



LINUX命令行大全

THE LINUX COMMAND LINE

[美] William E. Shotts, Jr. 著 郭光伟 郝记生 译

人民邮电出版社
北京

图书在版编目 (C I P) 数据

Linux命令行大全 / (美) 绍茨 (Shotts, W. E.) 著 ;
郭光伟, 郝记生译. — 北京 : 人民邮电出版社, 2013.3
ISBN 978-7-115-30745-3

I. ①L… II. ①绍… ②郭… ③郝… III. ①
Linux操作系统—程序设计 IV. ①TP316.89

中国版本图书馆CIP数据核字(2013)第003381号

版权声明

Copyright © 2012 by William E. Shotts, Jr. Title of English-language original: The Linux Command Line: A Complete Introduction, ISBN 978-1-59327-389-7, published by No Starch Press. Simplified Chinese-language edition copyright © 2012 by Posts and Telecom Press. All rights reserved.

本书中文简体字版由美国 No Starch 出版社授权人民邮电出版社出版。未经出版者书面许可，对本书任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

Linux 命令行大全

-
- ◆ 著 [美] William E. Shotts, Jr
 - 译 郭光伟 郝记生
 - 责任编辑 傅道坤
 - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京昌平百善印刷厂印刷
 - ◆ 开本：800×1000 1/16
印张：28.25
字数：677 千字 2013 年 3 月第 1 版
印数：1~3 000 册 2013 年 3 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-2972 号

ISBN 978-7-115-30745-3

定价：69.00 元

读者服务热线：(010) 67132692 印装质量热线：(010) 67129223
反盗版热线：(010) 67171154

内 容 提 要

本书主要介绍 Linux 命令行的使用，循序渐进，深入浅出，引导读者全面掌握命令行的使用方法。

本书分为四部分。第一部分开始了对命令行基本语言的学习之旅，包括命令结构、文件系统的导引、命令行的编辑以及关于命令的帮助系统和使用手册。第二部分主要讲述配置文件的编辑，用于计算机操作的命令行控制。第三部分讲述了从命令行开始执行的常规任务。类 UNIX 操作系统，比如 Linux，包含了很多“经典的”命令行程序，这些程序可以高效地对数据进行操作。第四部分介绍了 shell 编程，这是一个公认的初级技术，并且容易学习，它可以使很多常见的系统任务自动运行。通过学习 shell 编程，读者也可以熟悉其他编程语言的使用。

本书适合从其他平台过渡到 Linux 的新用户和初级 Linux 服务器管理员阅读。没有任何 Linux 基础和 Linux 编程经验的读者，也可以通过本书掌握 Linux 命令行的使用方法。

致 谢

我要感谢以下朋友帮助促成此书。

首先要感谢的是启发我的人：Wiley 出版社的组稿编辑 Jenney Watson，是她最初建议我写一本与 shell 脚本相关的书。尽管 Wiley 出版社最终没有接受我的书稿，但它成为本书的基础。John C. Dvorak，一位著名的专栏作家和权威人士，给了我重要的建议。在他的视频播客“Cranky Geeks”的一个片段中，他描述了这样一个写作过程：“哇喔！每天写 200 字，一年之后，你将拥有一部小说”。这个建议促使我每天写一页，直到完成本书。Dmitri Popov 在 *Free Software Magazine* 写了一篇名为 *Creating a book template with Writer* 的文章，它启发我使用 OpenOffice.org Writer 来排版文字，事实证明，结果还不错。

其次感谢那些帮助我制作本书最初的、可自由分发的电子版本（可在 LinuxCommand.org 找到）的志愿者：Mark Polesky 非常出色地审阅和检验了本书的文字；Jesse Becker、Tomasz Chrzczonek、Michael Levin 和 Spence Miner 也审阅和复审了部分文字；Karen M. Shotts 花了很多时间编辑我的书稿。

还要感谢在 No Starch 出版社长时间辛苦工作，保证本书出版、正常发售的朋友，他们是产品经理 Serena Yang、编辑 Keith Fancher 和 No Starch 出版社的其他员工。

最后要感谢 LinuxCommand.org 的读者，他们给我发了很多有价值的邮件。是他们的鼓舞让我感觉到，我真的是在做一件非比寻常的事情。

前　　言

我想给大家讲一个故事。故事内容不是 Linus Torvalds 在 1991 年怎样编写了 Linux 内核的第一个版本，因为这些内容你可以在很多 Linux 图书中找到。我也不想告诉你，更早之前，Richard Stallman 是如何开始 GNU 项目，设计了一个免费的类 UNIX 操作系统。那也是一个很有意义的故事，但大多数 Linux 图书也讲到了它。我想给大家讲一个如何才能夺回计算机控制权的故事。

在 20 世纪 70 年代后期，我刚刚开始和计算机打交道时，正在进行着一场革命，那时的我还是一名大学生。微处理器的发明使得你我这样的普通人真正拥有一台计算机成为可能。今天，人们难以想象，只有大公司和强大的政府机构才能够使用计算机的世界，是怎样的一个世界。让我说，你其实想象不出多少来。

如今，世界已经截然不同。计算机遍布各个领域，从小手表到大型数据中心，以及介于它们之间的每一样东西。除了随处可见的计算机之外，我们还有一个无处不在的连接所有计算机的网络。这开创了一个奇妙的个人授权和创作自由的新时代。但是在过去的二三十年里，一些事情在悄然发生。一个大公司不断地把它的控制权强加到世界绝大多数的计算机上，并且决定你对计算机的操作权力。幸运的是，世界各地的人们正在努力进行抗争。他们通过自己编写软件来争夺自己计算机的控制权。他们创造了 Linux！

很多人提到 Linux 的时候都会讲到“自由”，但是并不是所有人都明白这种自由到底意味着什么。自由就是能够决定计算机可以做什么，而获得这种自由的唯一方法就是知道你的计算机正在做什么；自由就是计算机没有秘密可言，

只要你仔细地寻找，就能了解其全部内容。

为什么使用命令行

读者之前应该注意到，电影中的“超级黑客”，就是那些能够在 30 秒内入侵到超级安全的军方计算机里的家伙，都是坐在计算机旁，从来不碰鼠标的。这是因为电影制片人意识到，我们人类从本能上会明白，能够让计算机执行任何任务的唯一途径，是通过键盘输入命令来实现的。

现在，大多数计算机用户只熟悉图形用户界面（GUI），并且产品供应商和专家还在不停地灌输一种思想，那就是命令行界面（CLI）是一种很糟糕的东西，而且已经过时。这是很不幸的，因为一个好的命令行界面是一种很神奇的人机交互方式，就和我们采用书信进行交流一样。据说“图形用户界面能让简单的任务更简单，而命令行界面能够处理复杂的任务”，这句话在今天看来仍然是正确的。

由于 Linux 系统参照了 UNIX 系列操作系统，它分享了 UNIX 系统丰富的命令行工具。UNIX 系统在 20 世纪 80 年代早期就占据了主流地位（尽管它只是在 20 世纪 70 年代才开发出来），结果，在普遍采用图形用户界面之前，开发了一种广泛使用的命令行界面。事实上，Linux 开发者优先使用命令行界面（而不是其他系统，比如 Windows NT）的一个原因就是因为其强大的命令行界面，使“完成复杂的任务成为可能”。

本书内容

这是一本全面讲述如何使用 Linux 命令行的图书。与那些仅涉及一个程序（比如 shell 程序、bash）的图书不同，本书从更广泛的意义上向读者传授如何使用命令行，它是如何工作的，它有哪些功能，以及使用它的最佳方式是什么。

这不是一本关于 Linux 系统管理方面的图书。任何一个关于命令行的重要讨论，都一定会涉及系统管理方面的内容，但是本书只涉及很少的管理方面的

问题。本书为读者准备了其他的学习内容，帮助你为使用命令行打下坚实的基础，这可是完成一个系统管理任务所必需的至关重要的工具。

本书以 Linux 为中心。其他许多图书为了扩大读者群体以及自身的影响力，会在书中包含其他平台，比如通用的 UNIX 和 Mac OS X 系统。而且为了达到这个目的，它们只能“淡化”书的内容，只讲解一些通用的主题。而本书只包括当前的 Linux 发行版本。尽管本书 95% 的内容对其他类 UNIX 系统用户也有帮助，但是本书主要还是面向现代的 Linux 命令行用户。

本书读者对象

本书适合从其他平台转到 Linux 的新用户阅读。这些新用户很可能原来是 Microsoft Windows 版本的超级用户；也可能是老板已经要求负责管理一个 Linux 服务器的管理员；还有可能是厌烦了桌面系统的安全问题，想要体验一下 Linux 系统的用户。没关系，不管你属于哪类用户，都欢迎阅读本书。

不过一般来说，Linux 的启蒙学习不存在任何捷径。命令行的学习具有挑战性而且颇费精力，这倒不是因为它太难，而是它涵盖的内容太多。一般的 Linux 系统有上千个程序可以通过命令行使用，这点毫不夸张。你需要提醒自己的是，命令行可不是随便就能学会的。

另一方面，学习 Linux 命令行也非常值得。如果你认为自己已经是一名“超级用户”了，那么请注意，你可能不知道什么才是真正的“超级用户”。不同于许多其他的计算机技术，命令行的知识是经久不衰的。今天学会的技能，在 10 年后仍然有用。换言之，命令行是能够历经时间考验的。

如果读者没有编程经验，也不用担心，你仍然可以从本书开始学习。

内容安排

本书精心编排，内容有序，就像有一位老师坐在你身旁，耐心指导你。许多作者都采用系统化的方法来讲解本书中的内容。对作者来讲，这很合理，但

是对初学者来讲，则可能摸不着头脑。

本书的另一个目的是使读者熟悉 UNIX 的思考方式，它与 Windows 的思考方式大不相同。在学习的过程中，本书还将帮助读者理解命令行的工作原理和方式。Linux 不仅仅是一个软件，它还是庞大的 UNIX 文化中的一小部分，有着自己的语言和历史。同时，我也许会说一些过激的言语。

本书分为四部分，每一部分都讲解了不同方面的命令行知识。

- **第一部分：学习 shell。** 开始对命令行基本语言的学习之旅，包括命令结构、浏览文件系统、编辑命令行，以及查找命令帮助文档。
- **第二部分：配置和环境。** 这部分主要讲述如何编写配置文件，通过配置文件，用命令行的方式控制计算机操作。
- **第三部分：常见任务和主要工具。** 这部分讲述了许多通过命令行来执行的常规任务。类 UNIX 操作系统，比如 Linux，包含了很多“经典的”命令行程序，这些程序可以对数据进行强大的操作。
- **第四部分：编写 shell 脚本。** 这部分介绍了 shell 编程，这是一个公认的基本技能，它很容易学习，而且它可以自动执行很多常见的计算任务。通过学习 shell 编程，你会逐渐熟悉一些关于编程语言方面的概念，这些概念也适用于其他编程语言。

如何阅读

建议读者从头到尾地阅读本书。本书并不是一本技术参考手册，实际上它更像一本故事书：有开头，有过程，有结尾。

预备知识

为了使用本书，你需要安装 Linux 操作系统。你可以通过两种方式来完成安装。

- 在计算机（不需要是最新的）上安装 Linux 系统。选择哪个 Linux 发行版本没有关系，虽然现在大多数人在开始的时候会选择 Ubuntu、Fedora，或者是 OpenSUSE。如果你拿不准，那就先试试 Ubuntu。由于主机硬件配置不同，安装 Linux 时，你可能不费吹灰之力就安装上了，也可能费了九牛二虎之力还安装不上。本书建议使用一台几年前的台式计算机，并且有至少 256MB 的内存和 6GB 的空闲磁盘空间。尽可能避免使用笔记本计算机和无线网络，在 Linux 环境下，它们经常不能工作。
- 使用 live CD。许多 Linux 发行版本都自带一个比较酷的功能，你可以直接从 CD-ROM 中运行 Linux，而不必安装 Linux。只需进入 BIOS 设置界面，设置计算机从 CD-ROM 启动，插入一个 live CD，然后重新启动。在进行安装之前，使用 live CD 可以检测计算机对 Linux 的兼容性。但是缺点是会比通过硬盘安装 Linux 要慢好多。Ubuntu 和 Fedora 都有 live CD 版本。

注意

无论采用何种方式来安装 Linux，都将需要有临时的超级用户（也就是管理员）特权来执行本书中所讲的内容。

在安装好 Linux 系统之后，就一边开始阅读本书，一边练习吧。本书大部分内容都可以自己动手练习，所以坐下来，开始敲入命令并体验吧。

本书为什么不称为“GNU/LINUX”

在某些领域，称 Linux 操作系统为“GNU/LINUX 操作系统”在政治立场上是正确的。没有一个完全正确的方式能命名它，因为它是由许多分布在世界各地的贡献者们合作开发而成的。从技术层面来讲，Linux 只是操作系统的内核名字，没有别的含义。当然，内核非常重要，因为有了它，操作系统才能运行起来，但是它不能构成一个完备的操作系统。

Richard Stallman 是一位天才的哲学家、自由软件运动的创始人、自由基金会创办者，他创建了自由软件项目，编写了 GNU C 语言编译器（GCC）的第一个版本，并且创建了 GNU 公用许可证（GPL）等。他坚持将 Linux 称为“GNU/LINUX”，为的是准确地反映 GNU 项目对 Linux 操作系统所做出的贡献。尽管 GNU 项目先于 Linux 内核出现，并且这个项目所做出的贡献得到了极高的赞誉，但是将 GNU 加入 Linux 名字里面则对其他每一个为 Linux

的发展做出巨大贡献的人们来说，就不公平了。除此之外，由于计算机在运行的时候首先启动内核，其他的所有软件都在其之上运行，所以本书认为“Linux/GNU”这个名字在技术上来讲更精准一些。

在目前流行的用法中，“Linux”指的是内核以及在一个典型的 Linux 发行版本中所包含的所有免费和开源软件，也就是说，整个 Linux 体系并非仅有 GNU 项目软件。在操作系统业界，好像更钟情于单个词的名字，比如 DOS、Windows、Solaris、Iris 和 AIX。所以本书选择使用流行的命名规则。然而，如果读者倾向于使用“GNU/LINUX”，那么在阅读本书时，可以搜索并替换“Linux”，我不会介意的。

目 录

第一部分 学习 shell

第 1 章 shell 是什么	3
1.1 终端仿真器.....	3
1.2 第一次键盘输入.....	4
1.2.1 命令历史记录.....	4
1.2.2 光标移动.....	4
1.3 几个简单的命令.....	5
1.4 结束终端会话.....	6
第 2 章 导航	7
2.1 理解文件系统树.....	7
2.2 当前工作目录.....	8
2.3 列出目录内容.....	9
2.4 更改当前工作目录	9
2.4.1 绝对路径名	9
2.4.2 相对路径名	9
2.4.3 一些有用的快捷方式	10
第 3 章 Linux 系统	13
3.1 ls 命令的乐趣.....	13

ii 目 录

3.1.1 选项和参数	14
3.1.2 进一步了解长列表格式	15
3.2 使用 file 命令确定文件类型	16
3.3 使用 less 命令查看文件内容	16
3.4 快速浏览	18
3.5 符号链接	20
第 4 章 操作文件与目录	23
4.1 通配符	24
4.2 mkdir——创建目录	26
4.3 cp——复制文件和目录	26
4.4 mv——移除和重命名文件	27
4.5 rm——删除文件和目录	28
4.6 ln——创建链接	29
4.6.1 硬链接	29
4.6.2 符号链接	30
4.7 实战演练	30
4.7.1 创建目录	30
4.7.2 复制文件	31
4.7.3 移动和重命名文件	31
4.7.4 创建硬链接	32
4.7.5 创建符号链接	33
4.7.6 移除文件和目录	34
4.8 本章结尾语	35
第 5 章 命令的使用	37
5.1 究竟是什么命令	38
5.2 识别命令	38
5.2.1 type——显示命令的类型	38
5.2.2 which——显示可执行程序的位置	39
5.3 获得命令文档	39
5.3.1 help——获得 shell 内置命令的帮助文档	39
5.3.2 help——显示命令的使用信息	40
5.3.3 man——显示程序的手册页	40
5.3.4 apropos——显示合适的命令	41

5.3.5 whatis——显示命令的简要描述	42
5.3.6 info——显示程序的 info 条目	42
5.3.7 README 和其他程序文档文件	43
5.4 使用别名创建自己的命令	43
5.5 温故以求新	45
第 6 章 重定向.....	47
6.1 标准输入、标准输出和标准错误	48
6.1.1 标准输出重定向	48
6.1.2 标准错误重定向	50
6.1.3 将标准输出和标准错误重定向到同一个文件	50
6.1.4 处理不想要的输出	51
6.1.5 标准输入重定向	51
6.2 管道	53
6.2.1 过滤器	53
6.2.2 uniq——报告或忽略文件中重复的行	54
6.2.3 wc——打印行数、字数和字节数	54
6.2.4 grep——打印匹配行	54
6.2.5 head/tail——输出文件的开头部分/结尾部分	55
6.2.6 tee——从 stdin 读取数据，并同时输出到 stdout 和文件	56
6.3 本章结尾语	57
第 7 章 透过 shell 看世界	59
7.1 扩展	59
7.1.1 路径名扩展	60
7.1.2 波浪线扩展	61
7.1.3 算术扩展	61
7.1.4 花括号扩展	62
7.1.5 参数扩展	63
7.1.6 命令替换	64
7.2 引用	65
7.2.1 双引号	65
7.2.2 单引号	67
7.2.3 转义字符	67
7.3 本章结尾语	68

第 8 章 高级键盘技巧	69
8.1 编辑命令行	69
8.1.1 光标移动	70
8.1.2 修改文本	70
8.1.3 剪切和粘贴 (Killing and Yanking) 文本	71
8.2 自动补齐功能	71
8.3 使用历史命令	73
8.3.1 搜索历史命令	73
8.3.2 历史记录扩展	75
8.4 本章结尾语	76
第 9 章 权限	77
9.1 所有者、组成员和其他所有用户	78
9.2 读取、写入和执行	79
9.2.1 chmod——更改文件模式	81
9.2.2 采用 GUI 设置文件模式	84
9.2.3 umask——设置默认权限	85
9.3 更改身份	87
9.3.1 su——以其他用户和组 ID 的身份来运行 shell	88
9.3.2 sudo——以另一个用户的身份执行命令	89
9.3.3 chown——更改文件所有者和所属群组	90
9.3.4 chgrp——更改文件所属群组	91
9.4 权限的使用	91
9.5 更改用户密码	93
第 10 章 进程	95
10.1 进程如何工作	96
10.1.1 使用 ps 命令查看进程信息	96
10.1.2 使用 top 命令动态查看进程信息	98
10.2 控制进程	100
10.2.1 中断进程	100
10.2.2 使进程在后台运行	101
10.2.3 使进程回到前台运行	101
10.2.4 停止 (暂停) 进程	102
10.3 信号	102

10.3.1 使用 kill 命令发送信号到进程	103
10.3.2 使用 killall 命令发送信号给多个进程	105
10.4 更多与进程相关的命令	105

第二部分 配置与环境

第 11 章 环境	109
11.1 环境中存储的是什么	109
11.1.1 检查环境	110
11.1.2 一些有趣的变量	111
11.2 环境是如何建立的	112
11.2.1 login 和 non-login shell	112
11.2.2 启动文件中有什么	113
11.3 修改环境	114
11.3.1 用户应当修改哪些文件	114
11.3.2 文本编辑器	115
11.3.3 使用文本编辑器	115
11.3.4 激活我们的修改	117
11.4 本章结尾语	118
第 12 章 VI 简介	119
12.1 为什么要学习 vi	119
12.2 VI 背景	120
12.3 启动和退出 vi	120
12.4 编辑模式	121
12.4.1 进入插入模式	122
12.4.2 保存工作	122
12.5 移动光标	123
12.6 基本编辑	124
12.6.1 添加文本	124
12.6.2 插入一行	125
12.6.3 删 除文本	126
12.6.4 剪切、复制和粘贴文本	127
12.6.5 合并行	128
12.7 查找和替换	128

12.7.1 行内搜索	128
12.7.2 搜索整个文件	129
12.7.3 全局搜索和替换	129
12.8 编辑多个文件	130
12.8.1 切换文件	131
12.8.2 载入更多的文件	132
12.8.3 文件之间的内容复制	132
12.8.4 插入整个文件	133
12.9 保存工作	134
第 13 章 定制提示符	135
13.1 提示符的分解	135
13.2 尝试设计提示符	137
13.3 添加颜色	138
13.4 移动光标	140
13.5 保存提示符	141
13.6 本章结尾语	141

