶部，b 表示底部，r 代表右边，以及 l 代表左边。下面的例子将绘制一个边长 2 英寸的正方形文本框，文本从左边开始。

＼framebox [2 in] [1] {Input File}

用户能够使用命令＼put 把一图元定位在一个特定的坐标系中。否则，图元将会紧挨着放置。＼put 把指定图元位置的 x 和 y 坐标作为它的第一组参数。然后的参数是绘制图元。下面的例子中绘制了一个圆，位于离图片底部 50 毫米，离图片左侧边 20 毫米处。

＼puts{50,20}{circle{100}} ＼＼

下面的例子建立了一个有文本"Input File"的文本框，框上面有一圆，在它们之间有一带箭头的线段。

```
\begin{picture}{300,300}{10,10} \\
\framebox［1 in］{Input File}\ \
\put {-20,10}{\vector(0,0){50}\ \
\put{-20,80}{\circle{100}}\ \
\end {picture}
```

22.4.13 信函

letter环境在信函文档类型中运行。用户可以在含有信函文档类型的TeX文件中建立好几个 letter 环境，这样用户就能够在同一个文档写不止一封信。在letter 环境里面，对一封信函的不同部分（例如，签名和回邮地址。见表 22-9）可以使用好几个不同的信函 LaTeX 命令。letter 环境把收信人的名字和地址作为它的参数。下面的例子在一个信函文档类中建立了一个 letter环境。

```
\documentstyle{letter}
\begin{document}
\begin{letter}{name \\ address \\ city ,state zip}
\end{letter}
\end{document}
```

信函以命令\opening{text}打头，然后紧跟着的是信函的文本。\opening的文本参数是用户在信函开始处对收信人的客气称呼，比如 Dear Sirs。

```
\opening {Dear Sirs}
```

信函的文本被认为是完整的信件的内容，在它里边没有任何 LaTeX 命令。

然后用户可以以命令＼closing{text}结束信函的文本。命令＼closing{text}的文本参数是用户在信末的问候语。例如：

＼closing{Sincerely yours}

在＼closing 命令后面，用户可以指定信函的其它一些特征，例如复写本和列表附寄物：＼cc 复写本和＼encl 列表附寄物。

如果用户想把信件格式化成在右上角有回邮地址的私人信件，就必须用命令＼address 定义回邮地址。没有＼address 命令，信函假设在信纸上已有信头。信函的回邮地址用命令＼address{Return address}指定。回邮地址应该在同一行上，并用字符＼＼ 把名字，地址，和城市分隔开。

＼address{name ＼＼ address ＼＼ city,state zip}

用户的名字由命令＼signature{your-name}指定。这个名字将在信纸底部出现。用户也可以添加其它行，用＼＼把它们分开。

＼documentstyle{letter}

＼begin{document}

＼begin{letter}{Mrs. Barbie Ken ＼＼ Beanie Inc. ＼＼5321 East St. ＼＼ Inland, MA 55555}

＼address{Larisa Petersen ＼＼ 7777 Book Drive ＼＼ Ourtown, CA 90000}

＼signature{Larisa Petersen}

＼opening {Dear Sir}

I can say that after extensive real world testing of your new products, that

some do not quite stand up to the everyday pressure that consumers may impose on them.

Most,however,survive ever the most reckless care.

\closing{Sincerely yours}

\cc{Aleina,Christopher, and Dylan}

\encl{Durability Report,Half-life Estimates}

\end {letter}

\end {document}

用户还可以使用 firstpagestyle 指定电话号码和地址。命令\telephone{number}将显示用户的电话号码，而命令\location{address}将指定一个不同于标准地址的一个地址。

22.4.14 定义新的命令，环境，以及字体

用户能够使用命令\newcommand 创建一个新命令。命令\renewcommand 允许用户重定义一个已经存在的宏。这些操作把命令的名字作为参数，紧跟在参数后面的是参数的个数（由方括号括起来），然后是被括在大括号之内的定义本身。

\newcommand{cmd-name} [number-args] {definition}

\renewcommand{cmd-name} [number-args] {definition}

用户也能够使用命令\newenvironment 定义自己的环境。命令\

renewenvironment 重定义一个已经存在的环境。对于这些操作，用户需要为起始宏定义起始命令，为结束宏定义结束命令，这两个命令将把环境包装起来。这些操作的第一个参数是环境名字，紧接着是参数个数，然后是起始宏命令的定义以及结束宏命令的定义。

＼newenvironment{name}[number-args]{begin-definition}{end-definition}

＼ renewenvironment{name} [number-args] {begin-definition}{end-definition}

命令 newfont 执行的操作与＼font 有很多相同之处，它允许用户为自己的文档定义新的字体。

＼newfont{def-name}{fontname}

表 22-10 列出了可以用来创建新的命令，环境，以及字体的 LaTeX 命令。

22.5TeX 应用程序

在用户的 TeTeX 软件包中包括了几个流行的 TeX 应用程序，这些应用程序扩展了 TeX 的功能。每一种包括在内的应用程序都有 man 帮助页，在文件／usr／TeX／man／whatis 里面有这些应用程序的清单。VirTeX 和 IniTeX 是用于快速处理 TeX 文档的实用程序。IniTeX 将生成一个扩展名为.fmt 的特殊格式二进制文件，它将用自己组元命令代替原来的字体和宏命令，从而减少处理时间。然后 VirTeX 读取.fmt 文件，快速生成 dvi 文件。VirTeX 和 IniTeX 都包括在用

户的 TeTeX 包中。

BibTeX 是用于生成书目的 TeX 应用程序。用户对含有支持 BibTeX 的宏的 LaTeX 文件执行命令 bibtex。关于 BibTeX 的文档资料在文件 doc／bibtex。程序 ps2frag 允许用户为一个引入图像生成 TeX 格式的标签。这对那些要在上面显示数学式子的图像很有帮助。slitex 是用来从 TeX 文件生成幻灯片的实用程序。程序 MetaFont(mt)允许用户设计自己的字体和成语活字。

AmSTeX 是一组提供强大数学排版能力的宏命令。它们可以 AmSLaTeX 包的形式被包括在 LaTeX 里面。LamsTeX 是一个使用自己命令的 TeX 应用程序，它的命令是按照面向内容格式化的原则设计的。它也包括 AmSTeX 宏命令。它有自己的文件处理命令 lamstex。

在 CTAN 因特网站点上还能找到许多其它的包和应用程序。其中一个特别有用的是应用程序 LaTeX2HTML，它使用户能够用 TeX 和 LaTeX 生成万维网页。另一个应用程序是 LyX，它是一种 X-Windows 环境下的 LaTeX 编辑器，在 OpenLinux 的 CD-ROM 中包含了此应用程序。

22.6Ghostscript 和 Ghostview

应用程序 Ghostscript 能够解释 PostScript 文件并且将文件打印到不支持 PostScript 的设备。现在，很多在线文档是 PostScript 型文件。为了打印这些文件，用户一般需要支持 PostScript 的打印机。Ghostscript 使用户在不支持

PostScript的打印机上仍能打印这些文件。

命令 gs 启动 Ghostscript 解释器。gs 把用户想打印的 PostScript 文件的名字作为参数。Ghostscript 解释器在处理文件时将会发布信息。处理结束后，输入命令 quit 退出解释器。

```
$ gs myfile.ps
```

下面就是一个 Ghostscript 对话的例子，它把输出存到文件 myd。当用户按回车键显示下一页时，Ghostscript 输出将逐页显示。显示完所有页后，显示 Ghostscript 提示符 GS>，等待用户输入命令。quit 命令将退出 Ghostscript 解释器。用户加选项-q 后可去掉这些信息。

```
$ gs -sOutputFile=myd   mydoc.ps
Aladdin Ghostscript 3.33 (4／10／1995)
Copyright (c) 1995 Aladdin Enterprises,Menlo Park,CA. All rights reserved.
This software comes with NO WARRANTY: see the file COPYING for details.
Loading NimbusMonL-Regu font from
／usr／lib／  Ghostscript／fonts／n0220031.pfb... 1689900 381057 1320152 28093 0 done.
Loading NimbusMonL-Bold font from
／usr／lib／  Ghostscript／fonts／n0220041.pfb... 1730052 410797 1320152 33311 0
done
```

```
>>showpage,press <return> to continue<<
GS>quit
$
```

gs 命令有许多选项。表 22-11 列出了 Ghostscript 的选项。较为重要的一个是-sDEVICE 选项，使用户能够设置输出设备。缺省情况下，Ghostscript 输出到标准打印机。如果联有其它打印机，用户可以把它们中的一个指定为 Ghostscript的输出设备。下面的例子把 Deskjet 打印机设置为输出设备。

```
-sDEVICE=deskjet
```

除了通过命令行，也可以在解释器中改变设备。 前接放置在括号内的设备名字的命令

selectdevice 能够选择设备。

```
(deskjet) selectdevice
```

用户也可以使用环境变量 GS_DEVICE 重新设置 Ghostscript的缺省输出设备（关于其它变量，见表 22-11）。

```
GS_DEVICE=deskjet
```

通过选项-sOutputFile，用户能够重定向输出到文件。以下的命令输出到文件 myfile.ot。

```
-sOutputFile=myfile.ot
```

如果用户想为每一页生成一个文件，用 name%d.ext 指定名字。由于%d,将生成一个数字。对应于选项 -sOutputFile=myfile%d.ot，将生成文件 myfile1.ot,myfile2.ot 等等。

用户可以用-sPAPERSIZE 选项设定纸张尺寸，可以指定已有的纸张尺寸，如 a4，4.5×11，11×13，或者 11×17。用户也能够在.Xdefaults 文件中使用特殊的项目来配置 X-Windows 下的 Ghostscript。用户可以配置窗口显示的特征，以及字体使用。

22.6.1Ghostview 和 GV

有了 Ghostview，用户就能够在 X-Windows 上显示 Ghostscript 的输出。这样，就可以在桌面上显示任何 PostScript 的输出，而不必把它打印出来。有了 Ghostview，用户能够显示以 PostScript 文件形式存放的任何文档资料，或者是显示正在对其进行操作的 PostScript文档。为了启动 Ghostview,需在 Xterm 窗口下使用后跟文件名的 ghostview 命令，如下所示（用户也可以通过 fvwm Workplace 菜单进行操作）：

```
$ghostview  filename
```

在 Ghostview 窗口的右边部分将显示格式化的 PostScript 文本。用鼠标单击窗口中的文本将打开一个放大窗口，里面显示放大了的文本。单击鼠标左键将以低分辨率显示文本，单击中键将以中分辨率显示，单击右键将以高分辨显示。

紧挨着文本的左边的是一个竖直窗口，列出了文本内容目录。对大多数文本来说，显示的将是一些页码。符号 A>将指出当前显示的页码。用户可以在页码上单击鼠标中键在各页之间移动。单击鼠标左键将选定一页而单击右键将扩

大选择范围。用户也可以用键盘命令执行很多屏幕操作。按空格键将显示下一页，按 b 键将显示前一页。按＋键将增加放大倍数，按-键将减小放大倍数。表 22-12 列出了 Ghostview 的键盘命令。

在 Ghostview 窗口的左边部分是一个有五个菜单按钮的菜单框。File 菜单使用户能够打开并打印文件，也可以退出 Ghostview 程序。Page 菜单使用户能够轻松自如地在各页之间移动并且标记页码。菜单 Magstep 使用户能够改变显示放大率。Orientation 菜单使用户能够旋转，镜象翻转文件内容，并且选择文件的横向视图。Media 允许用户选择页面尺寸。

Johannes Plass 的 GV 是一个用于在 X-Windows 上显示 Ghostscript 输出的较新的程序。它基于 Jim Theisen 编制的 Ghostview 程序。GV 增加了一些新的特征以及作为 Adobe PDF 文件浏览器的功能，用用户能够显示 Adobe PDF 文件。用户可以从放于 Redhat ftp 站点(ftp.redhat.com)下 contrib 目录中的 rpm 包得到 GV 的一个拷贝。源代码版本可在目前位于 wwwthep.physik.uni-mainz.de／~plass／gv／的 GV 主页上找到。GV 的网页包括了文档资料和有关 GV 的最新信息。

表 22-1TeX 命令

命令	操作
\bye	结束 TeX 文档
自然段	
\par or empty-line	起始一个新的自然段，首行缩排
\noindent	起始一个新的自然段，首行不缩排
预定义字体	
\rm	罗马字体
\tt	打印机字体
\bf	黑体
\it	意大利斜体

续表

＼sl	斜体
字体定义	
＼font ＼fontname=font	定义一种字体。Fontname 是用户取的字体名。Font 是字体的真实名字
间距单位	
em	字符宽度；随字体不同而变化
in	英寸
pt	点
mm	毫米
置空格命令	
＼hskip num-measure	把指定大小的间距插入某行
＼vskip num-measure	把指定大小的一行插入行与行之间
＼smallskip	把一高度较小的空行插入行与行之间
＼medskip	把一高度中等的空行插入行与行之间
＼bigskip	把一高度较大的空行插入行与行之间
＼hfill	填满一行中的空格。用于一行文本左对齐或右对齐
＼hfill text 右对齐文本 text＼hfill 左对齐文本 ＼hfill text ＼hfill 居中对齐	填充行与行之间的空间页面布局
＼vfill	

续表

＼headline={ text }	标题
＼footline={ text }	脚注
＼hsize=num-measure	自然段文本尺寸
＼baselineskip=num-measure	行与行之间的规则间距
＼parskip=num-measure	段与段之间的规则间距
分组	
＼begingroup	开始一个分组
＼endgroup	结束一个分组
{text}	分组{与＼begingroup 和＼endgroup效果一样}
{＼command text}	分组中的命令，只对分组中的文本起作用
定位对象	
＼topinsert object	定位一个对象，比如页面顶部的图片
＼endinsert	
＼pageinsert object	定位一个对象于其所在页面
＼endinsert	
宏	
＼def＼macro-name{text}	定义一个新的宏

续表

＼ def ＼ macro-name parameter-list{text} # num	定义一个新的带参数的宏 为一个宏定义一参数。用于参数列表 ＼ def ＼ myn#1#2{My name is #1 #2}

表 22-2TeX 特殊字符

命令	描述
＼c	引用紧跟着的那个字符
＼tex-command	命令
%tex-comment	注释
{text}	分组
#num	参数
$formula$	把数学式子嵌入文本
$$formula $$	在单独一行显示数学式子
^	上标
_	下标
&	
＼＼	新行
空格	分隔单词
空行	分隔段落

表 22-3TeX 和 LaTeX 数学符号

命令	符号	描述
^{expr}	x^{i}	上标
\alpha	α	Alpha
\delta	δ	Delta
\beta	β	Beta
\epsilon	ϵ	Epsilon
\sigma	σ	Sigma
\mu	μ	Mu

\theta	θ	Theta
\prod	$\prod$	连乘符号
\pi	π	Pi
\sum	$\sum$	求和符号
\int	$\int$	积分符号
\subset	$\subset$	子集
\supset	$\supset$	父集
\Delta	Δ	Delta
\equiv	$\equiv$	等价
\neq	$\neq$	不等于

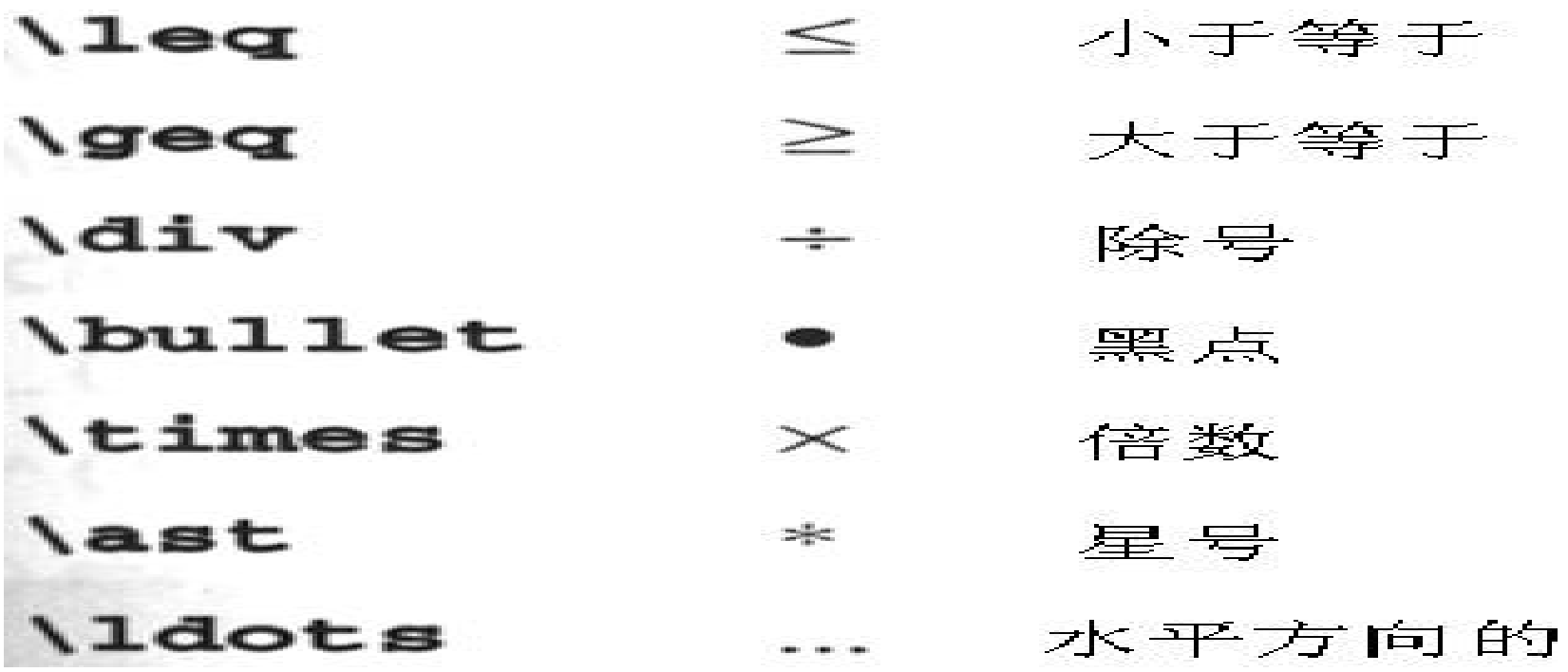


表 22-4 文档类

类	操作
\documentclass{classname}	为文件定义文档类
\begin{document}	起始文档体
\end{document}	结束文档体
文档类	
article	短文档，如杂志类文章

续表

book	分章节的书籍类文档
letter	商业或私人信函
report	分成几个章节的长篇报道
slides	生成幻灯片
文档类选项	
l1pt	11 点阵
l2pt	12 点阵
twoside	双面打印(非信函)
oneside	单面打印(非信函)
openright	在奇数页开始章节
openany	在奇数页或偶数页开始章节
leqno	对等式进行编号，编号位于左侧
draft	草稿输出
fleqn	使等式左对齐
leqno	置等式编号于底部左侧
acm	(对信函和书籍类不适用)
letterpaper,a4,paper	纸张尺寸

表 22-5LaTeX 字体类型和字符操作

字体类型	功能
\rm	罗马字体
\it	意大利斜体
\em	强调(在\it与\rm之间切换)
\bf	黑体
\sl	斜体
\sf	Sans Serif 字体
\sc	小体大写字母
\Tcl/Tk	打印机字体
字体尺寸	
\tiny	非常小的字体
\scriptsize	脚本字体尺寸
\footnotesize	注脚字体尺寸
\small	小体字体尺寸
\normalsize	缺省字体尺寸
\large	大型字体尺寸
\Large	首字母大写
\LARGE	所有字母大写
\huge	巨型字体尺寸
\Huge	首字母大写

续表

字符操作	
_ {text}	下标(用于数学模式中的表达式)
^{text}	上标(用于数学模式中的表达式)
\symbol{number}	显示字符集中对应编号的符号
\underline{text}	对文本添加下划线
\ldots	省略号(在所有模式下都可使用)

表 22-6LaTeX 命令

命令	功能
\addtocounter{counter}{value}	对计数器置值
\alph{counter}	使用字母字符打印计数器的值
\arabic{counter}	使用数字打印计数器的值
\roman{counter}	使用罗马数字打印计数器的值
\newcounter{name}	定义一个新的计数器
\setcounter{counter}{value}	分配一个新的值给计数器
\stepcounter{counter}	计数器值增加 1
\usecounter{counter}	在列表环境中使用计数器
\value{counter}	在表达式中使用一计数器的值
页面风格与标题	
\pgestyle{style}	定义整个文档的标题与注脚的放置
Empty	没有标题和注脚

续表

Plain	注脚中只打印页码
Headings	打印奇数页和偶数页的
标题	
Myheadings	定义自己的标题
\thispagestyle{style}	只定义对当前页的标题与页脚的放置
\markboth{even-header}{odd-header}	设定用户自己的偶数和奇数页的标题
\markright{single-header}	设定用户自己的标题
\twocolumn	开始一新的两栏页面
\onecolumn	开始一新的单栏页面
\item[label]	在列表环境中定义列表项，诸如枚举，分条，或者描述自然段与行
Empty-line , \par	开始新的自然段并首行缩排
\indent	对自然段进行缩排
\noindent	不对自然段进行缩排
\centering	居中段落
\raggedright	左对齐
\raggedleft	右对齐
\linebreak	断开行
\newline	提前断开行

续表

\nolinebreak	不断开当前行
\newpage	开始新的一页
\nopagebreak	在此处不设置段分隔符
\pagebreak	在此处设置段分隔符
\verb	对于全文文本，真实地按照输入打印节的标题
\section{heading-text}	一个节的标题
\subsection{heading-text}	一个子节的标题
\subsubsection{heading-text}	一个子子节的标题
\paragraph{heading-text}	一个自然段的标题
\subparagraph{heading-text}	一个子段的标题
\appendix	附录
\part{part-title}	一个局部标题
\chapter{chapter-title}	一章节标题
\tableofcontents	使用表头创建表格内容
标题	
\maketitle	为文档取一个标题
\title{title-text}	确定标题
\author{author-name}	确定作者
\date	生成日期

续表

注脚	
\footnote[number]{footnote-text}	创建一个脚注。用户可选择是否为页脚指定自己的编号
\footnotemark	在一个自然段中放置一个脚注引用(使用 footnotetext)
\footnotetext{text}	对文本做脚注。用于自然段中的脚注标记
\label{label-name}	使用标签名为交叉引用设定文本
\ref{label-name}	创建一个交叉引用，指定一个节
\pageref{label-name}	创建一个交叉引用，指定一页面环境
\caption{text}	插入一个插图说明(在 figure 与 table 环境中使用)
\begin{environment-name}	起始一个环境并以 environment-name 标识。将环境的格式应用于后面的文本
\end{environment-name}	关闭环境表格命令(tabular 环境)
\cline{coli-colj}	绘制一条跨越从第 coli 栏到第 colj 栏的水平线段
\hline	绘制一条跨所有栏的水平直线

