

PROGRAMMER TO PROGRAMMER™



Beginning Unix

Unix 入门经典

(美) Paul Love 等著
Joe Merlino
张楚雄 许文昭 译



清华大学出版社

Beginning Unix

Unix 操作系统是目前一些常用平台(如 Mac OS X 和 Linux)的基础。本书将讨论 Unix 的基础知识以及日益流行的 Sun Solaris 和 BSD 平台的基础知识。

首先,读者将学习 Unix 术语、核心概念、方法以及怎样登录和退出系统,然后开始定制工作环境并学习命令,最后将学习如何管理进程、处理安全事务、使用 Perl 脚本自动处理任务,以及如何安装 Unix 程序和备份数据等内容。

本书主要内容

- Unix shell 中不同的配置选项
- 高级工具和命令,包括正则表达式、Sed 和 AWK
- 增强 Unix 系统安全性的基本方法
- 基本编程技术,包括 shell 脚本编程和 Perl 编程
- 网络管理以及与其他操作系统通信有关的内容
- 如何将 Windows 和 Mac OS 中的命令和概念转换为 Unix 中等价的命令和概念

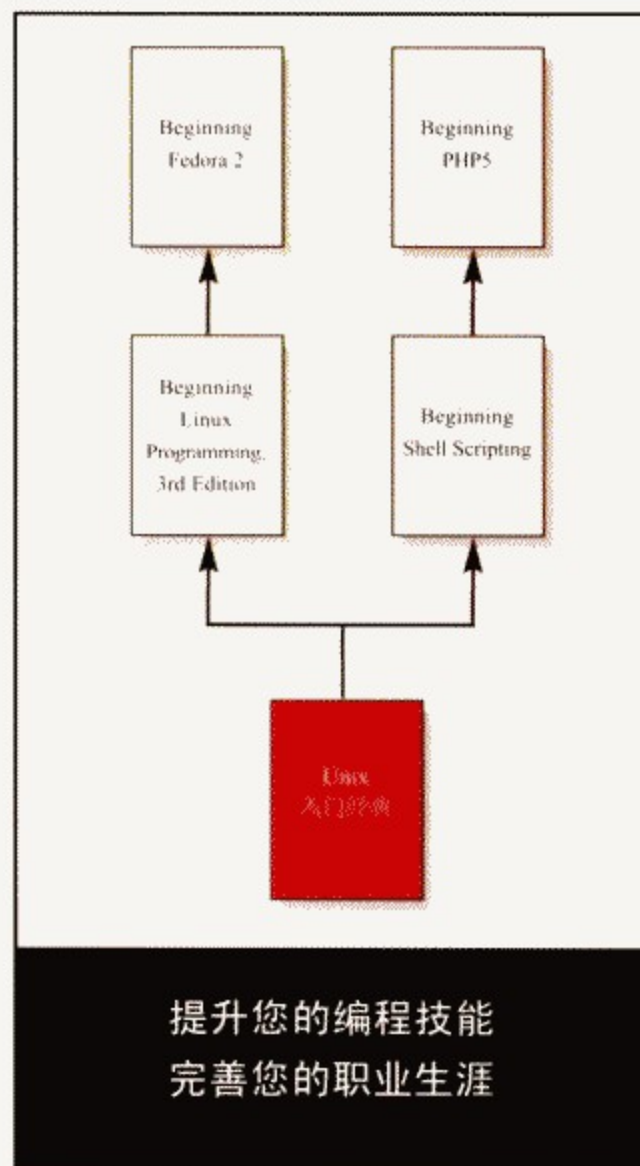
本书读者对象

本书适用于任何对 Unix 操作系统感兴趣的读者。虽然本书是一本入门级图书,但对于那些已经具有一定 Unix 知识的读者,仍颇具实用价值,另外,如果读者希望将 Mac OS 或 Windows 中的相关知识转换到 Unix 或其派生版本中,本书也可以作为一本优秀的参考手册。

源代码下载

<http://www.wrox.com>

<http://www.tupwk.com.cn/downpage>



Wrox Beginning guides are crafted to make learning programming languages and technologies easier than you think, providing a structured, tutorial format that will guide you through all the techniques involved.