第三部分 常见任务和主要工具

第 14 章 软件包管理	145
14.1 软件包系统	146
14.2 软件包系统工作方式	146
14.2.1 软件包文件	146
14.2.2 库	147
14.2.3 依赖关系	147
14.2.4 高级和低级软件包工具	147
14.3 常见软件包管理任务	148
14.3.1 在库里面查找软件包	148
14.3.2 安装库中的软件包	148
14.3.3 安装软件包文件中的软件包	149
14.3.4 删除软件包	149
14.3.5 更新库中的软件包	150
14.3.6 更新软件包文件中的软件包	150
14.3.7 列出已安装的软件包列表	150

14.3.8 判断软件包是否安装.....	151
14.3.9 显示已安装软件包的相关信息.....	151
14.3.10 查看某具体文件由哪个软件包安装得到.....	151
14.4 本章结尾语.....	152
第 15 章 存储介质.....	155
15.1 挂载、卸载存储设备	156
15.1.1 查看已挂载的文件系统列表.....	157
15.1.2 确定设备名称.....	160
15.2 创建新的文件系统	162
15.2.1 用 fdisk 命令进行磁盘分区.....	162
15.2.2 用 mkfs 命令创建新的文件系统.....	164
15.3 测试、修复文件系统	165
15.4 格式化软盘.....	166
15.5 直接从/向设备转移数据	166
15.6 创建 CD-ROM 映像	167
15.6.1 创建一个 CD-ROM 文件映像副本.....	167
15.6.2 从文件集合中创建映像文件.....	168
15.7 向 CD-ROM 写入映像文件	168
15.7.1 直接挂载 ISO 映像文件	168
15.7.2 擦除可读写 CD-ROM.....	169
15.7.3 写入映像文件	169
15.8 附加认证.....	169
第 16 章 网络.....	171
16.1 检查、监测网络.....	172
16.1.1 ping——向网络主机发送特殊数据包	172
16.1.2 traceroute——跟踪网络数据包的传输路径.....	173
16.1.3 netstat——检查网络设置及相关统计数据	174
16.2 通过网络传输文件	175
16.2.1 ftp——采用 FTP (文件传输协议) 传输文件	175
16.2.2 lftp——更好的 ftp (文件传输协议)	177
16.2.3 wget——非交互式网络下载工具	177
16.3 与远程主机的安全通信	178
16.3.1 ssh——安全登录远程计算机.....	178

16.3.2 scp 和 sftp——安全传输文件	181
第 17 章 文件搜索.....	183
17.1 locate——较简单的方式查找文件.....	184
17.2 find——较复杂的方式查找文件.....	185
17.2.1 test 选项	186
17.2.2 action 选项	190
17.2.3 返回到 playground 文件夹	194
17.2.4 option 选项	196
第 18 章 归档和备份.....	197
18.1 文件压缩.....	198
18.1.1 gzip——文件压缩与解压缩	198
18.1.2 bzip2——牺牲速度以换取高质量的数据压缩	200
18.2 文件归档.....	201
18.2.1 tar——磁带归档工具	201
18.2.2 zip——打包压缩文件	205
18.3 同步文件和目录.....	207
18.3.1 rsync——远程文件、目录的同步	207
18.3.2 在网络上使用 rsync 命令	209
第 19 章 正则表达式.....	211
19.1 什么是正则表达式	211
19.2 grep——文本搜索.....	212
19.3 元字符和文字.....	213
19.4 任意字符.....	214
19.5 锚.....	214
19.6 中括号表达式和字符类	215
19.6.1 否定	216
19.6.2 传统字符范围	216
19.6.3 POSIX 字符类	217
19.7 POSIX 基本正则表达式和扩展正则表达式的比较	220
19.8 或选项.....	221
19.9 限定符.....	222
19.9.1 ? ——匹配某元素 0 次或 1 次	222
19.9.2 * ——匹配某元素多次或零次	222

19.9.3 +——匹配某元素一次或多次	223
19.9.4 {}——以指定次数匹配某元素	223
19.10 正则表达式的应用	224
19.10.1 用 grep 命令验证号码簿	224
19.10.2 用 find 查找奇怪文件名的文件	225
19.10.3 用 locate 查找文件	226
19.10.4 利用 less 和 vim 命令搜索文本	226
19.11 本章结尾语	227
第 20 章 文本处理	229
20.1 文本应用程序	230
20.1.1 文件	230
20.1.2 网页	230
20.1.3 电子邮件	230
20.1.4 打印机输出	231
20.1.5 程序源代码	231
20.2 温故以求新	231
20.2.1 cat——进行文件之间的拼接并且输出到标准输出	231
20.2.2 sort——对文本行进行排序	232
20.2.3 uniq——通知或省略重复的行	238
20.3 切片和切块	239
20.3.1 cut——删除文本行中的部分内容	239
20.3.2 paste——合并文本行	242
20.3.3 join——连接两文件中具有相同字段的行	243
20.4 文本比较	245
20.4.1 comm——逐行比较两个已排序文件	245
20.4.2 diff——逐行比较文件	246
20.4.3 patch——对原文件进行 diff 操作	248
20.5 非交互式文本编辑	249
20.5.1 tr——替换或删除字符	249
20.5.2 sed——用于文本过滤和转换的流编辑器	251
20.5.3 aspell——交互式拼写检查工具	258
20.6 本章结尾语	260
20.7 附加项	261

x 目 录

第 21 章 格式化输出	263
21.1 简单的格式化工具	264
21.1.1 nl——对行进行标号	264
21.1.2 fold——将文本中的行长度设定为指定长度	266
21.1.3 fmt——简单的文本格式化工具	267
21.1.4 pr——格式化打印文本	270
21.1.5 printf——格式化并打印数据	270
21.2 文档格式化系统	273
21.2.1 roff 和 T _E X 家族	274
21.2.2 groff——文档格式化系统	274
21.3 本章结尾语	279
第 22 章 打印	281
22.1 打印操作简史	282
22.1.1 灰暗时期的打印	282
22.1.2 基于字符的打印机	282
22.1.3 图形化打印机	283
22.2 Linux 方式的打印	284
22.3 准备打印文件	284
22.3.1 pr——将文本文件转换为打印文件	285
22.4 向打印机发送打印任务	285
22.4.1 lpr——打印文件 (Berkeley 类型)	286
22.4.2 lp——打印文件 (System V 类型)	287
22.4.3 另外一个参数选项: a2ps	287
22.5 监测和控制打印任务	290
22.5.1 lpstat——显示打印系统状态	290
22.5.2 lpq——显示打印队列状态	291
22.5.3 lprm 与 cancel——删除打印任务	291
第 23 章 编译程序	293
23.1 什么是编译	294
23.2 是不是所有的程序都需要编译	295
23.3 编译一个 C 程序	295
23.3.1 获取源代码	296
23.3.2 检查源代码树	297

23.3.3 生成程序.....	298
23.3.4 安装程序.....	302
23.4 本章结尾语.....	302

第四部分 编写 shell 脚本

第 24 章 编写第一个 shell 脚本	305
24.1 什么是 shell 脚本	305
24.2 怎样写 shell 脚本	306
24.2.1 脚本文件的格式	306
24.2.2 可执行权限	307
24.2.3 脚本文件的位置	307
24.2.4 脚本的理想位置	308
24.3 更多的格式诀窍	309
24.3.1 长选项名	309
24.3.2 缩进和行连接	309
24.5 本章结尾语	310
第 25 章 启动一个项目	311
25.1 第一阶段：最小的文档	311
25.2 第二阶段：加入一点数据	313
25.3 变量和常量	314
25.3.1 创建变量和常量	314
25.3.2 为变量和常量赋值	316
25.4 here 文档	317
25.5 本章结尾语	319
第 26 章 自顶向下设计	321
26.1 shell 函数	322
26.2 局部变量	325
26.3 保持脚本的运行	326
26.4 本章结尾语	328
第 27 章 流控制：IF 分支语句	329
27.1 使用 if	330
27.2 退出状态	330

27.3 使用 test 命令	332
27.3.1 文件表达式	332
27.3.2 字符串表达式	334
27.3.3 整数表达式	335
27.4 更现代的 test 命令版本	336
27.5 (())——为整数设计	338
27.6 组合表达式	339
27.7 控制运算符：另一种方式的分支	341
27.8 本章结尾语	342
第 28 章 读取键盘输入	343
28.1 read——从标准输入读取输入值	344
28.1.1 选项	346
28.1.2 使用 IFS 间隔输入字段	347
28.2 验证输入	349
28.3 菜单	350
28.4 本章结尾语	351
28.5 附加项	352
第 29 章 流控制： WHILE 和 UNTIL 循环	353
29.1 循环	353
29.2 while	354
29.3 跳出循环	356
29.4 until	357
29.5 使用循环读取文件	358
29.6 本章结尾语	358
第 30 章 故障诊断	359
30.1 语法错误	359
30.1.1 引号缺失	360
30.1.2 符号缺失冗余	360
30.1.3 非预期的展开	361
30.2 逻辑错误	362
30.2.1 防御编程	363
30.2.2 输入值验证	364
30.3 测试	364

30.3.1 桩	365
30.3.2 测试用例	365
30.4 调试	366
30.4.1 找到问题域	366
30.4.2 追踪	366
30.4.3 运行过程中变量的检验	368
30.5 本章结尾语	369
第 31 章 流控制: case 分支	371
31.1 case	371
31.1.1 模式	373
31.1.2 多个模式的组合	374
31.2 本章结尾语	375
第 32 章 位置参数	377
32.1 访问命令行	377
32.1.1 确定实参的数目	378
32.1.2 shift——处理大量的实参	379
32.1.3 简单的应用程序	380
32.1.4 在 shell 函数中使用位置参数	381
32.2 处理多个位置参数	381
32.3 更完整的应用程序	383
32.4 本章结尾语	386
第 33 章 流控制: for 循环	389
33.1 for: 传统 shell 形式	389
33.2 for: C 语言形式	392
33.3 本章结尾语	393
第 34 章 字符串和数字	395
34.1 参数扩展 (Parameter Expansion)	395
34.1.1 基本参数	396
34.1.2 空变量扩展的管理	396
34.1.3 返回变量名的扩展	397
34.1.4 字符串操作	398
34.2 算术计算和扩展	400
34.2.1 数字进制	401

34.2.2 一元运算符	401
34.2.3 简单算术	401
34.2.4 赋值	402
34.2.5 位操作	404
34.2.6 逻辑操作	405
34.3 bc：一种任意精度计算语言	407
34.3.1 bc 的使用	407
34.3.2 脚本例子	408
34.4 本章结尾语	409
34.5 附加项	409
第 35 章 数组	411
35.1 什么是数组	411
35.2 创建一个数组	412
35.3 数组赋值	412
35.4 访问数组元素	413
35.5 数组操作	414
35.5.1 输出数组的所有内容	415
35.5.2 确定数组元素的数目	415
35.5.3 查找数组中使用的下标	416
35.5.4 在数组的结尾增加元素	416
35.5.5 数组排序操作	416
35.5.6 数组的删除	417
35.6 本章结尾语	418
第 36 章 其他命令	419
36.1 组命令和子 shell	419
36.1.1 执行重定向	420
36.1.2 进程替换	420
36.2 trap	422
36.3 异步执行	425
36.4 命名管道	426
36.4.1 设置命名管道	427
36.4.2 使用命名管道	427
36.5 本章结尾语	428

第一部分

学习 shell



第 1 章

shell 是什么

当谈到命令行时，我们实际上指的是 shell。shell 是一个接收由键盘输入的命令，并将其传递给操作系统来执行的程序。几乎所有的 Linux 发行版都提供 shell 程序，该程序来自于称之为 bash 的 GNU 项目。bash 是 Bourne Again Shell 的首字母缩写，Bourne Again Shell 基于这样一个事实，即 bash 是 sh 的增强版本，而 sh 是最初的 UNIX shell 程序，由 Steve Bourne 编写。

1.1 终端仿真器

当使用图形用户界面时，需要另一种叫做终端仿真器（terminal emulator）的程序与 shell 进行交互。如果我们仔细查看桌面菜单，那么很可能会找到一款终端仿真器。在 KDE 环境下使用的是 konsole，而在 GNOME 环境下使用的是 gnome-terminal，但是在桌面菜单上很可能将它们简单地统称为终端。在 Linux 系统中，还有许多其他的终端仿真器可以使用，但是它们基本上都做同样的事情：让用户访问 shell。因为不同的终端仿真器所具有的功能特性不尽相同，因

此，你可以根据自己的喜好进行选择。

1.2 第一次键盘输入

现在开始吧。启动终端仿真器！运行后的终端仿真器如下所示。

```
[me@linuxbox ~]$
```

这称为 shell 提示符，只要 shell 准备接受外部输入，它就会出现。在不同的发行版中，提示符的外观可能会有所差异，但是，它通常包括 `username@machinename`，其后是当前工作目录（长度更长一些）和一个\$符号。

如果 shell 提示符的最后一个字符是#，而不是一个\$符号，那么终端会话将享有超级用户特权。这就意味着要么我们是以根用户身份登录，要么我们选择的终端仿真器可以提供超级用户（管理）特权。

假定一切工作都很顺利，接下来尝试输入一些内容。在提示符后输入一些乱码，如下所示。

```
[me@linuxbox ~]$ kaekfjaejfj
```

由于这些命令没有任何意义，shell 会让我们重新输入。

```
bash: kaekfjaejfj: command not found  
[me@linuxbox ~]$
```

1.2.1 命令历史记录

如果按下向上方向指示键，将会看到先前的命令 `kaekfjaejfj` 再一次出现在提示符的后面，这称为命令历史记录。在默认情况下，大部分 Linux 发行版本能够存储最近输入的 500 个命令。按下向下方向指示键，则先前的命令消失。

1.2.2 光标移动

再次按下向上方向指示键，重新调用先前的命令，然后分别按下向左和向右方向指示键，看看如何将光标定位到命令行的任意位置。这可以让我们很容易地编辑命令。

关于鼠标与焦点

尽管 shell 与用户的交互全部是通过键盘来完成的，但是在终端仿真器中，也可以使用鼠标。内置到 X 窗口系统（驱动 GUI 的底层引擎）中的一种机制

可以支持快速的复制与粘贴技术。如果紧按鼠标左键选中一些文本并拖动鼠标（或双击选中一个词），该文本将复制到由 X 维护的一个缓冲区中。按下鼠标的中间按键可以将选中的文本粘贴到光标所在的位置。你可以试一下。

不要试图使用 Ctrl-C 和 Ctrl-V 在一个终端窗口内进行复制和粘贴操作，这不起作用。对于 shell 而言，这些组合键在很早之前就已经赋予了不同的含义，而那时微软的 Windows 操作系统还没有出现。

在操作上与 Windows 类似的图形桌面环境（很有可能是 KDE 或 GNOME），很可能拥有自己的焦点策略（focus policy）集合，用以通过“点击来获得焦点”。这意味着，如果一个窗口需要获得焦点（成为当前窗口），只需要点击一下即可。而传统的 X 窗口的行为是“焦点跟随着鼠标”，也就是说，当鼠标经过窗口时，窗口就会获得焦点。因此两者是截然不同的。如果没有点击窗口，那么它不会出现在前端，但此时它可以接受输入。将焦点策略设置为“焦点跟随鼠标”的方式会使终端窗口使用起来更容易。试一试吧，试过之后，你一定会喜欢上这种方式。你可在窗口管理器的配置程序中找到该设置。

1.3 几个简单的命令

在学习了键盘输入之后，我们来尝试几个简单的命令。首先是 date 命令，该命令显示当前系统的时间和日期。

```
[me@linuxbox ~]$ date
Thu Oct 25 13:51:54 EDT 2012
```

与之相关的一个命令是 cal，在默认情况下，cal 显示当月的日历。

```
[me@linuxbox ~]$ cal
October 2012
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

如果想要查看磁盘驱动器当前的可用空间，可以使用 df 命令。

```
[me@linuxbox ~]$ df
Filesystem      1K-blocks   Used Available Use% Mounted on
/dev/sda2        15115452   5012392   9949716  34% /
/dev/sda5        59631908  26545424  30008432  47% /home
/dev/sda1        147764     17370    122765  13% /boot
tmpfs            256856         0    256856  0% /dev/shm
```

6 Linux 命令行大全

同样，要显示可用内存，可以使用 free 命令。

```
[me@linuxbox ~]$ free
total        used        free      shared      buffers      cached
Mem:    513712     503976      9736          0       5312    122916
-/+ buffers/cache: 375748   137964
Swap: 1052248    104712   947536
```

1.4 结束终端会话

直接关闭终端窗口或是在 shell 提示符下输入 exit 命令，即可结束终端会话。

```
[me@linuxbox ~]$ exit
```

幕后的控制台

即使没有运行终端仿真器，一些终端会话也会在图形桌面的后台运行。这叫做虚拟终端或是虚拟控制台。在绝大多数系统中，通过依次按下 Ctrl-Alt-F1 键到 Ctrl-Alt-F6 组合键，可以访问大部分 Linux 发行版中的终端会话。每当访问一次会话，就会出现登录提示符，我们可以在其中输入用户名和密码。按 Alt 和 F1~F6 键，可从一个虚拟控制台转换到另一个虚拟控制台。按 Alt-F7 键可返回图形桌面环境。

第 2 章

导 航

除了在命令行进行输入操作之外，我们首先需要学习的是如何在 Linux 系统中导航文件系统。本章将介绍下述命令。

- `pwd`: 查看当前工作目录。
- `cd`: 改变目录。
- `ls`: 列出目录内容。

2.1 理解文件系统树

与 Windows 相同，类 UNIX 操作系统（比如 Linux）也是以称之为分层目录结构的方式来组织文件的。这意味着文件是在树形结构的目录（有时在其他系统中称为文件夹）中进行组织的，该树形结构目录可能包含文件和其他目录。文件系统的第一个目录叫做根目录，它包含了文件和子目录。子目录里包含了更多的文件和子目录，依此类推。

需要注意的是，在 Windows 系统中，每个存储设备都有一个独立的文件系统树。而在类 UNIX 系统中，如 Linux，无论多少驱动器或存储设备与计算机相连，通常只有一个文件系统树。根据系统管理员的设置，存储设备将会连接（更准确的说是“挂载”）到文件系统树的不同位置。系统管理员要负责系统的维护。

2.2 当前工作目录

可能大部分人都熟悉用于表示文件系统树的图形文件管理器，如图 2-1 所示。需要注意的是，树通常是倒立显示的。也就是说，顶部是根目录，依次向下排列的是子目录。

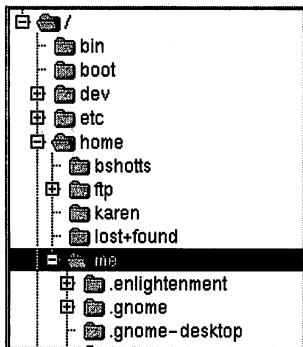


图 2-1 在图形文件管理器中显示的文件系统树

然而，由于命令行没有图像，若是要浏览文件系统树，就必须使用其他方法。

假设文件系统是一个迷宫，形如一棵倒置的树，并且用户处在文件系统之中。任何时刻，我们处在单个目录中，能够看到该目录中包含的文件、去往上一级目录（称为父目录）的路径，以及下一级的各个子目录。用户所处的目录叫做当前工作目录。使用 `pwd`（打印工作目录）命令可以显示当前工作目录。

```
[me@linuxbox ~]$ pwd
/home/me
```

第一次登录系统时（或是启动终端仿真器会话时），当前工作目录被设置成主目录。每个用户账号都有一个主目录，作为普通用户操作时，这是唯一一个允许用户写文件的地方。

2.3 列出目录内容

使用 ls 命令可以列出当前工作目录的文件和目录。

```
[me@linuxbox ~]$ ls
Desktop Documents Music Pictures Public Templates Videos
```

实际上，可以使用 ls 命令列出任何目录的内容，而不仅仅是当前工作目录。同时，它还拥有一些其他有趣的功能。我们会在第 3 章详细讨论 ls 命令。

2.4 更改当前工作目录

使用 cd 命令可以改变工作目录（即在文件系统树的位置）；只需输入 cd 命令，然后再输入目标工作目录的路径名即可。路径名指的是沿着分枝到达目标目录的路由。路径名分为两种：绝对路径名和相对路径名。首先来谈谈绝对路径名。

2.4.1 绝对路径名

绝对路径名从根目录开始，其后紧接着一个又一个文件树分支，直到到达目标目录或文件。例如，系统里有一个目录，大多数系统程序都安装到这个目录里，该目录的路径名是/usr/bin。这就意味着根目录（在路径名中用前导斜杠来表示）中有一个目录是 usr，该目录包含一个 bin 目录。

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
[me@linuxbox bin]$ ls
...Listing of many, many files ...
```

可以看到，我们已经将当前工作目录改变成/usr/bin，bin 目录中包含很多文件。请注意 shell 提示符是如何变化的。为方便起见，工作目录名通常被设置成自动显示。