续表

\multicolumn{cols}{align}{text}	创建跨指定数目栏的单元格,或者创建一个定制的单元格。Align 是对齐方式(l,r,或 c)
\vline	绘制一条垂直线段图形环境(picture 环境)
\put(x coord,y coord){objects}	put 进行图元的实际绘制。置图元于指定坐标
\circle[*]{diameter}	绘制一个直径为 diameter 的圆
\oval(width,height)[position]	绘制一个宽度和高度分别为 width,height 的椭圆并且放置文本于其中。使用定位选项 t,b,l,r 的组合，可定位文本于不同象限
\line(x slope,y slope){length}	绘制一条指定长度和斜率的线段
\vector(x slope,y slope){length}	绘制一条指定长度和斜率的带箭头线段
\makebox(width,height)[position]{text}	绘制一个宽度和高度分别为 width 和 height 的方框，并且只文本于其中。使用定位选项 t,b,l,r 的组合，可定位文本于不同象限

续表

\framebox(width,height)[position]{text}	绘制一个宽度和高度分别为 width 和 height 的方框，并且将指定对象包括起来。使用定位选项 t,b,l,r 的组合，可定位文本于不同象限
\dashbox{dash length}(width,height){text}	绘制一个宽度和高度分别为 width 和 height 的阴影框并且置文本于其中。可指定阴影线长度图形命令(picture 环境)
\frame{...}	将一边框直接置于一对象周围，两者之间不留空间
\multiput{x coord,y coord)(delta x,delta y)	绘制一个图元的多个拷贝{number of copies}{object}
目录	
\bibitem[label]{cite_key}	使用标签创建一个目录项。如果标签消失，使用枚举标签。cite_key 用来创建一系列引证索引
\cite[text]{key_list}	引证一个目录项。key_list 是一系列引证关键词
\nocite{key_list}	增加 key_list

表 22-7LaTeX 环境

环境	操作
Figure	对一幅图像进行编号和定位(使用\caption 来标识)
Table	对一张表格进行编号和定位(使用\caption 来标识)
Tabular	创建表格
Picture	创建草图
数学环境	
Equation	对公式进行标识和定位
Math	在文本中嵌入数学式子
Displaymath	杂单独一行上显示数学式子
Array	数组
Eqnarray	列式等式序列
Theorems	对一个定理进行标识和编号行的格式
Center	居中行
Flushleft	左对齐行
Flushright	右对齐行
Tabbing	在环境里面重新设定 tabs 键
Minipage	建立迷你页
Titlepage	建立特殊标题页

续表

列表	
Enumerate	编号列表。对应项目使用\item
Itemize	简报列表。对应项目使用\item
Description	标号列表。对应项目使用\item
List{label}{spacing}	创建指定标签和间距的列表标签可以是 LaTeX 命令。对应项目使用\item
专用环境	
Letter	创建一封信函(用于信函文档类)
Quote	引用一个自然段
Quotation	引用几个自然段
Verbatim	按照输入原样显示文本
Verse	诗作
TheBibliograph	目录项

表 22-8LaTeX 数学操作

Math 环境符号	说明
$	嵌入数学式子
$$	单独一行显示数学式子
Math 间距	
\;	大间距
\	中间距
\,	小间距
\!	负的小间距
LaTeX Math 操作	
\cdots	置于行中央的水平省略号
\ddots	对角省略号
\vdots	竖直方向的省略号
\ldots	省略号(在所有模式中都工作)
\frac{num}{den}	生成分数 num/den
\overbrace{text}	置 text 于大括号之中
\nderbrace{text}	用下面的大括号生成文本
\sqrt[root]{arg}	生成 arg 的平方根

表 22-9Letter 环境命令

命令	操作
\begin{letter}{name \\ address \\ city, state zip}	初始一个 letter 环境，开始一封新的信函。地址部分由\\分隔开
\address{Return address}	用户的回邮地址
\cc{name \\ name \\ etc}	Cc 列表。各项由\\分隔开
\closing{text}	结束信函文本并且指定告别语，如 Sincerely"
\encl{enclosure-list}	附寄材料清单。各项由\\分隔开
\location{address}	单位地址。只有在使用 firstpage 页面风格时起作用
\makelabels{number}	制作地址标签。在开始段中输入
\name{name}	为回邮地址而包括的名字
\opening{text}	开始信函文本并且指定开头敬语如'Dear Sir'
\ps	增加一个 PostScript 文件
\signature{name}	打印签名用名字
\startbreaks	允许分页
\stopbreaks	禁止分页
\telephone{number}	电话号码。只有在使用 firstpage 页面风格时起作用

表 22-10LaTeX 定义命令

命令	操作
\newcommand{cmd}[agrs][default]{definition}	定义一个新的命令
\renewcommand{cmd}[args][default]{definition}	改变一个已经存在的命令的定义
\newenvironment{env_name}[args][default]{begdef}{enddef}	定义一个新的环境
\renewenvironment{env_name}[args]{begdef}{enddef}	改变一个已经存在的环境的定义
\newtheorem{env_name}[numbered_like]{caption}	定义一个处理定理的新的环境
\newfont{cmd}{font_name}	创建一个引用字体的新的字体
使用包和包含文件	
\usepackage{package}	装入一个 LaTeX 包
\include{file}	有条件地包括一个文件
\includeonly{file-list}	确定包括哪个文件
\input{file}	无条件地包括一个文件

表 22-11Ghostscript 选项和变量

命令行选项	操作
-h or -?	显示系统各设备的帮助信息
@filename	Ghostscript 将处理的文件名清单保存在文件 filename 里面
-ffilename	处理名字以圆点开头的文件
-Idirectories	往库文件搜索路径添加目录
-Sname=str	定义一个串
-q	压缩标准启动信息
-gnum1xnum2	显示设备的高度和宽度
-rnumxnum	设置打印机的 x 和 y 分辨率
-	从标准输入设备而不是文件读入数据
-DdISKFONTS	为字体指定字符轮廓
-dNODISPLAY	压缩输出设备的初始化信息
-dNOPAUSE	取消提示符和每页尾部的暂停
-dNOPLATFONTS	取消系统字体
-dSAFER	允许文件的只读
-sDEVICE=device	选择一个输出设备
-sOutputFile =filename	选择一个输出文件环境变量
GS_DEVICE	缺省打印设备
GS_LIB	字体和文件的搜索目录

续表

GS_LIB_DEFAULT	除 GS_LIB 指定外的搜索目录
GS_FONTPATH	PostScript 字体目录列表

表 22-12Ghostview 键盘命令

键	操作
Q	退出
O	打开一个文件
R	重新打开一个文件
S	保存标识过的页
P	打印标识过的页
SHIFT-P	打印所有页
BACKSPACE,B	显示前一页
SPACE,RETURN,F	显示下一页
PERIOD,CTRL-L	重新显示页面
M	标识页面
N	取消对页面的标识
0- 5	选择显示文件的放大倍数
＋	提高放大倍数
-	减小放大倍数
U	向上滚动

续 表

D	向 下 滚 动
H	向 左 滚 动
L	下 右 滚 动

第 23 章编译器和库：gcc，g＋＋和 gdb

应用程序是程序开发人员使用某种编程语言创建的可执行文件。Linux 提供了一些工具程序，使开发人员能够控制应用程序的开发过程。在这些工具中，gcc 是最重要的一个，它可以调用 C 和 C＋＋编译器，生成相应的可执行文件。绝大多数的 Linux 应用程序是使用 C 或 C＋＋语言编写的。

在应用程序开发过程中经常使用到库，这些库可能是开发人员事先创建的自己的库，也可能是他人提供的专门用途的函数库。开发人员可以使用 X-Windows 库编写 X-Windows 程序的图形显示界面，也可以使用 gdbm 库实现对数据库文件的访问。库的使用正变得越来越灵活，现在可以实现库的共享或动态载入。

Linux 中还有一些程序开发中使用的其它工具。gdb 符号调试器可以帮助开发人员定位程序发生的错误，indent 和 cproto 可以对源代码进行优化处理。Autoconf 和 RPM 则可以为程序发行提供帮助。Linux 中还有其它主要编程语言的编译器，这些语言包括 Pascal、Lisp、Ada、Fortran、Basic 和 Modula-2 等。

23.1 获取信息：info

尽管所有的编译器和应用工具都有相应的帮助页，但使用 GNU 的 info 系统可以得到更为详细的软件信息。在/usr/info 目录下包含有各种 GNU 工具的详细描述和例子文件。这些文件就相当于是各种工具的在线手册。在这个目录中可以找到 gcc 编译器、C 和 C++函数库、Autoconf 应用程序，甚至 Indent 程序的详细信息文档。有些应用程序将相应的信息文件放在自己的目录下，如 LaTeX 的信息文件就存放在/usr/TeX/info 目录下。键入命令 info 就可以调出 info 文档的主菜单界面。

```
$ info
```

使用空格键可以向下翻页，按 M 键可以按照主题对文档进行搜索，此时在屏幕下方会出现一个空行，用户可以在行中输入菜单项的名。按回车键可以查看对应选择的主题。按 B 键可以回到文档的开始部位，按 U 键则可以回到上一次查看的菜单项。键入 info info 命令可以打开一个教用户如何使用 info 的教程。

23.2 C 编译器：gcc

UNIX 操作系统和 C 语言中存在一种特殊的关系。C 语言起先就是为了开发 UNIX 操作系统而设计的，UNIX 操作系统也是用 C 语言编写的。Linux 也与 C

语言存在这种特殊关系。绝大部分的Linux操作系统中都包含了C编译器的GNU版本：gcc。C语言是一种非常复杂的语言，拥有许多不同的特性。本节中简要描述了C语言中的基本组件，并使用这些组件创建了一个有用的程序例子。本节中还通过该例子展示了编译C程序的各种不同的方法。

键入gcc命令就可以在Linux系统中调用GNU C编译器。gcc命令将一次调用4个其它的组件。第一个组件为预编译器。预编译器是一个包含有特殊预编译命令的C程序，能够对需要编译的程序进行预编译处理。第二个组件为编译器。编译器会对经过预编译处理的程序进行编译，生成相应的汇编代码版本。第三个组件为汇编程序，它可以将编译器生成的汇编代码汇编为目标代码版本。最后一个组件为连接程序，它的功能是将目标代码连接为可执行程序。

该可执行程序的默认文件名为"a.out"。一般来说，应该使用参数指定自己选择的文件名。使用-o选项可以将文件名作为参数使用。这样，连接程序会使用指定的可执行文件名代替默认的"a.out"。表23-1中给出了可用的gcc选项。下面的例子使用gcc对greet.c程序进行编译，注意此时指定了可执行文件的文件名为"greet"。在Linux系统中可以直接键入可执行文件的文件名运行该文件。

```
$ gcc greet.c -o greet
$ greet
Hello, how are you
```

在使用多个文件构成的程序时，必须注意C编译器和连接程序的区别。C编译器的目的是生成目标代码文件，而连接程序的目的是使用目标代码文件生成可执行文件。C编译器单独对每一个源代码文件进行编译，生成多个对应的

目标代码文件，这些目标代码文件的扩展名为.o 而不再是.c。只需要在 gcc 后列出文件名就可以使用同一个 gcc 命令编译并连接多个文件。在下面的例子中，用户使用 gcc 命令对 bookrecs 程序进行编译。-o 选项指定了可执行文件的文件名。

```
$ gcc main.c io.c -o bookrecs
```

使用 gcc 命令也可以只完成连接工作，具体作法是在参数列表中只列出目标代码文件的文件名。目标代码文件的扩展名为.o。下面的例子就只进行了一次连接操作，而没有编译操作发生。当然，在执行连接工作前必须首先生成相应的目标代码文件。

```
$ gcc main.o io.o
```

在开发、调试程序时，需要经常对源代码文件进行修改，然后重新编译以查看运行效果。如果程序由多个文件组成且修改只涉及到少量的文件，每次都对所有文件进行编译是非常不经济的。那些没有修改过的文件不需要重新编译，只需要进行一次连接工作即可。将源代码和目标代码文件同时作为 gcc 的参数列出就可以完成这项工作。源代码文件的扩展名为.c，目标代码文件的扩展名为.o。gcc 将对列出的源代码文件进行编译，然后连接所有的目标代码文件。样就可以只对修改过的少量文件进行编译，提高了效率。例如，如果只是 main.c 发生了变化，io.c 没有变化，就应该在 gcc 的参数中指定源代码文件 main.c 和目标代码文件 io.o。这样就不会重新编译 io.c 文件。

```
$ gcc  main.c  io.o  -o  bookrecs
```

23.3 ELF 和 a.out 二进制格式

可执行文件之类的二进制文件存在两种格式。第一种是 UNIX 和 Linux 早期使用的 a.out 格式，该名称来自 UNIX C 编译器的默认可执行文件名。由于后来引入了共享库的概念，a.out 格式的文件在实现上遇到了困难，使 a.out 格式的可执行文件使用共享库是一种非常复杂的事情。因此，后来在 UNIX System 5 release 4 for Solaris 版本中又推出了一种新的可执行文件格式，即可执行连接格式(Executable and Linking Format，ELF)。该格式可以很方便地实现库的共享。

Linux 系统中随后也采用 ELF 格式作为可执行程序的标准格式。gcc 编译器生成的所有二进制文件都采用 ELF 格式(尽管默认的文件名仍然为 a.out)。但老的使用 a.out 格式的程序仍然可以在这些系统中运行。

23.3.1 C＋＋和 Objective C: g＋＋

gcc 也是一个 C＋＋程序编译器。它可以读入并编译任何 C＋＋程序。但 gcc 不能自动与 C＋＋类库进行连接，必须在命令行上指定使用的 C＋＋类库。使用 g＋＋就可以自动地调用 gcc 编译器并将其与 C＋＋类库进行连接。C＋＋源代码文件的扩展名与一般的 C 程序不同，下面列出的扩展名都可以作为 C＋＋源代码文件的扩展名：C，cc，cxx，cpp。除此之外，C＋＋程序的编译过程与 C

语言程序完全相同。下面的例子将对 C＋＋程序 myprog.cpp 进行编译。

```
$ g＋＋  myprog.cpp  -o  myprog
```

gcc 编译器也提供对 Objective-C 程序的支持。Objective-C 也是一种面向对象的 C 语言，它起先是为 NeXt 系统设计的。在使用 gcc 命令对 Objecticve-C 程序进行编译时，需要使用-lobjc 选项。-lobjc 选项将使用 Objective-C 类库，libobjc.so，进行连接。

23.3.2 其它编译器：Pascal、ADA、Lisp 和 Fortran

Linux 系统中还提供了许多对其它编程语言的支持。这些编译器都可以在 OpenLinux 光盘上找到。除了 C 和 C＋＋外，还可以对 Pascal、ADA、Lisp 和 Fortran 编写的程序进行编译。一般来说，编译过程仍然由 gcc 程序处理，该程序可以辨别出各种编程语言的源程序格式。例如，G77 是 GNU 版本的 Fortran 编译器。该程序也集成在 gcc 编译器中。g77 命令将调用 gcc 编译器对 Fortran 程序进行编译。ADA 95 的编译器名为 gnat。ADA 的信息文件提供了 gnat 的详细信息。使用 gnatmake 命令可以对 ADA 程序进行编译。

23.4 创建和使用库：静态、共享和动态

C 语言中有一些函数不需要经常进行编译，有一些函数也可以在多个文件

中使用。一般来说，这些函数都会执行一些标准任务，如数据库输入/输出操作或屏幕控制等。可以事先对这些函数进行编译，然后将它们放置在一些特殊的目标代码文件中，这些目标代码文件就称为库。库文件中的函数可以通过连接程序与应用程序进行连接。这样就不必在每次开发程序时都对这些通用的函数进行编译了。

不同类型的应用程序将会使用不同的函数库。例如，libdbm 库中组包含了对数据库文件进行访问的 dbm 函数，需要对数据库进行操作的程序就会与该库进行连接。数学应用程序将使用数学库 libm，X-Windows 应用程序将使用 Xlib 库，libX11。另外，所有的程序都将使用标准的 C 函数库，libc，该库中包含了诸如内存管理或输入输出操作的基本函数(最近将会推出 GNU libc 库的 2.0 版本)这些库都存放在/usr/lib 这些系统公用的目录中，系统中的任何用户都可以利用这些库。当然用户也可以建立自己专用的库函数，供自己或其它指定的人员使用。

库可以有三种使用的形式：静态、共享和动态。静态库的代码在编译时就已连接到开发人员开发的应用程序中，而共享库只是在程序开始运行时才载入，在编译时，只是简单地指定需要使用的库函数。动态库则是共享库的另一种变化形式。动态库也是在程序运行时载入，但与共享库不同的是，使用的库函数不是在程序运行开始，而是在程序中的语句需要使用该函数时才载入。动态库可以在程序运行期间释放动态库所占用的内存，腾出空间供其它程序使用。由于共享库和动态库并没有在程序中包括库函数的内容，只是包含了对库函数的引用，因此代码的规模比较小。

已经开发的大多数库都采取共享库的方式。ELF 格式的可执行文件使得共享库能够比较容易地实现，当然使用旧的 a.out模式也可以实现库的共享。Linux 系统中目前可执行文件的标准格式为 ELF 格式。

GNU 库的使用必须遵守 Library GNU Public License(LGPL 许可协议)。该协议与 GNU 许可协议略有不同，开发人员可以免费使用 GNU 库进行软件开发，但必须保证向用户提供所用的库函数的源代码。

系统中可用的库都存放在/usr/lib 和/lib 目录中。库文件名由前缀 lib 和库名以及后缀组成。根据库的类型不同，后缀名也不一样。共享库的后缀名由.so 和版本号组成，静态库的后缀名为.a。采用旧的 a.out 格式的共享库的后缀名为.sa。

libname.so.major.minor

libname.a

这里的 name 可以是任何字符串，用来唯一标识某个库。该字符串可以是一个单字、几个字符、甚至一个字母。数学共享库的库名为 libm.so.5，这里的标识字符为 m，版本号为 5。libm.a 则是静态的数学库。X-Windows 库名为 libX11.so.6，这里使用 X11 作为库的标识，版本号为 6。

使用 gcc 编译器就可以将库与自己开发的程序连接起来。例如，libc.so.5 中包含了标准的输入输出函数，当连接程序进行目标代码连接时会自动搜索该程序并将其连接到生成的可执行文件中。标准的输入输出库中包含了许多基本的输入输出函数，如 printf 函数等。也可以连接其它的一些系统函数库，如数学库等，但与 libc.so.5 不同，大部分其它的系统库需要在命令行中显式指定所

用的库名。

在/usr/lib 和/lib 目录中可以找到绝大多数的共享库。连接时将首先搜索这两个目录。有一些库也可能存放在特定的目录中，在/etc/ld.conf 配置文件中给出了这些目录的列表。连接程序也会对列出的这些目录进行搜索。在默认情况下，Linux 将首先搜索指定库的共享版本，如果找不到，才会去搜索静态版本。在对共享库进行更新或安装新库后，必须运行 ldconfig 命令更新/etc/ld.conf 文件中相应的项(如果使用 RPM 包进行安装，一般会自动进行更新，不过也不能保证这一点)

在 gcc 编译器中引用可搜索到的目录中的库文件时，需要使用-l 选项和库名。在 gcc 命令行上输入-lm 可以在程序中连接标准算术库，-l 将首先使用 libname.so 进行搜索，这里是 libm.so。下面的例子将使用算术库创建 bookrecs 程序，请注意这里的-lm 选项。

```
$ gcc main.c io.c -o bookrecs -lm
```

系统中还有一些其它可用的库，常用的是 libncurses.a 库，包含了一些简单的鼠标移动例程。在命令行中使用-lncurses 选项引用 libncurses.so 库。下面的例子同时调用了数学和光标库。

```
$ gcc main.c io.c -o bookrecs -lm -lncurses
```

在引用其它目录中的库时，需要使用-Ldir 选项指定该目录。该选项指定了搜索库函数时的其它路径。在下面的例子中，用户在连接时使用了 mydir 目录中的 myio.so 库文件。

```
$ gcc main.c -o bookrecs -Lmydir -lmyio
```

23.5 gdb 符号调试器

gdb 是 Linux 系统上的符号调试器。如果应用程序运行中出了问题，就可以使用符号调试器跟踪错误。符号调试器允许用户在运行程序时显示源代码。在调试过程中可以在特定的位置停止运行，显示变量的内容。使用符号调试器甚至可以检查特定地址或栈中的内容。

为了能够使用符号调试器对可执行程序进行调试，在编译和连接程序时必须使用-g 选项。下面的例子可以编译适合符号调试器进行调试的程序。在可执行程序生成后，就可以使用符号调试器进行调试。

```
$ gcc -g main.c io.c
```

键入关键字 gdb 和可执行文件名就可以调用 gdb 调试器。下面的例子中的可执行文件名为 a.out。

```
$ gdb a.out
```

键入 gdb 命令就可以进入调试器环境，此时的提示符由 Linux 提示符($)变为 gdb 提示符(gdb)。在 gdb 调试器中键入 run 命令可以运行指定的程序。

```
(gdb) run
```

如果程序中存在 fopen 或 open 语句，说明程序中某处会使用某个数据文件。这样 gdb 需要知道数据文件的文件名，在输入 run 命令时，应输入数据文件的文件名。

```
(gdb) run filename
```

在调试过程结束后，使用 q 或 quit 命令就可以退出调试器。

绝大多数的 gdb 命令都可以使用命令的第一个字母进行等价操作。键入 r 与键入 run 命令等价，q 代表 quit 命令，p 代表 print 命令，n 代表 next 命令。表 23-2 中列出了可用的 gdb 命令。

使用 print 命令和变量名可以查看该变量的内容。下面的例子就可以显示 count 变量的内容。

```
(gdb) print count
10
```

使用 where 命令，可以显示已经调用过的函数名和调用时使用的参数。在下面的例子中，用户正位于 calc 函数中，where 命令显示了 main 和 calc 函数调用使用的参数。

```
(gdb) where
#3calc(newcost = 2.0) at calc.c:25
#1main () at main.c:19
#20x8000455 in __crt_dummy__ ()
```

使用 info locals 命令可以列出当前函数中定义的所有变量和参数的值。下面的例子就显示了所有定义的变量的值。

```
(gdb) info locals
cost =2
name = "Richard\000\000"
count = 10
```

```
count2 = 10
nameptr = 0x8000570  "petersen"
countptr = (int *) 0xbffffde8
```

可以使用 break 命令在程序中设置断点。当程序运行到断点时程序将自动停止。然后，用户就可以使用 next 或 step 命令逐行运行程序，也可以随时使用 cont 命令运行到下一个断点处。

xxgdb

xxgdb 是 gdb 调试器的 X-Windows 版本，它由一些垂直放置的子窗口组成。在初始状态下，将显示 5 个窗口。最上面的子窗口称为文件窗口，在窗口中将显示正在进行调试的文件名。下面是显示源代码文件的源程序窗口，在窗口左边的滚动条可以滚动显示窗口中的内容。源程序窗口下是显示 gdb 状态和错误消息的消息窗口。最底部的子窗口称为对话窗口，可以用来输入 gdb 命令并显示命令返回的结果。窗口左边的滚动条允许用户查询已输入的命令和返回的结果。最后一个窗口为命令窗口，在该窗口中使用按钮列出了可用的 gdb 命令。当执行 run 或 step 命令时，只需要单击相应的按钮即可。