ISBN 7-302-12374-8



9 787302 123743 >

定价: 39.90 元

p2p.wrox.com
The programmer's resource center



www.wrox.com

Recommended
Computer Book
Categories

Operating Systems

Unix and Linux

Paul Love, Joe Merlino et al

Beginning Unix

EISBN: 0-7645-7994-0

Copyright © 2005 by John Wiley & Sons, Inc.

All Rights Reserved. Authorized translation from the English language edition published by John Wiley & Sons, Inc.

本书中文简体字版由 John Wiley & Sons, Inc. 授权清华大学出版社出版。未经出版者书面许可, 不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字: 01-2005-3316

版权所有, 翻印必究。举报电话: 010-62782989 13501256678 13801310933

本书封面贴有清华大学出版社防伪标签, 无标签者不得销售。

本书防伪标签采用特殊防伪技术, 用户可通过在图案表面涂抹清水, 图案消失, 水干后图案复现; 或将表面膜揭下, 放在白纸上用彩笔涂抹, 图案在白纸上再现的方法识别真伪。

图书在版编目(CIP)数据

Unix 入门经典/(美)洛费(Love,P.), (美)默显罗(Merlino,J.)等著; 张楚雄, 许文昭 译. —北京: 清华大学出版社, 2006. 4

书名原文: Beginning Unix

ISBN 7-302-12374-8

I. U… II. ①洛… ②默… ③张… ④许… III. UNIX 操作系统 IV. TP316.81

中国版本图书馆 CIP 数据核字(2006)第 002730 号

出版者: 清华大学出版社 地址: 北京清华大学学研大厦

<http://www.tup.com.cn> 邮编: 100084

社总机: 010-62770175 客户服务: 010-62776969

组稿编辑: 曹 康

文稿编辑: 李 阳

封面设计: 孔祥丰

版式设计: 康 博

印装者: 北京鑫海金澳胶印有限公司

发行者: 新华书店总店北京发行所

开本: 185×260 印张: 23.25 字数: 595 千字

版次: 2006 年 4 月第 1 版 2006 年 4 月第 1 次印刷

书号: ISBN 7-302-12374-8/TP·7933

印数: 1~4000

定价: 39.90 元

前 言

计算机的许多领域,从具有巨大存储空间的新存储形式到比第一代计算机用户所能想象的强大得多的操作系统,都发生了很大的变化。30 多年前,人们设计并开发了 Unix 操作系统,它已成为计算机演化进程的一部分,而且 Unix 仍然是执行关键任务的最为流行的操作系统之一。

Unix 是当前一些用得最多的操作系统的基础,从 Apple 公司的 Mac OS X 系统到 Linux,再到更加广为人知的 Unix 版本,例如 Sun 公司的 Solaris Unix 和 IBM 公司的 AIX 系统。现在,许多 Unix 版本对用户和企业都是免费的,这使得 Unix 拥有一个很大的用户群,比当初刚开发时人们所想象的要大得多。现在 Unix 被看作是一个用户友好的、非常安全和健壮的操作系统,而不是曾经被认为只对计算机专家有用的冷冰冰的、只有命令行的操作系统。

本书涵盖了 Unix 操作系统的各方面的内容。其独特之处在于它不仅包含标准的 Unix 系统,如 Sun 公司的 Solaris Unix 和 IBM 公司的 AIX 系统,还包含 Unix 的派生系统,如 Apple 公司的 Mac OS X 系统以及各种 Linux。此外,本书包括一个独有的转换章节,解释了如何将已知的 Mac OS X-specific 或者 Windows 操作系统命令转换为与它们等价的 Unix 指令,这大大简化了从其他操作系统到 Unix 的转化。

为了提高学习本书的效率,您可以使用 KNOPPIX 操作系统。这个功能完整的 Linux 版本可以让您重启计算机进入 Linux 环境。KNOPPIX 不需要技术经验,并且它不会破坏或改变当前的操作系统。对读者来说,利用 KNOPPIX 来理解本书是一个简单易行的方法,在学习 Unix 的同时避免出现丢弃计算机上的数据或者操作系统的风险。