2.4.2 相对路径名

绝对路径名是从根目录开始，通向目标目录，而相对路径名则是从工作目录开始的。为了实现这个目的，它通常使用一些特殊符号来表示文件系统树中的相对位置，这些特殊符号是“.”（点）和“..”（点点）。

10 Linux 命令行大全

符号“.”代表工作目录，符号“..”代表工作目录的父目录。下面演示它们是如何工作的。让我们再次将工作目录改变成/usr/bin。

```
[me@linuxbox ~]$ cd /usr/bin  
[me@linuxbox bin]$ pwd  
/usr/bin
```

好的，下面来说明一下，我们希望将工作目录改变成/usr/bin的父目录，即/usr。有两种方法可以实现，一种是使用绝对路径名。

```
[me@linuxbox bin]$ cd /usr  
[me@linuxbox usr]$ pwd  
/usr
```

另一种是使用相对路径名。

```
[me@linuxbox bin]$ cd ..  
[me@linuxbox usr]$ pwd  
/usr
```

由于两种不同的方法产生同样的结果。那么我们究竟应该用哪一种方法呢？那就选择输入字符最少的吧。

同样，可以用两种方法将工作目录从/usr 变到/usr/bin。我们可以使用绝对路径名。·

```
[me@linuxbox usr]$ cd /usr/bin  
[me@linuxbox bin]$ pwd  
/usr/bin
```

我们也可以使用相对路径名。

```
[me@linuxbox usr]$ cd ./bin  
[me@linuxbox bin]$ pwd  
/usr/bin
```

必须在这里指出来的是，几乎在所有的情况下都可以省略“./”，因为它是隐含的。输入以下代码。

```
[me@linuxbox usr]$ cd bin
```

该代码与使用相对路径名的代码具有相同效果。一般而言，如果没有指定路径名，则默认为工作目录。

2.4.3 一些有用的快捷方式

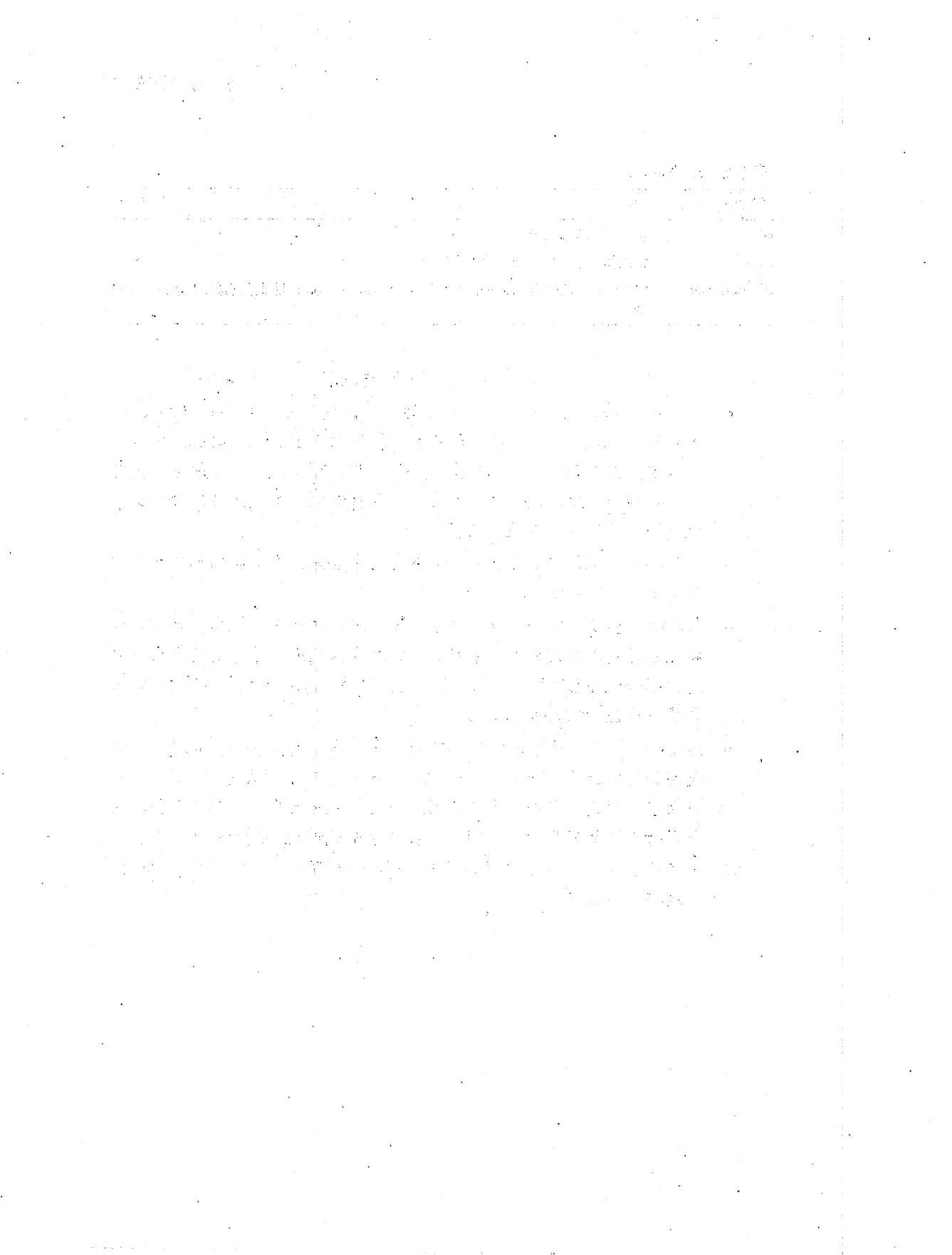
表 2-1 列出了一些可以快速改变当前工作目录的方法。

表 2-1 cd 快捷方式

快捷方式	结果
cd	将工作目录改变成主目录
cd-	将工作目录改变成先前的工作目录
cd~username	将工作目录改变为 username 的主目录。例如, cd~bob 将目录改变成用户 bob 的主目录

有关文件名的一些重要说明

- 以“.”字符开头的文件名是隐藏的。这仅说明 ls 不会列出这些文件，除非输入 ls-a。在创建用户账号时，主目录里会放置一些隐藏文件，用来配置账号信息。仔细观察这样的文件，可以使我们了解如何自定义工作环境。此外，一些应用程序也会将它们的配置文件和设置文件以隐藏文件的形式放在主目录下面。
- 与 UNIX 一样，在 Linux 中，文件名与命令是区分大小写的。文件名 File1 和 file1 指向不同的文件。
- 与其他一些操作系统一样，Linux 没有文件扩展名的概念。我们可以按照自己的喜好随意给文件命名。文件的内容或用途由其他方式来决定。尽管类 UNIX 操作系统不使用文件扩展名来决定文件内容或用途，但是一些应用程序却这么做了。
- Linux 支持长文件名，文件名可能包含了嵌入的空格和标点符号。但是在创建文件名的过程中，仅句号、连字符和下划线是可以使用的。更为重要的是，文件名中不要嵌入空格。文件名中嵌入空格会使很多命令行任务的实现变得困难，在第 7 章中我们就会发现这个问题。如果想要表示文件名词语间的空格，可以使用下划线，以后我们就会知道这样的好处了。



第 3 章

Linux 系统

既然已经知道了如何在文件系统中跳转，是时候开始 Linux 操作系统之旅了。但是，在开始之前，我们要先学习一些对研究 Linux 系统很有帮助的命令。

- `ls`: 列出目录内容。
- `file`: 确定文件类型。
- `less`: 查看文件内容。

3.1 `ls` 命令的乐趣

有充分的理由证明，`ls` 命令很可能是用户最常使用的命令。通过 `ls` 命令可以查看目录内容，确定各种重要文件和目录的属性。我们已经看到，只需输入 `ls` 命令，即可查看当前工作目录中包含的一系列文件和子目录。

```
[me@linuxbox ~]$ ls
Desktop Documents Music Pictures Public Templates Videos
```

除了当前工作目录之外，我们还可以指定要显示的目录，如下所示。

```
[me@linuxbox ~]$ ls /usr
bin games kerberos libexec sbin src
etc include lib local share tmp
```

我们甚至可以指定多个目录。下面这个例子就列出了用户主目录（由符号“～”表示）和/usr目录的内容。

```
[me@linuxbox ~]$ ls ~ /usr
/home/me:
Desktop Documents Music Pictures Public Templates Videos
/usr:
bin games kerberos libexec sbin src
etc include lib local share tmp
```

我们也可以改变输出格式来得到更多细节。

```
[me@linuxbox ~]$ ls -l
total 56
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Desktop
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Documents
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Music
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Pictures
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Public
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Templates
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Videos
```

在命令中加上-l，我们可以将输出以长格式显示。

3.1.1 选项和参数

下面，让我们来了解一下大部分命令是如何工作的，这也是非常重要的一点。通常，命令后面跟有一个或多个选项，带有不同选项的命令其功能也不一样。此外，命令后面还会跟有一个或多个参数，这些参数是命令作用的对象。所以大部分命令看起来如下所示：

command -options arguments

大部分命令使用的选项是在单个字符前加上连字符，如-l。但是，很多命令，包括GNU项目里的命令，也支持在单字前加两个连字符的长选项。而且，很多命令也允许多个短选项串在一起使用。在下面的例子中，ls命令包含了两个选项；l选项产生长格式输出，而t选项则表示以文件修改时间的先后将结果进行排序。

```
[me@linuxbox ~]$ ls -lt
```

加上长选项--reverse，则结果会以相反的顺序输出：

```
[me@linuxbox ~]$ ls -lt --reverse
```

ls 命令有大量可用的选项。最常用的选项如表 3-1 所示。

表 3-1 ls 命令的常用选项

选项	长选项	含义
-a	--all	列出所有文件，包括以点号开头的文件，这些文件通常是不列出来的（比如隐藏的文件）
-d	--directory	通常，如果指定了一个目录，ls 命令会列出目录中的内容而不是目录本身。将此选项与-l 选项结合使用，可查看目录的详细信息，而不是目录中的内容
-F	--classify	选项会在每个所列出的名字后面加上类型指示符（例如，如果名字是目录名，则会加上一个斜杠）
-h	--human-readable	以长格式列出，以人们可读的方式而不是字节数来显示文件大小
-l		使用长格式显示结果
-r	--reverse	以相反的顺序显示结果。通常，ls 命令按照字母升序排列显示结果
-S		按文件大小对结果排序
-t		按修改时间排序

3.1.2 进一步了解长列表格式

前面看到，-l 选项使得 ls 命令以长格式显示其结果。这种格式包含了大量的有用信息。下面的例子来自 Ubuntu 系统。

```
-rw-r--r-- 1 root root 3576296 2012-04-03 11:05 Experience ubuntu.ogg
-rw-r--r-- 1 root root 1186219 2012-04-03 11:05 kubuntu-leaflet.png
-rw-r--r-- 1 root root 47584 2012-04-03 11:05 logo-Eduubuntu.png
-rw-r--r-- 1 root root 44355 2012-04-03 11:05 logo-Kubuntu.png
-rw-r--r-- 1 root root 34391 2012-04-03 11:05 logo-Ubuntu.png
-rw-r--r-- 1 root root 32059 2012-04-03 11:05 oo-cd-cover.odf
-rw-r--r-- 1 root root 159744 2012-04-03 11:05 oo-derivatives.doc
-rw-r--r-- 1 root root 27837 2012-04-03 11:05 oo-maxwell.odt
-rw-r--r-- 1 root root 98816 2012-04-03 11:05 oo-trig.xls
-rw-r--r-- 1 root root 453764 2012-04-03 11:05 oo-welcome.odt
-rw-r--r-- 1 root root 358374 2012-04-03 11:05 ubuntu Sax.ogg
```

再来看一下其中一个文件的不同字段，表 3-2 列出了这些不同字段的含义。

表 3-2 ls 长列表字段

字段	含义
-rw-r--r--	对文件的访问权限。第一个字符表示文件的类型。在不同类型之间，开头的“-”表示该文件是一个普通文件，d 表示目录。紧接着的三个字符表示文件所有者的访问权限，再接着的三个字符表示文件所属组中成员的访问权限，最后三个字符表示其他所有人的访问权限。详细解释请见第 9 章

续表

字段	含义
1	文件硬链接数目。链接的内容将在本章后面讨论
root	文件所有者的用户名
root	文件所属用户组的名称
32059	以字节数表示的文件大小
2012-04-03 11:05	上次修改文件的日期和时间
oo-cd-cover.odf	文件名

3.2 使用 file 命令确定文件类型

在我们探索系统的过程中，知道文件包含的内容是非常有用的。为此，我们可以使用 `file` 命令来确定文件类型。先前讲过，Linux 系统中的文件名不需要反映文件的内容。例如，当我们看到 `picture.jpg` 这样一个文件名，会很自然地觉得该文件中包含一张 JPEG 压缩图像，但是在 Linux 中却没有这个必要。我们可以这样调用 `file` 命令：

```
file filename
```

调用后，`file` 命令会打印出文件内容的简短说明。例如：

```
[me@linuxbox ~]$ file picture.jpg
picture.jpg: JPEG image data, JFIF standard 1.01
```

文件的种类有很多。事实上，在类 UNIX 操作系统（比如 Linux 系统）中，有个普遍的观念是“所有的东西都是一个文件”。随着课程的深入，大家会明白这句话的真谛。

尽管我们已经很熟悉系统中的许多文件，比如 MP3 和 JPEG 文件。但是也有一些文件比较含蓄，还有一些文件对我们而言相当陌生。

3.3 使用 less 命令查看文件内容

`less` 命令是一种查看文本文件的程序。纵观 Linux 系统，很多文件都含有人们可以阅读的文本。`less` 程序为我们查看文件提供了方便。

为什么要查看文本文件呢？因为包含系统设置的多数文件（即配置文件）是以这种格式存储的，阅读这些文件有利于更好地理解系统是如何工作的。此外，系统使用的许多实际程序（称之为脚本）也是以这种格式存储的。在随后的章节中，我们将学习如何编写文本文件来修改系统设置，并编写自己的脚本，

而现在我们只需看看它们的内容即可。

什么是文本

有很多方式可在计算机中表达信息。所有的方式都涉及在信息与一些数字之间确立一种关系，而这些数字可以用来表达信息。毕竟，计算机只能理解数字，并且所有的数据都将转换成数值来表示。

有些表示方法非常复杂（例如压缩后的视频文件），也有一些其他方法相当简单。其中出现最早也是最简单的是 ASCII 文本。ASCII（发音是“As-Key”）是美国信息交换标准码的简称。这个简单的编码方案最早用于电传打字机。

文本是字符与数字之间简单的一对一映射，它非常紧凑。由 50 个字符构成的文本在转换为数据时，也是 50 个字节。这与文字处理器文档中的文本是不一样的，比如由 Microsoft Word 或 OpenOffice.org Write 创建的文档。与简单的 ASCII 文本相比，那些文件包含了很多用来描述结构和格式的非文本元素。而普通的 ASCII 文本文件仅包含字符本身和一些基本的控制代码，比如制表符、回车符和换行符。

纵观 Linux 系统，很多文件是以文本格式存储的，并且也有很多 Linux 工具来处理文本文件。甚至连 Windows 系统也认识到这种格式的重要性。众所周知的记事本程序就是一款普通的 ASCII 文本文件编辑器。

less 命令的使用方式如下。

`less filename`

一旦运行起来，less 程序允许我们前后滚动文件。例如，想要查看定义了系统用户账户的文件，可输入下面的命令。

[me@linuxbox ~]\$ less /etc/passwd

一旦 less 程序运行起来，我们就可查看文件内容。如果文件不止一页，可以上下滚动文件。按 Q 键可退出 less 程序。

表 3-3 列出了 less 程序最常使用的键盘命令。

表 3-3 less 命令

命令	功能
PAGE UP 或 b	后翻一页
PAGE DOWN 或 Spacebar	前翻一页
向上箭头键	向上一行

续表

命令	功能
向下箭头键	向下一行
G	跳转到文本文件的末尾
1G 或 g	跳转到文本文件的开头
/characters	向前查找指定的字符串
n	向前查找下一个出现的字符串，这个字符串是之前所指定查找的
h	显示帮助屏幕
q	退出 less

少即是多 (LESS IS MORE)

设计 less 程序是为了替换早期 UNIX 中的 more 程序。less 这个名字是对短语 “less is more” 开了个玩笑，该短语是现代派建筑师和设计师们的座右铭。

less 命令属于“页面调度器 (pagers)”程序类，这些程序允许通过一页一页的方式，轻松浏览很长的文本文档。而 more 程序只允许向前翻页，使用 less 命令既可以前后翻页，还具有很多其他的特性。

3.4 快速浏览

在 Linux 系统中，文件系统布局与其他类 UNIX 系统很相似。实际上，一个已经发布的名为 Linux 文件系统层次标准 (Linux Filesystem Hierarchy Standard) 的标准，已经详细阐述了这个设计。并不是所有 Linux 发行版都严格符合该标准，但大部分与之很接近。

接下来，我们将通过对文件系统的探索来找到 Linux 系统正常运行所依赖的基础。这将提供一个练习导航技巧的机会。此时我们会发现，很多有趣的文件都是普通的可读文本。请尝试下面的步骤。

1. 使用 cd 命令进入一个给定的目录。
2. 使用 ls-l 命令列出目录内容。
3. 如果看到一个感兴趣的文件，使用 file 命令确定文件内容。
4. 如果文件看起来像是一个文本，试着使用 less 命令浏览其内容。

注意

牢记复制与粘贴技巧！如果使用鼠标的话，可以双击文件名来复制，中键单击将其粘贴进命令行。

当我们浏览文件系统时，不要担心将文件系统的布局弄得混乱不堪。普通用户不具有管理文件系统的权限，那是系统管理员的工作！如果一条命令无法执行某些功能，那么继续选择其他命令。多花些时间浏览一下。系统是需要大家探索的。记住，Linux 没有秘密可言。

表 3-4 只列出了一些探索到的目录，剩余的目录请大家自由地探索！

表 3-4 在 Linux 系统中找到的目录

目录	内容
/	根目录，一切从这里开始
/bin	包含系统启动和运行所必需的二进制文件（程序）
	包含 Linux 内核、最初的 RAM 磁盘映像（系统启动时，驱动程序会用到），以及启动加载程序
/boot	有趣的文件： <ul style="list-style-type: none">• /boot/grub/grub.conf 或 menu.lst，用来配置启动加载程序• /boot/vmlinuz，Linux 内核
/dev	这是一个包含设备节点的特殊目录。“把一切当成文件”也适用于设备。内核将它能够识别的所有设备存放在这个目录里
	/etc 目录包含了所有系统层面的配置文件，同时也包含了一系列 shell 脚本，系统每次启动时，这些 shell 脚本都会打开每个系统服务。该目录中包含的内容都应该是可读的文本文件。
/etc	有趣的文件：尽管/etc 目录中的任何文件都很有趣，但这里只列出了一些我一直喜欢的文件： <ul style="list-style-type: none">• /etc/crontab，该文件定义了自动化任务运行的时间• /etc/fstab，存储设备以及相关挂载点的列表• /etc/passwd，用户账号列表
/home	在通常的配置中，每个用户都会在/home 目录中拥有一个属于自己的目录。普通用户只能在自己的主目录中创建文件。这一限制可以保护系统免遭错误的用户行为的破坏
/lib	包含核心系统程序使用的共享库文件。这与 Windows 系统中的 DLL 类似
/lost+found	每个使用 Linux 文件系统的格式化分区或设备，例如 ext3 文件系统，都会有这个目录。当文件系统崩溃时，该目录用于恢复分区。除非系统真的发生很严重的问题，否则这个目录一直是空的
/media	在现代 Linux 系统中，/media 目录包含可移除媒体设备的挂载点。比如 USB 驱动、CD-ROM 等。这些设备在插入计算机后，会自动挂载到这个目录节点下
/mnt	在早期的 Linux 系统中，/mnt 目录包含手动挂载的可移除设备的挂接点
/opt	/opt 目录用来安装其他可选的软件。主要用来存放可能安装在系统中的商业软件
/proc	/proc 目录很特殊。从文件的角度来说，它不是存储在硬盘中的真正的文件系统，反而是一个 Linux 内核维护的虚拟文件系统。它包含的文件是内核的窥视孔。该文件是可读的，从中可以看到内核是如何监管计算机的
/root	root 账户的主目录
/sbin	该目录放置“系统”二进制文件。这些程序执行重要的系统任务，这些任务通常为超级用户预留的