23.6 编 程 工 具

Linux 系统中存在许多编程工具，可以帮助开发人员准备和组织源代码文件。indent 应用程序可以使用特定的格式对源代码文件中的语句块进行缩进处

理，使程序易于阅读。cproto 应用程序可以根据文件中的所有函数定义生成头文件中使用的函数声明。f2c 和 p2c 程序可以分别将 Fortran 程序和 Pascal 程序翻译为相应的 C 语言程序。xwpe 是一个类似 Turbo-C 的提供 C 编程环境的 X-Windows 应用程序。还有其它的一些应用工具，可以在 Development 下的 Linux Applications and Utilities 页下找到相应的说明。

在软件开发完成后，需要生成软件的发行版本。一般情况下，可以将开发出的程序压缩成一个 tar 文件，用户可以从网络上下载该文件并将其解压。注意在提供软件时，应该提供详细的安装和使用文档，并说明技术支持文档和函数库的存放位置。如果提供源代码文件，可能还需要告诉用户如何对源代码进行修改，以使其能够适合不同的系统。

Redhat Package Manager(RPM)和 Autoconf 是为自动安装而设计的。Autoconf 可以根据指定的系统对源代码进行设定，RPM 则可以自动地将软件、支持文档、函数库和支持程序安装在指定的目录。这两个应用程序都有非常复杂和强大的功能，可以对最复杂的程序进行处理。一些 Linux 发行版本(如 Redhathe Caldera)支持 RPM 软件包。

23.7 开 发 工 具

应用程序是编程人员使用编程语言创建的可执行程序。Linux 提供了一些应用程序可以控制应用程序的开发过程。在这些程序中最重要的是 make 工具，

它可以方便有效地对程序进行维护和编译。RCS 应用程序使开发人员能够较好地控制程序的修改。它将程序的修改储存在文件的不同版本中，这样可以保证修改的正确性和有效性。甚至可以使用 man 工具为自己开发的程序建立自己的在线帮助文档。所有这些工具的功能都非常复杂而强大。

23.7.1make 应用程序

一般来说，软件程序的开发会涉及到许多源代码文件。在开发程序时，需要反复地对文件进行修改和编译。但在编译时只需要对那些修改过的文件进行编译。连接程序会将新编译的修改过的文件和已编译的未修改过的文件进行连接，生成新的可执行文件。这样，就使编译器的工作大大减轻，不需要重新对每一个源文件进行编译。

在大软件的开发过程中，跟踪文件的修改状况，确定某个源代码文件是否修改过是一件并不容易的事情。make 应用程序可以帮助用户完成这项任务。make 就是为大工程中经常修改的源文件而设计的，它可以记录每个文件的修改过程。然后只对修改过的文件进行编译，并连接成新的可执行文件，确保编译连接工作的有效性和效率。在下面的例子中，用户键入 make 命令调用 make 应用程序。然后 make 将编译已被修改的文件然后生成新的可执行文件。make 将显示每一条执行的 Linux 命令。

```
$ make
cc -c main.c
```

```
cc -c io.c
cc main.o io.o
```

make 程序是使用源代码文件的时间戳决定是否需要对其进行编译的。当文件创建或修改时，Linux 操作系统将会对文件相应的时间戳进行修改。如果文件创建于 1:00，则该文件的时间戳为 1:00。如果随后在 6:00 对文件进行了修改，文件的时间戳也会修改为 6:00。当重新对程序进行编译时，只有那些被修改的文件，也就是说，相应的时间戳在旧的可执行文件之后的文件才会被重新编译。make 重新就是这样实现选择性编译的。

make 程序使用相关行指定文件中的相关关系。编译过程中，源程序代码文件编译成目标代码文件，然后再连接为可执行程序。因此，可以说可执行程序依赖于目标代码文件，进一步依赖于源代码文件。所以需要在相关行中指定源代码文件和目标代码文件的依赖关系，以及可执行程序和目标代码文件的依赖关系。

可以将相关行看作一种条件语句，相关关系即为测试表达式。如果目标代码文件所依赖的源代码文件被修改，相当于测试表达式为真，该文件将重新进行编译。不过，相关行的语法比一般标准的条件测试语句略为复杂一些，相关行包括三个组件：目标文件，所依赖的文件列表和需要执行的 Linux 命令。如果目标文件所依赖的文件列表中最近修改过，就会执行指定的 Linux 命令。目标文件和所依赖的文件写在同一行上，中间采用冒号分隔。相应的 Linux 命令可以放在同一行上，使用分号和前面的内容分开，也可以放在下一行上，但在命令前应该有一个制表符。如果需要，可以列出多个 Liunx 命令，在相关行后

应有一空行标志该相关行的结束。在下面的例子中，Linux 命令调用 cc 编译器，对源代码文件进行编译或对目标代码文件进行连接。相关行的语法如下：

目标文件 ： 相关文件 ； Linux 命令

空行

目标文件 ： 相关文件

制表符 Linux 命令

空行

在下面的 makefile 文件中，对包含两个源代码文件 main.c 和 io.c 的 C 程序创建了相关行。在这样一个两文件的程序中，实际上存在五个文件需要管理。对应于每一个.c 文件都存在一个.o 文件，然后还存在一个可执行文件 bookrecs。目标代码文件(.o)依赖于源代码文件(.c)，而可执行文件则依赖于多个目标代码文件。在这个例子中，bookrecs 可执行文件依赖于两个目标代码文件 main.o 和 io.o。每一个目标代码文件又依赖于一个源代码文件，main.o 依赖于 main.c，io.o 依赖于 io.c。

在 makefile 文件中，针对 bookrecs，main.o 和 io.o 三个文件，需要三个相关行。注意这里对程序的编译和连接操作分别位于不同的相关行。bookrecs 行的命令为连接两个目标代码文件，生成新的可执行文件，它只进行连接操作。main.o 和 io.o 行的命令为对源代码进行编译，并不涉及到连接过程。

makefile

bookrecs : main.o io.o

gcc main.o io.o -o bookrecs

```
main.o : main.c
gcc -c main.c
io.o : io.c
gcc -c io.c
```

23.7.2 修改控制系统：RCS

在开发大型工程时，由于发现程序的错误或需要添加新功能，开发人员需要不断地对源代码文件进行修改。有时对程序的修改会带来一些新的错误。如果能够对修改的过程作记录，就可以帮助开发人员定位错误的产生过程。修改控制系统(Revision Control System, RCS)就是一个可以记录开发者对源代码文件修改过程的 Linux 应用程序。RCS 可以提供一套程序修改前的版本，便于开发人员进行比较检查。

在开发小组进行程序开发时，RCS 就非常有用了。小组中的每一个开发人员都会对程序进行修改。RCS 可以记录下每一个开发者所作的修改和修改时间，甚至可以为每一次修改记录下相应的注释信息。

RCS 将程序的原始版本储存在某个文件中，然后记录所有对文件进行的修改。使用记录下来的信息，RCS 可以生成文件开发过程中的任何一个版本。RCS 并不真正储存所有的版本文件，它只是使用原始版本和记录下的修改信息来生成相应的版本。表 23-3 中列出了在管理 RCS 文件时可用的命令。

对文件的一套已记录的修改称为一个版本。每一个版本都有自己的一个版

本号，版本号一般由主版本号和次版本号组成。在默认情况下，程序的第一个版本的版本号为 1.1。其后继版本则依次对次版本号进行增加，如 1.2、1.3 等。当然也可以手工修改主版本号。

在创建 RCS 文件时，需要首先创建一个 RCS 目录存放程序所用的 RCS 文件。然后使用 ci 目录创建 RCS 文件，ci 命令使用程序的原始文件名作为参数，在 RCS 目录中生成后缀名为,v 的 RCS 文件。main.c 文件的 RCS 文件名为 main.c,v。如果程序由多个源文件组成，需要为每一个源文件创建单独的 RCS 文件。下面的例子可以为 main.c 程序创建 RCS 文件。

```
$ ci main.c
RCS/main.c,v  <-- main.c
enter desciption, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> Bookrecs main program
>> .
initial revision : 1.1
done
```

在使用 RCS 对源代码文件进行编辑时，必须首先让 RCS 生成一份源代码文件的复件。该复件与 RCS 文件名相同，但没有,v 后缀。对于 main.c,v 文件，RCS 将生成一个名为 main.c 的文件。

使用 co 命令可以生成源代码文件的一个复件。co 命令可以使用许多选项，不带任何选项的 co 命令将生成一个只读的复件，-l 选项则可以生成一个可以编

辑的复件，这里的-l表示锁定，在使用该选项时，就可以锁定RCS main.c,v文件。其它的开人员此时将不能对该文件进行访问，这样就确保了在某一时刻只能有一个开发人员对程序进行修改。在对程序的修改完成后，需要对所作的修改进行登记，然后对已锁定的文件解锁供他人使用。下面的例子就使用co命令生成了源代码文件main.c,v的可编辑复件。

```
$ co -l main.c
RCS/main.c,v --> main.c
revision 1.1 (locked)
done
```

在源代码文件修改完成后，可以使用ci命令登记自己所作的修改。键入ci命令和RCS文件名，命令将提示用户输入动修改所作的注释。RCS将自动为源程序代码文件分配一个新的版本号。在下面的例子中，用户对main.c进行了修改，在存盘时RCS将自动生成新的1.2版本。

```
$ ci main.c
RCS/main.c,v <-- main.c
new version: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Added prompts
>> .
done
```

23.8 应用程序的在线手册：man

在开发 Linux 下的程序时，一般应该提供相应的文档资料。如果该程序是由多个程序员同时开发的的大程序，文档就变得更加重要。一般来说，文档都采取手册的形式，对程序中的不同命令和特点进行描述。对于用户和开发人员来说，提供详细正确的文档是非常重要的。在本书的第 3 章中已经提到，可以使用 man 命令显示 Linux 命令的有关信息，它提供了命令的在线手册功能。开发人员也可以使用 man 命令为自己开发的应用程序建立自己的在线帮助文档。使用 man 的文本处理宏命令就可以生成这些文档，然后可以使用 man 命令进行阅读。当键入 man 命令和文档名时，man 将使用 gnroff 对该文档进行格式处理并显示该文档。开发人员建立的真实文档是无格式的，但文档中包含了相应的 man 宏。因此，可以使用带-man 选项的 gnroff 命令直接显示手册文档。下面的两个例子都将显示 ls 的在线文档资料。

```
$ man ls
$ gnroff -man /usr/man/man1/ls | more
```

man 文档文件

可以使用任何标准的文本编辑器建立 man 文档文件。文本文件的文件名为相关的命令或主题和一个表示节的数字。因此，who 命令的帮助文档的文件名

可能为 who.1。在下面的例子中，bookrec 命令的帮助文档名为 bookrec.1，这里的 1 表示第 1 节。

man 帮助文档有许多节组成，各节的内容可以由开发人员自己设定。但一般来说，Linux 命令的帮助文档都将有以下几节：NAME、SYNOPSIS 和 DESCRIPTION。真正的帮助文档是没有格式的，但在文档中包含有相应的 man 宏命令。在文档中至少存在两个不同的 man 宏：TH 和 SH。TH 命令可以指定在显示文档的每一页时，屏幕上的标题和相应的节号。而 SH 命令则用来指定每一节的题头。当然在文档中也可以使用其它的宏，如 PP 宏用来进行段落格式设置，IP 宏用来实现缩进等。表 23-4 中列出了可用的 man 宏命令。

宏命令应放置在文档中某一行的开始处，在命令前应使用一个点号作为宏命令的表示。在命令后的所有文本都将按命令指定的格式进行处理。如果不出现下一个宏命令，该宏命令就一直有效。有些宏，如 SH 和 TH 可以带参数。参数应与宏命令位于同一行。SH 的参数指定节的名字，下面即为该节的内容。一般某节的内容由一系列的短行组成，man 将自动对其进行格式处理。下面的例子指定了名为 DESCRIPTION 的节。

```
.SH DESCRIPTION
mylistopt provides alternative ways
of displaying your grocery list.
You can display it with numbers
the list in a file.
```

man 帮助文档由一系列的小章节组成，下面的模板给出了如何组织帮助文

档的办法。Linux 命令的在线帮助文档一般都采用这种格式。

.TH COMMAND Section -number

.SH NAME

命令和函数功能的简单介绍

.SH SYNOPSIS

命令和选项。每一个选项都用括号进行封装。本节有时也称为 SYNTAX。

.SH DESCRIPTION

详细介绍命令和各选项的意义，可以使用 PP、LP 和 IP 进行格式处理。

.SH OPTIONS

命令可用的选项。

.SH EXAMPLES

如何使用命令的例子。

.SH SEE ALSO

可以用来参考的其它文档。

.SH DAGNOSTICS

对不正常输出的描述。

.SH WARNINGS

使用该命令可能带来的危险后果。

.SH BUGS

可能出现的错误。

在章节的文本中，可以使用其它一些宏进行特定的文本处理操作。.PP 宏

命令开始新的一段，.IP 宏对随后的文本进行缩进处理。一般用来显示程序所用的选项。.I 命令可以使文本加上下划线，.B 则是字体变为黑体。.I 和.B 选项都使用需要处理的文本作为参数。一般来说，NAME 节中的命令名和选项应使用.B 加黑，用到的其它命令则应使用.I 加上下划线。另外，如果在文本中使用了连字符，应在连字符前加上一个反斜线。

下面的例子创建了一个 bookrec 程序的在线文档。注意这里对文本进行格式处理的方法。

```
bookrec.1 --- man 文档文件
.TH  BOOKREC 1
.SH  NAME
bookrec \-Input and display a book record
.SH  SYBOPSIS
.B bookrec
[\-t] [\-p] [\-f]
.SH  DESCRIPTION
.I  bookrec
allows the user to input a title and price
for a book.  Then both elements
of the record are displayed
.SH  OPTIONS
.IP t
```

Display only the title
.IP p
Display only the price
.IP f
Save the record to a file
.SH DIAGNOSTICS
Date output has the form of m/d/y.
.SH BUGS
The program can onlu read and display one record.
.br
It does not as yet allow you to read records from a file.
.SH FILES
The command uses no special files.
.SH "SEE ALSO"

man 应用程序在类似/usr/man 的系统命令中寻找指定的帮助文档，但该命令下实际上并不存在这些文档文件，在该命令下存在的是一些称为节目录的子目录，相应的帮助文档都存放在这些子目录下。子目录名由 man 和章节号数字组成。一般说来会存在 7 个目录，目录名从 man1 到 man7。在自己的文档目录中，子目录的数目没有限制，但必须存在 man1 目录。

节目录的使用使得开发人员可以为某个目录创建多个详细程度不同的文档。例如，man1 目录中的文档可能只是目录一般的介绍，而 man7 目录中则

可能会给出所有宏的列表。这些文档根据.TH 宏指定的章节号存放在指定的目录，用户在查看这些文档时，可以在目录前指定需要查看的章节号。下面的例子将显示 bookrec 命令的第 3 章节的文档，如果没有指定指定章节号，将默认显示第 1 节的内容。

```
$ man 3 bookrec
```

表 23-1gcc 应用程序

gccgcc	应用程序使用预编译器、编译器、汇编程序和连接程序生成可执行文件。预编译器对源代码文件进行预编译，只进行一些简单的文本替换工作。编译器将源代码文件编译为汇编代码。然后汇编程序将汇编代码汇编为目标代码，最后使用连接程序将目标代码连接为可执行文件。gcc 可以使用各种类型的文件作为参数，它根据后缀名判断文件的类型
	.c C 源代码文件
	.o 目标代码文件
	.s 汇编代码文件
	.C C＋＋文件
	.cpp C＋＋文件
选项	
-S	只输出汇编代码。汇编代码文件的后缀名为.s
-P	输出预编译后的结果
-c	只生成目标代码，目标代码文件的后缀名为.o
-g	生成可执行文件的调试版本，供符号调试器使用
-o	文件名指定可执行文件的文件名，默认的文件名为 a.out
-O	编译优化
-l	文件名使用系统库名或文件名进行连接；-l 选项必须放在源代码文件和目标代码文件之后

表 23-2gdb 符号调试器

在 gbd 中运行程序		
r	run	运行程序
q	quit	退出 gdb
显示变量和参数		
p 变量	print 变量	显示变量内容
p &变量	print &变量	显示变量地址
set var = 值		在调试器中对变量进行赋值
where		显示函数调用情况以及使用的参数
	info locals	显示已定义的变量和参数
显示行		
l 行号	list 行号	显示指定行号开始的源代码
l 函数	list 函数	显示函数中的源代码
l 数字，数字	list 数字，数字	显示指定行号范围中的源代码
单步和继续执行		
n	next	单步执行程序

续表

s	step	单步执行程序
c	cont	继续执行程序
设置和删除断点		
b	break	在当前行设置断点
	break	行在指定行设置断点
	break	函数在指定函数的第一行设置断点
	info break	列出所有断点
d 数字	delete 数字	删除断点，必须指定需要删除的断点号
	delete	删除所有断点

表 23-3RCS 应用程序

RCS	修改控制系统(RCS)允许对程序开发过程进行控制；使用 RCS，可以保留程序开发过程中的所有版本；RCS 将记录程序的修改过程；RCS 文件由 ci 目录创建并使用 rcs 命令进行管理；使用 co 命令可以检索版本号，使用 rcs -o 命令可以删除指定的版本
ci	ci 命令对 RCS 文件进行更新，生成新的版本；如果不存在已有的 RCS 文件，cu 将使用后缀,v 创建该文件；使用 ci 命令储存编辑过的文件；储存的同时将生成新的版本号
	$ ci main.c
	选项
	-r 版本该选项允许开发人员指定版本号$ci -r5.2 main.c
co	co 命令获取 RCS 文件的某个版本；如果没有选项，该命令将返回一个只读的版本；如果没有指定版本号，将返回最新的版本选项
	-l 该选项将返回 RCS 文件的可编辑版本，锁住源代码文件防止他人对其进行修改
	$ co -l main.c
rcs	rcs 命令对 RCS 文件进行管理，用来控制其它用户对该文件的访问权限选项
	-a 用户名使指定的用户能够访问特定的 RCS 文件
	$ rcs -arobert main.c
	-e 用户名取消指定的用户访问特定的 RCS 文件的权限

续表

	$ rcs -erobert main.c
	-l 版本号锁定指定的版本号，除了文件的创建者外，其它人无法对其访问
	$ rcs -l2 main.c
	-u 版本号对指定的版本号解锁
	$ rcs -u2 main.c
	-L 版本号锁定指定的版本号，包括文件的创建者在内的所有人都无法对其访问
	$ rcs -L2 main.c
	-U 版本号对指定的版本号解锁
	$ rcs -U2 main.c
	-o 版本号从 RCS 文件中删除指定的版本
	$ rcs -o2.3 main.c
rlog	rlog 命令输出 RCS 文件中不同版本的信息；如果不使用任何参数，将显示每一个版本的简要信息选项
	-r 版本显示指定版本的信息
	-d 日期显示在指定日期创建的版本的信息；日期的格式为年/月/日，和小 时：分：秒;必须指定年份，其余为可选项
	$ rlog -d93/04/12 main.c
	-d<显示指定日期前创建的版本的信息

续表

	$rlog -d<93/04/12 main.c
	->d 显示指定日期后创建的版本的信息
	$rlog ->d93/04/12 main.c

表 23-4man 命令和创建在线帮助文档时可用的宏命令

man	搜索并显示命令的在线帮助文档；可以创建自己的帮助文档；man 命令根据指定的章节号在目录中搜索文档；在创建自己的帮助目录时，应同时创建相应章节的子目录
man 命令名	搜索并显示指定命令的在线帮助文档
	$man who
MANPATH	Shell 特殊变量，保存了 man 命令将自动搜索的路径；可以使用赋值操作添加新的路径名
	$MANPATH = $MANPATH:/$HOME/man
选项	
数字	只显示指定的章节
	$ man 3 bookrec
-M 目录名	只对指定的目录进行搜索
	$ man -M $HOME/man bookrec
man 宏命令	

续表

.TH 标题章节号	指定在线帮助文档的标题
.SH 章节号	指定章节的题头
.B 单字	使指定单字以黑体显示
.I 单字	使指定单字以下划线方式显示
.IP 选项	段落缩进格式处理。一般用来在 OPTIONS 节中显示选项内容
.PP	开始新段落

第 24 章 Perl

Perl 是 Practical Extraction and Report Language---实用摘录和报告语言的简称。这种编程语言可以完成 awk 和 sed 程序的所有功能，并增加了许多其他的特性。Perl 的最初设计目标是象 awk 和 sed 程序那样，帮助用户对文件进行处理，产生报告并对大文件进行处理。不过，作为核心程序，开发者很容易对 Perl 进行扩充。经过多年的发展，Perl 的功能得到了极大的加强。现在它可以对网络连接、进程交互进行控制，甚至可以对许多数据库管理文件提供支持。在此同时，Perl 仍然保持了良好的可移植性。Perl 程序可以在任何 UNIX 系统下运行，在 Windows 和 Mac 等大多数其他操作系统上也可以顺利运行。另外，Perl 也广泛地应用于网站的 CGI 程序实现等场合。

本书所附的 CDROM 光盘中包含有 Perl 的 Perl5 和 Perl4 版本。一般说来，应使用最新的 Perl5 版本，提供 Perl4 只是为了保持与旧程序的兼容。在 man" 随机帮助文档中可以找到许多详细描述 Perl 的说明，这些说明中用大量的例子对 Perl 语言的各方面进行了描述。这些相关的说明项都以 perl 开头，如 perlfunc 对 Perl 语言中内建的函数进行了说明，而 perlsyn 则描述了 Perl 语言的各种程序控制结构。

在因特网上存在大量与 Perl 语言相关的资源。在 Perl 的网址

www.perl.com，用户可以找到相应的文档、软件、新闻组和技术支持。专业 Perl 网站集中对 Perl 的编程技巧、会议安排和参考资源进行描述。而综合 Perl 文件网络(CPAN)则对相应的 ftp 站点进行维护，在这些 ftp 站点上收集了大量的应用工具、程序模块、技术文档以及最新的 Perl 发行版本。可以通过 Perl 网址连接到类似下面列出的 CPAN 网址：

- www.perl.com
- language.perl.com
- refernce.perl.com
- republic.perl.com
- www.perl.com/CPAN/CPAN.html
- www.cis.ufl.edu/Perl

网址 language.perl.com 主要对 Perl 编程进行讨论，用户可以在这里得到 CGI、模块化和安全性等话题的详细疑难解答，也可以获取更详细的软件文档。网址 refernce.perl.com 提供通过主题的资源访问，如对网络、CGI、图形等主题的访问。网址 conference.perl.com 则提供了交互式的网络会议以及以往的会议资料。在网址 republic.perl.com 可以加入 Perl 编程协会。

因特网上还有一些讨论各种 Perl 话题的 Usenet 新闻组。在这些新闻组上可以提出问题，对当前热门的话题进行讨论。下面列出了一些这样的新闻组：

comp.lang.perl.announce
comp.lang.perl.misc
comp.lang.perl.modules

comp.lang.perl.tk
comp.infosystems.www.authoring.cgi

24.1 Perl命令行操作

用户可以在shell命令行下使用带-e参数的perl命令执行任何Perl命令。希望执行的Perl命令应该用单引号进行引用。命令采用分号进行结尾。

```
perl -e 'print "hello";'
```

可以采用这种命令行方式完成简单的Perl操作，就像使用单行的gawk命令一样。除非显示地在while循环中说明，Perl不会自动地从标准输入设备中读取输入。这一点和gawk程序不同，Perl不会默认读取标准输入。下面的例子在文件books中对字符串"Dickens"进行搜索。首先采用cat命令读取文件books，然后通过管道将其作为标准输入传送给perl命令。

```
cat books | perl -e 'while (<STDIN>) { if (/Dickens/) { print; } } '
```