本书读者对象

本书适用于任何对理解 Unix 操作系统,包括 Unix 的任何可用的派生系统(例如 Apple OS X、Linux,或者 BSD)的概念和操作感兴趣的人。本书是为那些绝对的 Unix 新手设计的,包括那些只用过不同 Unix 系统(Apple 的 Aqua 界面、KDE、GNOME 等)的图形界面的用户。本书对资深的 Unix 用户也有用,因为没有人了解 Unix 的所有内容,它可以作为对已知概念的回顾或者作为填补某些知识领域内的漏洞的工具。

本书对读者的技术水平或使用计算机的经历不做任何要求。如果读者曾经使用过计算机和其他操作系统,如 Mac OS X 或者 Microsoft Windows,那么对某些概念就会理解得更快一些,但是所有读者都会从本书获得一定的收益。

本书主要内容

本书涵盖所有 Unix 版本的最基本的形式,以及所有 Unix 版本及其派生系统通用的命令和概念,包括:

- Apple 公司的 Mac OS X
- Red Hat Linux
- Mandrakelinux
- IBM 公司的 AIX
- 任何版本的 Linux
- 任何版本的 BSD(FreeBSD, OpenBSD, NetBSD)

本书重点放在 Sun 公司的 Solaris、Mac OS X 和 Linux 上，因为获得它们最容易。不同的 Unix 版本使用的原理和命令相同，只有很小的差异，因此任何版本的 Unix 都可以使用本书。

本书还涉及到基本的编程，包括 shell 脚本编程和 Perl 编程，这可以让我们的系统尽可能地自动化——这也是 Unix 操作系统功能强大的一个表现。掌握这些编程概念将为用户学习其他书籍所涵盖的更高级的编程知识打下一个坚实的基础。

本书组织结构

本书首先讲解 Unix 操作系统的基本概念，然后逐步深入到后面更高级的主题和编程技术。如果您对某一章所涵盖的概念或命令比较熟悉，可以直接跳到需要学习的章节。

第 1~4 章介绍了理解 Unix 方法学所需的基本信息，Unix 是如何设计的，以及登录和退出 Unix 系统的基础知识。

- **第 1 章：Unix 基础。** Unix 的基础知识，包括历史和术语以及 Unix 设计和哲学体系的一些核心概念。本章有助于理解 Unix 操作系统背后的一些文化。
- **第 2 章：起步。** 本章描述了为有效地使用 Unix 操作系统所必须了解的首要内容，包括在 Unix 的启动过程中会发生哪些事情、如何登录、用户环境(shell)是怎样构造的，以及如何正确地退出 Unix 系统。
- **第 3 章：理解用户和组。** 了解系统中用户和组的工作原理对于理解怎样才能有效地使用系统至关重要。本章涵盖了用户账户和组的所有方面，包括如何添加、修改和删除用户账户，以及如何利用 su 命令变成另外一个用户。
- **第 4 章：文件系统。** Unix 文件系统是 Unix 系统整体中最关键的组成部分之一。文件系统允许用户存储和操作自己的文件。本章将从用户和系统管理员的角度来说明 Unix 文件系统是什么以及要如何使用它。读者将会学会如何有效地使用文件系统，从而可以避免与文件系统管理有关的一些常见问题。

第 5~7 章使读者可以进行实际的操作，从定制自己的工作环境到编辑 Unix 上的文件。这些章节将扩大读者对 Unix 命令的了解。

- **第 5 章：定制工作环境。** shell 是在 Unix 中完成日常工作所用到的最基本的环境。Unix 提供了很多方法来定制工作环境以适应各种需要。本章将介绍多种配置方法，这些方法适用于各种 Unix shell 用户。
- **第 6 章：深入 Unix 命令。** Unix 有数百条实现各种任务的不同命令。为了有效地利用系统来完成日常工作，需要理解那些最常用的命令，本章将给出其中一些命令的基础。
- **第 7 章：用 vi 编辑文件。** vi 编辑器是 Unix 中最古老而且使用最广泛的文本编辑器之一。通常认为它是一个整体式的且难以使用的编辑器，但是，就像我们将要学到的那