续表

目录	内容
/tmp	/tmp 是供用户存放各类程序创建的临时文件的目录。某些配置使得每次系统重启时都会清空该目录
/usr	/usr 目录可能是 Linux 系统中最大的目录树。它包含普通用户使用的所有程序和相关文件
/usr/bin	/usr/bin 目录中放置了一些 Linux 发行版安装的可执行程序。该目录通常会存储成千上万个程序
/usr/lib	/usr/bin 目录中的程序使用的共享库
/usr/local	这个/usr/local 目录是并非系统发行版自带，但却打算让系统使用的程序的安装目录。由源代码编译好的程序通常安装在/usr/local/bin 中。在一个新安装的 Linux 系统中，就存在这一个目录，但却是空目录，直到系统管理员向其中添加内容
/usr/sbin	包含更多的系统管理程序
/usr/share	/usr/share 目录里包含了/usr/bin 中的程序所使用的全部共享数据，这包括默认配置文件、图标、屏幕背景、音频文件等
/usr/share/doc	安装在系统中的大部分程序包包含一些文档文件。在/usr/share/doc 中，文档文件是按照软件包来组织分类的
/var	除了/tmp 和/home 目录之外，目前看到的目录相对来说都是静态的；也就是说，其包含的内容是不变的。而那些可能改变的数据存储在/var 目录树里。各种数据库、假脱机文件、用户邮件等都存储在这里
/var/log	/var/log 目录包含的日志文件，记录了各种系统活动。这些文件非常重要，并且应该时不时地监控它们。其中最有用的文件是/var/log/messages。注意，为了安全起见，在一些系统里，必须是超级用户才能查看日志文件

3.5 符号链接

在浏览过程中，我们可能会看到带有如下条目的目录信息。

```
lrwxrwxrwx 1 root root 11 2012-08-11 07:34 libc.so.6 -> libc-2.6.so
```

注意，该条目信息的第一个字母是 1，而且看起来像是有两个文件名。这种特殊的文件叫做符号链接（又叫软链接或 symlink）。在大多类 UNIX 系统中，一个文件很可能采用多个名字来引用。虽然这种特性的意义并不明显，但是它真的很有用。

想象这样一个场景：一个程序需要使用包含在 foo 文件中的一个共享资源，但 foo 版本变化很频繁。这样，在文件名中包含版本号会是一个好主意，因此管理员或其他相关方就能够看到安装了 foo 的哪个版本。这就出现一个问题。如果改变了共享资源的名称，就必须跟踪每个可能使用了该共享资源的程序，

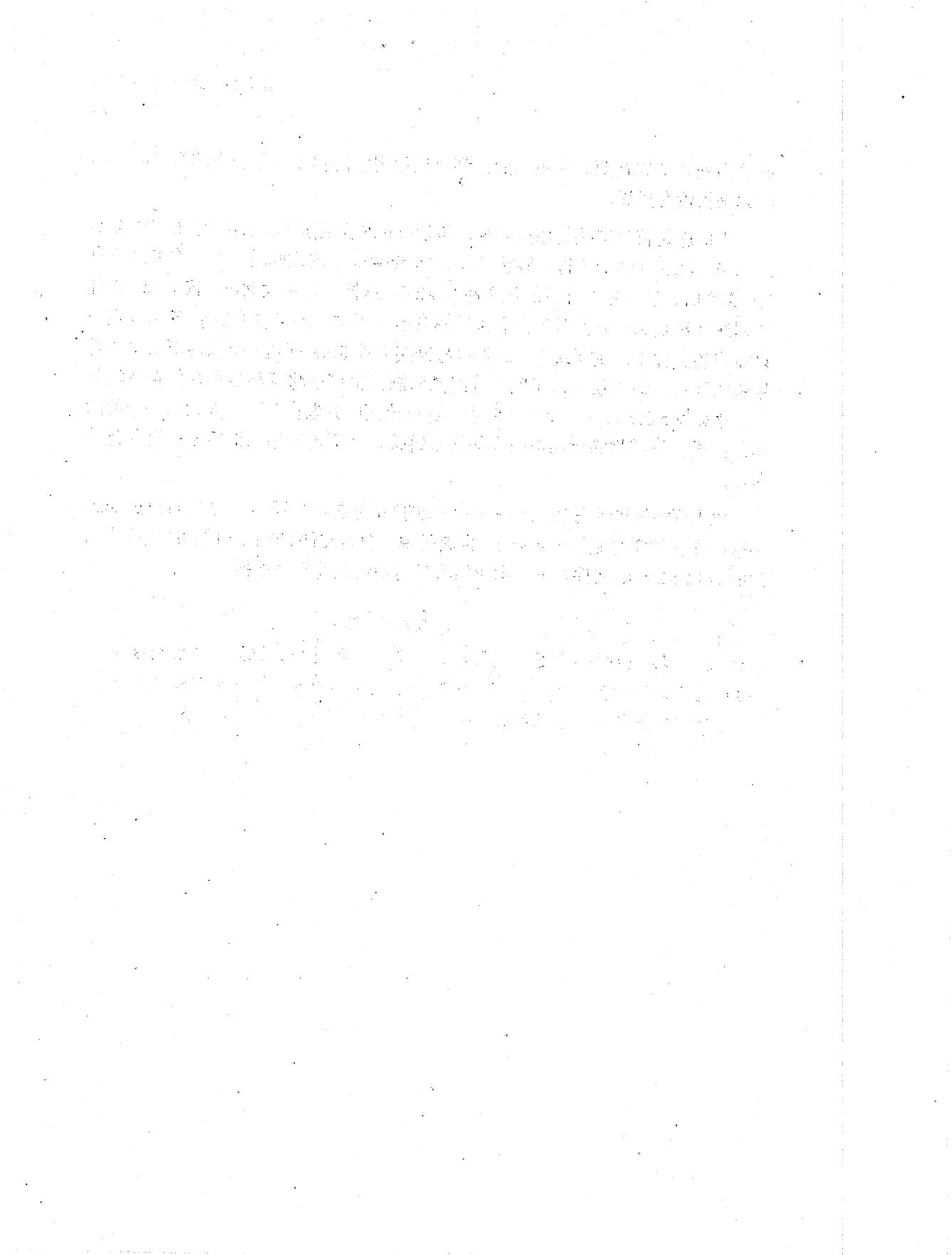
并且当安装了该资源新的版本后，都要让使用它的程序去寻找新的资源名。这听起来很没有意思。

下面就是符号链接的用武之处。假设 `foo` 的安装版本是 2.6，它的文件名是 `foo-2.6`，然后创建一个符号链接 `foo` 指向 `foo-2.6`。这意味着，当一个程序打开 `foo` 文件时，它实际上打开的是文件 `foo-2.6`，这样一来皆大欢喜。依赖 `foo` 文件的程序能够找到它，并且也能看到实际安装的版本。当需要升级到 `foo-2.7` 时，只需将该文件添加到系统里，删除符号链接文件 `foo`，创建一个指向新版本的符号链接即可。这不仅解决了版本升级的问题，也可以将两种版本都保存在机器里。假如 `foo-2.7` 存在一个程序错误（都怪该死的开发商！），需要切换到旧的版本。同样，只需删除指向新版本的符号链接，重新创建指向旧版本的符号链接即可。

以上列举的目录（来源于 Fedora 系统的 `/lib` 目录）中显示了一个指向 `libc-2.6.so` 共享库文件的符号链接 `libc.so.6`。也就是说，搜索文件 `libc.so.6` 的程序实际上访问的是 `libc-2.6.so` 文件。下一章我们将学习如何创建符号链接。

硬链接

既然在讨论链接问题，就需要提一下另一种类型的链接，即硬链接。它同样允许文件有多个名字，但是处理的方式是不同的。下一章我们将进一步讨论符号链接与硬链接的区别。



第 4 章

操作文件与目录

现在，我们准备好做些实际工作了！本章将介绍如下命令。

- `cp`: 复制文件和目录。
- `mv`: 移动或重命名文件和目录。
- `mkdir`: 创建目录。
- `rm`: 移除文件和目录。
- `ln`: 创建硬链接和符号链接。

这 5 个命令属于最常使用的 Linux 命令之列，可用来操作文件与目录。

坦率地说，使用图形文件管理器来执行一些由这些命令执行的任务要容易得多。使用文件管理器，我们可以将文件从一个目录拖放到另一个目录，我们可以剪切和粘贴文件，我们可以删除文件。那么，为什么还要用这些命令行程序呢？

原因就在于命令行程序具有强大的功能和灵活的操作。虽然使用图形文件

管理器能轻松实现简单的文件操作，但是对于复杂的任务，使用命令行程序更容易完成。例如，怎样仅因为文件在目标目录中不存在或存在旧的版本，就将所有 HTML 文件从一个目录复制到目标目录里呢？要完成这个任务，使用文件管理器则相当困难，而使用命令行则很容易。

```
cp -u *.html destination
```

4.1 通配符

在开始使用命令之前，我们需要介绍一个使命令行如此强大的 shell 特性。由于 shell 需要经常使用文件名，因此它提供了一些特殊字符来帮助你快速指定一组文件名。这些特殊字符称为通配符。通配符（也叫文件名替换）允许用户依据字符模式选择文件名。表 4-1 列出了通配符以及它们所选择的对象。

表 4-1 通配符

通配符	匹配项
*	匹配任意多个字符（包括 0 个和 1 个）
?	匹配任一单个字符（不包括 0 个）
[characters]	匹配任意一个属于字符集中的字符
[!characters]	匹配任意一个不属于字符集中的字符
[:class:]	匹配任意一个属于指定字符类中的字符

表 4-2 列出了最常用的字符类。

表 4-2 常用的字符类

字符类	匹配项
[:alnum:]	匹配任意一个字母或数字
[:alpha:]	匹配任意一个字母
[:digit:]	匹配任意一个数字
[:lower:]	匹配任意一个小写字母
[:upper:]	匹配任意一个大写字母

通配符的使用使得为文件名构建复杂的筛选标准成为可能。表 4-3 列出了一些通配符模式及其匹配内容的示例。

表 4-3 通配符示例

模式	匹配项
*	所有文件

续表

形式	匹配项
g*	以 g 开头的任一文件
b*.txt	以 b 开头，中间有任意多个字符，并以.txt 结尾的任一文件
Data???	以 Data 开头，后面跟 3 个字符的任一文件
[abc]*	以 abc 中的任一个开头的任一文件
BACKUP.[0-9][0-9][0-9]	以 BACKUP.开头，后面紧跟 3 个数字的任一文件
[:upper:]*	以大写字母开头的任一文件
![:digit:]*	不以数字开头的任一文件
*[:lower:]123]	以小写字母或数字 1、2、3 中的任一个结尾的任一文件

通配符可以与任一个使用文件名为参数的命令一起使用，在第 7 章我们会进一步讨论。

字符范围

如果你之前使用过其他的类 UNIX 环境或是读过该主题的相关书籍，可能遇到过[A-Z]或[a-z]形式的字符范围表示法。这些都是传统的 UNIX 表示法，在早期的 Linux 版本中仍然奏效。尽管它们仍然起作用，但使用时请务必小心，因为一旦配置不当，就会产生非预期的结果。目前，我们要避免使用它们，而是使用字符类。

通配符在 GUI 中也奏效

通配符相当有用，不仅仅因为它们在命令行中使用频繁，而且在于一些图形文件管理器也支持通配符操作。

- 在 Nautilus (GNOME 的文件管理器) 中，你可以使用 Edit->Select Pattern 选择文件。仅仅输入一个用通配符表示的文件选择模式后，则当前查看的目录中，所匹配的文件就会突出显示。
- 在 Dolphin 和 Konqueror (KDE 的文件管理器) 的一些版本中，你可以直接在地址栏输入通配符。比如，如果想要在/usr/bin 目录中查找所有以小写字符 u 开始的文件，只需在地址栏输入/usr/bin/u*，即可显示匹配的结果。

最初源于命令行界面的许多理念，也同样适用于图形界面。而这恰恰是使 Linux 系统桌面如此强大的原因之一。

4.2 mkdir——创建目录

`mkdir` 命令是用来创建目录的，格式如下。

`mkdir directory...`

注意表示法：本书在描述命令时，如果参数后面带有 3 个点号（如上所示），这表示该参数重复。因此，下面这种情况：

`mkdir dir1`

可创建单个 `dir1` 目录，而输入：

`mkdir dir1 dir2 dir3`

可创建 3 个目录，分别命名为 `dir1`、`dir2` 和 `dir3`。

4.3 cp——复制文件和目录

`cp` 命令用来复制文件和目录。它有两种不同的使用方式，如下所示。

`cp item1 item2`

将单个文件或目录 `item1` 复制到文件或目录 `item2` 中。

`cp item... directory`

将多个项目（文件或目录）复制进一个目录中。

表 4-4 和表 4-5 列出了 `cp` 常用的选项（短选项和等价的长选项）。

表 4-4 `cp` 命令选项

选项	含义
<code>-a, --archive</code>	复制文件和目录及其属性，包括所有权和权限。通常来说，复制的文件具有用户所操作文件的默认属性
<code>-i, --interactive</code>	在覆盖一个已存在的文件前，提示用户进行确认。如果没有指定该选项， <code>cp</code> 会默认覆盖文件
<code>-r, --recursive</code>	递归地复制目录及其内容。复制目录时需要这个选项（或-a 选项）
<code>-u, --update</code>	当将文件从一个目录复制到另一个目录时，只会复制那些目标目录中不存在的文件或是目标目录相应文件的更新文件
<code>-v, --verbose</code>	复制文件时，显示信息性消息（informative message）

表 4-5 cp 命令示例

命令	结果
<code>cp file1 file2</code>	将 file1 复制到 file2。如果 file2 存在，则会被 file1 的内容覆盖。如果 file2 不存在，则创建 file2
<code>cp -i file1 file2</code>	同上，区别在于当 file2 存在时，覆盖之前通知用户确认
<code>cp file1 file2 dir1</code>	将 file1 和 file2 复制到目录 dir1 里。dir1 必须已经存在
<code>cp dir1/* dir2</code>	通过使用通配符，将 dir1 中的所有文件复制到 dir2 中。dir2 必须已经存在
<code>cp -r dir1 dir2</code>	将 dir1 目录（及其内容）复制到 dir2 目录中。如果 dir2 不存在，创建 dir2，且包含与 dir1 目录相同的内容

4.4 mv——移除和重命名文件

mv 命令可以执行文件移动和文件重命名操作，这具体取决于如何使用它。在这两种情况下，完成操作之后，原来的文件名将不存在。mv 的使用方法与 cp 基本相似。

`mv item1 item2`

将文件（或目录）item1 移动（或重命名）为 item2，或是

`mv item... directory`

将一个或多个条目从一个目录移动到另一个目录下。

mv 命令很多选项与 cp 命令是共享的，如表 4-6 和表 4-7 所示。

表 4-6 mv 选项

选项	含义
<code>-i, --interactive</code>	覆盖一个已存在的文件之前，提示用户确认。如果没有指定该选项，mv 会默认覆盖文件
<code>-u, --update</code>	将文件从一个目录移动到另一个目录，只移动那些目标目录中不存在的文件或是目标目录里相应文件的更新文件
<code>-v, --verbose</code>	移动文件时显示信息性消息

表 4-7 mv 示例

命令	结果
<code>mv file1 file2</code>	将 file1 移到 file2。如果 file2 存在，则会被 file1 的内容覆盖。如果 file2 不存在，则创建 file2。无论哪一种情况，file1 不再存在
<code>mv -i file1 file2</code>	同上，仅当 file2 存在时，覆盖之前通知用户确认
<code>mv file1 file2 dir1</code>	将 file1 和 file2 移到目录 dir1 下。dir1 必须已经存在
<code>mv dir1 dir2</code>	将目录 dir1（和其内容）移到目录 dir2 下。如果目录 dir2 不存在，创建目录 dir2，将目录 dir1 的内容移到 dir2 下，同时删除目录 dir1

4.5 rm——删除文件和目录

`rm` 命令用来移除（删除）文件和目录，如下所示。

`rm item...`

其中，`item` 是一个或多个文件（或目录）的名称。

小心 `rm` 命令

类 UNIX 操作系统（如 Linux）并不包含还原删除操作的命令。一旦使用 `rm` 命令，就彻底地删除了。Linux 系统默认用户是明智的，并清楚自己在干什么。

`rm` 命令与通配符在一起使用时要特别小心。来看下面这个经典的示例。假设我们只希望删除目录中的 HTML 文件，为此需要输入以下正确的命令：

`rm *.html`

如果不小心在`*`与`.html`之间多打了一个空格，如下所示：

`rm * .html`

`rm` 命令将会删除目录中所有文件，并提示说明目录中没有叫做`.html`的文件。

提示：当 `rm` 命令与通配符一起使用时，除仔细检查输入内容外，可使用 `ls` 命令预先对通配符做出测试，这将显示欲删除的文件。紧接着，你可以按上下方向键调用之前的命令，并使用 `rm` 代替 `ls`。

表 4-8 和表 4-9 列出了 `rm` 命令的一些常用选项。

表 4-8 `rm` 选项

选项	含义
<code>-i, --interactive</code>	删除一个已存在的文件前，提示用户确认。如果没有指定这个选项， <code>rm</code> 命令会默认删除文件
<code>-r, --recursive</code>	递归地删除目录。也就是说，如果删除的目录有子目录的话，也要将其删除。要删除一个目录，则必须指定该选项
<code>-f, --force</code>	忽略不存在的文件并无需提示确认。该选项会覆盖 <code>--interactive</code> 选项
<code>-v, --verbose</code>	删除文件时显示信息性消息

表 4-9 rm 实例

命令	结果
rm file1	在不提示用户的情况下，删除 file1
rm -i file1	删除 file1 前，提示用户确认
rm -r file1 dir1	删除 file1、dir1 以及它们的内容
rm -rf file1 dir1	同上，当时在 file1 或 dir1 不存在时，rm 仍会继续执行，且不提示用户

4.6 ln——创建链接

ln 命令可用来创建硬链接或是符号链接。它的使用方式有两种。

`ln file link`

用来创建硬链接，而

`ln -s item link`

用来创建符号链接，这里的 item 可以是文件也可以是目录。

4.6.1 硬链接

硬链接是最初 UNIX 用来创建链接的方式，符号链接较之更为先进。默认情况下，每个文件有一个硬链接，该硬链接会给文件起名字。当创建一个硬链接的时候，也为这个文件创建了一个额外的目录条目。硬链接有两条重要的局限性。

- 硬链接不能引用自身文件系统之外的文件。也就是说，链接不能引用与该链接不在同一磁盘分区的文件。
- 硬链接无法引用目录。

硬链接和文件本身没有什么区别。与包含符号链接的目录列表不同，包含硬链接的目录列表没有特别的链接指示说明。当硬链接被删除时，只是删除了这个链接，但是文件本身的内容依然存在（也就是说，该空间没有释放），除非该文件的所有链接都被删除了。

因为会经常遇到它们，了解硬链接就显得特别重要。但是现在大多使用的是符号链接，下面将会有所介绍。

4.6.2 符号链接

符号链接是为了克服硬链接的局限性而创建的。符号链接是通过创建一个特殊类型的文件来起作用的，该文件包含了指向引用文件或目录的文本指针。就这点来看，符号链接与 Windows 系统下的快捷方式非常相似，但是，符号链接要早于 Windows 的快捷方式很多年。

符号链接指向的文件与符号链接自身几乎没有区别。例如，将一些东西写进符号链接里，那么这些东西同样也写进了引用文件。而当删除一个符号链接时，删除的只是符号链接而没有删除文件本身。如果先于符号链接之前删除文件，那么这个链接依然存在，但却不指向任何文件。此时，这个链接就称为坏链接。在很多实现中，ls 命令会用不同的颜色来显示坏链接，比如红色，从而显示它们的存在。

链接的概念似乎很令人迷惑，但是不要害怕。我们要经常性地使用，它们慢慢的就会清晰起来。

4.7 实战演练

由于我们要做一些实际的文件操作，我们先来创建一个安全的地带，来执行文件操作命令。首先，我们需要一个工作目录。我们在主目录里创建一个目录并命名为 playground。

4.7.1 创建目录

mkdir 命令用来创建一个目录。为了创建 playground 目录，我们首先要保证当前目录是主目录，然后再创建新目录。

```
[me@linuxbox ~]$ cd  
[me@linuxbox ~]$ mkdir playground
```

为了让我们的实战演练更有意思，我们在 playground 目录中新建两个目录，命名为 dir1、dir2。为此，我们应先将当前的工作目录切换为 playground，然后再次执行 mkdir 命令。

```
[me@linuxbox ~]$ cd playground  
[me@linuxbox playground]$ mkdir dir1 dir2
```

需要注意的是，mkdir 命令可以接受多个参数，从而允许我们用一个命令创建两个目录。

4.7.2 复制文件

接下来，让我们在创建的目录中放入一些数据。这一过程可通过文件复制来完成。通过使用 cp 命令，我们可以将/etc 目录中的 passwd 文件复制到当前工作目录里。

```
[me@linuxbox playground]$ cp /etc/passwd .
```