这条命令与下面的gawk命令等价：

```
cat books | gawk '/Dickens/ {print}'
```

这种Perl命令行操作使用非常少。一般来说，Perl命令用于编写脚本程序文件然后执行，和shell脚本程序的运行方式一致。

24.2 Perl 程序

在通常情况下，perl 命令存放在一个文件中，然后由 perl 命令读入该文件并执行。包含 Perl 命令的文件必须使用.pl 后缀名。这样，就可以使用 perl 命令读入该程序文件。可以使用两种不同的方式读入 Perl 脚本程序。第一种方式是在 shell 命令行环境下键入 perl 命令，然后键入 Perl 程序的文件名。Perl 将会根据输入的文件名读入并执行该文件中的命令。键入下面的例子可以执行名为 hello.pl 的 Perl 程序。

```
$ perl hello.pl
```

第二种方法是与 shell 程序一样，在 Perl 程序文件中调用 perl 命令。在文件的第一行键入下面给出的 shell 指令。这样就可以自动调入 Perl shell，然后执行程序中剩下的 Perl 命令。/usr/bin/perl 是 OpenLinux 系统中 perl 命令存放的地址。在其他的系统中，该命令可能存放在/usr/local/bin 目录中。使用 which perl 命令可以返回本系统中 Perl 命令的存放地址。

```
#! /usr/bin/perl
```

然后，必须更改脚本程序的访问权限，使其可以直接执行。使用 755 参数的 chmod 命令可以设置文件的执行权限，这样就可以在命令行下直接执行该程序文件。对于某一个 Perl 程序，只需要执行一次权限更改操作即可。如果采用上面提到的命令行执行方法，就不必执行权限更改操作。下面的例子赋予 hello.pl 程序文件执行权限。

```
$ chmod 755 hello.cp
```

Perl与gawk和C语言有很多相似之处。与C语言一样，Perl命令以分号结尾。Perl中用于输出文本的print命令与gawk相同。Perl中也使用相同的转义序列输出新行\n和制表符\t(参见表24-1)。Perl中的注释行也和shell和gawk中相同，采用#表示注释行的开始。下面给出了Perl程序的一个例子，它在标准输出设备上输出单词Hello和一个换行府。注意程序中的第一行调用了perl命令。

```
helloprg
# ! /usr/bin/perl
print "hello \n";
$helloprg
hello
```

尽管Perl是一种解释性的语言，在程序执行前还是会首先进行全局的语法检查。如果检查中发现错误，将会在屏幕上显示错误所在的行号和错误类型。许多的错误信息都不是非常明显。这里列出了常见的几种错误。

· 最常见的错误是在命令行的末尾没有加入分号。Perl语言在很多方面与shell和gawk命令非常相似，但在这一点上却完全不同，因此很容易忘记加上分号。

· 控制结构必须由块组成，而不能象C语言中那样，仅仅包含一个单独的命令。

· 变量前必须使用$前缀，这一点与gawk和C中完全不同。

Perl 程序可以分成多个文件，然后在主文件使用 use 命令读入。这种文件采用.pm 后缀名，表明该文件为 Perl 模块。通常，这些模块为文件处理或文本搜索提供了增强的操作方法。标准的 Perl 模块集位于/usr/lib/perl5 目录。联机帮助的 perlmod 页对 Perl 模块进行了详细讨论，包括如何动态载入模块等高级话题。下面的命令读入 find.pm 程序模块。该模块提供了 shell 中搜索目录的 find 命令的 Perl 版本。

```
use /usr/lib/perl5/File/find.pm ;
```

24.3 Perl 输入输出：<>和打印

Perl 程序可以从许多不同的输入源接收输入。它可以从各种不同的文件中、从标准输入设备、甚至从管道中读取输入。因此，必须在程序中指明输入源。在这一点上，Perl 与 shell 程序一致，必须显式指定程序读取输入源，而在 gawk 中，是不需要这么做的。特定的输入源通过文件句柄来识别，程序通过该句柄引用输入源。Perl 中预先为标准输入设备、标准输出设备和标准错误设备都分配了相应的句柄。标准输入设备相应的句柄为 STDIN。

Perl 中的输出也同样是这种情况。Perl 可以输出到许多不同的目的地，如管道、文件或标准输出设备等。文件句柄用来识别输入或输出时的文件和管道。句柄 STDOUT 表示标准输出设备，STDERR 表示标准错误设备。下面我们首先对 Perl 如何使用标准输入输出设备进行描述，然后讨论如何对特定的文件进

行操作。

Perl 可以从标准输入设备或任何指定的文件读取输入。读取输入的命令由小于符号<和大于符号>组成。从文件中读入时，应在符号间插入相应的文件句柄名，如<MYFILE>。从标准输入设备输入时，只需要简单地使用 STDIN 句柄，<STDIN>。<STDIN>与 Bourne shell 中的 read 命令相似。

<STDIN>

<STDIN>命令读入的输入存放于一个特殊的名为$_的变量。用户可以使用该变量对读入的数据进行操作。例如，可以在 print 命令中使用$_变量显示读入的内容。$_是一个特殊的字符串，与 gawk 中的$0 非常相似。它是许多命令的默认操作对象。如果使用<STDIN>读入的行没有赋值给任何变量，它将储存在$_中。如果 print 命令没有参数，将显示出$_的值。如果 chomp 命令没有参数，该命令将对$_进行操作，去掉末尾的换行符。

myread

```
# ! /usr/bin/perl
#从键盘中读取输入数据然后显示该数据的程序
$_ = <STDIN>#从标准输入设备中读入数据
print "This is what I entered: $_";#输出读入的数据
$ myread
laraisa and aleina
This is what I entered: larisa and aleina
```

可以使用 print 命令将数据写入任何文件或输出到标准输出设备。输出时，

所用的文件句柄名应放置在 print 命令后，输出的数据前。如果没有指定文件句柄，print 命令将输出到标准输出设备。标准输出设备的显式文件句柄为 STDOUT。如果没有指定任何参数，print 命令将输出所有从标准输入设备输入的设备。

```
print STDOUT "hello\n";
print "hello\n";
```

空句柄<>是将引发一个特殊的输入操作，程序将从启动 Perl 程序的命令行中指定的文件读取输入。Perl 将自动为命令行中指定的文件分配一个句柄并从中读取数据。如果命令行中同时列出了多个文件，Perl 将使用空句柄从所有列出的这些文件中读取输入。这有点象进行了一次 cat 操作将这些文件连接成一个大文件，然后将这个大文件的内容读入 Perl 程序。

24.4 Perl 文件句柄

使用 open 命令可以为文件或管道建立一个文件句柄。open 命令包括两个参数，即文件句柄名和文件名字符串。文件句柄名可以任意进行设置，传统上应全部使用大写字符。文件名字符串可以是文件名或表示文件名的变量。该字符串中还可以包含文件打开的模式。在默认情况下，文件将以读模式打开。但是也可以通过相应的参数将一个文件以写模式、追加模式或同时读写模式打开。open 命令采用的语法如下：

open (文件句柄，文件名字符串);

在下面的例子中，用户打开 reports 文件，相应的文件句柄为 REPS。

```
open (REPS, "reports");
```

通常文件名将保存在变量中。可以使用$加变量名的方式表示文件名。例如，下面的例子将文件名"reports" 储存在变量 filen 中。

```
filen = "reports";
open (REPS, $filen);
```

如果需要以特定的模式(如写模式或追加模式)打开文件，需要在文件名字符串中加入相应的模式符号。该符号放在文件名前，并用空格分开。表 24-2 中列出了各种不同的模式串。>表示采用写模式打开文件，而＋>表示打开的文件可以同时进行读写。如下面的例子打开的 reports 文件可以同时进行读写。

```
open (REPS, "＋> reports");
```

如果采用变量存储文件名，也可以采取同样的方法指定文件打开的模式，如：

```
open (REPS, "＋> $filen");
```

使用文件句柄从文件中进行读写时，只需要将该句柄放在<和>符号之间即可。<REPS>表示从 reports 文件中读入一行。在下面的 myreport 程序中，将打开 reports 文件并显示其内容。

myreport.pl

```
#! /usr/bin/perl
# 从 reports 文件中读取并显示其内容的程序
```

```
open(REPS, "< reports");#以只读模式打开 reports 文件
while ($ldat = <REPS> )#从 reports 文件中读入一行
{
print $ldat;#显示读入的内容
}
close REPS;#关闭文件
```

24.5 Perl变量和表达式

Perl 中的变量包括数字和字符串。变量的类型是根据变量使用的环境来决定的，不需要提前对变量进行申明。赋给数值或在算术表达式中进行操作的变量为数字变量，其余则为字符串变量。同 shell 程序一样，在程序中引用变量时，应在变量前加上$前缀。

Perl 变量可以使用的运算符与 C 语言相似，不过在对字符串变量进行运算时略有差别。在 Perl 中，字符串的比较运算采用 Bourne shell 使用的运算符，而不是标准的比较运算府。数字变量的比较则采用标准的运算符。不过，其他的各种运算符对数字变量和字符串变量来说都完全一致。在下面的例子中，myname 变量赋值为"Larisa"。

```
$myname = "Larasa";
```

对于数字变量，可以赋值为一个数字。数字的值可以是整形，也可以是浮

点类型。Perl中的所有浮点数都是双精度类型。

```
$mynum = 45;
$price = 54.72;
```

Perl也提供对算术表达式的支持。所有其他编程语言中存在的标准算术表达式都可以在Perl中使用。表达式中可以通过括号进行层次划分，操作数可以为数值常数、数字变量或其他算术表达式。在下面的例子中，首先通过表达式给变量$mynum赋值，然后该变量也参与运算，并将计算结果赋值给变量$price。

```
$mynum = 3 + 6;
$price = ( 5 * ($mynum / 3) );
```

Perl中支持gawk和C中使用的所有赋值运算符。++和--操作符使变量加一或减一。+=和-=操作符可以用来更新变量的数据。例如，i++与i=i+1等价，i+=5与i=i+5等价。在循环中广泛使用诸如i++之类的增量运算符。

如果需要将变量的值添加到字符串中，只需要简单地将该变量放到字符串中即可。在下面的例子中，$nameinfo变量的值即为"My name is Larisa \n"。

```
print "The number of items is $mynum \n"
$nameinfo = "My name is $myuname \n"
```

如果需要将从文件中读入的数据赋值给某个变量，只需要将读入操作的结果付给该变量即可。下面的例子将从标准输入设备中读入的数据赋给变量$mydata。

```
$mydata = <STDIN>;
```

当从键盘上读入数据赋给变量时，输入字符串末尾将包含一个回车换行符。一般来说，不希望在变量中保留该符号。可以使用 chomp 命令删除换行符。chomp 命令会删除字符串末尾的字符，对于键盘输入的数据，该字符正好是换行符。

```
chomp $myinput;
```

在下面的例子中，用户在键盘上输入自己的姓名，该输入将赋值给 myname 变量，然后作为字符串的一部分在屏幕上进行输出。在输出前使用 chomp 命令删除$myname 字符串末尾的换行符。

readname.pl

```
#! /usr/bin/perl
$myname = <STDIN>;
chomp $myname;
print "$myname just ran this program\n";
```

$myread.pl

```
larisa Petersen
larisa Petersen just ran this program
```

24.6 数组和列表

在 Perl 中， 可以使用值的列表建立数组。Perl 中的列表包括一系列括号中

的值，列表中的各个元素采用逗号进行分隔。下面给列出了拥有 4 个元素的列表的一个例子：

{23, 41, 92, 7}

可以将该列表赋值给需要建立的数组，变量名前的@符号表示该变量为数组。这项赋值操作将首先对该数组进行初始化。

@mynums = {23, 41, 92, 7};

在建立数组后，就可以单独引用数组中的元素。元素下标从 0 开始，而不是从 1 开始。mynums 数组中有 4 个元素，下标则从 0 到 3。可以使用方括号中的数字单独对数组元素进行引用。[0]表示第一个元素，[2]表示第三个元素。下面的例子中，将依次打印数组的第 1 个和第 4 个元素。注意这里，数组名前使用的是$前缀符号。

```
print $mynums[0] ;
print $mynums[3] ;
```

可以对数组中单个元素的值进行修改。注意，在单独引用数组元素时必须使用符号$，而不是@作为单个数组元素的前缀。@前缀用来引用整个数组，在对整个数组赋值时使用。而$前缀则用来引用一个特定的元素，这时实际进行操作的是一个单个变量。

```
$mynums[2] = 40;
```

Perl 中对数组中的元素个数没有限制。只要引用一个新的元素并对其进行赋值就可以对数组进行扩展。下面的赋值语句就在 mynums 数组中添加了第 5 个元素。

```
$mynums[4]=63;
```

在每一个数组中都存在一个特殊的变量，该变量由符号#和数组的名称构成。它表示了数组中当前元素的个数。例如，#mynums变量中储存了 mynums 数组中元素的个数。采用下面的语句可以显示出元素的个数。注意这里的$前缀符。

```
print "$#mynums";
```

当采用列表给数组赋值时，列表中的值不一定必须类型相同。在列表中可以存在数字、字符串甚至变量。因此，数组中的元素也并不一定属于同一类型。数组中可以同时存在数字和字符串，如下例所示：

```
@myvar = { "aleina", 11, 4.5, "a new car"};
```

在数组名前使用符号@可以同时引用数组中的所有元素，这时该变量代表的是一个列表。下面的例子可以同时输出 mynums数组中的所有值。

```
print @mynums;
```

在这里使用了符号@作为前缀，因为数组名本身不是一个变量。数组名只是用来表示一个列表，只有数组中的单个元素才是变量。这意味着在引用数组中的所有元素时，应该使用符号@，而不是符号$。在数组间相互进行赋值时也遵守这项规则。在下面的例子中，mynums 数组中的每一个元素都赋值给 newnums数组相应的位置。注意这里使用了前缀@，可以将@mynums想象为一种运算，该运算提取 mynums 数组中相应的列表，然后通过赋值运算将该列表赋值给 newnums变量。

```
@newnums = @mynums;
```

采用数组名引用的列表可以和变量一样在字符串中使用。在下面的例子中，mynums 数组中的元素值成为了字符串的一部分，然后该字符串的值赋给 myreport 变量。注意@mynums 中@前缀的用法。

```
$myreport = "Here are the numbers I have: @mynums \n";
print $myreport;
```

在利用下标单独引用数组元素时，仍然应该使用$前缀。下面的例子中使用了 mynums 数组中的第三个元素：

```
$myelement = "This is the calue of the third element: $mynums[2] \n";
print $myelement;
```

24.6.1 数组管理函数：列表操作

Perl 中定义了一套函数来帮助用户方便地对数组进行管理。使用非常简单的命令，就可以完成常见的数组操作，如列出数组元素、对数组进行排序、依序引用数组中的元素等。从这一方面来看，可以将数组想象成一个列表，这些函数就像在对列表进行操作。表 24-3 中列出了这些数组和列表函数。

push 和 pop 函数对数组的末尾进行操作，添加或删除最后一个元素。push 函数在数组的最后面加入一个新的元素。push 函数的第一个参数即为数组名，然后是加入数组的数值列表。采用该函数可以在数组中一次加入多个值。pop 函数则将数组中的最后一个数值删除并将其作为返回的参数。shift 和 unshift 函数则正好与之相反，对数组的开始处进行操作。shift 函数将移去列表中的第一

个元素，使第二个元素代替第一个的位置。而 unshift 函数则会在列表的开头加入一个新的元素。shift 函数使用数组名作为参数，返回结果为移去的第一个元素的值。unshift 函数则使用数组名和试图加入的元素作为参数。该函数返回数组中现有的元素个数。sort 操作则会按照系统的字符集(通常为 ASCII 字符集)对数组中的元素按升序进行排序。sort 操作返回排序后的列表。split 操作用来创建一个数组并从字符串中给该数组进行赋值。这些字符串由某个特定的分隔符进行划分，然后按照指定的规则赋给相应的数组。split 通常用来对大量的输入数据进行处理，这些数据一般已经由特定的分隔符(如冒号或逗号)划分为不同的域。也可以使用 grep 函数对数组或列表进行搜索。这里的 grep 函数与 UNIX 中的 grep 命令非常相似，它使用一个搜索模式和一个列表作为参数，返回列表中与模式匹配的值。

24.6.2 关联数组

关联数组使用字符串对数组中的元素进行引用，而不是数组中使用的数字。可以将用于索引的字符串认为是访问数组元素的关键字。在 Perl 中，在数组名前采用%前缀就可以定义一个关联数组。数组中的值列表包括索引串-元素值对。列表中首先出现的是索引串，然后是元素值，然后又是索引串、元素值，依此类推。在下面给出的例子中，关联数组 city 拥有 4 个元素，每个元素中包含一个整数值和一个代表城市名称的索引串。

```
%city = ('Sacramento', 4,
```

```
'Fallon', 86,
'Napa', 7,
'Alameda', 53 ) ;
```

使用索引串可以引用关联数组中的值，索引串放在单引号和花括号中。

```
print $city{'Fallon'};
```

在关联数组中添加新的元素时，必须同时提供元素的索引字符串和元素的值。

```
$city{'LA'} = 45;
```

在关联数组名前加上%符号前缀可以引用整个数组列表的值。下面的例子显示了 city 数组中所有的值：

```
print %city
```

Perl 中为关联数组的提供了一系列的操作。使用这些操作，可以分别得到索引字符串和元素值的列表。keys 操作使用关联数组作为参数，返回数组中索引字符串的列表。得到该列表后，就可以方便地对数组中的元素值进行访问。例如，使用 krys 操作得到数组中索引字符串的列表后，就可以将其在一个 foreach 循环中使用，依次得到关联数组中的元素数值并进行打印。

24.7 控 制 结 构

Perl 中的控制结构体系和 gawk、C-shell 和 C 语言非常相似。Perl 中存在

循环结构可以重复执行命令，存在条件结构允许执行不同的命令集。在表达式测试中，对于字符串和数值变量存在两套不同的操作符。表 24-4 和 24-5 列出了这些相应的操作符。在使用正则表达式时也可以使用模式操作。表 24-6 列出了 Perl 控制结构的语法。

24.7.1 循环

Perl 中的循环结构包括 while、do-until、for 和 foreach 循环。应用得最广泛的是 while 循环，而 for 和 foreach 循环则提供了一些特殊的功能。在对列表和数组进行处理时 foreach 循环就显得特别有用。while、do-until 和 for 循环的执行过程和 C 语言中相应的结构非常相似。尤其是 for 循环，它的三表达式结构与 C 语言中完全一致。foreach 循环则和 C-shell 中相应的结构非常相似，可以使用它来简单地处理列表结构。

只要稍微作一下改动，就可以在处理数组时使用 while 循环结构。一般的做法是在将控制循环的变量同时在循环内部用来对数组进行索引。在下面的例子里，title 数组经过赋值后使用 loop 循环输出数组元素的值。注意变量$#nm 中存放着数组中的元素个数。在 for 循环中使用$#num 变量作为循环的上界检查循环是否应该结束。

titlearr.pl

```
#!/usr/bin/perl
#标量数组的赋值与输出
```

```
@title = ("Tempest", "Iliad", "Raven");#定义三个元素的数组
for ($i = 0; $i <= $#title; $i++ )#循环，$#title 为数组的大小
{
print "$title[$i] \n";#输出 title 数组的元素
}
$ titlearr.pl
Tempest
Iliad
Raven
```

foreach 循环在对数组进行管理时非常有用。在 foreach 循环中，可以使用数组名生成数组中所有元素值的列表。foreach 循环就可以使用该列表来控制循环的范围。循环的范围也可以使用数组元素的一个子集，或几个单独元素组成的元素集。在下面的例子里，数组名@mylist 用来生成一个数组元素值的列表，foreach 循环可以使用该列表依次对$mynum 变量进行操作。

```
mynumlist.pl
#!/usr/bin/perl
#使用 foreach 循环输出数组元素的值
@mylist = (34, 21, 96, 85);#定义一个存在 4 个元素的数组
foreach $mynum( @mylist ) #将元素值依次赋给变量$mynum
{
print "$mynum \n";
```

```
}
```

命令行参数可以通过@AGRV 数组来访问。使用 foreach 循环可以依次调用运行程序时的所带命令行参数。在第一次循环中引用该数组时，变量的值即为第一个命令参数，第二次循环时，则为第二个命令行参数，依此类推。

用户在命令行键入的参数个数可能发生变化，用户实际输入的参数个数储存在#ARGV 变量中，这也正是 ARGV 数组的元素个数。如果需要访问 AGRV 数组中的所有元素，必须首先通过#ARGV 变量获取元素的个数。例如，在使用 foreach 循环引用 ARGV 数组中的所有元素时，可以使用..元素符生成相应的索引表。0..$#ARGV 生成一个从 0 到$#ARGV 的数值列表。

Perl 提供一些特别的命令使得用户能够更加简洁地控制循环和语句块。这些命令和 C 语言中使用的一些命令比较相似。在使用这些命令时，需要引入语句标号的概念。如果在循环内部使用这些命令，需要对循环进行标记，一旦在循环内部执行到这些命令，就需要通过标号寻找程序下一步执行的位置。

Perl 中的 last 命令对应于 C 语言中的 break 语句。该命令用来停止循环的执行,也可以用来跳出某个程序块。last命令也可以和程序块配合使用完成 switch 命令的功能。next 命令跳过循环中的其余部分，与 C 语言中的 continue 语句相似。continue 语句块由 continue 语句和程序块组成，该程序块在循环的末尾执行。即使使用 next 命令跳过了循环中的其余部分，仍然必须执行 continue 程序块中的语句。可以使用 continue 语句块完成如循环变量变化等循环中必须进行的操作。redo 命令将重新执行循环的内容，即使在循环条件为假的情况下也是如此。在 C 语言中找不到与之相应的语句。redo 和 last 命令都将忽略

continue 语句块的存在。

24.7.2 条件命令：if、else、unless 和 switch

Perl 支持编程语言中的 if-else 条件语句。if 结构(包括 else 和 elseif)可以使程序在不同的情况下执行不同的分支。在程序中可以只使用 if 命令选择一种方案，也可以使用 if-else-elseif 结构在多种方案中选择执行。if 结构由括号中的测试表达式和随后的语句块构成。如果测试表达式值为真，将执行语句块中的命令，否则，就将跳过该语句块。与其他的编程语言不同，在测试表达式后只能有一个程序块，另外，即使语句块中只有一条语句，也必须用花括号将其封装起来。在下面的例子里，将检查文件的打开操作是否执行成功。如果不成功，就使用 die 命令终止本程序。表达式中的感叹号！为取反运算符，这样当文件打开失败时测试表达式就会返回真值，从而能够执行 die 命令。

```
if (! open (REPS, "< $filen"))) {
die "Can't open $filen";
}
else {
print "Opened $filen successfully" ;
}
```

24.8 字符串函数

Perl对字符串提供了一系列的函数和操作(见表 24-7)。length、index和 substr函数执行一些标准的字符串运算，如获取字符串长度、拷贝子串等。split 函数执行的功能则比较特殊。它生成一个数组，将字符串划分为一些数组元素。该函数与 gawk 中的 split 函数比较类似。在前面对数组和列表的操作中我们已经讨论了 split 函数的一些问题。点操作符用来连接字符串，而 x 操作符则将生成包含有重复字符的新串。