样，它是编辑文件的一种快速且强大的方式。本章研究了有效地使用 vi 编辑器创建和编辑文件的各个方面。

有了一个好的基础，下面将转入几个更高级的主题。第 8~11 章讨论了如何利用一些强有力的 Unix 工具，怎样管理进程，以及如何调度程序使其在指定的时间运行。第 12 章则阐述了安全性这一重要主题。

- **第 8 章：高级工具。**本章介绍了正则表达式的概念和 Unix 用户可以使用的一些更高级的工具。
- **第 9 章：高级 Unix 命令：Sed 和 AWK。**sed 和 awk 是两个非常强大的工具，它们使得用户能以一种高效的方式来操作文件。这些命令很重要，读者将会发现自己会频繁地用到它们。本章将说明如何使用这些命令。
- **第 10 章：作业控制和进程管理。**本章涵盖了 Unix 进程的基础知识以及如何控制和管理 Unix 操作系统的这些重要组成部分。作为对进程的扩展，将回顾和解释工作控制。
- **第 11 章：在指定时间运行程序。**在指定时间运行程序而无需用户或管理员干涉，这使得用户或管理员能够在最少用户利用系统时在系统影响最小的情况下运行程序。本章介绍如何在不同时间运行命令并讨论影响进程的环境变量。
- **第 12 章：安全性。**多年以来，Unix 的安全特性已经颇为彻底，但是，与很多操作系统一样，可以使它更加安全以防范来自外部或内部的恶意实体。本章回顾了系统安全的基础，然后介绍了一些使系统更加安全所能采取的基本步骤。

第 13~17 章研究 shell 脚本编程以及使 Unix 系统中的普通任务“自动化”的其他方法。虽然这些任务通常属于系统管理员的职责范围，但是其他用户(包括家庭用户)，也会从中受益。

- **第 13 章：基本 shell 脚本编程。**对许多用户来说，shell 脚本编程是走向更高级编程语言的必由之路。本章研究主要的 Unix shell 中的编程基础，从而使用户转变为初级程序员的过程更为简单。
- **第 14 章：高级 shell 脚本编程。**本章在第 13 章的基础上更进一步，转入更高级的编程主题，使读者具备可以为任何任务编写 shell 脚本的能力。
- **第 15 章：系统日志。**对用户、管理员和程序员来说，日志极为重要。它是系统与用户进行交互的出口。无论是出现问题还是成功地系统操作，所有的事情都通过日志与用户进行通信。
- **第 16 章：Unix 网络互联。**本章涵盖了与其他系统进行通信的所有方面，包括网络管理和为通常的网络任务编写脚本。
- **第 17 章：Perl 编程实现 Unix 自动化。**Perl 是在 Unix 以及其他操作系统上使用得最为普遍的编程语言之一。Perl 能使用户很快地写出简洁有用的程序。本章介绍了 Perl 语言的编程基础并告诉读者如何使用 Perl 来自自动化通常的 Unix 任务。

第 18~19 章包含了两个重要的主题：备份数据和安装 Unix 程序。

- **第 18 章：备份工具。**本章描述了 Unix 系统上一些可用于备份和恢复系统的工具，当发生意外删除、主系统失灵或者其他灾难时就可以恢复系统。
- **第 19 章：从源代码安装软件。**虽然 Unix 包含许多默认安装的程序，但还是会有很多其他需要安装的程序。本章讲述如何从源代码和预编译的二进制文件出发来安装软件。

第 20~21 章为那些熟悉 Microsoft Windows、Microsoft DOS、Mac OS 9 和 Mac OS X 的读者提供了一个到 Unix 操作系统的映射。对那些已经用过其他操作系统并且想把 Unix 和他们已

经知道的内容作比较的人来说，这些章节是一个很好的参考。

- **第 20 章：转换：适用于 Mac OS 用户的 Unix。** Mac OS X 建立在 Unix 的基础上，但是在标准 Unix 和 Apple 公司的 Mac OS X 之间存在一些细小的差别。本章将典型的 Mac OS(X, 9 及以下版本)命令和概念转换为与它们等价的 Unix 命令或概念。对 Apple 公司任何版本的操作系统用户而言，本章将使他们转移到 Unix 的过程变得更为简单。
- **第 21 章：转换：适用于 Windows 用户的 Unix。** Microsoft Windows 是当前处于主流地位的操作系统。本章将最常用的 Windows 和 MS-DOS 命令转换为与它们等价的 Unix 命令，从而简化从这些操作系统到 Unix 的转移过程。