注意我们是如何使用当前工作目录的快捷方式的，即在命令末尾加单个句点。如果我们此时执行 ls 命令，将会看到我们的文件。

```
[me@linuxbox playground]$ ls -l
total 12
drwxrwxr-x 2 me me 4096 2012-01-10 16:40 dir1
drwxrwxr-x 2 me me 4096 2012-01-10 16:40 dir2
-rw-r--r-- 1 me me 1650 2012-01-10 16:07 passwd
```

现在让我们使用-v 选项，重复操作复制命令，来看看结果如何。

```
[me@linuxbox playground]$ cp -v /etc/passwd .
'/etc/passwd' -> './passwd'
```

cp 命令再次执行复制操作，但是，这一次显示了一条简洁的信息来指明它正在执行什么操作。需要注意的是，在没有任何警告的情况下，cp 命令覆盖了第一次的复制内容。cp 命令会假定用户清楚自己当前的操作。加上-i（交互式）选项可以获得警告信息。

```
[me@linuxbox playground]$ cp -i /etc/passwd .
cp: overwrite './passwd'?
```

通过在提示符下输入 y，文件就会被重写；任何其他的字符（比如，n）会使 cp 命令保留该文件。

4.7.3 移动和重命名文件

现在，passwd 这个名字似乎没有那么有趣，而我们毕竟是在进行实战演练，因此我们给它改个名字。

```
[me@linuxbox playground]$ mv passwd fun
```

现在传送 fun 文件，这是通过将重命名的文件移动到各个目录，然后再移动回当前目录来实现的：

```
[me@linuxbox playground]$ mv fun dir1
```

首先移到目录 dir1 下，然后：

```
[me@linuxbox playground]$ mv dir1/fun dir2
```

将文件从目录 dir1 移到 dir2，然后：

```
[me@linuxbox playground]$ mv dir2/fun .
```

再将文件 fun 重新移到当前工作目录下。下面来看 mv 命令的效果。首先，再次将数据文件移到目录 dir1。

```
[me@linuxbox playground]$ mv fun dir1
```

然后将目录 dir1 移到 dir2 并且使用 ls 命令进行确认。

```
[me@linuxbox playground]$ mv dir1 dir2
[me@linuxbox playground]$ ls -l dir2
total 4
drwxrwxr-x 2 me me 4096 2012-01-11 06:06 dir1
[me@linuxbox playground]$ ls -l dir2/dir1
total 4
-rw-r--r-- 1 me me 1650 2012-01-10 16:33 fun
```

注意，因为目录 dir2 已经存在，mv 命令将目录 dir1 移到 dir2。如果 dir2 不存在，mv 将 dir1 重命名为 dir2。最后，我们将所有东西放回原处。

```
[me@linuxbox playground]$ mv dir2/dir1 .
[me@linuxbox playground]$ mv dir1/fun .
```

4.7.4 创建硬链接

现在，我们试着创建一些链接。首先是创建硬链接，我们先按照如下方式创建一些指向数据文件的链接：

```
[me@linuxbox playground]$ ln fun fun-hard
[me@linuxbox playground]$ ln fun dir1/fun-hard
[me@linuxbox playground]$ ln fun dir2/fun-hard
```

目前有 4 个文件 fun 的实例。来看一下 playground 目录。

```
[me@linuxbox playground]$ ls -l
total 16
drwxrwxr-x 2 me me 4096 2012-01-14 16:17 dir1
drwxrwxr-x 2 me me 4096 2012-01-14 16:17 dir2
-rw-r--r-- 4 me me 1650 2012-01-10 16:33 fun
-rw-r--r-- 4 me me 1650 2012-01-10 16:33 fun-hard
```

可以注意到，在列表中，文件 fun 和 fun-hard 的第二个字段都是 4，这是文件 fun 存在的硬链接数目。你要记得，由于文件名是由链接创建的，所以一个文件通常至少有一个链接。那么，我们是如何知道 fun 和 fun-hard 实际上是同一个文件的呢？这种情况下，ls 命令无济于事。虽然从第 5 个字段得知 fun 和

`fun-hard` 文件大小相同，但是我们的列表并没有提供可靠的方式来确认它们是否是同一个文件。要解决这个问题，必须做进一步研究。

提到硬链接时，可以想象文件是由两部分组成的，即包含文件内容的数据部分和包含文件名的名称部分。创建硬链接时，实际上是创建了额外的名称，这些名称都指向同一数据部分。系统分配了一系列的盘块给所谓的索引节点（inode），该节点随后与文件名称部分建立关联。因此，每个硬链接都指向包含文件内容的具体索引节点。

`ls` 命令有一种方式可以显示上述信息。它是通过在命令中加上`-i` 选项来实现的。

```
[me@linuxbox playground]$ ls -li
total 16
12353539 drwxrwxr-x 2 me me 4096 2012-01-14 16:17 dir1
12353540 drwxrwxr-x 2 me me 4096 2012-01-14 16:17 dir2
12353538 -rw-r--r-- 4 me me 1650 2012-01-10 16:33 fun
12353538 -rw-r--r-- 4 me me 1650 2012-01-10 16:33 fun-hard
```

在这个列表中，第一个字段就是索引节点号，可以看到，`fun` 和 `fun-hard` 共享同一个索引节点号，这就证实它们是相同的文件。

4.7.5 创建符号链接

之所以创建符号链接，是为了克服硬链接的两大不足，即硬链接无法跨越物理设备，也无法引用目录，只能引用文件。符号链接是一种特殊类型的文件，它包含了指向目标文件或目录的文本指针。

创建符号链接与创建硬链接相似，如下所示。

```
[me@linuxbox playground]$ ln -s fun fun-sym
[me@linuxbox playground]$ ln -s ../fun dir1/fun-sym
[me@linuxbox playground]$ ln -s ../fun dir2/fun-sym
```

第一个命令相当直接，我们只是在 `ln` 命令中添加`-s` 选项，就可以创建符号链接而不是硬链接。但是，接下来的两个命令是什么呢？牢记一点，创建符号链接时，同时也创建一个文本来描述目标文件在哪里与与符号链接有关联。如果看看 `ls` 命令的输出就更容易明白了。

```
[me@linuxbox playground]$ ls -l dir1
total 4
-rw-r--r-- 4 me me 1650 2012-01-10 16:33 fun-hard
lrwxrwxrwx 1 me me       6 2012-01-15 15:17 fun-sym -> ../fun
```

在目录 `dir1` 中，`fun-sym` 的列表显示它是一个符号链接，这是通过第 1 个

字段中的首字符“1”来确认的，并且它也指“`../fun`”，这也是正确的。相对于`fun-sym`的实际位置，文件`fun`在它的上一级目录。还要注意到，符号链接文件的长度是6，这是“`../fun`”字符串中字符的数目，而不是它所指向的文件的长度。

创建符号链接时，可以使用绝对路径名，如下所示：

```
[me@linuxbox playground]$ ln -s /home/me/playground/fun dir1/fun-sym
```

也可以使用相对路径，如前面的示例所示。因为相对路径允许包含符号链接的目录被重命名和/或移动，而且不会破坏链接，因此更可取一些。

除了普通文件之外，符号链接也可以引用目录。

```
[me@linuxbox playground]$ ln -s dir1 dir1-sym
[me@linuxbox playground]$ ls -l
total 16
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir1
lrwxrwxrwx 1 me me 4 2012-01-16 14:45 dir1-sym -> dir1
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir2
-rw-r--r-- 4 me me 1650 2012-01-10 16:33 fun
-rw-r--r-- 4 me me 1650 2012-01-10 16:33 fun-hard
lrwxrwxrwx 1 me me 3 2012-01-15 15:15 fun-sym -> fun
```

4.7.6 移除文件和目录

前面讲到，使用`rm`命令可以删除文件和目录。那么我们就用它来清空`playground`目录。首先，我们删除目录中的一个硬链接。

```
[me@linuxbox playground]$ rm fun-hard
[me@linuxbox playground]$ ls -l
total 12
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir1
lrwxrwxrwx 1 me me 4 2012-01-16 14:45 dir1-sym -> dir1
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir2
-rw-r--r-- 3 me me 1650 2012-01-10 16:33 fun
lrwxrwxrwx 1 me me 3 2012-01-15 15:15 fun-sym -> fun
```

不出所料，文件`file-hard`被删除了，文件`fun`的链接数相应的也由4变成了3，如目录列表的第二个字段所示。接下来，我们删除文件`fun`，为了好玩，我们还会加上`-i`选项，看看执行了哪些操作。

```
[me@linuxbox playground]$ rm -i fun
rm: remove regular file 'fun'?
```

在提示符下输入字符`y`，文件就被删除了。现在看一下`ls`命令的输出。注意`fin-sym`文件发生了什么变化？由于它是一个符号链接，且指向的文件现在已经不存在，所以链接也就破坏了。

```
[me@linuxbox playground]$ ls -l
total 8
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir1
lrwxrwxrwx 1 me me      4 2012-01-16 14:45 dir1-sym -> dir1
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir2
lrwxrwxrwx 1 me me      3 2012-01-15 15:15 fun-sym -> fun
```

大多数 Linux 发行版会配置 ls 命令，来显示破坏的链接。在 Fedora 系统中，破坏的链接是以闪烁的红色来显示的！受破坏的链接并不危险，但是会相当混乱麻烦。如果试图调用破坏的链接，将会看到如下情况：

```
[me@linuxbox playground]$ less fun-sym
fun-sym: No such file or directory
```

稍微清理一下。我们准备删除符号链接。

```
[me@linuxbox playground]$ rm fun-sym dir1-sym
[me@linuxbox playground]$ ls -l
total 8
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir1
drwxrwxr-x 2 me me 4096 2012-01-15 15:17 dir2
```

有关符号链接，需要记住一点，即大部分文件操作是以链接目标为对象的，而非链接本身。而 rm 命令是个例外。当删除一个链接的时候，链接本身被删除，但是目标文件依旧存在。

最后，我们需要删除目录 playground。为此，我们将返回主目录，使用 rm 命令的递归选项 (-r) 来删除 playground 目录以及包括子目录在内的所有内容。

```
[me@linuxbox playground]$ cd
[me@linuxbox ~]$ rm -r playground
```

使用 GUI 创建符号链接

GNOME 和 KDE 中的文件管理器提供了一种自动创建符号链接的简单方法。在 GNOME 环境下，拖拽文件时同时按住 Ctrl-Shift 键将会新建链接文件，而不是执行复制（移动）操作。在 KDE 环境下，无论什么时候放下 (drop) 一个文件都会弹出一个小菜单，它提供了复制、移动或创建链接文件等选项。

4.8 本章结尾语

到此为止，我们已经学习了大量的基础知识，可能要花一段时间来完全消

化吸收。我们要反复地操作 `playground` 实例，直到完全掌握。其中，能够很好地理解基本的文件操作命令和通配符是很重要的。我们可以通过增加文件和目录来自由地拓展 `playground` 实例，使用通配符来指明各种操作需要的文件。刚开始的时候，链接的概念可能有些模糊，但在花些时间学习之后，我们就会发现它真的是大有裨益。

第 5 章

命令的使用

到此为止，我们已经学习了多个神秘的命令，而且每个命令带有神秘的选项和参数。在本章中，我们将进一步揭开命令的神秘面纱，甚至尝试创建自己的命令。本章中介绍的命令如下。

- **type:** 说明如何解释命令名。
- **which:** 显示会执行哪些可执行程序。
- **man:** 显示命令的手册页。
- **apropos:** 显示一系列合适的命令。
- **info:** 显示命令的 info 条目。
- **whatis:** 显示一条命令的简述。
- **alias:** 创建一条命令的别名。

5.1 究竟是什么命令

一条命令不外乎以下 4 种情况。

- **可执行程序。** 可执行程序就像在 /usr/bin 目录里看到的所有文件一样。在该程序类别中，程序可以编译为二进制文件，比如 C、C++ 语言编写的程序，也可以是 shell、Perl、Python、Ruby 等脚本语言编写的程序。
- **shell 内置命令。** .bash 支持许多在内部称之为 shell builtin 的内置命令。例如，cd 命令就是 shell 内置指令。
- **shell 函数。** shell 函数是合并到环境变量中的小型 shell 脚本。在后面的章节中，我们将讨论环境变量的配置以及 shell 函数的编写，但是目前我们只需知道它们的存在就好了。
- **alias 命令。** 我们可以在其他命令的基础上定义自己的命令。

5.2 识别命令

能够准确地识别我们使用的命令是上述 4 种命令类型中的哪一种是很有用的。为此，Linux 提供了两个方法来识别命令类型。

5.2.1 type——显示命令的类型

`type` 命令是一个 shell 内置命令，可根据指定的命令名显示 shell 将要执行的命令类型。格式如下。

```
type command
```

这里的 `command` 是想要查看的命令名。一些实例如下所示。

```
[me@linuxbox ~]$ type type
type is a shell builtin
[me@linuxbox ~]$ type ls
ls is aliased to 'ls --color=tty'
[me@linuxbox ~]$ type cp
cp is /bin/cp
```

这里将看到 3 种不同命令的查看结果。需要注意的是，`ls` 命令（来自 Fedora 系统）实际上是带有 `--color=tty` 选项的 `ls` 命令的别名。现在我们知道了 `ls` 命令的输出为什么会有颜色了。

5.2.2 which——显示可执行程序的位置

有时候，系统中可能会安装了一个可执行程序的多个版本。这种现象虽然在桌面系统中不常见，但是在大型服务器中却是很常见的。使用 which 命令可以确定一个给定可执行文件的准确位置。

```
[me@linuxbox ~]$ which ls
/bin/ls
```

which 命令只适用于可执行程序，而不适用于内置命令和命令别名（真正可执行程序的替代物）。试图在 shell 内置命令（例如，cd）中使用 which 命令时，要么没有响应，要么得到一条错误信息：

```
[me@linuxbox ~]$ which cd
/usr/bin/which: no cd in (/opt/jre1.6.0_03/bin:/usr/lib/qt-3.3/bin:/usr/kerberos/bin:/opt/jre1.6.0_03/bin:/usr/lib/ccache:/usr/local/bin:/usr/bin:/bin:/home/me/bin)
```

which 命令是一种是“无法找到命令”的奇特方式。

5.3 获得命令文档

了解了什么是命令后，我们可以查看每一类命令的可用文档。

5.3.1 help——获得 shell 内置命令的帮助文档

bash 为每一个 shell 内置命令提供了一个内置的帮助工具。输入 help，然后输入 shell 内置命令的名称即可使用该帮助工具。例如：

```
[me@linuxbox ~]$ help cd
cd: cd [-L|-P] [dir]
Change the current directory to DIR. The variable $HOME is the default DIR.
The variable CDPATH defines the search path for the directory containing DIR.
Alternative directory names in CDPATH are separated by a colon (:). A null
directory name is the same as the current directory, i.e. '.'. If DIR begins
with a slash (/), then CDPATH is not used. If the directory is not found, and
the shell option 'cdable_vars' is set, then try the word as a variable name.
If that variable has a value, then cd to the value of that variable. The -P
option says to use the physical directory structure instead of following
symbolic links; the -L option forces symbolic links to be followed.
```

注意表示法：出现在命令语法描述中的方括号表示一个可选的选项。竖线符号代表的是两个互斥的选项。比如上边的 cd 命令：cd [-L|-P] [dir]。

这种表示法说明，cd 命令后可能有一个-L 参数，也可能是-P 参数，甚至可以跟

参数 dir。

尽管 cd 命令的帮助文档简明而又准确，但这绝不是一个辅导教程，我们所可以看到，帮助文档中也似乎提到了很多我们还没有讨论到的内容！别担心，稍后我们会详加说明。

5.3.2 help——显示命令的使用信息

很多可执行程序都支持--help 选项，--help 选项描述了命令支持的语法和选项。例如：

```
[me@linuxbox ~]$ mkdir --help
Usage: mkdir [OPTION] DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

-Z, --context=CONTEXT (SELinux) set security context to CONTEXT
Mandatory arguments to long options are mandatory for short options too.
-m, --mode=MODE    set file mode (as in chmod), not a=rwx - umask
-p, --parents no   error if existing, make parent directories as
                  needed
-v, --verbose     print a message for each created directory
--help            display this help and exit
--version         output version information and exit
Report bugs to <bug-coreutils@gnu.org>.
```

一些程序不支持--help 选项，但是我们还是要试试。这通常会产生一条错误消息，该错误消息也能揭示相同的命令使用信息。

5.3.3 man——显示程序的手册页

大多数供命令行使用的可执行文件，提供一个称之为 manual 或者是 man page 的正式文档。该文档可以用一种称为 man 的特殊分页程序来查看，用法如下。

`man program`

这里的 program 是需要查看的命令名称。

手册文档在格式上会有所不同，但是通常都包括标题、命令句法的摘要、命令用途的描述、命令选项列表以及每个命令选项的描述。但是，手册文档通常不包括实例，更多的是作为一个参考使用，而不是教程。例如，尝试查看 ls 命令的手册文档。

```
[me@linuxbox ~]$ man ls
```

在大多数 Linux 系统中，man 命令调用 less 命令来显示手册文档。所以，当显示手册文档时，你熟悉的所有 less 命令都能奏效。

`man` 命令显示的“手册文档”被分成多个部分（section），它不仅包括用户命令，也包括系统管理命令、程序接口、文件格式等。表 5-1 描述了手册文档的结构安排。

表 5-1 手册文档的组织结构

部分	内容
1	用户命令
2	内核系统调用的程序接口
3	C 库函数程序接口
4	特殊文件，如设备节点和驱动程序
5	文件格式
6	游戏和娱乐，例如屏幕保护程序
7	其他杂项
8	系统管理命令

有时候我们需要查看手册文档的具体部分，以查找我们需要的信息。当我们所查找的一个文件格式同时也是一个命令名的时候，这一点就尤为重要了。如果没有指明部分编号（section number），通常我们会获得第一次匹配的实例（它可能会出现在第一部分）。为了指明具体在哪个部分，我们可以这样使用 man 命令。

man section search term

例如：

```
[me@linuxbox ~]$ man 5 passwd
```

该命令将会显示文件/etc/passwd 的文件格式描述手册。

5.3.4 apropos——显示合适的命令

我们有可能会搜索参考手册列表，才进行基于某个搜索条目的匹配。尽管有些粗糙，但是这种方法有时还是很有用的。下面是一个使用 `floppy` 为搜索条目，来搜索参考手册的例子。

```
[me@linuxbox ~]$ apropos floppy
create_floppy_devices (8) - udev callout to create all possible
                           floppy device based on the CMOS type
fdformat                (8) - Low-level formats a floppy disk
floppy                  (8) - format floppy disks
gfloppy                 (1) - a simple floppy formatter for the GNOME
mbadblocks              (1) - tests a floppy disk, and marks the bad
                           blocks in the FAT
mformat                 (1) - add an MSDOS filesystem to a low-level
                           formatted floppy disk
```

在输出中，每一行的第一个字段是手册页的名称，第二个字段显示部分（section）。注意，带有-k 选项的 man 命令与 apropos 命令在功能上基本是一致的。

5.3.5 whatis——显示命令的简要描述

whatis 程序显示匹配具体关键字的手册页的名字和一行描述。

```
[me@linuxbox ~]$ whatis ls
ls          (1) - list directory contents
```

最难以忍受的参考手册

我们已经看到，Linux 和其他类 UNIX 系统中的手册文档是作为参考文档而非教程来使用的。很多手册文档都难以阅读，其中 bash 提供的手册页最为困难。在本书的编写过程中，我仔细复查，以确保覆盖了手册文档中的大部分主题。如果打印出来的话，将超过 80 页，而且排版非常紧凑，对一个初学者而言，这种结构是毫无意义的。

另一方面，bash 手册文档非常准确，也非常简要，同时也相当完备。所以，如果有胆量就去尝试一下，并期待有一天能读懂它。

5.3.6 info——显示程序的 info 条目

GNU 项目提供了 info 页面来代替手册文档。info 页面可通过 info 阅读器来显示。info 页面使用超链接，这与网页结构很相似。实例如下。

```
File: coreutils.info, Node: ls invocation, Next: dir invocation, Up:
Directory listing
=====
10.1 'ls': List directory contents
=====

The 'ls' program lists information about files (of any type, including
directories). Options and file arguments can be intermixed arbitrarily, as
usual.

For non-option command-line arguments that are directories, by default 'ls'
lists the contents of directories, not recursively, and omitting files with
names beginning with '.'. For other non-option arguments, by default 'ls'
lists just the filename. If no non-option argument is specified, 'ls' operates
on the current directory, acting as if it had been invoked with a single
argument of ''.

By default, the output is sorted alphabetically, according to the
--zz-Info: (coreutils.info.gz)ls invocation, 63 lines --Top-----
```

info 程序读取 info 文件，该文件是树形结构，分为各个单独的节点，每一个节点包含一个主题。info 文件包含的超链接可以实现节点间的跳转。通过前置星号可以识别超链接，将光标放在超链接上并按 Enter 键，可以激活它。