点操作符(.)用来连接两个字符串。下面的例子用来在文件名后加入".dat"后缀。如果变量$curfile 的值为 myaddress，那么操作完成后变量$newfile 的值应为 myaddress.dat。

```
$newfile = $curfile.".dat" ;
```

x 操作符用来进行字符串复制。使用时在 x 操作符的左边为需要复制的字符串，右边为需要复制的次数。操作符左边的字符串可以是单个字符，也可以是多个字符组成的字符串。使用 x 操作符还可以生成元素相同的列表，然后使用该列表对数组进行赋值。在下面的例子中，myarray 数组赋值为 5 个 0，即(0,0,0,0,0)。使用 x 操作符的好处是，在语句中不必重复地列出相同的 0 元素。

```
@myarray = (0) x 5;
```

24.9 模 式 匹 配

与 Sed 和 gawk 类似，Perl 也可以在给定的行中进行模式搜索。搜索模式一般由两个斜杠表示。/Dickens/表示在行中搜索模式 Dickens。在默认情况下，模式搜索的对象是特殊变量_$中的值。一般来说，该变量中保存了最新的从输入源输入的内容。如果最近的输入命令为<STDIN>，则_$变量中的值为从标准输入设备读入的值，随后的模式搜索工作就会对这个变量中的值进行。在下面的例子里，用户从标准输入设备读入一行，然后在读入的行中搜索模式/Dickens/。

```
<STDIN>;
/Dickens/;
```

使用=~操作符可以在变量中进行模式搜索。操作符左边为变量名，右边为需要匹配的字符串。表 24-8 中列出了 Perl 中的模式匹配操作和对应选项。下面的例子将在$title 变量中查找模式串"Christmas"。

Perl 中的模式匹配操作支持包括扩展特殊字符在内的所有正则运算表达式。可以使用特殊字符*，＋，?，{}，.，^，$和[]完成非常复杂的搜索操作。第 14 章中已经对正则表达式的使用作了深入讨论。另外，Perl 中还定义了一些转义字符代表一些常用的匹配模式。如\w 可以匹配任何字母或数字字符，与正则表达式 [A-Za-z0-9]+等价。\d 匹配数字，\s 则与空格相匹配。单字前的\b 前缀表示对单字，而不是对单字代表的模式进行匹配。大写的转义字符则代表完全

相反的含义。\W 匹配任何不是字母和数字的字符，\D 则可以对不是数字的任何字符进行匹配。表 9-24 中列出了 Perl 正则表达式中可用的这些转义字符。

24.10 函数：子程序

Perl 脚本程序中可以定义子函数，这样更容易对程序的结构进行组织。可以将程序按照不同的功能划分为几个子程序，而不必将这些语句全部写在一段代码中。可以为多次使用的功能单独定义一段子程序然后在需要的地方进行调用，而不必多次重复地书写相同的代码。Perl 中的子程序由关键字 sub、子程序名和语句块组成。在语句块中可以书写任意所需的 Perl 语句。Perl 中的子程序与 C 语言中的函数或其他编程语言中的子程序类似，可以预定义调用时的参数。表 24-10 中列出了构成 Perl 子程序的不同组件。下面的例子定义了一个名为 dispfile 的子程序。

```
sub dispfile
{
print "Please enter in name of file to be displayed:";
chomp($filename = <STDIN>);
system("cat $filename");
}
```

子程序可以接收主程序调用时使用的参数并向主程序返回调用后的值。调

用参数和返回值采用列表方式进行传递。调用子程序时使用的参数存放在列表中，通过@_数组传递给子程序。@_数组中将保存当前调用的子程序所使用的调用参数。使用数组或列表操作引用@_数组中的元素就可以访问特定的参数。

@_数组中的参数从 0 开始计数，使用时与一般数组的访问方法完全相同。使用$_[0]可以对第一个参数进行访问，$_[1]为第二个数组，依此类推。在下面的例子中，disfilearg 子程序将文件名作为主程序送来的参数。它使用$_[0]来访问@_数组中传递过来的文件名。

```
sub dispfielarg
{
system("cat $_[0]");
}
```

子程序的调用可以使用下面两种形式中的任一种：

```
dispfilearg ("myfile");
&dispfilearg "myfile";
```

对于参数中的常数，@_数组中只保存参数的值。但如果使用变量或数组作为参数，@_将直接引用相应的对象，子程序在对@_数组中的元素进行操作时，实际上是对原始的对象进行操作。因此，假设使用字符串变量作为函数的参数，如果子程序中的语句通过@_数组引用了该参数，并对其进行了赋值操作，该字符串变量的值也将被改变。

表 24-1Perl 转义字符

转义字符	描述
\a	响铃
\b	后退
\cC	控制字符 C.如\cD 为 CTRL-D
\f	Form feed
\e	ESC
\n	新行
\r	回车
\t	制表符
\\	反斜杠\
\"	引号"
\uC	大写字符 C
\lC	小写字符 C
\L 文本\E	\L 和\E 之间文本的所有字符小写
\U 文本\E	\L 和\E 之间文本的所有字符大写

表 24-2Perl 操作和命令行选项

Perl 命令行选项	功能
-e	输入一行 Perl 程序
-n	从命令行上列出的文件中读取程序
-p	向标准输出设备输出所有读入的数据
Perl 文件命令	
open(文件句柄, 文件名)	按指定权限打开文件
close(文件句柄)	关闭文件
<文件名>	从文件中读入数据
<STDIN>	从标准输入设备读入数据
<>	从调用程序时的参数列表列出的文件中读入数据
print <文件句柄> 文本	输出到文件。如果没有指定文件句柄,将输出到标准输出设备。如果没有指定输出的文本，将输出变量$_的值
printf <文件句柄> "格式串", 值;	输出到文件。使用格式串对输出的值进行转换。如果没有指定文件句柄,将输出到标准输出设备。如果没有指定输出的值，将输出变量$_的值

续表

sprintf 字符串变量"格式串", 值;	输出到字符串。使用格式串对输出的值进行转换。如果没有指定输出的值，将输出变量$_的值打开文件的权限
<	文件名只读
>	文件名只写
＋>	文件名读写
>>	文件名追加(在文件末尾添加数据)
命令 |	管道输入，从管道中读取数据
| 命令	管道输出，向管道输出数据

表 24-3Perl 数组操作

标量数组操作, @array	
push(数组，值列表)	在数组末尾处添加元素
pop(数组)	删除数组末尾处的元素
shift(数组，值列表)	在数组开始处添加元素
unshift(数组)	删除数组开始处的元素
sort(数组)	按字母升序排序
reverse(数组)	按字母降序排序
split(分隔符，字符串)	将字符串按分隔符分割为数组元素。分隔符可以是模式或字符串

续表

join(分隔符，字符串)	将数组元素联合成字符串
grep(字符串，模式串)	在数组中搜索模式
splice(数组，索引号)	删除数组中指定的元素，索引号从 0 开始
splice(数组，索引号，数字)	删除数组中多个元素
splice(数组，数字，0，字符串)	在数组中添加一个值为指定字符串的元素标量数组操作,@array
splice(数组，$#argv, 0,字符串)	在数组的末尾添加新的元素关联数组操作，%array
keys(%关联数组)	生成关联数组中索引串的列表
values(%关联数组)	生成关联数组中值的列表
each(%关联数组)	返回数组中下一个元素的值和相应的索引串
delete(%关联数组，索引串)	在关联数组中删除指定的元素一般数组操作
undef(数组)	删除整个数组

表 24-4 数字运算符

算术运算符	功能
*	乘
/	除
＋	加
-	减
%	取模-除法所得的余数
**	幂
关系运算符	
>	大于
<	小于
>=	大于等于
<=	小于等于
==	等于
!=	不等于
增量运算符	
＋＋	变量加 1
--	变量减 1
算术赋值运算符	
＋=	增加指定值
-=	减少指定值

续表

/=	变量等于本身除以指定值后的值
*=	变量等于本身乘以指定值后的值
%=	变量等于本身对指定值取模后的值

表 24-5 字符串、逻辑、文件和赋值操作符

字符串比较	功能
gt	大于
lt	小于
ge	大于等于
le	小于等于
eq	等于
ne	不等于
逻辑操作	
表达式 1 && 表达式 2 表达式 1 and 表达式 2	如果两个表达式都为真值 0，逻辑与操作返回真值 0；如果表达式中有一个为假，逻辑与操作也返回非 0 假值；逻辑与操作在第一个为假的表达式处停止；and 操作符与&&操作符相同但优先级较低

续表

表达式 1 || 表达式 2 表达式 1 or 表达式 2	如果两个表达式中有一个为真值 0，逻辑或操作返回真值 0；如果两个表达式都为假，逻辑或操作也返回非 0 假值；逻辑或操作在第一个为真的表达式处停止；or 操作符与||操作符相同但优先级较低
! 表达式 not 表达式	逻辑非操作对表达式的值取反；not 操作符与!操作符相同但优先级较低
文件测试	
-e	文件存在
-f	文件存在且为一般文件
-s	文件非空
-z	文件为空，字节数为 0。
-r	文件可读
-w	文件可写
-x	文件可以执行
-d	文件名为目录名
-B	文件为二进制文件
-T	文件为文本文件
赋值操作符	
=	对变量赋值

表 24-6Perl 条件、循环和函数

标号:{语句;}	语句块由花括号之间的一些语句组成，语句块中的语句将顺序执行，可以为语句块加上标号
条件控制结构:	
if, else, elsif, case	
if (表达式) {语句;}	如果测试表达式为真，执行 if 语句，语句中必须包含在一个块中
if (表达式) {语句 1;}	如果测试表达式为真，执行 if-else 语句；如果为假则执行 else 语句
else {语句 2;}	
if (表达式) {语句;}	elsif 可以实现语句执行的多种选择。当测试表达式为真时，将执行对应的语句并跳出整个 elsif 块
elsif (表达式) {语句;}	
else (表达式){语句;}	
unless (表达式) [语句;}	如果测试表达式为真，将执行 unless 语句
条件?语句 1:语句 2	条件表达式。如果条件为真，执行语句 1，否则执行语句 2

续表

标号: {if (表达式) { 语句;last 标号};}	通过使用块中的最后一个 if 语句指向该块的标号来模拟 switch 结构
循环控制结构 :while, until, for, foreach 标号: while (表达式) { 语句;}	while 结构在测试表达式为真时，反复执行语句。标号为可选项
do {语句;} until(表达式)	until 结构在测试表达式为假时，反复执行语句
foreach 变量 (值列表){语句;}	foreach 结构是为使用数组这样的值列表设计的。循环过程中变量将依次取列表中的值
for (初始表达式; 测试表达式;增量表达式) {语句;}	for 循环结构在测试表达式为真时反复执行相应的语句。初始化表达式在循环开始时执行一次，增量表达式则在每次循环执行完语句后执行
标号: 语句块或循环	对语句块或循环做标记。为 next，last 和 redo 命令做准备
Next	跳过循环中剩下的语句，直接开始下一次迭代；与 C 语言中的 continue 命令类似；但仍然会执行 continue 块中的语句

续表

continue {语句;}	continue 块中的语句在循环末尾执行；即使使用 next 语句直接开始下一次迭代，仍然会执行 continue 块中的语句
redo 标号	即使测试表达式为假，仍然重新执行循环；跳过 continue 块中的语句
last 标号	跳出语句块或循环；跳过 continue 块中的语句；与 C 语言中的 break 命令类似
函数	
sub 函数名;	申明一个函数
sub 函数名 {语句;	定义一个名为指定函数名的函数
& 函数名(参数列表)	使用参数列表中的指定的参数调用函数
@_	保存传递给当前函数的参数值。使用$_和索引号可以对参数进行引用。$[0]为第一个参数
$#_	传递给当前函数的参数个数。

表 24-7 字符串操作

字符串操作	描述
字符串1 . 字符串2	点号操作符连接字符串
字符串 x 数字	将指定字符串或字符重复指定次数
字符串函数	
chomp(字符串)	删除字符串末尾的换行符
substr(字符串，起始位置，长度)	返回字符串中的子串
substr(字符串，起始位置，长度) = 字符串	将字符串中指定的部分替换为指定的字符串
length(字符串)	返回字符串的长度
index(字符串，模式)	返回字符串中指定模式出现的位置
rindex(字符串，模式)	返回字符串中指定模式最后一次出现的位置

表 24-8 模式匹配操作符和选项

模式匹配操作符	动作
变量名 =~ /正则表达式/	查看字符串变量中是否存在指定模式
变量名 !~ /正则表达式/	检查字符串变量中是否不存在指定模式
/正则表达式/	使用正则表达式进行模式匹配
模式匹配选项	

续表

I	模式匹配时忽略字母大小写
M	字符串为多行
S	字符串为单行
X	忽略匹配过程中的空格和注释

表 24-9 正则表达式

特殊字符	描述
^	行开始
$	行末尾
c*	匹配一个或多个连续出现的字符 c
.	匹配任意的单个字符
c＋	匹配两个或多个连续出现的字符 c
c?	匹配一个或两个连续出现的字符 c
c{数字}	匹配数字＋1 个连续出现的字符 c
c{最小值，最大值}	对字符 c 后连续出现的相同字符的个数范围进行指定匹配
[字符列表]	字符列表
[^字符列表][^字符 1-字符 2]	任何不在字符列表或字符范围中的字符正则表达式 | 正则表达式匹配出现的任意一个正则表达式，与逻辑或操作类似

续表

(模式段)\n	使用\n 代码指定模式段中用来进行匹配的模式。\1 将引用第一段，在 s///替换中使用
转义字符正则表达式	
\w	任意字母或数字字符，等价于[a-zA-Z0-9]+
\d	任意数字，等价于[0-9]+
\s	空格符(空格，制表符，新行，form feed)，等价于[\n\t\r\f]+
\b	单字匹配单字
\W	任何不是字母或数字的字符，等价于[^a-zA-Z0-9]+
\D	任何不是数字的字符，等价于[^0-9]+
\S	任何不是空格的字符，等价于[^\n\t\r\f]+
\B	模式任何不是单字的模式

表 24-10Perl 子程序

子程序组件	描述
@_	存放参数列表的数组。调用时如采用参数数组，该参数数组也是@_数组中的一个元素 sub 程序名 {语句；}定义一个子程序
sub 程序名;	声明一个子程序
sub 程序名 {定义列表};	子程序原形声明
程序名；	调用子程序
程序名(参数)；	调用子程序
&程序名 参数；	调用子程序
my (对象列表)	限制在子程序或语句块中的局部变量
local (对象列表)	全局对象的局部版本，只在当前子程序或语句块及其调用的子程序中有效
\@数组名	数组引用(在参数列表中使用保持数组的一致性)

第 25 章 Tcl, Tk 和 Expect

Tcl 是 John Ousterhount 1987 年在加州大学伯克利分校开发的一种通用命令语言。该语言起先是为应用程序进行定制时设计的，但它现在已经成为一种功能强大的语言。与 Perl 和 gawk 相同，用户可以建立自己的 Tcl 脚本文件，开发自己的 Tcl 程序。Tcl 是一种使用非常简单的语言。

随着 Tcl 应用程序的不断发展，Tcl 语言本身的功能也越来越强。使用合适的应用程序，Tcl 程序可以创建图形化界面，与其它程序相互通讯，甚至可以对声音记录进行管理。使用 Tk 应用程序可以非常简单地开发交互式图形界面的应用程序，开发者可以很简单地实现带有按钮和文本框的窗口和对话框。XF 在 Tk 的基础上又前进了一步，它为开发者提供了图形组件的实现方法。Expect 应用程序则可以实现 Tcl 程序与交互式应用程序(如 ftp, Telnet)的相互通讯。Tcl-DP 为 Tcl 应用程序行发提供了支持。AK 则提供了音频功能，如录音、回放和电话控制等。

Tcl 通常与 Tk 同时使用以建立图形化的应用程序。Tk 用来创建窗口等图形组件，Tcl 则用来完成相应的程序动作，如对用户输入进行响应等。同 Java 一样，Tcl/Tk 是一种跨平台的应用程序。用 Tcl/Tk 编写的程序可以在任何安装了 Tcl/Tk 解释程序的平台上运行。目前，在 Windows、Mac 和 UNIX 系统(包括 LINUX)

上都有 Tcl/Tk 的解释程序。用户可以在 LINUX 上开发 Tcl 应用程序，然后在 Windows 或 Mac 上直接运行。新版的 Tcl 和 Tk 8.0 甚至可以自动搜索当前使用的操作系统，然后相应地在 Mac 系统上使用类似 Mac 的窗口，而在 Windows 95 上则使用 Windows 风格的窗口。

与 Perl 相同，Tcl 是一种解释语言，拥有自己的 shell 解释程序。在命令行上使用 tclsh 命令可以调入 Tcl shell。用户可以在 shell 中执行 Tcl 命令，也可以将 Tcl shell 调用语句和 Tcl 命令储存在一个 Tcl 程序中，然后直接调用该程序。Tcl 语言及其应用程序的一个显著优点是它与 C 语言完全兼容，可以在 C 语言中直接调用 Tcl 库。这样，用户就可以开发出编译后的执行速度非常快的 Tcl 程序。

在系统中安装 Tk 和 Tcl 的同时,也将安装 Tcl/Tk 命令的帮助文件。使用 man 命令和 Tcl/Tk 命令名可以得到该命令的详细信息。例如，man switch 将显示 Tcl switch 命令的详细信息，man button 则将显示 Tk button 组件的详细信息。在安装了 Tk 后，可以运行一个名为饰件的演示程序，该程序将显示所有可用的 Tk 组件。饰件程序使用 Tcl/Tk 示例程序并可以显示每个程序的源代码。饰件演示程序位于 Tk 的 demos 目录下(这里的 tk*代表 Tk 所在的目录，目录名由 tk 和相应的版本号组成，如 4.1 版本的目录名为 tk4.1，而 8.0 版本的目录名则为 tk8.0)。

cd /usr/lib/tk*/demos

在 Xterm 窗口中只需要键入命令饰件即可。也可以单独查看某一个演示文件并对其进行修改。如果在安装 Tk 时不是使用默认的/usr/bin 目录而采用了

/usr/local/bin 目录，那么 demos 目录将位于 /usr/local/lib/tk*下。

25.1 Tcl/Tk 产品和版本

目前，Tcl、Tk 和其它相关的软件都是由 Sun 公司开发并提供支持的，John Ousterhout 领导 Tcl/Tk 的研究工作。当前最新的 Tcl 和 Tk 发行版本为 8.0。Sun 公司还同时为因特网和其它应用程序开发提供 3 种其它的 Tcl/Tk 产品：Tcl/Tk 网页插件、TclHttpd 网络服务器和 SpecTcl Tcl/Tk GUI 开发工具。表 25-1 中给出了可用的 Tcl/Tk 软件列表。Tcl/Tk 网页插件允许用户在自己建立的网页中插入 Tcl/Tk 程序，这些 Tcl/Tk 程序通常被称作 Tclets。Tclhttpd 是一种网络服务器，可以简单地插在应用程序中使其能够提供对网络的支持。SpecTcl 则是一个 Tcl/Tk 编辑器，通过它可以方便地建立应用程序的图形界面。Spec 可以同时生成相应的 Tcl/Tk 和 Java 代码，可以用来建立可单独运行的 Tcl/Tk 应用程序或网络 Tclets。SpecTcl 是一个基于窗口系统的程序，可以通过菜单和图标方便地创建应用程序的图形界面。

所有这些产品，包括最新版的 Tcl 和 Tk 程序，都可以从下面列出的 Tcl/Tk 网址免费获取。在这个网址上还可以得到每个产品的详细文挡说明，包括 PostScript, Adobe PDF 和 HTML 等格式。HTML 格式的文档可以在线进行查看。

http://sunscipt.sun.com

目前的 OpenLinux 光盘中只包含 Tcl 和 Tk 的旧版本，版本号分别为 7.5 和 4.1。大部分的 Tcl/Tk 产品都不能在这些旧版本上运行，因此应该从以上的网址下载安装新的 8.0 版本。从 RedHat 公司的 ftp 站点 ftp.redhat.com/pub/contrib/i386 可以得到最新的 Tcl/Tk 8.0RPM 软件包。用户在下载时应该同时下载 Tcl 和 Tk 的 RPM 包和相应的开发程序包。这里列出了这些文件的文件名。在下载之后，就可以使用 rpm -i 命令安装这些程序。注意，如果系统中存在旧的 Tcl/Tk 版本，在安装前，必须使用 Lisa(或 rpm -e tk 和 rpm -e tcl)将其删除(使用 rpm qi tk 可以检查是否存在这些版本)。

```
rpm -i tcl-8.0-1.i386.rpm
rpm -i tk-8.0-1.i386.rpm
rpm -i tcl-devel-8.0-2.i386.rpm
rpm -i tk-devel-8.0-2.i386.rpm
```

在删除旧的 Tcl/Tk 包前，应该对它们的共享库，/usr/lib/libtcl7.5.so 和 /usr/lib/libtk4.1.so 作一个备份(改名)。有一些 Tcl/Tk 应用程序如 XF86setup 需要旧共享库的支持。因此在运行 XF86Setup 之前需要暂时将 libtk.so 和 libtcl.so 目录连接到备份的旧目录，在程序运行结束后再将其指回原来的目录 /usr/lib/libtcl8.0.so 和 /usr/lib/libtk8.0.so。

从 Tcl/Tk 网站上可以下载最新的源程序代码。这些源程序代码都经过压缩，后缀名为 .tar.gz。使用 gunzip 和 tar xvf 命令就可以对该文件进行解压缩，解压后的文件分别存放在 tcl8.0 和 tk8.0 目录下。解压后可以查看 tcl8.0 目录中的 README 文件得到相应的指令信息，在 unix 目录下的 README 文件可以得

到如何编译并使用这些软件代码的办法。这些源代码使用一个配置脚本文件可以极大地简化编译的操作。下面的命令可以在 OpenLinux 系统上运行，-enable-gcc 选项选择 gcc 编译器。

```
./configure -enable-gcc
make
make install
```

对 Tk 的源程序代码也需要进行同样的工作。tk8.0 目录中也存在相应的 unix 子目录。执行完 make 命令后，将建立两个名为 tchsh8.0 和 wish8.0 的程序。这两个程序存放在/usr/local/bin 或/usr/bin 目录中。make install 操作则安装相应的帮助文档、函数库和例子程序。

```
# ln /usr/local/bin/tclsh8.0   /usr/bin/tclsh
# ln /usr/local/bin/wish8.0   /usr/bin/wish
```

25.2 T cl

Tcl 是一种易于使用的编程语言。Tcl 中的语句由相应的命令和所带的参数组成，当然在 Tcl 中也存在包括 while 和 for 循环在内的一套完善的控制结构。分号和换行都可以表示一条语句的结束。Tcl 命令就像一条函数调用语句，命令名即为函数名，所带的参数就是调用使用的参数。不过，与函数调用不同的是，在调用参数的前后没有使用括号或引号等符号来显式进行说明，只需要使用空格将函数名和参数分开即可。

在 Tcl 赋值语句 set 中可以很清楚地看到这种格式的特点。在对变量进行赋值时，首先输入赋值命令 set，然后输入变量名和所赋的值。赋值命令、变量名和所赋的值中间用空格隔开。