可以通过输入 info 以及程序名（可选的）来调用 info。表 5-2 列出了显示 info 页面时，用于控制阅读器的命令。

表 5-2 info 命令

命令	功能
?	显示命令帮助
PAGE UP or BACKSPACE	返回上一页
PAGE DOWN or Spacebar	翻到下一页
n	Next——显示下一个节点
p	Previous——显示上一个节点
u	Up——显示目前显示节点的父节点（通常是一个菜单）
ENTER	进入光标所指的超链接
q	退出

到目前为止，我们讨论的大部分命令行程序都是 GNU 项目 coreutils 包的一部分，输入以下内容可以看到更多的信息。

[me@linuxbox ~]\$ info coreutils

我们将会看到一个菜单页面，该菜单页面包含了 coreutils 包提供的每个程序的文档的超链接。

5.3.7 README 和其他程序文档文件

系统中安装的很多软件包都有自己的文档文件，它们存放在 /usr/share/doc 目录中。其中大部分文档文件是以纯文本格式存储的，因此可以用 less 命令来查看。有些文件是 HTML 格式，并且可以用 Web 浏览器来查看。我们可能会遇到一些以.gz 扩展名结尾的文件。这表明它们是使用 gzip 压缩程序压缩过的。gzip 包包含一个特殊的 less 版本，称之为 zless。zless 可以显示由 gzip 压缩的文本文件的内容。

5.4 使用别名创建自己的命令

现在我们可以尝试编写程序了！我们可以使用 alias 命令来创建自己的命令。

但是在开始之前，我们需要展示一个命令行的小技巧。通过使用分号来分隔多条命令，就可以将多条命令输入在一行中。其工作方式如下：

```
command1;command2;command3...
```

我们将要使用的例子如下：

```
[me@linuxbox ~]$ cd /usr; ls; cd -
bin games    kerberos lib64   local share tmp
etc include   lib      libexec sbin   src
/home/me
[me@linuxbox ~]$
```

可以看到，我们将 3 条命令放置在同一行中。首先我们将当前目录改变成 /usr，然后列出这个目录内容，最后返回到原始目录（使用 cd-）。那么程序结束的位置恰恰是开始的位置。现在，我们通过使用 alias 命令将以上命令整合成一条新的命令。首先要为新命令虚构出一个名称，试试名称 test。不过输入前，我们最好检查一下名称 test 是否已经被使用过了。为此，我们可以再次使用 type 命令。

```
[me@linuxbox ~]$ type test
test is a shell builtin
```

啊哦！这个名字已经用过了，试试 foo。

```
[me@linuxbox ~]$ type foo
bash: type: foo: not found
```

太好了！foo 还没有被使用。下面创建新命令的别名。

```
[me@linuxbox ~]$ alias foo='cd /usr; ls; cd -'
```

注意这个命令的结构。

```
alias name='string'
```

在 alias 命令之后输入 name，紧接着是一个等号（没有空格），等号之后是一个用单引号括起来的字符串，该字符串中的内容将赋值给 name。定义好的别名可以用在 shell 期待的任何地方。

尝试如下命令：

```
[me@linuxbox ~]$ foo
bin games    kerberos lib64   local share tmp
etc include   lib      libexec sbin   src
/home/me
[me@linuxbox ~]$
```

也可以再次使用 type 命令来查看别名。

```
[me@linuxbox ~]$ type foo  
foo is aliased to 'cd /usr; ls ; cd -'
```

要删除别名，可以使用 unalias 命令，如下所示。

```
[me@linuxbox ~]$ unalias foo  
[me@linuxbox ~]$ type foo  
bash: type: foo: not found
```

尽管我们有意避免使用已经存在的命名名称来给我们的别名命名，但有时也会期待这么做。这样做的目的是，为每一个经常调用的命令添加一个普遍会用到的选项。例如，前面讲到的为 ls 命令添加别名，以添加颜色支持。

```
[me@linuxbox ~]$ type ls  
ls is aliased to 'ls --color=tty'
```

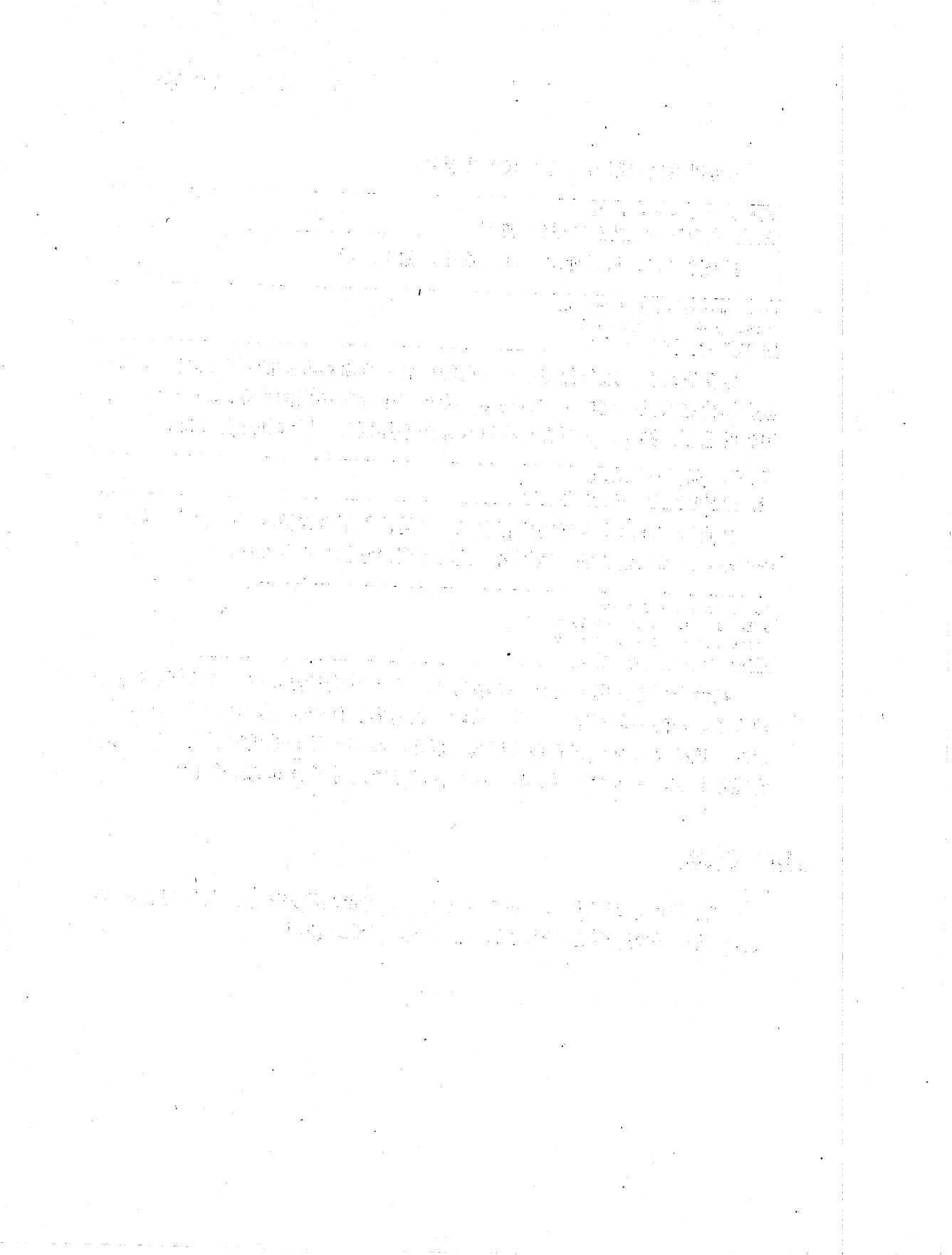
要查看在环境中定义的所有别名，可是使用不带参数的 alias 命令。以下是 Fedora 系统默认定义的一些别名。试着弄明白它们是干什么的。

```
[me@linuxbox ~]$ alias  
alias l.='ls -d .* --color=tty'  
alias ll='ls -l --color=tty'  
alias ls='ls --color=tty'
```

在命令行定义别名还有一个小问题。当 shell 会话结束时，这些别名也随之消失了。在随后的章节中，我们将学习如何向文件中添加别名。每一次登录系统时，这些文件都会建立系统环境。现在，我们已经开始迈出了第一步，纵然它微不足道，可毋庸置疑的是，现在我们已经走进了 shell 编程的世界。

5.5 温故以求新

我们已经学习了如何查找命令文档，现在我们就来查看之前遇到的所有命令的文档，学习一下这些命令的其他选项，并练习使用。



第 6 章

重 定 向

本章我们将要探讨命令行最酷的功能——I/O 重定向。I/O 是输入/输出 (input/output) 的缩写。这个功能可以把命令行的输入重定向为从文件中获取内容，也可以把命令行的输出结果重定向到文件中。如果我们将多个命令行关联起来，将形成非常强大的命令——管道。接下来，我们将通过介绍以下命令来展示这一功能。

- **cat:** 合并文件。
- **sort:** 对文本行排序。
- **uniq:** 报告或删除文件中重复的行。
- **wc:** 打印文件中的换行符、字和字节的个数。
- **grep:** 打印匹配行。
- **head:** 输出文件的第一部分内容。
- **tail:** 输出文件的最后一部分内容。

- **tee**: 读取标准输入的数据，并将其内容输出到标准输出和文件中。

6.1 标准输入、标准输出和标准错误

到目前为止，我们使用过的很多程序生成了不同种类的输出。这些输出通常包含两种类型。一种是程序运行的结果，即该程序生成的数据；另一种是状态和错误信息，表示程序当前的运行情况。比如输入 `ls` 命令，屏幕上将显示它的运行结果以及它的相关错误信息。

与 UNIX “一切都是文件”的思想一致，类似 `ls` 的程序实际上把它们的运行结果发送到了一个称为标准输出（standard output，通常表示为 `stdout`）的特殊文件中，它们的状态信息则发送到了另一个称为标准错误（standard error，表示为 `stderr`）的文件中。默认情况下，标准输出和标准错误都将被链接到屏幕上，并且不会被保存在磁盘文件中。

另外，许多程序从一个称为标准输入（standard input，表示为 `stdin`）的设备来得到输入。默认情况下，标准输入连接到键盘。

I/O 重定向功能可以改变输出内容发送的目的地，也可以改变输入内容的来源地。通常来说，输出内容显示在屏幕上，输入内容来自于键盘。但是使用 I/O 重定向功能可以改变这一惯例。

6.1.1 标准输出重定向

I/O 重定向功能可以重新定义标准输出内容发送到哪里。使用重定向操作符“>”，后面接文件名，就可以把标准输出重定向到另一个文件中，而不是显示在屏幕上。为什么我们需要这样做呢？它主要用于把命令的输出内容保存到一个文件中。比如，我们可以按照下面的形式把 `ls` 命令的输出保存到 `ls-output.txt` 文件中，而不是输出到屏幕上。

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

这里，我们将创建`/usr/bin` 目录的一个长列表信息，并把这个结果输出到 `ls-output.txt` 文件中。检查下该命令被重定向的输出内容。

```
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me me 167878 2012-02-01 15:07 ls-output.txt
```

这是一个不错的大型文本文件。如果使用 `less` 命令查看这个文件，我们可以看到 `ls-output.txt` 文件确实包含了 `ls` 命令的执行结果。

```
[me@linuxbox ~]$ less ls-output.txt
```

现在，让我们重复重定向测试，但是这次做一点变换。我们把目录名称换成一个不存在的目录。

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt
ls: cannot access /bin/usr: No such file or directory
```

我们会收到一条错误信息。因为我们指定的是一个不存在的目录/bin/usr，所以这个错误信息是正确的。但是为什么这个错误信息显示在屏幕上，而不是重定向到 ls-output.txt 文件中呢？原因是 ls 程序并不会把它运行的错误信息发送到标准输出文件中。而是与大多数写得很好的 UNIX 程序一样，它把错误信息发送到标准错误文件中。因为我们只重定向了标准输出，并没有重定向标准错误，所以这个错误信息仍然输出到屏幕上。稍后我们将讲述如何重定向标准错误，但是首先，先让我们看看这个输出文件发生了什么变化。

```
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me me 0 2012-02-01 15:08 ls-output.txt
```

当前这个文件大小为零！这是因为当使用重定向符“>”来重定向标准输出时，目的文件通常会从文件开头部分重新改写。由于 ls 命令执行后没有输出任何内容，只是显示一条错误信息，所以重定向操作开始重新改写这个文件，并在出现错误的情况下停止操作，最终导致了该文件内容被删除。事实上，如果我们需要删除一个文件内容（或者创建一个新的空文件），可以采用这样的方式。

```
[me@linuxbox ~]$ > ls-output.txt
```

仅仅使用了重定向符，并在它之前不加任何命令，就可以删除一个已存在的文件内容或者创建一个新的空文件。

那么，我们如何能够不从文件的首位置开始覆盖文件，而是从文件的尾部开始添加输出内容呢？我们可以使用重定向符“>>”来实现，比如：

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
```

使用重定向符>>将使得输出内容添加在文件的尾部。如果这个文件并不存在，将与操作符>的作用一样创建这个文件。下面验证一下该操作符。

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me me 503634 2012-02-01 15:45 ls-output.txt
```

重复执行这条命令三次，系统将最终生成一个为原来三倍大小的输出文件。

6.1.2 标准错误重定向

标准错误的重定向并不能简单地使用一个专用的重定向符来实现。要实现标准错误的重定向，不得不提到它的文件描述符（file descriptor）。一个程序可以把生成的输出内容发送到任意文件流中。如果把这些文件流中的前三个分别对应标准输入文件、标准输出文件和标准错误文件，那么 shell 将在内部用文件描述符分别索引它们为 0、1 和 2。shell 提供了使用文件描述符编号来重定向文件的表示法。由于标准错误等同于文件描述符 2，所以可以使用这种表示法来重定向标准错误。

```
[me@linuxbox ~]$ ls -l /bin/usr 2> ls-error.txt
```

文件描述符“2”紧放在重定向符之前，将标准错误重定向到 `ls-error.txt` 文件中。

6.1.3 将标准输出和标准错误重定向到同一个文件

在许多情况下，我们会希望把一个命令的所有输出内容都放在同一个独立的文件中。为此，我们必须同时重定向标准输出和标准错误。有两种方法可以满足要求。第一种是传统的方法，在旧版本的 shell 中使用。

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt 2>&1
```

使用这个方法，将执行两个重定向操作。首先重定向标准输出到 `ls-output.txt` 文件中，然后使用标记符 `2>&1` 把文件描述符 2（标准错误）重定向到文件描述符 1（标准输出）中。

注意

这些重定向操作的顺序是非常重要的。标准错误的重定向操作通常发生在标准输出重定向操作之后，否则它将不起作用。在上面的例子中，`>ls-output.txt 2>&1` 把标准错误重定向到 `ls-output.txt` 文件中，但是如果顺序改变为 `2>&1>ls-output.txt`，那么标准错误将会重定向到屏幕上。

最近的 bash 版本提供了效率更高的第二种方法来实现这一联合的重定向操作。

```
[me@linuxbox ~]$ ls -l /bin/usr &> ls-output.txt
```

在这个例子中，只使用一个标记符“`&>`”就把标准输出和标准错误都重定

向到了 *ls-output.txt* 文件中。

6.1.4 处理不想要的输出

有时候“沉默是金”，命令执行后我们并不希望得到输出，而是想把这个输出丢弃，尤其是在输出错误和状态信息的情况下更为需要。系统提供了一种方法，即通过把输出重定向到一个称为 /dev/null 的特殊文件中来实现它。这个文件是一个称为位桶（bit bucket）的系统设备，它接受输入但是不对输入进行任何处理。以下命令可以用来抑制（即隐藏）一个命令的错误信息。

```
[me@linuxbox ~]$ ls -l /bin/usr 2> /dev/null
```

UNIX 文化中的/DEV/NUL

位桶（bit bucket）是一个古老的 UNIX 概念，由于它的普适性，它出现在 UNIX 文化的很多地方。因此当某人说他正把你的意见发送到“dev null”的时候，现在你知道他是什么意思了。你可以在 <http://en.wikipedia.org/wiki/Dev/null> 中查看维基百科的相关文章，了解更多的相关示例。

6.1.5 标准输入重定向

到目前为止，我们还没有接触过使用标准输入的命令（实际上已经遇到了，稍后将揭晓这个谜底），接下来我们先介绍一个命令。

cat——合并文件

cat 命令读取一个或多个文件，并把它们复制到标准输出文件中，格式如下。

```
cat [file...]
```

在大多数情况下，你可以认为 **cat** 命令和 DOS 中的 TYPE 命令类似。使用它显示文件而不需要分页，例如：

```
[me@linuxbox ~]$ cat ls-output.txt
```

将显示 *ls-output.txt* 文件的内容。**cat** 经常用来显示短的文本文件。由于 **cat** 可以接受多个文件作为输入参数，所以它也可以用来把文件连接在一起。假设我们下载了一个很大的文件，它已被拆分为多个部分（Usenet 上的多媒体文件经常采用拆分这种方式），现在我们想要把各部分连接在一起，并还原为原来的文件。如果这些文件命名为

```
movie.mpeg.001 movie.mpeg.002...movie.mpeg.099
```

我们可以使用这个命令让它们重新连接在一起。

```
[me@linuxbox ~]$ cat movie.mpeg.* > movie.mpeg
```

通配符一般都是按照顺序来扩展的，因此这些参数将按正确的顺序来排列。

虽然这样很好，但是这跟标准输入有什么关系呢？确实没有任何关系，但是我们可以试试其他的情况。如果输入 cat 命令却不带任何参数，会出现什么样的结果呢？

```
[me@linuxbox ~]$ cat
```

没有任何结果——它只是停在那边不动，好像它已经挂起了。看起来好像是这样的，但是它实际上正在执行我们期望它做的事情。

如果 cat 命令没有给定任何参数，它将从标准输入读取内容。由于标准输入在默认情况下是连接到键盘，所以实际上它正在等待着从键盘输入内容！

试下这个。

```
[me@linuxbox ~]$ cat
The quick brown fox jumped over the lazy dog.
```

下一步，按下 Ctrl-D（按住 Ctrl 键同时按下 D），告知 cat 命令它已经达到标准输入的文件尾（end-of-file, EOF）。

```
[me@linuxbox ~]$ cat
The quick brown fox jumped over the lazy dog.
The quick brown fox jumped over the lazy dog.
```

在缺少文件名参数的情况下，cat 将把标准输入内容复制到标准输出文件中，因此我们将看到文本行重复显示。用这种方法我们可以创建短的文本文件。如果想要创建一个名叫 lazy_dog.txt 的文件，文件中包含之前例子中的文本内容，我们可以这样做：

```
[me@linuxbox ~]$ cat > lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

在 cat 命令后输入想要放在文件中的文本内容。记住在文件结束时按下 Ctrl-D。使用这个命令行，相当于执行了世界上最愚蠢的文字处理器！为了看到结果，我们可以使用 cat 命令再次把文件复制到标准输出文件中。

```
[me@linuxbox ~]$ cat lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

现在我们已经知道 `cat` 命令除了接受文件名参数之外，是如何接受标准输入的。接下来尝试一下标准输入的重定向。

```
[me@linuxbox ~]$ cat < lazy_dog.txt
The quick brown fox jumped over the lazy dog.
```

使用重定向符“`<`”，我们将把标准输入的源从键盘变为 `lazy_dog.txt` 文件。可以看到，得到的结果和只传递单个文件名参数的结果一样。和传输一个文件名参数的方式作对比，这种方式并不是特别有用，但是可以用来说明把一个文件作为标准输入的源文件。还有其他的命令更好地使用了标准输入，稍后会讲到。

在继续学习下面内容之前，我们可以查看 `cat` 命令的手册文档，因为它有几个有趣的选项。

6.2 管道

命令从标准输入到读取数据，并将数据发送到标准输出的能力，是使用了名为管道的 shell 特性。使用管道操作符“`|`”（竖线）可以把一个命令的标准输出传送到另一个命令的标准输入中。

`Command1 | command2`

为了充分证明这一点，我们需要一些命令。还记得之前说过有一条已知的命令可以接受标准输入吗？它就是 `less` 命令。使用 `less` 命令可以分页显示任意命令的输入，该命令将它的结果发送到标准输出。

```
[me@linuxbox ~]$ ls -l /usr/bin | less
```