下面的例子将字符串"larisa"赋值给 myname 变量，将 11 赋值给变量 age。

```
set myname "larisa"
set age 11
```

与 gawk 和 Perl 相同，变量的类型由变量的使用方式决定。赋值为整数的变量将为整数类型，赋值为字符串的变量则为一个字符数组。

25.2.1tclsh shell 和脚本文件

在 Tcl shell 中可以执行相应的 Tcl 命令。执行命令存在两种不同的方式，可以在 Tcl shell 提示符下交互式地输入命令，一步一步地执行，也可以将这些命令写入一个脚本文件，然后一次执行这些命令。键入命令 tclsh 可以启动 Tcl shell。shell 的提示符为%，在 shell 中可以输入 Tcl 命令，当按下回车键后就可以执行该命令。输入 exit 命令或按下 CTRL-D 组合键都可以退出 Tcl shell。

```
$ tclsh
% set age 11
% puts $age
11
% exit
```

```
$
```

可以将 Tcl 脚本程序以单独的程序运行，也可以使用 Tcl shell 命令 tclsh 显式读入该文件然后运行。Tcl 脚本程序的后缀名为.tcl。例如，myread.tcl 脚本程序可以使用下面列出的方式来读入并执行：

```
$ tclsh myread.tcl
```

如果需要将该脚本作为单独的程序运行，应该在脚本中调用 tclsh 命令。可以采取两种方式完成这项操作。第一种是在脚本程序的第一行中显式指出调用 tclsh 命令的路径，如下所示：

```
# ! /usr/bin/tclsh
```

第二种方式不需要显式指定 tclsh 所在的路径，而是在指定的系统上执行 tclsh 命令。这样做的好处是，即使不同系统中 tclsh 所在的目录不同，脚本程序也可以正常运行。

```
# ! /bin/sh
# the next line restarts using tclsh \
exec tclsh "$0" "$@"
```

本节中的 myread 例子脚本程序将使用显式调用的方案，而 myreport 脚本则使用第二种非显式调用的办法。

25.2.2 表达式

Tcl 中表达式的处理与命令相同。expr 命令计算表达式的值并将计算的结果

作为字符串返回。它的参数即为表达式的运算数和运算符。Tcl支持标准的算术、比较和逻辑操作。算术表达式的结果与参与计算的运算数的类型相同。如果运算数均为实型，计算的结果也是实型。在表达式中可以混合使用多种类型，Tcl会自动进行相应的转换。如果操作数中同时存在实型和整型，计算时 Tcl 会将整形转换为实型。在下面的例子中，expr 命令计算了 4 和 3 的和以及 25 和 2 的乘积。

```
expr 4 + 3
expr 25 * 2
```

可以使用括号创建复杂的表达式，最里层的括号中的表达式将首先进行计算。在下面的例子中，将首先计算 25*2，然后从结果中减去 20。

```
expr (25 * 2) - 20
```

任何 Tcl 命令返回的结果值都是一个字符串。对于算术表达式来说，Tcl 将首先将计算的结果转换为字符串，然后再将转换后的字符串返回。

25.2.3 嵌套命令

Tcl 中可以将命令嵌套使用。在使用表达式的结果对变量进行赋值时经常使用嵌套命令。这里将使用两个命令，set 命令用来执行赋值操作，expr 命令用来计算表达式的值。命令嵌套时应使用方括号，方括号中的命令返回的结果作为外层命令的某个参数。下面的例子将算术运算 25*2 的结果赋值给 num 变量。expr 25 * 2 在 set 命令中嵌套使用。方括号中的表达式，25*2 将首先被执行，

随后将计算的结果 50 赋给变量 num。

```
set num [expr 25 * 2]
```

25.2.4 变量

Tcl 支持算术和字符串变量，也支持包括关联数组在内的数组变量。所有变量的内容都以字符串的形式进行储存。不过，如果字符串由数字组成，该字符串仍然可以直接在算术表达式作为整数或实数使用,Tcl 将自动对字符串进行相应的转换。转换后参与运算完毕后，得到的结果再转换为字符串返回。因此，开发人员不必对变量的类型进行声明，甚至根本就不必对变量进行定义。变量都将在第一次使用时自动定义。

可以使用 set 命令对变量进行赋值，set 命令使用的参数为变量名和赋给该变量的值。变量名可以由字母或数字字符以及下划线组成，变量名中不允许使用点号或其它字符。在语句中使用变量值时需要对变量进行计算，即使用变量的值代替变量名参与语句运算。在变量名前加上$符号前缀就可以完成这种转换工作。在下面的例子中，首先将 5 赋值给变量 mynum。然后使用$mynum 将变量的值在表达式中使用。

```
set mynum 5
expr 10 * $mynum
```

如果需要将变量的值在字符串中使用，只需要在字符串中对变量进行计算即可，变量的值将成为字符串的一部分。在下面的例子中，变量 myname 的值

就作为字符串的一部分使用，该字符串为 "My name is Larisa"。

```
set myname "Larisa"
set mystr "My name is $myname"
```

Tcl 中存在一些特定的对变量进行操作的命令。append 命令可以连接字符串和变量，incr 命令可以对整数进行增值操作，而 unset 命令则可以取消对变量的定义，释放其所占的空间。表 25-3 中列出了这些操作。

25.2.5 数组

使用 set 命令可以对数组元素定义并赋值。下面的例子对 weather 数组中的第二个元素赋值为字符串"rain"。

```
set weather(2) rain
```

在对数组元素进行引用时，需要在数组名的前面加上$前缀。

```
puts $weather(2)
rain
```

Tcl 允许使用任何字符串作为数组的索引串，也就是说，它提供了对关联数组的支持。下面的例子可以在 city 数组中添加两个索引串为 Napa 和 Alameda 的元素。

```
set city(Napa) 34
set city(Alameda) 17
```

关联数组中的元素引用与一般数组相同，在数组名前加上前缀$，然后在数

组后的括号中写入索引串即可。

```
puts $city(Napa)
34
```

25.2.6 列表

Tcl 支持列表类型，这是一种在其它的编程语言中很少可以见到的类型。列表是使用括号包围起来的一组单字、字符串或数字。Tcl 中存在一套非常灵活的列表命令，使用这些命令可以对列表进行联接、分隔、添加或删除元素等操作(见表 25-2)。在引用列表时，可以简单地引用列表名，也可以列出括号和括号中的所有元素。

使用 set 命令同样可以将列表的值赋给变量，下面的例子将三个单字的列表赋给变量 weather。注意这里使用空格来分隔列表中的组件。

```
set weather {sun rain wind}
```

Tcl 提供了一系列的命令，通过这些命令可以对列表进行访问和操作。大部分这些命令都根据元素在列表中的位置对该元素进行访问。列表中的位置从 0 开始。因此，在上面的例子中，sun 的位置为 0，rain 为 1，wind 为 2。

lindex 命令根据元素的索引位置在列表中返回该元素，它使用列表和所需的索引位置作为参数。lreplace 命令将列表中指定位置的元素替换为新的值，linsert 命令的参数与 lreplace 相同：列表，索引位置和新元素。下面的例子将列表中的第二个元素(索引位置为 1)blue 替换为 green。这条语句将返回列表

(red green yellow)。

```
lreplace (red blue yellow) 1 green
(red green yellow)
```

也可以将列表赋值给变量，然后使用给变量名对列表进行操作。注意这里花括号和$前缀的使用方法。

```
set mycolors {red blue yellow}
lreplace $mycolors 1 green
puts $mycolors
```

也可以使用类似数组的方式对列表进行操作，如使用(1)可以引用第二个元素。

```
set mycolors(1) green
```

25.2.7 控制结构

Tcl中的控制结构与Perl、gawk、C-shell和C语言中的结构非常相似。Tcl中也存在重复作某事的循环结构和作条件选择的条件结构。表25-3中列出了这些控制结构的具体情况。Tcl中的控制结构使用Tcl命令块进行操作，命令块则由括号中的Tcl命令行组成。开括号必须与使用该命令块的语句放在同一行，语句块以闭括号结束，该闭括号单独位于一行。一个语句块在语法上与一条Tcl命令等价。控制语句将顺序执行语句块中的语句。

if控制结构允许程序选择不同的执行路线。if命令使用两个参数，一个测试

表达式和 Tcl 命令或命令块。表达式用来测试是否执行相应的命令，如果表达式为真，将执行该命令，否则将跳过该命令。下面给出了 if 结构的语法。可以使用 else 块指定表达式为假时执行的命令，使用 elseif 命令可以实现 if 语句的嵌套。

```
if {测试表达式}{
Tcl 命令
} elseif (测试表达式){
Tcl命令
} else {
Tcl命令
}
```

switch 结构允许在多个可能的执行路线中进行选择，选择的标准是将字符串的值和一些可能出现的模式进行比较。每一种模式都对应自己的一块 Tcl 命令语句。如果出现某种匹配情况，就可以执行对应的语句块。如果所有可能出现的模式都不能形成匹配，程序将执行 default 关键字后的语句。switch 结构以关键字 switch 开始，然后是前缀-和匹配时使用的选项，需要进行匹配的字符串，以及所有可能出现的模式和对应的动作。下面给出了 switch 结构的语法。

```
switch -选项 字符串 {
模式 {
Tcl 命令
}
```

模式 {
Tcl 命令
}
default {
Tcl 命令
}
}
选项指定了模式匹配使用的方法，下面给出了命令可以支持的选项：
-exact 在比较字符串和模式时使用精确匹配，缺省模式
-glob 使用全局模式进行匹配
-regexp 使用正则表达式模式进行匹配
--选项结束标识，--参数后的字符串为需要进行匹配的字符串(即使该字符串以-开始)
-regexp 选项可以使用正则表达式进行匹配，-glob 则可以使用与 shell 文件名搜索类似的匹配方式。在使用全局模式时，可以使用特殊字符 *、[]和?对字符串进行匹配。使用-regexp 选项时，可以对非常复杂的正则表达式进行匹配。

while 循环可以重复执行命令。在 Tcl 中，while 循环由 while 命令和两个参数组成，一个测试表达式和一个 Tcl 语句或语句块。Tcl 命令块由与 while 关键字位于同一行的开括号表示开始,单独位于一行的闭括号结束。下面给出了 while 循环结构的语法。

while (测试表达式){

Tcl 命令
}
for 循环
for 循环的功能与 while 循环相同，但格式略有差别。for 循环使用 4 个参数，前 3 个参数为表达式，最后一个参数为 Tcl 命令或语句块。三个表达式分别为初始表达式、测试表达式和增量表达式，每个表达式都使用花括号进行封装。
for {表达式 1} {表达式 2} {表达式 3}{
Tcl 命令;
}
foreach 循环是为顺序使用列表中的值设计的，它和 C-shell 中的 for-in 结构非常相似。foreach 结构使用三个参数：变量、列表和 Tcl 命令块。列表中的每一个值都顺序赋给指定的变量，然后参与循环中的命令块操作。当到达列表的结尾时，循环停止。和 while 循环相同，Tcl 命令块前后都有括号。foreach 循环的语法为：
foreach 变量 (值列表){
Tcl 命令
}

25.2.8Tcl 子程序： proc

在程序中使用 proc 命令可以定义自己的 Tcl 命令，即可以建立新的子程序，

这一点与其它编程语言中的子程序和子函数相似。proc 命令存在 3 个参数：新子程序的程序名，子程序使用的参数列表和需要执行的语句。程序名后所跟的参数列表前后都有花括号，表示使用子程序时所接收的参数。proc 命令的语法为：

```
proc 子程序名 {参数} {
Tcl 命令
}
```

下面的例子实现了 lsearch 命令，它的功能是在列表中搜索某个特定的模式。

```
proc mysearch {mylist pat} {
set i 0
set len [ length $mylist ]
while { $i < $len } {
if { [lindex $mylist $i] == $pat}
return $i
}
}
```

25.2.9Tcl 字符串命令：string

Tcl 中的 string 命令可以对字符串进行各种操作。根据完成的任务不同，它

有各种不同的选项形式。这些选项在 string 命令后输入，根据选项的不同完成字符串搜索、取子串或获取字符串长度等功能。命令使用的参数根据选项的不同也有变化。表 25-3 中列出了这些 Tcl 字符串命令。

在使用 string match 命令时，只能在字符串中完成基本的搜索操作，该命令不提供对正则表达式进行匹配的功能。但该命令提供了对*、?和[]特殊字符的支持。该命令使用两个参数，搜索使用的模式和需要进行搜索的字符串。

```
string match report* $fname
```

string range 命令返回字符串中指定的子串。该命令带有 3 个参数：字符串、开始和结束位置。结束位置上的 end 表示对字符串的剩下所有的部分进行拷贝。

```
string range 4 9
string range 3 end
```

string index 命令返回字符串中指定位置的单个字符。该命令使用 2 个参数：字符串和需要返回的字符位置。

```
string index $mystring 5
```

string length 命令返回字符串的长度。

```
string length $mystring
```

使用 regexp 命令可以完成正则表达式的搜索工作，regexp 命令和 egrep 命令类似，可以支持所有的正则表达式扩展属性。它使用 2 个参数：正则表达式模式和需要搜索的字符串。如果搜索到指定的模式，该命令返回 1，否则返回 0。下面的例子将在 mystring 变量中寻找数字。

```
regexp [0-9]+ $mystring
```

regsub 是 Tcl 中的替换命令。它和 regexp 类似，在字符串中进行使用正则表达式进行匹配，然后将匹配的部分替换为指定的字符串，将返回的值储存在一个新的变量中。regsub 使用 4 个参数，前面两个参数与 regexp 相同，分别为正则表达式模式和需要搜索的字符串。第 3 个参数为替代字符串，第 4 个参数为修改后的字符串存放的变量名。

25.2.10Tcl 输入、输出

Tcl 使用 gets 命令从标准输入设备或文件中读取输入数据，使用 puts 命令向标准输出设备或文件输出数据。下面的例子将从标准输入设备中(通常是键盘)读入一行，读入的内容将储存在变量 line 中。

```
gets line
```

puts 命令将向标准输出设备或文件输出数据，命令的参数为需要输出的字符串。

```
puts $line
```

gets 命令将读入的一行内容储存在指定的变量中，可以使用该变量对读入的内容进行处理。

例如，可以在 puts 命令中使用 line 显示输入的内容。

```
myread
#! /usr/bin/tclsh
gets line
```

```
puts "This is what I entered: $line"
$myread
larisa and aleina
This is what I entered: larisa and aleina
```

使用 print 命令可以向文件或标准输出设备输出数据，文件句柄名放在 print 命令和输出的数据之间。如果没有指定文件名，print 将输出到标准输出设备。

25.2.11Tcl 文件句柄

使用 open 命令可以为某个文件或管道建立句柄(表 25-3 中列出了可用的 Tcl 文件命令)。open 命令使用两个参数：文件名字符串和文件打开的模式，该命令将返回一个文件句柄，通过该句柄就可以对文件进行操作。文件名字符串可以是一个文件名，也可以是储存该文件名的一个字符串变量。文件模式是打开文件时的权限，r 表示只读，w 表示只写，a 表示为追加模式。如果需要对文件进行覆盖或更新，应在文件模式中使用＋。r＋表示可读写模式。open 命令的语法为：

```
open ( 文件名字符串， 文件模式 );
```

一般来说，应在 set 命令中使用 open 命令，这样可以直接将 open 命令返回的文件句柄赋值给某个变量。使用该变量就可以对打开的文件进行访问。下面的例子使用只读权限打开 reports 文件，然后将返回的文件句柄赋值给变量 myfile。

```
set myfile [open "reports" r]
```

文件名一般储存在某个变量中，这样就需要使用$前缀对该文件名进行引用。在下面的例子中，文件名"reports"储存在 filen 变量中。

```
set myfile [open $filen r]
```

在文件使用完毕后，应该使用 close 命令关闭该文件。close 命令的参数为需要关闭的文件使用的文件句柄。

```
close $myfile
```

gets 和 puts 命令可以使用文件句柄对特定的文件进行读写操作。这里的 gets 命令将使用 2 个参数：文件句柄和变量。该命令将从文件句柄指定的文件中读取一行，然后将读入的结果储存在指定的变量中。如果没有指定的文件句柄，gets 命令将从标准输入设备中读取数据。下面的命令将从 myfile 变量指定的文件中读取一行，读入的结果储存在 line 变量中。

```
gets $myfile line
```

puts 命令也使用两个参数：文件句柄和字符串。该命令将字符串的内容写入到文件句柄指定的文件中。如果没有指定的文件句柄，puts 命令将向标准输出设备输出数据。下面的命令将 line 变量中储存的字符串输出到 myfile 变量指定的文件。注意这里的 line 变量前有一个$前缀，但前面的 gets 命令中没有。这是因为 puts 对字符串进行操作，而 gets 的操作对象为一个变量。

```
puts $myfile $line
```

myreport

```
#! /bin/sh
```

```
# 下一行使用 tclsh
exec tclsh "$0" "$@"
set reps [open "reports" r]
while (gets $reps line)
{
puts $line;
}
close reps
```

25.3 Tk

Tk 应用程序对 Tcl 进行了扩展，它提供了一系列创建和管理图形对象的命令，如窗口、图标、按钮、文本框等。Tk 命令使用 X-Windows 系统创建图形对象，与直接使用 X11 工具箱相比，使用 Tk 命令编写 X-Windows 图形程序要容易得多。使用 Tk 可以很方便地为应用程序创建复杂的基于窗口的用户界面。

Tk 语言根据不同的图形对象(如窗口、按钮、菜单和滚动条等)进行分类。这些对象通常称为饰件。每一种饰件都有自己的一些命令来完成创建过程。例如，使用 button 命令可以创建一个按钮，使用 window 命令可以创建一个窗口。每一种类型的饰件即为一个类，创建该饰件的命令称为类命令。该命令将创建该类的一个实例，一个该类型的饰件。button 是创建按钮的类命令，表 25-4 中列出了 Tk 中可用的饰件。

25.3.1wish shell 和脚本

Tk 在 X-Windows 系统下运行，在系统中，Tk 使用自己的 wish shell 来运行 Tk 命令。在运行 Tk 程序时，需要首先启动 X-Windows 系统，然后使用 wish 命令启动 wish shell，这样会打开一个可以运行 Tk 命令的窗口。

在 wish shell 中可以交互式地运行 Tk 命令，即一步一步地输入命令并执行，也可以将这些命令存放在一个脚本文件中，然后执行该脚本文件。一般都使用后面的执行脚本程序的方法。

同 Tcl 脚本相同，Tk 脚本文件的后缀名也为.tcl。例如，可以在 Xterm 窗口中输入以下命令，读取名为 mydir.tcl 的 Tk 脚本文件并执行文件中的命令。

```
$ wish mydir.tcl
```

如果希望象命令那样，创建可独立执行的脚本，需要在脚本中调用 wish 命令。同 tclsh 相同，有两种方法可以达到该目标。第一种方法是显式给出 wish 命令的路径名，脚本程序的第一行为：

```
# ! /usr/bin/wish
```

第二种方法不需要显式指定 wish 所在的路径，这样就提高了该脚本程序的通用性。在下面的 mydir1 和 mydir2 脚本中都使用了这种不进行显式指明的方法。

```
#! /bin/sh
#下一行调用 wish
exec wish "$0" "$@"
```

在建立独立执行的脚本后，必须使用 chmod 命令修改文件的权限，使其能够直接执行。修改完成后，就可以简单地键入文件名来执行该程序了。

```
$chmod 755 mydir1
$ mydir1
```

25.3.2Tk 饰件

Tk 程序有许多类命令组成，使用这些类命令可以创建各种图形饰件。类命令的参数为该饰件的名字和该饰件的选项参数值。Tk 命令的格式与 Tcl 相同，首先是命令名，然后是命令使用的参数。但 Tk 命令比 Tcl 要复杂得多，因为图形界面需要大量的参数来对相应的饰件进行设定。例如，一个按钮就需要一个对象名、按钮上需要显示的标题和选取该按钮后需要进行的操作。许多 Tk 命令都将使用各种选项指定对应的饰件的特性，表 25-5 中列出了 Tk 饰件常用的一些选项。下面的例子使用 button 命令建立了一个按钮。button 命令的第一个参数为该按钮的对象名，然后给出了各个选项的值。-text 选项及其后的字符串指定了按钮上需要显示的标题，-command 选项及其后的命令则给出了按钮被单击后应执行的操作。该 button 命令将显示一个标题为"Click me"的按钮，单击该按钮将退出 Tk shell。

```
button .mybutton -text "Click me" -command exit
```

在创建工作界面时，为了一个特定的任务一般需要定义多个不同的饰件。其中一些饰件是为管理其它饰件而创建的，如滚动条就是为管理窗口而创建的。

其它的一些类似文本输入框的饰件可以与 Tcl 程序交互通讯，而菜单选项可以根据用户的选择运行不同的程序组件。

饰件的组织方式为分层结构。例如，为建立一个用来输入数据的窗口，需要创建一个图形框，然后在框中放置相应的文本框和按钮。饰件的对象名就反映了这种层次性，饰件的对象名即采用其上层饰件的对象名作为前缀。如果该图形框的对象名为 report，文本框的对象名则可能为 report.monday。在每一层中以点号作为表示层次的间隔。这样，在 report 框中名为 ok 的按钮，其对象名可能为 report.ok。

在建立饰件时，需要指定该饰件的几何尺寸。这些几何尺寸给定了饰件的大小和位置。Tk 中存在三种几何尺寸管理命令，pack, place 和 grid。在下面的例子中，都将使用 pack 命令。在创建某个饰件后，就应该使用几何尺寸命令决定饰件的大小和位置。只有在决定了大小位置后，该饰件才会在屏幕上显示出来。下面的例子使用 pack 命令决定.mybutton 饰件的几何尺寸。表 25-6 中列出了几种不同的几何命令。

```
pack .mybutton
```

mydir1 程序是一个简单的 Tcl/Tk 程序，该程序可以在 Tk 列表框饰件中显示文件和目录名，程序中还附加了一个滚动条饰件。图 25-1 给出了列表框的显示效果。在列表框中，可以显示一系列项的列表，通过滚动条可以在列表框中显示超出列表框设定大小范围的项。滚动条的建立过程非常简单，首先使用 scrollbar 命令对其进行定义，给定滚动条的对象名.scroll 和绑定的命令，".list yview"。该命令将滚动条附加到名为.list的列表框的垂直方向上。

scrollbar .scroll -command ".list yview"