这相当方便！通过使用该技术，可以很方便地检查任意一条生成标准输出的命令的运行结果。

6.2.1 过滤器

管道功能经常用来对数据执行复杂的操作。也可以把多条命令合在一起构成一个管道。这种方式中用到的命令通常被称为过滤器（filter）。过滤器接受输入，按照某种方式对输入进行改变，然后再输出它。第一个要用到的命令是 `sort`。假设要把`/bin` 和`/usr/bin` 目录下的所有可执行程序合并成一个列表，并且按照顺序排列，最后再查看这个列表。

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | less
```

由于我们指定了两个目录（/bin 和/usr/bin），ls 的输出将包含两个排好序的列表，每个对应一个目录。通过在管道中包含 sort 命令，我们改变输出数据，从而产生一个排好序的列表。

6.2.2 uniq——报告或忽略文件中重复的行

uniq 命令经常和 sort 命令结合使用。uniq 可以接受来自于标准输入或者一个单一文件名参数对应的已排好序的数据列表（可以查看 uniq 命令的 man 页面获取详细信息）。默认情况下，该命令删除列表中的所有重复行。因此，在管道中添加 uniq 命令，可以确保所有的列表都没有重复行（即在 /bin 和 /usr/bin 目录下都出现的相同名字的任意程序）。

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | less
```

在这个例子中，我们使用了 uniq 命令来删除来自 sort 命令输出内容中的任意重复行。如果反过来想要查看重复行的列表，可以在 uniq 命令后面添加 -d 选项，如下所示。

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq -d | less
```

6.2.3 wc——打印行数、字数和字节数

wc（字数统计，word count）命令用来显示文件中包含的行数、字数和字节数。例如：

```
[me@linuxbox ~]$ wc ls-output.txt
7902 64566 503634 ls-output.txt
```

在这个例子中，我们打印输出了三个数据，即 ls-output.txt 文件中包含的行数、字数和字节数。和前面的命令一样，如果在执行 wc 时没有输入命令行参数，它将接受标准输入内容。-l 选项限制命令只报告行数，把它添加在管道中可以很方便地实现计数功能。如果我们要查看已排好序的列表中的条目数，可以按以下方式输入。

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | wc -l
2728
```

6.2.4 grep——打印匹配行

grep 是一个功能强大的程序，它用来在文件中查找匹配文本，其使用方式如下。

```
grep pattern [file...]
```

当 grep 在文件中遇到“模式”的时候，将打印出包含该模式的行。grep 能够匹配的模式内容可以是非常复杂的，不过这里，我们只关注简单文本的匹配。在第 19 章，我们将介绍“正则表达式 (regular expression)”的高级模式。

如果想我们从列出的程序中搜索出文件名中包含 zip 的所有文件，该搜索将获悉系统中与文件压缩相关的程序，操作如下。

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

grep 存在一对方便的选项：-i，该选项使得 grep 在搜索时忽略大小写（通常情况下，搜索是区分大小写的）；-v，该选项使得 grep 只输出和模式不匹配的行。

6.2.5 head/tail——打印文件的开头部分/结尾部分

有的时候，你并不需要命令输出的所有内容，可能只是需要开头几行或者最后几行。head 命令将输出文件的前 10 行，tail 命令则输出文件的最后 10 行。默认情况下，这两条命令都是输出文件的 10 行内容，不过可以使用-n 选项来调整输出的行数。

```
[me@linuxbox ~]$ head -n 5 ls-output.txt
total 343496
-rwxr-xr-x 1 root root      31316 2011-12-05 08:58 [
-rwxr-xr-x 1 root root      8240 2011-12-09 13:39 411toppm
-rwxr-xr-x 1 root root     111276 2011-11-26 14:27 a2p
-rwxr-xr-x 1 root root     25368 2010-10-06 20:16 a52dec
[me@linuxbox ~]$ tail -n 5 ls-output.txt
-rwxr-xr-x 1 root root      5234 2011-06-27 10:56 znew
-rwxr-xr-x 1 root root      691 2009-09-10 04:21 zonetab2pot.py
-rw-r--r-- 1 root root      930 2011-11-01 12:23 zonetab2pot.pyc
-rw-r--r-- 1 root root      930 2011-11-01 12:23 zonetab2pot.pyo
lrwxrwxrwx 1 root root       6 2012-01-31 05:22 zsoelim -> soelim
```

这些命令选项也可以应用在管道中。

```
[me@linuxbox ~]$ ls /usr/bin | tail -n 5
znew
```

```
zonetab2pot.py
zonetab2pot.pyc
zonetab2pot.pyo
zsoelim
```

tail 中有一个选项用来实时查看文件，该选项在观察正在被写入的日志文件的进展状态时很有用。在下面的例子中，我们将观察 /var/log 目录下的 messages 文件。因为 /var/log/messages 文件可能包含安全信息，所以在一些 Linux 发行版本中，需要超级用户的权限才能执行该操作。

```
[me@linuxbox ~]$ tail -f /var/log/messages
Feb  8 13:40:05 twin4 dhclient: DHCPACK from 192.168.1.1
Feb  8 13:40:05 twin4 dhclient: bound to 192.168.1.4 -- renewal in 1652
seconds.
Feb  8 13:55:32 twin4 mountd[3953]: /var/NFSv4/musicbox exported to both
192.168.1.0/24 and twin7.localdomain in 192.168.1.0/24,twin7.localdomain
Feb  8 14:07:37 twin4 dhclient: DHCPREQUEST on eth0 to 192.168.1.1 port 67
Feb  8 14:07:37 twin4 dhclient: DHCPACK from 192.168.1.1
Feb  8 14:07:37 twin4 dhclient: bound to 192.168.1.4 -- renewal in 1771
seconds.
Feb  8 14:09:56 twin4 smartd[3468]: Device: /dev/hda, SMART Prefailure
Attribute: 8 Seek_Time_Performance changed from 237 to 236
Feb  8 14:10:37 twin4 mountd[3953]: /var/NFSv4/musicbox exported to both
192.168.1.0/24 and twin7.localdomain in 192.168.1.0/24,twin7.localdomain
Feb  8 14:25:07 twin4 sshd(pam_unix)[29234]: session opened for user me by
(uid=0)
Feb  8 14:25:36 twin4 su(pam_unix)[29279]: session opened for user root by
me(uid=500)
```

使用 -f 选项，tail 将持续监视这个文件，一旦添加了新行，新行将会立即显示在屏幕上。该动作在按下 Ctrl-C 后停止。

6.2.6 tee——从 stdin 读取数据，并同时输出到 stdout 和文件

为了和我们的管道隐喻保持一致，Linux 提供了一个叫做 tee 的命令，就好像安装了一个“T”在管道上。tee 程序读取标准输入，再把读到的内容复制到标准输出（允许数据可以继续向下传递到管道中）和一个或更多的文件中去。当在某个中间处理阶段来捕获一个管道中的内容时，会很有用。这里我们重复使用之前的一个例子，这次在使用 grep 命令过滤管道内容之前，我们先使用 tee 命令来获取整个目录列表并输出到 ls.txt 文件中，具体操作如下。

```
[me@linuxbox ~]$ ls /usr/bin | tee ls.txt | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
```

```
zipgrep  
zipinfo  
zipnote  
zipsplit
```

6.3 本章结尾语

和以前一样，请查看本章介绍的各个命令的相关文档。本章只介绍了这些命令最基本的用法，它们都还有很多其他有趣的选项。在有一定 Linux 使用经验的时候，我们将会发现命令行的重定向功能对于解决某些特定的问题相当有用。很多命令使用了标准输入和输出，而且几乎所有的命令行程序都使用了标准错误来显示它们的提示性信息。

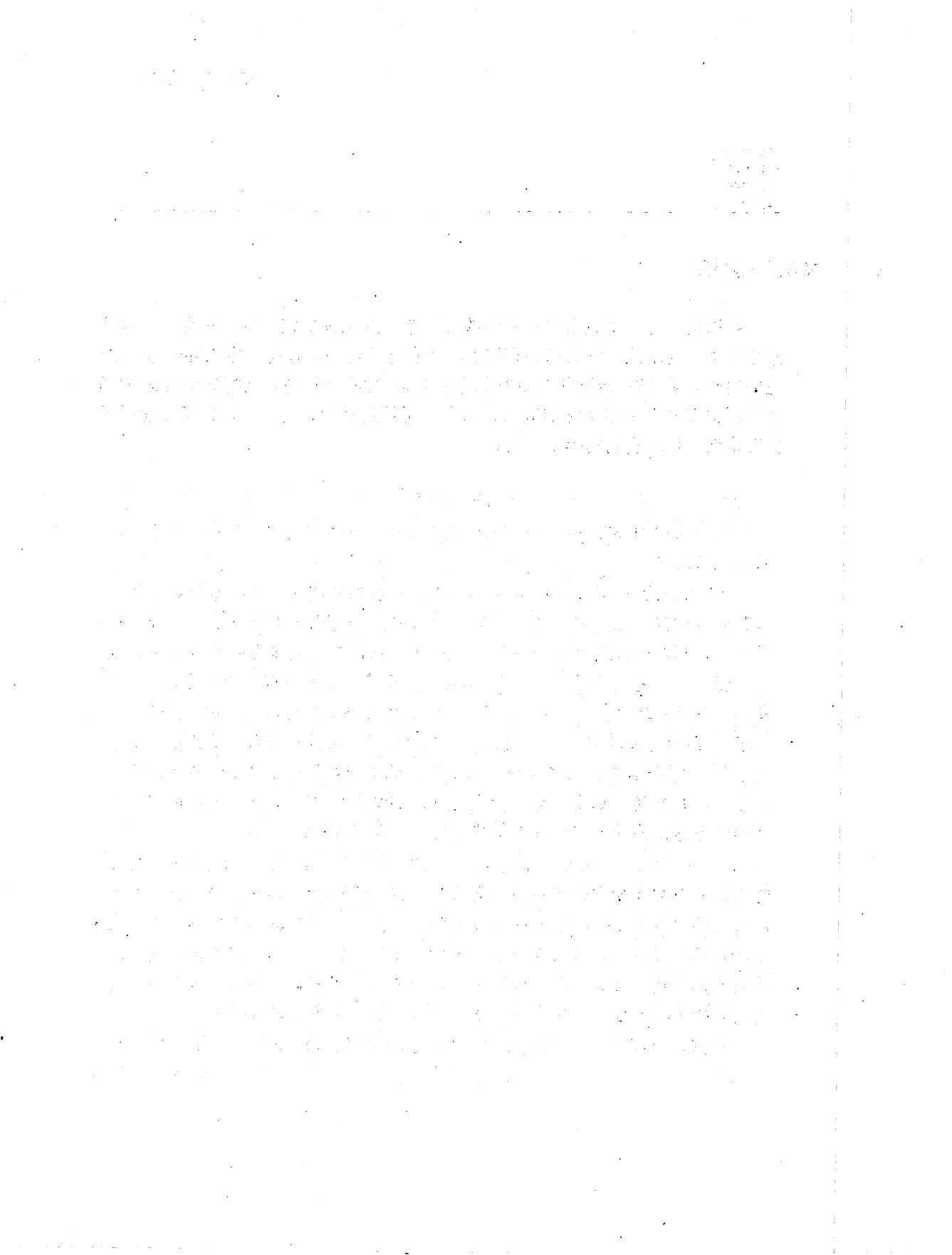
富有想象力的 Linux

每当被问到 Windows 和 Linux 的区别时，我经常通过用玩具打比方的方式来解释。

Windows 就像是 Game Boy 游戏机。你去商店买了一个全新的游戏机。你把它带回家，启动它，开始玩这个游戏机。漂亮的画面，可爱的声音。但是不久，你对这款游戏机玩腻了，于是你回到商店，买了另一款游戏机。这个过程一遍一遍地重复着。最后，你再次回到商店，对柜台后的售货员说“我想要一款可以玩这个游戏的游戏机！”但是却被告知因为没有针对它的“市场需求”，所以并不存在这种游戏机。然后你会说“但是我只需要更换这一个东西就行了”。柜台后的售货员将会对你说，你不能更换它。这个游戏机盒子都是完全密封好的。你发现你的玩具选择范围被限定了，你只能选择由别人决定的认为你需要的游戏，并没有其他更多的游戏可选。

而另一方面，Linux 就像是世界上最大的建筑拼装玩具。你打开它，发现它只是一个超大的零件集——一大堆的钢架、螺丝钉、螺帽、齿轮、滑轮组以及马达，另附上可拼装的一些参考样式图案。你开始玩这个玩具。你拼装了一种参考样式后，再接着拼装另一种。不久，你发现你可以拼装出自己想要的样式。你不再需要非得回到商店，因为你已经有你需要的所有东西。这个建筑拼装玩具可以呈现出你想象的形状，它可以实现你所想要的。

当然，选择哪个玩具是你自己的事情，你觉得你会更加钟情于哪种玩具呢？



第 7 章

透过 shell 看世界

在本章，我们将介绍在按下 Enter 键时，命令行中发生的一些“神奇”事情。虽然我们会介绍 shell 的几个有趣而复杂的特性，但是我们只使用一条新命令来处理。

- echo：显示一行文本。

7.1 扩展

每次输入命令行按下 Enter 键时，bash 都会在执行命令之前对文本进行多重处理。前面已经见过一个简单的字符序列（比如*）在 shell 中被识别为多种意思的几个例子。产生这个结果的处理过程称为扩展(expansion)。有了扩展功能，在输入内容后，这些内容将在 shell 对其执行之前被扩展成其他内容。为了证明这点，让我们先来看看 echo 命令。echo 是 shell 的一个内置命令，它执行的任务非常简单，即把文本参数内容打印到标准输出。

```
[me@linuxbox ~]$ echo this is a test
this is a test
```

这个例子相当简单，传递给 echo 的任何参数都将显示出来。让我们再看另一个例子。

```
[me@linuxbox ~]$ echo *
Desktop Documents ls-output.txt Music Pictures Public Templates Videos
```

刚刚发生了什么？为什么 echo 不是输出“*”呢？回想一下之前我们对通配符的使用。“*”字符意味着“匹配文件名中的任意字符”，但是之前我们并没有讨论 shell 是如何实现这个功能的。答案很简单，shell 会在执行 echo 命令前把“*”字符扩展成其他内容（在这个例子中，扩展为当前工作目录下的所有文件名）。在按下 Enter 键的时候，shell 会在执行命令前自动扩展命令行中所有符合条件的字符，因此 echo 命令将不可能看到“*”字符，只能看到“*”字符扩展后的结果。知道了这些，我们就会发现 echo 命令输出的正是预期的结果。

7.1.1 路径名扩展

通过使用通配符来实现扩展的机制称为路径名扩展（pathname expansion）。试试在前面章节中使用过的一些技术，将会发现它们实际上就是扩展。下面给定一个主目录，如下所示：

```
[me@linuxbox ~]$ ls
Desktop  ls-output.txt  Pictures  Templates
Documents  Music        Public    Videos
```

执行下面的扩展：

```
[me@linuxbox ~]$ echo D*
Desktop Documents
```

以及

```
[me@linuxbox ~]$ echo *s
Documents Pictures Templates Videos
```

甚至是

```
[me@linuxbox ~]$ echo [[:upper:]]*
Desktop Documents Music Pictures Public Templates Videos
```

查看除主目录之外的目录：

```
[me@linuxbox ~]$ echo /usr/*/*share
/usr/kerberos/share /usr/local/share
```

隐藏文件的路径名扩展

众所周知，文件名以一个“.”点字符开头的文件都将被隐藏。路径名扩展功能也遵守这个规则。类似 echo *这样的扩展并不能显示隐藏的文件。

乍一看，好像可以通过在扩展的模式中以一个点字符开头来包含隐藏文件，如下所示。

```
echo *
```

这似乎是可行的。但是如果我们仔细地检查结果，会发现文件名“.”和“..”也将出现在结果中。由于这两个名字分别指的是当前的工作目录以及当前目录的父目录，使用这种模式匹配可能会生成不正确的结果。执行命令行 ls -d .*|less 可以发现这个结果是不正确的。

在这种情况下，要正确地执行路径名扩展，必须采用一种更精确些的模式，它会正确地工作。

```
ls -d .[!.]*
```

这种模式将扩展为以一个点字符开头的所有文件名，文件名中并不包含第二个点字符，但包含至少一个额外的字符，后面也可能还跟着其他的字符。

7.1.2 波浪线扩展

回顾前面对 cd 命令的介绍，你会发现波浪线字符（～）具有特殊的含义。如果把它用在一个单词的开头，那么它将被扩展为指定用户的主目录名；如果没有指定用户名，则扩展为当前用户的主目录。

```
[me@linuxbox ~]$ echo ~  
/home/me
```

如果有用户 foo 这个账户，那么：

```
[me@linuxbox ~]$ echo ~foo  
/home/foo
```

7.1.3 算术扩展

shell 支持通过扩展来运行算术表达式。这允许我们把 shell 提示符当作计算器来使用。

```
[me@linuxbox ~]$ echo $((2 + 2))  
4
```

算术扩展使用如下格式。

```
$((expression))
```

其中，*expression* 是指包含数值和算术操作符的算术表达式。

算术扩展只支持整数（全是数字，没有小数），但是可以执行很多不同的运算。表 7-1 列出了一些支持的操作符。

表 7-1 算术运算符

运算符	描述
+	加
-	减
*	乘
/	除（但是记住，因为扩展只支持整数运算，所以结果也是整数）
%	取余，即余数
**	取幂

空格在算术表达式中是没有意义的，而且表达式是可以嵌套的。例如把 5^2 和 3 相乘。

```
[me@linuxbox ~]$ echo $(((5**2) * 3))
75
```

你可以使用一对括号来组合多个子表达式。通过该技术，可以把上面的例子重写，用一个扩展来代替两个，可以得到同样的结果：

```
[me@linuxbox ~]$ echo $(((5**2) * 3))
75
```

下面的例子使用了除运算符和取余运算符，注意整数相除的结果。

```
[me@linuxbox ~]$ echo Five divided by two equals $((5/2))
Five divided by two equals 2
[me@linuxbox ~]$ echo with $((5%2)) left over.
with 1 left over.
```

在第 34 章我们将更详细地介绍算术扩展。

7.1.4 花括号扩展

花括号扩展（brace expansion）可能算是最奇怪的扩展方式了。有了它，你可以按照花括号里面的模式创建多种文本字符串。实例如下。

```
[me@linuxbox ~]$ echo Front-{A,B,C}-Back
Front-A-Back Front-B-Back Front-C-Back
```

用于花括号扩展的模式信息可以包含一个称为前导字符 (preamble) 的开头部分和一个称为附言 (postscript) 的结尾部分。花括号表达式本身可以包含一系列逗号分隔的字符串，也可以包含一系列整数或者单个字符。这里的模式信息不能包含内嵌的空白。下面的例子使用了一系列的整数。

```
[me@linuxbox ~]$ echo Number_{1..5}
Number_1 Number_2 Number_3 Number_4 Number_5
```

下面输出一系列逆序排列的字母。

```
[me@linuxbox ~]$ echo {Z..A}
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

花括号扩展支持嵌套。

```
[me@linuxbox ~]$ echo a{A{1,2},B{3,4}}b
aA1b aA2b aB3b aB4b
```

那么花括号扩展一般应用在哪些地方呢？最普遍的应用是创建一系列的文件或者目录。比如说，摄影师有一个很大的图片集，想要按年份和月份来对这些图片进行分组，那么要做的第一件事就是创建一系列以年月格式命名的目录。这样，这些目录名将会按照年代顺序排列，输出目录的一个完整的列表。但是这样做工作量大，而且容易出错。为此我们可以这样操作。

```
[me@linuxbox ~]$ mkdir Pics
[me@linuxbox ~]$ cd Pics
[me@linuxbox Pics]$ mkdir {2009..2011}-0{1..9} {2009..2011}-{10..12}
[me@linuxbox Pics]$ ls
2009-01 2009-07 2010-01 2010-07 2011-01 2011-07
2009-02 2009-08 2010-02 2010-08 2011-02 2011-08
2009-03 2009-09 2010-03 2010-09 2011-03 2011-09
2009-04 2009-10 2010-04 2010-10 2011-04 2011-10
2009-05 2009-11 2010-05 2010-11 2011-05 2011-11
2009-06 2009-12 2010-06 2010-12 2011-06 2011-12
```

相当巧妙！

7.1.5 参数扩展

本章我们只是简要地介绍参数扩展 (parameter expansion)，在之后的章节中我们将会更深入地介绍它。参数扩展用在 shell 脚本中比直接用在命令行中更为有用。它的许多特性与系统存储小块数据以及给每个小块数据命名的性能有关。很多这样的小块数据（称为变量 [variable] 会更合适）可用于扩展。例如，命名为 USER 的变量包含你的用户名，为了触发参数扩展，并显示出 USER 的内容，你可以进行如下操作。