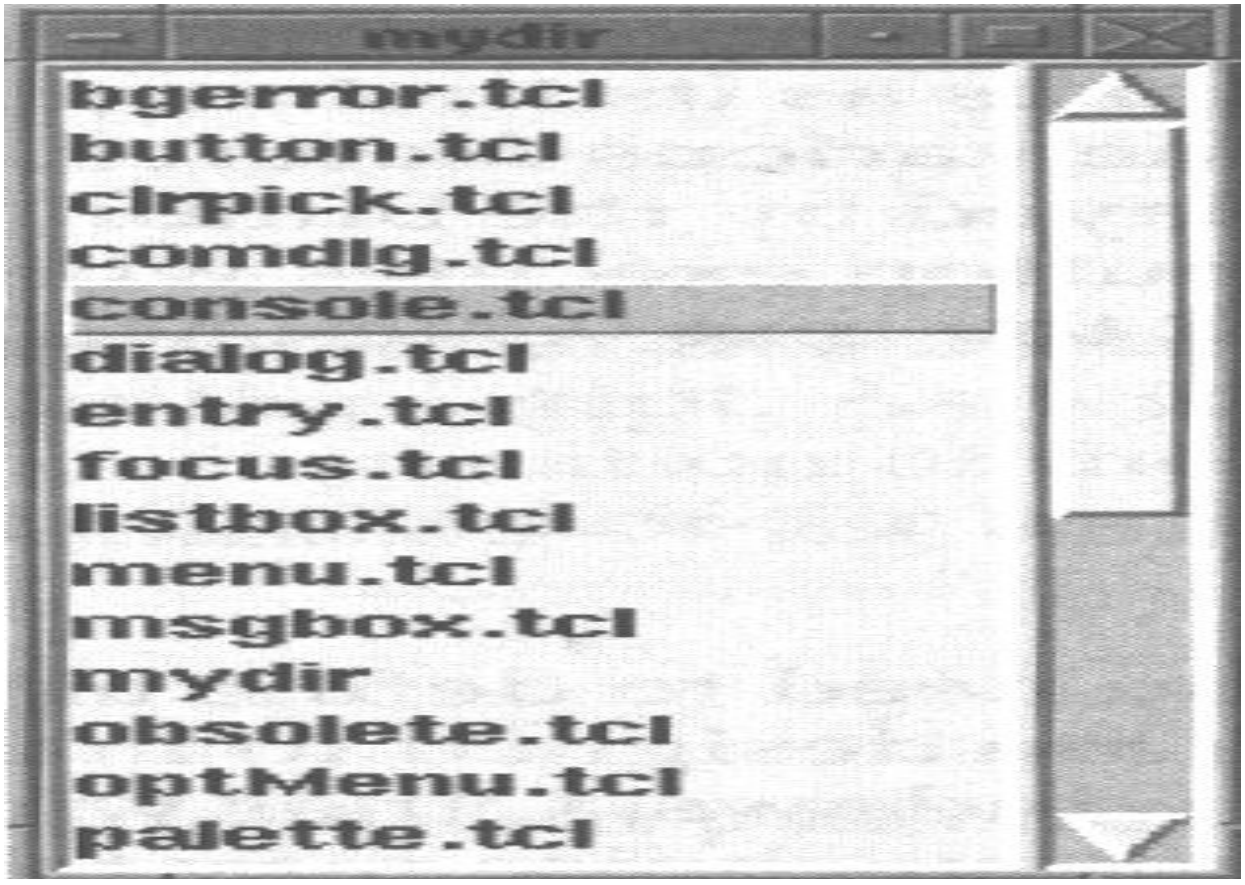


图 25-1 列表框和滚动条 mydir1

定义滚动条后，下一步是用 listbox **命令定义一个名为**.list **的列表框，该列表框垂直方向将附加一个名为**.scroll **的滚动条。**

listbox .list -yscroll ".scroll set" -relief sunken -width 15 -height 15 -setgrid yes

最后使用 pack 命令建立以上定义的两个饰件，并在窗口中进行显示。这两个饰件将放置在窗口的左边界并将扩展到整个窗口。首先将放置名为.list 的列表框，然后才是名为.scroll 的滚动条。

pack .list .sroll -side left -fill both -expand yes -anchor w

在界面显示后，使用一条 if 语句检查用户在调用该程序时是否输入了参数，argv 列表中将储存调用程序时的参数。如果没有使用参数进行调用，将使用句点符号对当前目录进行操作。

在将选中的目录赋值给 dir 变量后，使用 foreach 循环填满列表框。循环使用的列表采用 shell 的 ls 命令产生，为指定目录下的文件和目录列表。在循环体中将使用 Tk 的列表框 insert 操作将每一个文件名填进列表框.list。insert 目录使用位置和插入所需的值作为参数。这里，插入的值为$i 中储存的文件名，插入的位置则为 end 指定的列表末尾。

.list insert end $i

CTRL-C 控制字符将绑定到 exit 命令，这样在按下 CTRL-C 组合键后，就将关闭本程序窗口。

mydir1

#! /bin/sh

下一行启动 wish

exec wish "$0" "$@"

#创建滚动条和列表框

scroll bar .scroll -command ".list yview"

```
    listbox .list -yscroll ".scroll set" -relief sunken -width 15 -height 15 -setgrid
yes
    pack .list .sroll  -side left  -fill both  -expand  yes  -anchor w
    #如果用户输入了目录参数，将对该目录进行操作，否则将对当前目录进行
操作
    if ($argc > 0) then {
    set dir [lindex $argv 0]
    } else {
    set dir "."
    }
    在列表框中插入从 ls 命令获取的文件和目录列表
    cd $dir
    foreach i [exec ls -a] {
    if [file isfile $i] {
    .list insert end $i
    }
    }
    #设定文件管理器的绑定状况，CTRL-C 将关闭该窗口
    bind all <Control-c> {destroy.}
```

25.4 事件和绑定

Tk 程序采用事件驱动方式，在程序运行时，它将等待事件的发生，如鼠标事件或键盘事件等。鼠标事件可能是鼠标键的单击或双击，甚至鼠标键按下和松开也对应有相应的事件。键盘事件可能是 CONTROL 键、meta 键或输入数据结束的回车键被按下。当程序检测到一个特定的事件时，就会进行相应的动作。该动作可能是一个图形操作(如显示子菜单)，也可以是另外一个 Tcl、Perl 或 shell 程序。

使用 bind 命令可以显式绑定相应的事件和动作。bind 命令使用 3 个参数：饰件或类名、需要绑定的事件和绑定到该事件的动作。一旦在指定的饰件中发生指定的事件，就会执行相应的动作。

```
bind .myframe <CTRL-H> {.myframe delete insert}
```

可以使用 bind 命令将 Tk 饰件绑定到需要执行的 Tcl 命令。在这种情况下，整个 Tcl 程序将分为几段，每一段对应 Tk 饰件中的一个事件。当饰件中的事件发生时，就会去执行与之关联的 Tcl 命令。Tk 命令也可以与事件关联，这样就可以实现饰件及其事件的嵌套。一个 Tk 事件引发的 Tcl 命令可能会引发其它的 Tk 事件和 Tcl 命令(请参看表 25-6 Tk 命令列表)。图 25-2 中显示的 mydir2 程序说明了函数和饰件的绑定情况。该程序对前面的 mydir1 程序进行了增强，可以显式两个列表框，分别用来显示目录和文件。每一个列表框都有自己的滚动条。目录列表框名为.listdir，相应的滚动条为.scrolldir。程序首先建立两个对

话框和相应的两个滚动条，.listdir 和.scrolldir 首先建立，因此它们的位置位于窗口的左边。下一步是定义 Tcl 函数 listdirsfuncs，该函数的作用是填满列表框。该函数的参数为需要显示的目录名。程序将首先使用 cd 目录切换到指定的目录，然后使用 isdirectory 函数检查列出的文件是否为目录名，如果是，则将其放到.listdir 列表框中。在该函数定义后是对该函数的调用。在程序运行时应该首先调用一次 listdirfuncs 函数，用当前目录中的内容填满列表框。

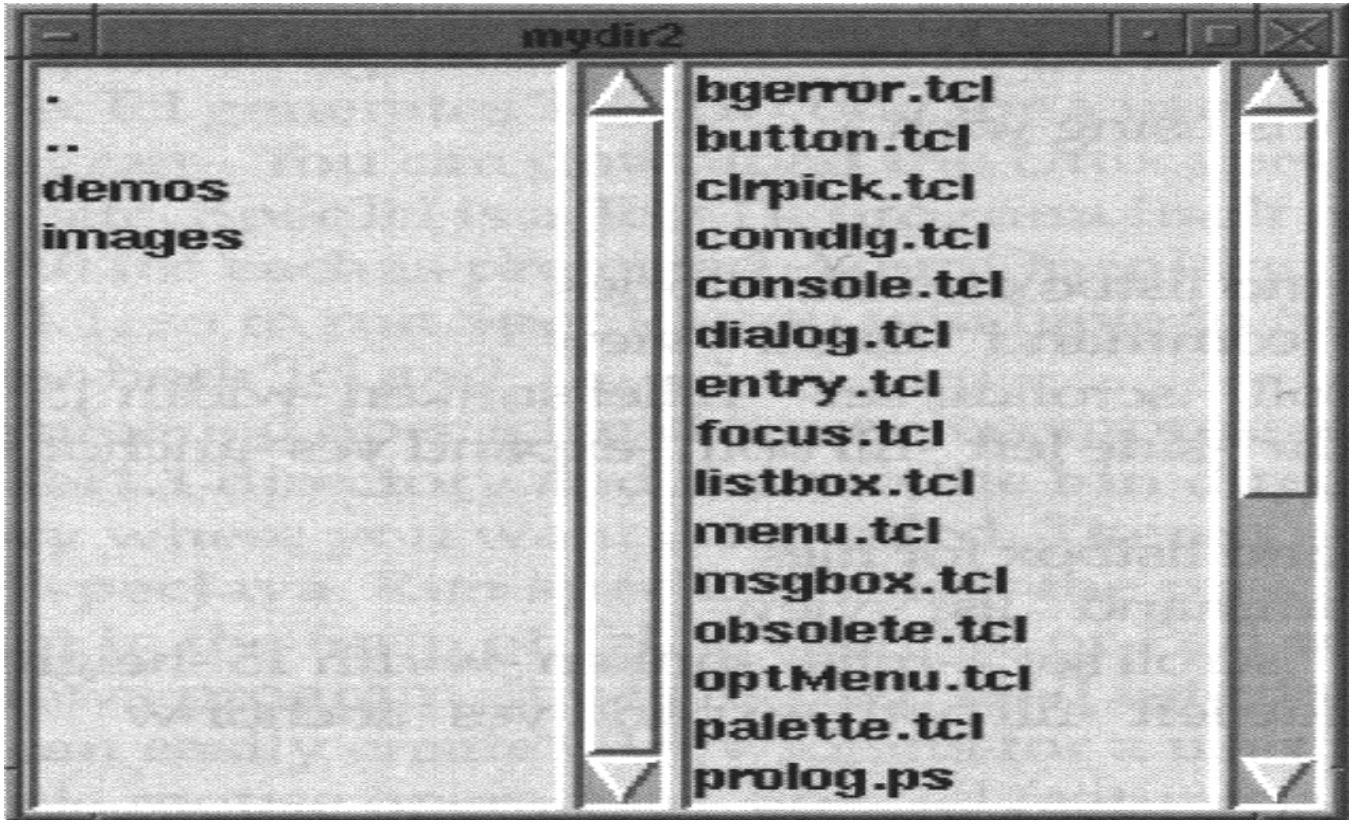


图 25-2 目录和文件列表框 mydir2

程序中存在.listdir 饰件的一个绑定。鼠标左键双击事件,Double-Button-1,绑定到了.listdir 列表框的 listdirsfuncs 函数。该绑定使得在.listdir 列表框中的任何鼠标左键双击事件都将调用一次 listdirsfuncs 函数。由于在.list 列表框中不存在相应的绑定,因此在该列表框中的双击事件将没有任何效果。

.listdir 列表框中的双击事件绑定了两个动作。这两个动作都放置在事件后的括号中。第一个动作是从列表框中获取选定的目录名。当用户在列表框中的某一项上单击时,将同时选定该项。[selection get]操作将返回选定的项,此时为选定的文件名。文件名将赋值给变量 i。然后执行第二个动作,使用该变量作为参数调用 listdirsfuncs 函数。这两个动作说明了如何在绑定中获取所需的值,如何使用这些参数进行函数调用。

绑定是 Tk 程序中最主要的功能组件。绑定将检测能够驱动 Tk 程序的事件。可以将 Tk 程序想象成一个无限的循环,在循环中程序不断监视特定事件的发生。一旦程序检测到这种事件,如鼠标单击或控制键按下,程序就会自动去运行与该事件绑定的动作。这些动作可以是任何 Tcl/Tk 命令或命令序列。通常,事件引发的动作将调用一些函数,可以完成非常复杂的操作。当动作完成后,程序又会回到无限的循环中,重复监视事件发生的工作。该循环只在使用 exit 或 destroy 命令时退出,通常这两个命令将绑定到 CTRL-C 组合键。从某种意义上来说,绑定就是程序的一种多重入口结构,从不同的入口将执行不同的操作。但绑定与传统的层次顺序结构的程序完全不同,每一个绑定事件都会引发它自己的命令序列,自己的程序。

在对 Tk 程序的控制流程进行跟踪时,也是从绑定开始,每一个绑定都会有

自己的控制流程。

```
mydir2
#! /bin/sh
#下一行启动 wish
exec  wish  "$0" "$@"
#创建目录所用的列表框和滚动条
scrollbar .scrolldir -command ".listdir yview"
listbox .listdir -yscroll ".scrolldir set" -relief sunken -width 15 -height 15 -setgrid yes
pack  .listdir .scrolldir -side left -fill both -expand yes -anchor w#创建文件所用的列表框和滚动条
scrollbar .scroll -command ".list yview"
listbox .list -yscroll ".scroll set" -relief sunken -width 15 -height 15 -setgrid yes
pack  .list .scroll -side left -fill both -expand yes -anchor w#检查调用时是否使用了参数
if $argc>0 {set dir [lindex $argv 0]} else {set dir "."}
#列出目录和文件的函数
#当前项首先被删除
proc listdirsfunc dir {
        .listdir delete 0 end
```

```
        .list delete 0 end
cd $dir
foreach i [exec ls -a ] {
        if [file isdirectory $i] {
                .listdir insert end $i
                } else {
                        .list insert end $i
                        }}
}
#初次显示目录和文件
listdirsfunc $dir
#设定绑定情况，对列表框中的鼠标双击双击进行绑定
bind all <Control-c>   {destroy.}
bind .listdir <Double-Button-1> {set i [selection get]
listdirsfunc $i}
```

25.5 SpecTcl

在建立 GUI 图形界面时，除了显式使用 Tk 代码进行编程外，还可以使用 SpecTcl GUI builder 来进行编程。SpecTcl 可以为 Tcl/Tk 应用程序简单地建立图形方式的用户界面。从网址 sunscipt.sun.com 可以下载 SpecTcl 的 Unix 版

本。SpecTcl本身也是一个Tcl/Tk应用程序，至少需要Tcl 7.6和Tk 4.7的支持，推荐使用Tcl和Tk的8.0版本。OpenLinux光盘中只有Tcl 7.4和Tk 4.1，因此在运行SpecTcl前需要首先下载、编译并安装Tcl和Tk的8.0版本。

在下载SpecTcl并将其展开后，更换到SpecTel1.1目录下的bin目录(当然也可以将其安装到任何指定的目录).在该目录中可以找到SpecTcl和SpecJava程序。键入SpecTcl以运行该程序，程序将询问用户需要将设计的结果输出为何种格式：Tcl、Java、http还是Perl格式。Linux中的SpecTcl是一个X-Windows程序，它拥有自己的图形界面，包括窗口、菜单和图标等饰件。使用该程序，可以很方便地使用菜单选项和鼠标操作建立复杂的图形界面。

SpecTcl提供了网格状的几何位置管理器，可以方便地建立跨平台的图形界面。图25-3显示了SpecTcl窗口。

Tcl/Tk网站包含SpecTcl的全部详细使用文档。创建一个饰件时只需要简单地单击调色板上的图标并将其拖到网格中即可。调色板即为窗口左边的两排图标，将鼠标停留在图标上时，在SpecTcl窗口的底部将显示出相应的操作信息。

对饰件进行属性设置时，只需要选中该饰件，并从工具条上选中适当的选项即可。工具条由窗口上方的图标组成。工具条的下拉菜单中包含了一些常用的选项,如字体或大小等。选中某个饰件,然后在Edit菜单中选择饰件properties选项，可以对该饰件的所有属性进行设置。单击Additional Properties框可以列出所有可以进行设置的属性。

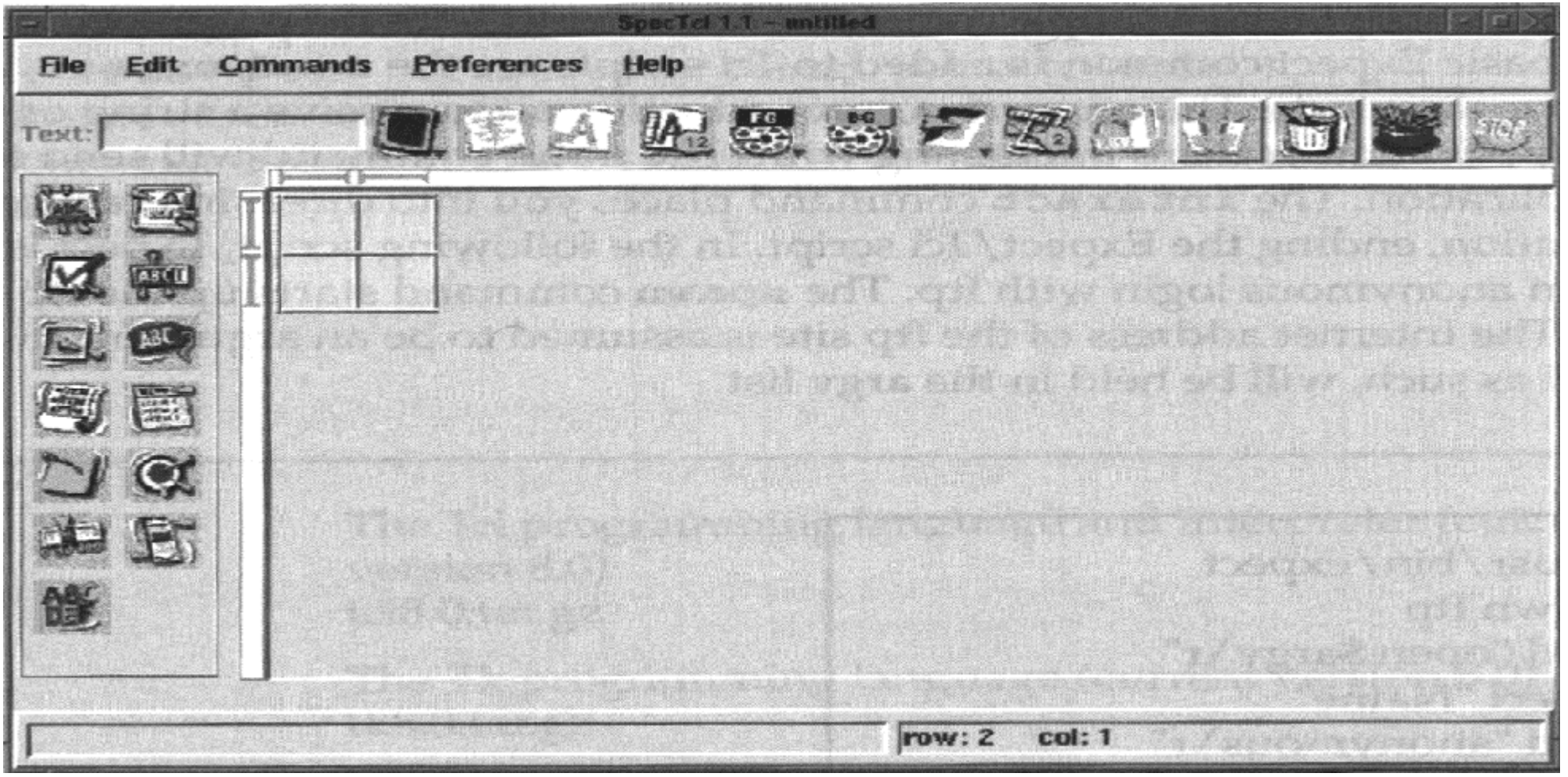


图 25-3SpectTcl,the Tcl/Tk GUI Builder

选择并拖动某个饰件，可以移动该饰件的位置。选择饰件后按 DEL 键可以删除该饰件，单击右上角的回收站图标也可以完成同样的功能。在界面建立后，可以单击右上角的测试按钮对其进行测试。

25.6 Expect

通过 Expect 提供的命令，程序可以自动地与任何需要交互式操作的 Unix 应用程序提供响应。例如，使用 Expect 命令可以建立能够自动进行 ftp 或 telnet 登录操作的程序。Expect 是为与任何需要交互式操作的程序相互通讯而设计的。它可以等待应用程序返回的消息，然后向应用程序发送脚本中指定的响应。这样，用户就可以执行一个简单的命令，来完成复杂的交互操作功能。

Expect 中存在 3 个基本的命令：send、expect 和 interact。expect 命令将等待从应用程序返回的字符串或值，send 命令则可以向应用程序发送指定的字符串。interat 命令结束 Expect/Tcl 脚本程序，使用户可以直接与应用程序进行交互式操作。在下面的脚本程序中，Expect 将用来完成一次 ftp 的自动匿名登录。spawn 命令启动 ftp 程序，ftp 站点的地址作为调用参数传递给 ftp 程序，该参数将储存在 argv 列表中。

myftp

```
#! /usr/bin/expect
spawn ftp
send "open $argv \r"
expect "Name"
send "anonymous \r"
expect "word:"
send richlp@turtle.trek.com \r
```

interact

在运行 Expect 命令前，需要首先进入 Expect shell。在前面的脚本程序中，使用#! /usr/bin/expect 命令调用 Expect shell。注意，在执行 myftp 程序前必须首先使用 chmod 755 myftp 命令赋给该脚本程序可执行权限。

$myftp ftp.caldera.com

expect 命令可以使用两个参数，一个期待接收到的模式串和收到该模式串后需要作的动作。expect 也可以使用一个模式/动作语句块作为相应的参数。在这种情况下，expect 命令可以对多个模式串进行匹配，然后根据接收到的消息执行相应的动作。例如，ftp 命令可能会返回一个"Connection refused"字符串，而不是"Name"字符串。在这种情况下，需要显示该信息并退出 Expect 脚本程序。如果针对接收到的消息需要执行多个操作时，可以将这些操作使用分号隔开，然后前后加上花括号封装起来。

另外一个有用的 Expect 命令是 timeout 命令。可以使用 timeout 命令设定超时所需的时间，然后用 expect 命令检查联接是否超时。设定时间时，应使用 set 命令对 timeout 变量进行赋值(单位为秒，默认值为 10 秒)。在 expect 块中使用 timeout 作为接收模式，就可以检测联接是否超时。下面是这样一个 expect 脚本的例子：

```
set timeout 20
send "open $argv"
expect {
timeout { puts "Connection timed out\n"; exit }
```

```
"Connection refused"   { puts   "Failed to connect\n"; exit }
"Unkown Host"      { puts "$argv is unknown\n"; exit }
"Name"
}
```

表 25-1Tcl/Tk **软件产品**

TclTcl	**编程语言及解释器**(**最新版本** 8.0)
	tcl8.0.tar.gz
Tk	Tk **编程语言**(**最新版本** 8.0)
	tk8.0.tar.gz
SpecTcl	Tcl/Tk GUI builder，**建立图形方式的用户界面**
	SpecTcl1.1.tar.gz
Tcl Plugin	Tcl/Tk **网页插件**，(**最新版本** 2.0);
	Tcl/Tk **应用程序**(Tclets)
	查看器 tclplugin1.1.386.rpm
TclHttpd	Tcl/Tk **网络服务器**
	tclhttpd.tar.Z
sunscript.sun.com	Tcl/Tk **网址**

表 25-2Tcl 列表操作

命令	操作
set 列表 值	建立列表并赋值
lsearch(列表，模式)	在列表元素中搜索指定模式
lindex(列表，索引)	返回列表中指定索引位置的值
llength(列表)	返回列表中元素的个数
linsert(列表，索引，值列表)	在列表中指定索引位置后插入指定的值
lreplace(列表，索引，值列表)	将列表中指定索引的值替换为新值
lappend(列表，值列表)	在列表的末尾添加新值
concat(列表)	将多个列表的值联接到一个列表中
list(列表)	将多个列表组合为一个列表，该列表的各个元素的值即为每个列表
split(字符串，分隔符)	使用指定的分隔符将字符串进行分隔，然后将分隔后的结果储存在列表中
join(列表)	将列表中的元素组合为字符串
lsort(列表)	对列表按字符或数字顺序进行排序

表 25-3 常用的 Tcl 命令

赋值和变量	说明
set	对变量进行赋值
global	声明全局变量
incr	对某个变量增加指定的整数值
unset	删除变量
upvar	在变量的声明范围外引用该变量
variable	声明变量名
array	数组引用
expr	数学表达式
控制结构和子程序	
if	条件命令，可以使用 else 和 elseif 语句块
switch	switch 选择结构
while	while 循环
for	for 循环
foreach	为列表操作设计的循环
break	强行退出循环
continue	跳过循环块中的语句，直接开始下一次迭代
proc	定义 Tcl 子程序
return	从子程序中返回值
source	读入并执行另外一个文件中的 Tcl 命令

续表

uplevel	执行上一层的命令
文件	
file	获取文件信息
open	打开文件
close	关闭文件
eof	检查是否到文件末尾
fcopy	拷贝文件
flush	刷新文件缓冲区
glob	使用全局匹配字符匹配文件名
read	从文件中读取字符块
seek	确定文件指针的位置
tell	返回当前文件指针的位置
socket	打开一个 TCP/IP 网络联接
输入/输出	
format	使用指定的格式对字符串进行格式处理，与 C 语言中的 sprintf 类似
scan	使用指定的格式读入数据并将其转化为字符串元素，与 C 语言中的 scanf 类似
gets	读入一行
puts	输出一行

续表

字符串	
binary	在字符串和二进制数据中进行转换
regexp	使用正则表达式对字符串进行模式匹配
regsub	使用正则表达式对字符串进行替换操作
split	将字符串分隔为列表元素赋值和变量说明
string 对字符串进行操作	subst 对字符串进行替换操作
系统	
catch	跟踪错误
cd	更改工作目录
clock	返回当前时间和日期
error	产生错误
eval	执行命令
exec	执行 UNIX 命令
exit	结束程序并退出
pwd	返回当前工作目录
info	查看当前 Tcl 解释器的信息
trace	检查变量的值

表 25-4 常用的 Tk 命令选项

选项	描述
-anchor	饰件上的信息如何显示，必须为下列值之一：n,ne,e,se,s,sw,w,nw,center
-background	显示饰件时的背景颜色
-bitmap	饰件中显示的位图
-borderwidth	饰件边缘三维边界的宽度
-cursor	饰件中使用的鼠标光标形状
-font	在饰件中显示文本时使用的字体
-foreground	显示饰件时的前景颜色
-geometry	饰件窗口的几何尺寸
-highlightbackground	当饰件中没有输入焦点时的背景颜色
-highlightcolor	当饰件中存在输入焦点时的高亮度颜色
-image	饰件中显示的图形
-insertbackground	插入光标覆盖处的背景颜色
-justify	多行文本的对齐方式
-orient	饰件放置的位置
-padx	饰件的 X 方向上请求分配的多余空间
-pady	饰件的 Y 方向上请求分配的多余空间
-relief	饰件使用的 3 维效果
-selectbackground	显示被选择的项时使用的背景颜色

续表

-text	饰件内显示的文本
-textvariable	变量名
-troughcolor	饰件中滚动条等矩形对象使用的颜色
-underline	饰件中需要使用下划线的字符位置
-xscrollcommand	与水平滚动条进行通讯的命令
-yscrollcommand	与垂直滚动条进行通讯的命令按钮选项
-command	与按钮绑定的 Tcl 命令
-selectimage	复选框被选择时显示的图形
-height	按钮的高度
-selectcolor	按钮被选择时的背景颜色
-state	指定单选框的状态：一般、选中或无效
-variable	表明按钮是否被选中的全局变量

表 25-5 标准的 Tk 饰件

饰件	描述
button	按钮
canvas	窗口
checkbutton	复选框
entry	输入框
frame	框架，主要任务是在复杂的窗口设计中起分隔美观作用
image	图形
label	标签
listbox	列表框
menu	菜单
menubutton	菜单按钮，可以对菜单进行访问
message	消息
radiobutton	单选框
scrollbar	滚动条
text	可编辑的文本框
scale	标尺

表 25-6Tk 操作

选项	描述
事件操作	描述
bind	绑定 X 事件与 Tcl 脚本
bindtags	绑定命令和标记
selection	使用鼠标选取饰件或文本
几何尺寸管理命令	
pack	将饰件紧密放置
grid	将饰件按网格行列放置
place	将饰件放置在框架指定位置上
窗口操作	destroy
关闭 Tk 窗口	toplevel
选择最顶端的窗口	wm
设置窗口特性	uplevel
将窗口上移一个层次	

第 26 章 gawk

gawk filter 和其他的一些 filter 的不同之处在于它并没有预先定义需要执行的任务。本书中前面提到的 filter 程序都是为某种特定的任务而设计的，如搜索文本或对行进行排序等。但 gawk 没有事先定义的任务，而是由用户来定义 gawk 需要完成的任务。gawk 拥有自己的一套编程语言、命令和操作，用户可以根据需要完成的任务自己编写相应的 gawk 程序。从这种意义上来说，gawk 实际上是一种可编程的 filter。用户可以自己在程序中对数据进行相应的处理。使用 gawk 可以象 cat 命令一样简单地显示文件中的文本，也可以象 grep 命令一样完成文本搜索工作，甚至可以完成 wc 命令的字数统计工作。在这些程序中，用户都可以按照自己的需要加入特殊的功能，如只显示一行的某一部分，在某一特定的域中搜索模式串，或只统计大写的单字等。

gawk 可以直接执行，也可以放置在 shell 文件中然后执行。此时 shell 文件的文件名可以认为是用户自己创建的新 filter。因此，通过使用 gawk 可以定义自己的 filter。从这种意义上来说，gawk 就有了双重含义。首先，gawk 本身是一个可以直接在命令行上调用的 filter，这一点与其他的 filter 没有差别。另外，gawk 又是一个可编程的 filter，通过它用户可以定义自己的 filter。本章中将对 gawk 的这两方面都进行描述。本章的前面部分将对作为普通 filter 的 gawk 的

特点进行描述。然后，本章还会对如何使用 gawk 定义自己的 filter 的方法进行探讨。

gawk 应用程序拥有编程语言所具有的全部灵活性和复杂性。gawk 中拥有自己的一套运算符可以用来完成决策和计算工作。在 gawk 中也可以申明变量并在控制结构中使用这些变量来控制程序执行的顺序。gawk 中的许多编程特性都来自 C 语言，相应的语法也完全一致。从所有的这些特性可以看出，gawk 确实是一种非常强大的编程工具。

gawk 是 Unix 的 awk 应用程序的 GNU 版本。awk 起先是作为 UNIX 操作系统的标准引用程序设计的。它的作者之一 Brian kernighan，也是 UNIX 操作系统的开发者。随后，又推出了 awk 的增强版本，nawk，该版本支持对文件的操作，可以在同一个程序中访问多个文件。gawk 则在 awk 和 nawk 的基础上又有增强，它支持 awk 和 nawk 的全部功能。

26.1 gawk 命令

gawk 命令使用 gawk 指令和文件名列表作为参数。gawk 指令用单引号封装作为一个参数读入。gawk 指令本身则由两段组成，模式段和动作段。动作段用前后的括号表示。模式段则可以考虑为一个条件，它可以是一个模式搜索操作或编程语言中定义的测试条件。gawk 应用程序中存在一整套操作符可以构成非常复杂的条件。可以将模式搜索操作看成是检索记录的一种条件。这样，与

grep 命令中简单地进行模式匹配操作不同，用户可以指定一个非常复杂地条件，符合这个条件的记录就可以被检索出来，然后采用动作段中指定的动作对记录进行相应的操作。下面的例子给出了 gawk 指令的语法，可以将其看做 条件 {动作}。

模式 {动作}

gawk 可以对文件或标准输入设备进行操作。对文件进行操作时，应在命令行上的指令后列出文件名，如果没有列出文件名，gawk 将从标准输入设备读取输入。下面的例子给出了 gawk 命令的完整结构。调用时在命令行上输入 gawk 关键字和相应的 gawk 指令即可，与 sed 命令类似，为避免 shell 对 gawk 指令进行解释操作，需要用单引号将指令封装起来。由于条件和动作都是 gawk 指令的一部分，因此它们应该使用同一对引号。下面的例子给出了 gawk 命令的完整语法(参见表 26-1)。

$ gawk '模式 {动作}' 文件名

gawk 执行时使用指令中的模式串引用文件中符合条件的行，然后使用动作段中的动作对这些行进行处理。不过，动作和模式都有自己缺省的值，用户在使用这些缺省的值时不必显式地进行指明。默认的动作为打印输出，如果没有指定动作，将对选定的行进行打印输出。默认的模式为选择文本中的每一行，因此，如果没有指明模式，gawk 将对文本中的所有行进行操作。下面的两个例子都可以打印输出含有模式串"Penguin"的所有行。这里的模式段是一个模式搜索操作。模式搜索操作由两个斜杠中的模式表示。所有符合这个模式的文本行都将被检索出来。在第一个例子的动作段中，指明了执行的动作为 print 命令。

print 命令将在标准输出设备上输出选定的文本行。而在第二个例子中没有指定动作段，因此 gawk 程序将执行默认的动作，即 print 命令。

```
books
TempestShakespeare15.
75Penguin
ChristmasDickens3.50Academic
IliadHomer10.25Random
RavenPoe2.50Penguin
$ gawk '/Penguin/{print}' books
TempestShakespeare15.75Penguin
Raven        Poe                    2.50Penguin
$gawk '/Penguin/' books
Tempest          Shakespeare      15.75Penguin
RavenPoe 2.50 Penguin
```

26.2 模式搜索和特殊字符

和 sed 应用程序相同，gawk 也可以使用包含特殊字符的模式搜索检索文本中的各行。模式串由开始和结尾的两个斜杠组成，出现在 gawk 指令的模式段。

/模式/ {动作}

模式搜索操作将对文件中的所有行进行。如果在某行上找到指定的模式，gawk 将会对该行进行指定的操作。从这种角度上看，gawk 与编辑程序的操作很相似。与 sed 相似，gawk 中的行也被看作是一行文本，匹配操作将对整行进行。在下面的例子中，gawk 将查找所有包含模式'Poe'的行，然后输出该行。

```
$ gawk '/Poe/{print}' books
RavenPoe2.50Penguin
```

在 gawk 中可以使用 sed 和 Ed 编辑器的正则表达式中可用的所有特殊字符。下面的第一个例子在每一行的开始处进行模式匹配，特殊字符^表示行的开始。第二个例子则在每一行的末尾处进行模式搜索。

```
$ gawk '/^Christmas/{print}' books
ChristmasDickens                3.50            Academic
$gawk 'Random$/{print}' books
Iliad               Homer               10.25       Random
```

与 sed 和 Ed 相同，可以在模式中使用特殊字符指定模式串的变化。句点(.)匹配任何字符，星号(*)匹配任意重复的字符，中括号([])则对指定的字符集进行匹配。在下面的例子中，句点用来匹配后面跟有字符串"en"的任意字符。

```
$ gawk '/.en/{print}' books
ChristmasDickens                3.50    Academic
Raven           Poe             2.50    Penguin
```

下面的例子则使用括号和星号特殊字符指定一个数字序列。[0-9]表示任意的数字字符，这里的星号表示对任意重复的数字进行匹配。该模式可以匹配任

何以.50 结尾的数字串。注意这里的句点前有一个反斜杠，表示这里的句点为点号字符，而不是相应的特殊字符。

```
$ gawk '/[0-9]*\.50/ {print}' books
Christmas    Dickens          3.50          Academic
Raven Poe                     2.50          Penguin
```

在 gawk 中还可以使用扩展的特殊字符：＋、?和|。＋和？是特殊字符星号*的变化形式，＋可以匹配字符后重复出现的一个或多个相同的字符，？则用来匹配字符后不出现的或重复出现的一个相同的字符。|则表示可以在符号两侧的模式进行选择。在下面的例子中，可以对包含有模式串"penguin"或"Academic"的行进行匹配操作。

26.2.1 变量、常量和模式

gawk 提供了定义变量的功能。在 gw 中存在三种变量：域变量、特殊 gawk 变量和用户定义的变量。gawk 自动定义前面两种变量，第三种变量是用户自己定义的。在 gawk 中还可以定义算术和字符串常量。算术常量由数字构成，而任何一对双引号之间的字符都可以看作是字符串常量。

域变量用来引用一行中的域。域是由域分隔符分隔开的字符集，默认的分隔符为空格或制表符。同其他数据库 filter 相同，gawk 的域从 1 开始计数。gawk 为文件中的每一个域都会定义一个域变量。域变量由美圆符号和符号后的数字组成，$2 表示第二个域。变量$0 是一个特殊的域变量，它包含了一行中所有

的内容。域变量可以用在 gawk 指令的模式段或动作段中。如果同时列出多个变量，变量间用逗号加以分隔。注意这里美圆符号$的用法和 shell 程序中的用法并不一致。下面的例子将打印 books 文件的第二个和第四个域。$2 和$4 分别引用这两个域。

```
books
Tempest    Shakespeare    15.75    Penguin
Christmas    Dickens      3.50     Academic
Iliad          Homer      10.25    Random
Raven          Poe         2.50    Penguin
$ gawk '{print $2 $4}' books
Shakespeare          Penguin
Dickens               Academic
Homer                Random
Poe                   Penguin
```

在下面的例子中，用户将输出包含有模式"Dickens"的行两次，第一次逆向输出各域，第二次则采用正向输出。这里使用$0 变量代表整行的内容。

```
$ gawk '/Dickens/ {print $4, $3, $2, $1; print $0}' books
Academic          3.50          Dickens          Christmas
Christmas         Dickens       3.50             Academic
```

gawk 中还定义了一套特殊变量用来提供当前处理文档的信息。NR 变量中保存了当前行的行号，NF 变量的值则为当前行中域的个数。还有一些储存域和

记录分隔符的其他变量。gawk 中甚至还存在一个名为 FILENAME 的变量，该变量储存了当前正在处理的文件名。表 26-1 中列出了 gawk 中的这些特殊变量。

特殊变量和用户自己定义的变量在引用时都不用在变量名前加上$符号。下面的例子使用特殊变量 NR 和域变量$2、$4 显示文件中各行的行号和行中的第 2 和第 4 个域。NR 变量中储存了正在处理的行的行号。

```
$ gawk '{print NR, $2, $4}' books
1        Shakespeare      Penguin
2         Dickens         Academic
3         Homer           Random
4         Poe             Penguin
```

用户可以定义自己的变量，可以使用任意字母或数字以及下划线组成的变量名。变量名必须以字母开头。在第一次使用变量时，gawk 会自动对其进行定义。变量的类型不需事先声明，而有变量使用的方式来决定。如果使用变量储存数字，变量则为数字类型；如果使用变量储存字符，则变量为字符串类型。在使用变量的过程中必须保持变量的一致性。不能使用字符串变量来进行算术操作，反之亦然。

使用赋值操作符，=，可以对变量进行赋值操作。赋值运算符的左边为变量，右边为需要给变量赋的值。值可以是某个变量的值，也可以是一个常量。下面的例子将第二个域变量的内容赋值给变量 myfield。

```
$ gawk '{myfield = $2; print myfield}' books
Shakespeare
```

```
Dickens
Homer
Poe
```

26.2.2 模式搜索操作符

使用特殊操作符~和!~可以在域中进行模式搜索工作。与使用等于操作符比较字符串和整个域不同，使用~操作符可以在查看域中是否存在指定的模式。在使用匹配操作符时，操作符的左边是被搜索的域，右边是搜索使用的模式。下面的例子将在第一个域中搜索模式"mas"。

```
$ gawk '{$1 ~ /mas/} {print}' books
Christmas       Dickens         3.50       Academic
```

下面的例子则将显示第一个域中不包含模式"mas"的所有记录：

```
$ gawk '($1 !~ /mas/) {print}' books
Tempest         Shakespeare     15.75    Penguin
Iliad           Homer           10.25    Random
Raven           Poe             2.50     Penguin
```

和一般的模式搜索操作相同，在对域进行搜索时也可以使用特殊字符。下面的例子将对第四个域中"Penguin"的大小写版本进行匹配。特殊字符[]用来定义一个大写字母 P 和小写字母 p 的集合。

```
$ gawk '($4 ~ /[Pp]enguin/) {print}' books
Tempest          Shakespeare          15.75       Penguin
Raven             Poe                  2.50       Penguin
```

26.2.3BEGIN 和 END 模式

gawk 中存在两个 BEGIN 和 END 这两个特殊的模式，允许用户指定处理开始前和对输入行处理完毕后的相应动作。在下面的例子中，在对文本行进行处理前将输出表头，"Book List"，在处理完成后，则输出变量 NR 的值。在进行处理时，每处理一行，NR 的值就加 1。因此在处理完成后，NR 变量中将储存文件中的记录总个数。

```
$ gawk 'BEGIN {print "Book List"} {print} END {print "Total record is ", NR}' books
Book List
Tempest      Shakespeare        15.75          Penguin
Christmas    Dickens             3.50           Academic
Iliad         Homer              10.25          Random
Raven         Poe                 2.50          Penguin
Total records is 4
```

26.3 gawk 指令文件

由于 gawk 指令变得越来越复杂，因此比较好的方法是将这些指令存放在一个 gawk 指令文件中，然后使用 gawk 读出该文件并执行。如果需要改变执行的操作，只需要对该文件进行修改即可。-f 选项表示从 gawk 指令文件而不是命令行上读入 gawk 指令。在下面的例子中，列出 Penguin 出版的书的指令存放在名为 findbk的 gawk指令文件中。执行这些指令时需要在命令行上输入 gawk 命令，并带上-f选项和 findbk文件名。

```
findbk
BEGIN {print "Book List";
count = 1;
}
/$4 ~ "Penguin"/ {
count = count = 1;
print;
}
END {
print "Total records found is ", count
 }
$ gawk -f findbk books
```

```
Book List
TempestShakespeare15.75Penguin
RavenPoe2.50Penguin
Total records found is 2
```

注意在这里不需要在指令前后加上单引号。在指令文件中可以将指令的不同部分放在不同的行上，增强指令的可读性，这一点对控制结构的使用非常有用。

26.3.1 控制结构

gawk 与编程语言非常类似，可以在 gawk 中定义变量、创建表达式或进行赋值操作。在 gawk 中还包含了一套控制结构可以完成迭代和选择功能。在 gawk 中存在 3 种循环控制结构: while, for 和 for-in 循环。另外，还存在一种选择控制结构:if 选择语句。表 26-1 中列出了 gawk 中的控制结构。除了 for-in 循环外，这些控制结构和 C 语言中的相应结构非常类似。在 gawk 中也存在一些用于读入和输出数据的命令。表 26-1 中列出了这些命令。

26.4 作为用户定义 Filter 的 gawk

将整个 gawk 指令放在一个脚本文件中,就可以使用 gawk 定义自己的 filter。

可以将改脚本文件设为可执行文件，这样该文件的文件名就成为了一个新的 LINUX 命令。在这个脚本文件中，必须在指令的前后加上单引号。脚本文件中的内容必须能够在 shell 上直接执行，因此在书写脚本文件时必须严格按照在命令行上书写 gawk 指令的格式来进行书写。在脚本文件中，可以将指令的模式段和动作段写在不同的行上，但所有的回车换行符必须都包含在指令前后的两个单引号之间。也就是说，脚本文件的第一行必须是关键字 gawk 和一个单引号，然后可以任意在不同的行上书写相应的模式和动作。但文件的末尾必须也是一个单引号，然后在同一行上输入所需的文件名参数。这种格式的语法为：

gawk '
模式 {
gawk 动作;
}' 文件名

在下面的例子中，用户将整个 gawk 指令存放在一个名为 field3 的脚本文件中。该指令将输出每一行的前三个域。注意，这里的 gawk 指令前后都存在单引号。在使用 chmod 命令给改文件赋予可执行属性后，就可以简单地输入脚本文件名来执行该文件。

field3
gawk '{
for (i = 1; (i <=3); i＋＋)
{
printf("%s\t", $i);

```
}
printf("\n");
} 'books'
$ chmod 755 field3
$ field3
TempestShakespeare15.75
ChristmasDickens3.50
IliadHomer10.25
RavenPoe2.50
```

表 26-1gawk 选项、操作符、变量和控制结构

组件	描述
-f 文件名	从指定文件中读入 gawk 指令
-Fc	指定输入文件的域分隔符。默认的分隔符为制表符或空格$ gawk -F: books
输出操作	
print	在标准输出设备上输出当前行的内容
print 变量名	在标准输出设备上输出变量
print 变量名 >> 文件名	将变量的值输出到文件
输入操作	
getline 变量名	读入下一行输入。如果已经到达文件末尾，返回 0。如果指定了变量，将读入的值赋给该变量
getline 变量名 << 文件名	从指定文件中读入下一行输入。如果已经到达文件末尾，返回 0。如果指定了变量，将读入的值赋给该变量
控制结构	
{ 动作 }	语句块由语句组和前后的花括号组成
if (表达式) 动作 1	
else 动作 2	如果表达式为真，if 控制结构将执行语句 1。如果表达式为假，将执行语句 2

续表

while (表达式)动作	如果表达式为真，while 循环将反复执行循环中的指定的动作
for (表达式 1；表达式 2；表达式 3)动作	如果表达式 2 为真，for 循环将反复执行循环中的指定的动作。表达式 1 将在循环开始前执行，表达式 3 则在每次迭代过程结束后执行
for (变量 in 数组名)	动作 for-in 控制结构是为关联数组设计的。数组中的索引串将将依次赋值给指定变量
next	next 命令停止对当前记录的操作，直接跳到下一个记录
exit	exit 命令停止所有对记录的操作，直接执行 END 命令(如果有)
操作符	
>	大于
<	小于
>=	大于或等于
<=	小于或等于
==	等于
!=	不等于
&&	逻辑与
||	逻辑或

续表

!	逻辑非
字符串 ~ 正则表达式	在字符串中搜索指定的正则表达式
字符串 !~ 正则表达式	查看在字符串中是否不存在指定的正则表达式
初始化和中止模式	
BEGIN	在 gawk 开始处理前执行的操作
END	在 gawk 结束处理后执行的操作
变量	
NR	当前行的记录号
NF	当前记录中域的个数
$0	整个当前记录
$n	当前记录中的域，从 1 开始计数，如 $1
FS	输入的域分隔符，默认的分隔符为空格或制表符