```
[me@linuxbox ~]$ echo $USER
me
```

想要查看可用的变量列表，试试如下操作。

```
[me@linuxbox ~]$ printenv | less
```

你可能已经注意到，对于其他的扩展类型来说，如果你误输入了一个模式，就不会发生扩展，这时 echo 命令将只是显示这些误输入的模式信息。但是对于参数扩展来说，如果变量名拼写错误，仍然会进行扩展，只不过结果是输出一个空字符串而已，如下所示。

```
[me@linuxbox ~]$ echo $SUER
[me@linuxbox ~]$
```

7.1.6 命令替换

命令替换可以把一个命令的输出作为一个扩展模式使用，如下所示。

```
[me@linuxbox ~]$ echo $(ls)
Desktop Documents ls-output.txt Music Pictures Public Templates Videos
```

我最喜欢的一种用法如下。

```
[me@linuxbox ~]$ ls -l $(which cp)
-rwxr-xr-x 1 root root 71516 2012-12-05 08:58 /bin/cp
```

这里，把 which cp 命令的运行结果作为 ls 命令的一个参数，因此我们无需知道 cp 程序所在的完整路径就能获得 cp 程序对应的列表。这个功能并不只是局限于简单的命令，也可以应用于整个管道中（只不过只显示部分输出内容）。

```
[me@linuxbox ~]$ file $(ls /usr/bin/* | grep zip)
/usr/bin/bunzip2:      symbolic link to 'bzip2'
/usr/bin/bzip2:      ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
, dynamically linked (uses shared libs), for GNU/Linux 2.6.9, stripped
/usr/bin/bzip2recover: ELF 32-bit LSB executable, Intel 80386, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.9, stripped
/usr/bin/funzip:      ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
, dynamically linked (uses shared libs), for GNU/Linux 2.6.9, stripped
/usr/bin/gpg-zip:     Bourne shell script text executable
/usr/bin/gunzip:      symbolic link to '../bin/gunzip'
/usr/bin/gzip:        symbolic link to '../bin/gzip'
/usr/bin/mzip:        symbolic link to 'mtools'
```

在这个例子中，管道的输出为 file 命令的参数列表。

在早期的 shell 程序中，存在命令替换的另一种语法格式，bash 也支持这种格式。它用反引号代替美元符号和括号，具体如下所示。

```
[me@linuxbox ~]$ ls -l `which cp'
-rwxr-xr-x 1 root root 71516 2012-12-05 08:58 /bin/cp
```

7.2 引用

我们已经知道，shell 有多种方式可以执行扩展，现在我们来学习如何控制扩展。先看下面的例子。

```
[me@linuxbox ~]$ echo this is a test
this is a test
```

再看这个例子。

```
[me@linuxbox ~]$ echo The total is $100.00
The total is 00.00
```

在第一个例子中，shell 会对 echo 命令的参数列表进行单词分割 (*word splitting*)，去除多余的空白。在第二个例子中，因为 \$1 是一个未定义的变量，所以参数扩展将把 \$1 的值替换为空字符串。shell 提供了一种称为引用 (quoting) 的机制，用来有选择性地避免不想要的扩展。

7.2.1 双引号

我们要看的第一种引用类型是双引号 (double quote)。如果把文本放在双引号中，那么 shell 使用的所有特殊字符都将失去它们的特殊含义，而被看成普通字符。字符 “\$” (美元符号)、“\” (反斜杠)、“” (反引号) 除外。这就意味着单词分割、路径名扩展、波浪线扩展和花括号扩展都将失效，但是参数扩展、算术扩展和命令替换仍然生效。使用双引号能够处理文件名中包含空白的情况。假设不幸地有一个名为 two words.txt 的文件，如果在命令行中使用该文件名，那么单词分割功能将把它当成两个独立的参数，而不是当成我们希望的单个参数，具体运行结果如下所示。

```
[me@linuxbox ~]$ ls -l two words.txt
ls: cannot access two: No such file or directory
ls: cannot access words.txt: No such file or directory
```

使用双引号可以阻止单词分割，得到预期的结果。另外，使用双引号甚至可以修复破损的文件名，参见下面的例子。

```
[me@linuxbox ~]$ ls -l "two words.txt"
-rw-rw-r-- 1 me me 18 2012-02-20 13:03 two words.txt
[me@linuxbox ~]$ mv "two words.txt" two_words.txt
```

看！现在我们就不需要一直输入那些让人讨厌的双引号了。

请记住，参数扩展、算术扩展和命令替换在双引号中依然生效：

```
[me@linuxbox ~]$ echo "$USER $((2+2)) $(cal)"
me 4 February 2012
Su Mo Tu We Th Fr Sa
      1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29
```

接下来，让我们看看双引号对字符替换的影响。我们首先深入了解一下单词分割是怎么工作的。在前面的例子中，我们已经看到单词分割去除文本中多余空白的情况，如下所示。

```
[me@linuxbox ~]$ echo this is a test
this is a test
```

默认情况下，单词分割会先查找是否存在空格、制表符以及换行（换行字符），然后把它们当作单词见的界定符（delimiter）。这就意味着没有用引号包含起来的空格、制表符和换行字符都不会被当成文本的一部分，而只是被当成分割符。因为它们把这些单词分割成不同的参数，所以例子中的命令行被识别为命令后面跟着 4 个不同的参数。但是如果加上双引号，单词分割功能将失效，嵌入的空格将不再被当成界定符，而是被当成参数的一部分，如下所示。

```
[me@linuxbox ~]$ echo "this is a test"
this is a    test
```

一旦加上双引号，那么命令行将被识别为命令后面只跟着一个参数。

单词分割机制会把换行字符当成界定符，这一点在命令替换时将会产生微妙有趣的效果。参考下面的例子。

```
[me@linuxbox ~]$ echo $(cal)
February 2012 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29
[me@linuxbox ~]$ echo "$(cal)"
      February 2012
Su Mo Tu We Th Fr Sa
      1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29
```

在第一个例子中，没有加上引号的命令替换将导致命令行被识别为命令后面跟着 38 个参数；而在第二个例子中加了双引号，使得命令行被识别为命令后面只跟一个参数，这个参数包含着嵌入空格和换行字符。

7.2.2 单引号

如果我们希望抑制所有的扩展，那么应使用单引号。下面是不使用引号、使用双引号和使用单引号的情况对比。

```
[me@linuxbox ~]$ echo text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
text /home/me/ls-output.txt a b foo 4 me
[me@linuxbox ~]$ echo "text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER"
text ~/*.txt {a,b} foo 4 me
[me@linuxbox ~]$ echo 'text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER'
text ~/*.txt {a,b} $(echo foo) $((2+2)) $USER
```

可以看到，随着引用级别的加强，越来越多的扩展将被抑制。

7.2.3 转义字符

有时候我们只是想要引用单个字符。这种情况可以通过在该字符前加上反斜杠来实现。这里的反斜杠称为转义字符。转义字符经常在双引号中用来有选择性地阻止扩展。如下所示。

```
[me@linuxbox ~]$ echo "The balance for user $USER is: \$5.00"
The balance for user me is: $5.00
```

转义字符也常用来消除文件名中某个字符的特殊含义。比如，文件名中可以使用在 shell 中通常具有特殊含义的字符。这些字符包括“\$”、“!”、“&”、空格等。要想在文件名中包含特殊字符，可执行如下操作。

```
[me@linuxbox ~]$ mv bad\&filename good_filename
```

如果想要显示反斜杠字符，可以通过使用两个反斜杠 “\\” 来实现。需要注意的是，单引号中的反斜杠将失去它的特殊含义，而只是被当成一个普通字符。

反斜杠转义字符序列（BLACKSLASH ESCAPE SEQUENCES）

反斜杠除了作为转义字符外，也是一种表示法的一部分，这种表示法代表称为控制码的某些特殊字符。ASCII 码表的前 32 个字符用来向电传打字类设备传送命令。其中有一些控制码很常见（比如制表符、退格符、换行符和回车符），但是其他的都不太常见（空字符、结束符和确认符），如表 7-2 所示。

表 7-2 反斜杠转义字符序列

转义字符	含义
\a	响铃（警告声—计算机发出哔哔声）
\b	退格
\n	新的一行（在类 UNIX 系统中，产生的是换行效果）
\r	回车
\t	制表

表中列出了一些常用的反斜杠转义字符序列。使用反斜杠来表示转义字符表示的理解来源于 C 语言，其他语言也采用了这种表示方法，包括 shell。

在 echo 命令中带上-e 选项，就能够解释转义字符序列。也可以将其放在“\$”中。在下面的例子中，只需要使用 sleep 命令（它是一个简单的程序，在等待指定的秒数之后就会退出），就可以创建一个简单的倒计时的计时器：

```
sleep 10; echo -e "Time's up!\a"
```

也可以这样做：

```
sleep 10; echo "Time's up" $'\a'
```

7.3 本章结尾语

随着我们输入学习 shell，就会发现扩展和引用的使用频率逐渐多起来，所以我们有必要很好地理解它们的工作方式。事实上，甚至可以说它们是 shell 中最重要的主题。如果不能正确地理解扩展，那么 shell 将会一直是个神秘和让人困惑的资源，它的潜在能力也就被浪费了。

第 8 章

高级键盘技巧

我经常将 UNIX 戏称为“它是为喜欢敲键盘的人设计的操作系统”。当然，UNIX 中存在命令行的这一事实充分证明了这点。但是用户使用命令行时往往不喜欢敲入太多字，所以命令中存在很多类似 cp、ls、mv 和 rm 的短命令。

事实上，省事 (laziness) (即用最少的击键次数执行最多的任务) 是命令行最希望达到的目标之一。命令行的另一个目标是，用户在执行任务时手指无需离开键盘，用不使用鼠标。本章我们将学习可以令键盘使用得更快和更高效的 bash 功能。

我们将使用到以下命令。

- `clear`: 清屏。
- `history`: 显示历史列表的记录。

8.1 编辑命令行

bash 使用了一个名为 Readline 的库(供不同的应用程序共享使用的线程集合)

来实现命令行的编辑。在前面我们曾提到过相关内容。比如，通过箭头键移动光标。除此之外，bash 还有很多其他的功能，它们可以当作在工作中使用的附加工具。虽然并不要求你们学会所有的这些功能，但是学会其中的一些功能还是非常有用的。请选择自己需要的功能。

注意 下面的有些组合键（尤其对于那些使用了 Alt 键的组合键）可能会被 GUI（图形用户界面）识别为其他功能。当使用虚拟控制台时，所有的组合键应该能够正常工作。

8.1.1 光标移动

表 8-1 中列出了用来移动光标的组合键。

表 8-1 光标移动命令

组合键	作用
Ctrl-A	移动光标到行首
Ctrl-E	移动光标到行尾
Ctrl-F	光标向前移动一个字符；和右箭头键作用一样
Ctrl-B	光标向后移动一个字符，和左箭头键作用一样
Alt-F	光标向前移动一个字
Alt-B	光标向后移动一个字
Ctrl-L	清屏并把光标移到左上角；clear 命令可以完成相同的工作

8.1.2 修改文本

表 8-2 列出了用来编辑命令行字符的键盘指令。

表 8-2 文本编辑命令

组合键	作用
Ctrl-D	删除光标处的字符
Ctrl-T	使光标处的字符和它前面的字符对调位置
Alt-T	使光标处的字和它前面的字对调位置
Alt-L	把从光标到字尾的字符转换成小写字母形式
Alt-U	把从光标到字尾的字符转换成大写字母形式

8.1.3 剪切和粘贴 (Killing and Yanking) 文本

Readline 文档中使用术语 killing 和 yanking 来指代通常所说的剪切和粘贴。表 8-3 列出了用来剪切和粘贴的命令。被剪切的内容存放在一个称为 kill-ring 的缓冲区中。

表 8-3 剪切和粘贴命令

组合键	作用
Ctrl-K	剪切从光标到行尾的文本
Ctrl-U	剪切从光标到行首的文本
Alt-D	剪切从光标到当前词尾的文本
Alt-Backspace	剪切从光标到词头的文本。如果光标在一个单词的开头，则剪切前一个单词
Ctrl-Y	把 kill-ring 缓冲区中的文本粘贴到光标位置

元键

在 bash 帮助文档的“READLINE”部分可以查看 Readline 文档，在这里你将会看到元键 (meta key) 这个术语。它对应于现代键盘中的 Alt 键，不过也并不总是这样。

回到混沌时代 (PC 时代前, UNIX 时代后)，并不是每个人都有自己的计算机。当时的用户可能只有一台称为终端的设备。终端是一种通信设备，它包含一个文本显示屏、一个键盘以及一些用来显示文本字符和移动光标的电子器件。终端 (通常通过串行电缆) 连接到一台大型计算机或者大型计算机通信网。它有很多不同的品牌，因此有不同的键盘和不同的显示特性集。由于它们至少都能识别 ASCII 码，因此软件开发者想要编写符合最低标准的可移植的应用程序。UNIX 系统有一套非常巧妙的方法来处理这些终端以及它们不同的显示特性。因为 Readline 的开发者们不能确定是否存在一个专门的附加控制键，所以他们发明了一个，并把它称之为“元”。现代键盘上的 Alt 键相当于元键。如果你仍然在使用终端，则按下和释放 Esc 键和长按住 Alt 键的效果是相同的 (对于 Linux 系统也是如此)。

8.2 自动补齐功能

shell 的一种称为“自动补齐”的机制为用户提供了很大的帮助。在输入命

令时，按 Tab 键将触发自动补齐功能。下面让我们看看它是如何工作的。假设用户目录如下。

```
[me@linuxbox ~]$ ls
Desktop  ls-output.txt  Pictures  Templates  Videos
Documents  Music          Public
```

输入如下命令，但是不要按 Enter 键。

```
[me@linuxbox ~]$ ls l
```

此时按 Tab 键：

```
[me@linuxbox ~]$ ls ls-output.txt
```

观察 shell 是如何补齐这一行的。再看另一个例子，同样，也不要按 Enter 键。

```
[me@linuxbox ~]$ ls D
```

按下 Tab 键：

```
[me@linuxbox ~]$ ls D
```

没有自动补齐——只有哔哔声。这是因为字母 D 和目录中一个以上的名称匹配。要让自动补齐功能生效，要保证输入的内容不模棱两可，即必须是确定性的。如果我们继续输入：

```
[me@linuxbox ~]$ ls Do
```

此时按下 Tab：

```
[me@linuxbox ~]$ ls Documents
```

自动补齐功能这次生效了。

这个例子给出的是路径名的自动补齐，这也是最常用的方式。自动补齐也可以针对变量（如果单词以\$开头）、用户名（如果单词以~开头）、命令（如果单词是命令行的第一个单词）和主机名（如果单词以@开头）起作用。主机名的自动补齐只对/etc/hosts 目录下的主机名生效。

有一些控制和元键序列与自动补齐功能相关联（见表 8-4）。

表 8-4 自动补齐命令

组合键	作用
Alt-\$	显示所有可能的自动补齐列表。在大多数系统中，可通过按两次 Tab 键实现，而且也会更容易一些
Alt-*	插入所有可能的匹配项。当需要用到一个以上的匹配项时，将比较有用

除了以上这些，还有相当多的组合键，可以在 bash man 页面的 README 部分获取更多的相关内容列表。

可编程的自动补齐

bash 的当前版本提供了一种称为“可编程的自动补齐”的工具。可编程自动补齐允许用户（更可能是发行版本提供商）添加附加的自动补齐规则。

一般来说，这样做是为了支持特定的应用。例如，可以为一个命令的可选列表，或者是为了匹配某种应用支持的特定的文件类型，而添加自动补齐。默认情况下，Ubuntu 定义了一个相当大的规则集合。可编程自动补齐通过 shell 函数来实现的，shell 函数是一种小型 shell 脚本，这个将在后面的章节介绍。如果你好奇的话，试一下：

```
set | less
```

看看是否可以找到它们。默认情况下，并不是所有的发行版本都包含它们。

8.3 使用历史命令

第1章我们已经提到，bash 会保存使用过命令的历史记录。这些命令的历史记录列表保存在用户主目录的.bash_history 文件中。这些历史记录非常有用，可以大大减少用户敲打键盘的次数，特别是和命令行编辑结合使用的时候。

8.3.1 搜索历史命令

任何情况下，我们都可以通过如下命令查看历史记录的内容列表。

```
[me@linuxbox ~]$ history | less
```

bash 默认会保存用户最近使用过的 500 个命令。其中，500 是个默认值，关于如何改变这个默认值将在第 11 章介绍。假设我们想找到用来列出/usr/bin 目录下内容的命令，我们可以这样做：

```
[me@linuxbox ~]$ history | grep /usr/bin
```

假设得到的搜索结果中有一行包含如下有趣的命令。

```
88 ls -l /usr/bin > ls-output.txt
```

数字 88 表示这个命令行在历史记录列表中所处的行号，我们可以通过使用

名为历史记录扩展（history expansion）的扩展类型来立即使用它。为了使用我们发现的命令行，可以如下操作：

```
[me@linuxbox ~]$ !88
```

bash 将把!88 扩展为历史列表中第 88 行的内容。稍后将介绍历史记录扩展的其他形式。

bash 也支持以递增方式搜索历史记录。也就是说，当搜索历史记录时，随着输入字符数的增加，bash 会相应地改变搜索范围。按下 Ctrl-R 键，接着输入你要查找的内容，可以开始递增式的搜索。当找到要查找的内容时，按 Enter 键表示执行此命令，而按 Ctrl-J 将把搜索到的内容从历史记录列表中复制到当前命令行。当要查找下一个匹配项时（即向前搜索历史记录），再次按下 Ctrl-R 键。若要退出搜索，按下 Ctrl-G 或者 Ctrl-C 即可。请看下面的例子。

```
[me@linuxbox ~]$
```

首先按下 Ctrl-R。

```
(reverse-i-search) :::
```

提示符发生改变，提示正在进行逆向递增式搜索。称为“逆向”是因为查找的是从“现在”到过去的某个时间之间的操作。接下来，输入要查找的内容，这个例子中是查找/usr/bin。

```
(reverse-i-search) '/usr/bin': ls -l /usr/bin > ls-output.txt
```

很快搜索操作返回了结果。此时我们可按 Enter 键执行搜索结果，也可按下 Ctrl-J 把搜索结果复制到当前命令行以便作进一步的编辑。假定按下 Ctrl-J，把搜索结果复制到当前命令行。

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

shell 将实时响应，命令行将被加载，准备运行。

表 8-5 列出了一些用来手动操作历史记录的组合键。

表 8-5 历史记录命令

组合键	作用
Ctrl-P	移动到前一条历史记录。相当于向上箭头键
Ctrl-N	移动到后一条历史记录。相当于向下箭头键
Alt-<	移动到历史记录列表的开始处
Alt->	移动到历史记录列表的结尾处。即当前命令行

续表

组合键	作用
Ctrl-R	逆向递增地搜索。从当前命令行向前递增搜索
Alt-P	逆向非递增地搜索。按下这个组合键，接着输入待搜索的字符串，在按 Enter 键后，搜索才真正开始执行
Alt-N	向前非递增地搜索
Ctrl-O	执行历史记录列表中的当前项，执行完跳到下一项。若要把历史记录中的一系列命令重新执行一遍，使用该组合键将很方便

8.3.2 历史记录扩展

shell 提供了一种专门用来扩展历史记录项的方式——使用!字符。前面我们曾提到过如何通过在感叹号后面跟数字的方式，将来自历史记录列表中的命令插入到命令行中。除了这种方式，还有很多其他的扩展特性（见表 8-6）。

当使用“! string”和“! ? string”时，请务必小心谨慎，除非对历史记录中的内容非常确信。

历史记录扩展机制中还有很多其他的可用特点，但是该主题太过晦涩难懂，此处不再讨论。你可以查阅 bash 帮助页面中的“HISTORY EXPANSION”部分获取更多细节。

表 8-6 历史记录扩展命令

序列	行为
!!	重复最后一个执行的命令。按向上箭头键再按 Enter 键也可实现相同的功能，而且操作更简单
!number	重复历史记录中第 number 行的命令
! string	重复最近的以 string 开头的历史记录
!?string	重复最近的包含 string 的历史记录

脚本

除了 bash 中的命令历史特性外，大部分 Linux 发行版本都包含一个称为脚本（script）的程序，它记录了 shell 的整个会话，并且将会话保存到一个文件中。该命令的基本语法是：

```
script [file]
```

其中 file 为用来保存会话记录的文件名。如果没有指定文件，默认使用文件 typescript。脚本（script）的 man 页面给出了该程序的所有可选项和特性。

8.4 本章结尾语

本章介绍了 shell 提供的一些键盘操作技巧，它们能够帮助打字员减少工作量。随着时间的推移，你会越来越多地接触到命令行，到时候你会翻阅这一章的内容，以获得更多的键盘使用技巧。当前，只需将它们当做一个虽然有用但是当前没有必要掌握的可选技能即可。