本书包含两个附录。附录 A，“答案”，给出大部分章节后面所附习题的解答。这些习题可以用来检验读者对相应章节中讨论的概念的掌握情况。附录 B，“有用的 Unix 站点”，给出了因特网上一些最好的 Unix 相关站点的链接。

源代码

如果希望剪切和粘贴而不是手工输入代码，本书中的源代码可以在 www.wrox.com 网站或 www.tupwk.com.cn/downpage 网站上在线获得。在 Wrox 站点上，可以通过查找书名(Beginning Unix)或 ISBN(0-7645-7994-0)找到本书的源代码。

勘误表

本书已经检查过技术和语法错误，但是还是难免会存在差错。本书的勘误记录可在 www.wrox.com 上的书籍详细资料部分获得。如果在书中发现没有列出的错误，请进入 www.wrox.com/contact/techsupport.shtml 页面并填写表格提交该错误，作者将不胜感激。通过提交发现的差错，可以帮助我们使本书更完善。

目 录

第 1 章	Unix 基础	1
1.1	简史	1
1.2	Unix 的版本	1
1.3	操作系统组件	3
1.3.1	Unix 内核	3
1.3.2	shell	4
1.3.3	其他组件	5
1.4	小结	5
第 2 章	起步	6
2.1	系统启动	6
2.2	登录和退出 Unix	9
2.2.1	GUI 登录	9
2.2.2	命令行登录	11
2.2.3	远程登录	13
2.2.4	shell	16
2.2.5	退出	16
2.3	关闭系统	16
2.4	使用联机帮助页	17
2.5	小结	19
第 3 章	用户和组	20
3.1	账户基础知识	20
3.1.1	根账户	20
3.1.2	系统账户	20
3.1.3	用户账户	21
3.1.4	组账户	21
3.2	管理用户和组	21
3.2.1	/etc/passwd	21
3.2.2	/etc/shadow	24
3.2.3	/etc/group	26
3.2.4	Mac OS X 的不同之处	28
3.3	管理账户和组	29
3.3.1	账户管理	29
3.3.2	组管理	32

3.3.3	使用图形用户界面工具进行用户管理	32
3.4	变成另一个用户	34
3.5	与用户和组相关的命令	35
3.6	小结	37
3.7	练习	37
第 4 章	文件系统	38
4.1	文件系统基础	38
4.1.1	目录结构	39
4.1.2	根的基本目录	40
4.2	路径和大小写	41
4.3	文件系统导航	41
4.3.1	pwd	42
4.3.2	cd	42
4.3.3	which 和 whereis	43
4.3.4	find	44
4.3.5	file	44
4.3.6	ls	44
4.4	文件类型	46
4.5	链接	46
4.6	文件和目录权限	51
4.7	修改权限	52
4.7.1	以符号模式使用 chmod	52
4.7.2	以绝对模式使用 chmod	53
4.8	查看文件	54
4.9	创建、修改和删除文件	55
4.9.1	删除文件	55
4.9.2	创建和删除目录	56
4.10	基本的文件系统管理	57
4.11	使文件系统可访问	58
4.12	小结	61
4.13	练习	61
第 5 章	定制工作环境	62
5.1	环境变量	62
5.1.1	PS1 变量	62
5.1.2	其他环境变量	63
5.2	路径	64
5.2.1	PATH 环境变量	65
5.2.2	相对路径和绝对路径	66
5.2.3	切换文件系统	66

5.3	选择 shell	67
5.3.1	临时修改 shell	67
5.3.2	修改默认的 shell	68
5.3.3	各种 shell	68
5.4	配置 shell	72
5.4.1	运行控制文件	73
5.4.2	环境变量	78
5.4.3	别名	80
5.4.4	选项	80
5.5	动态共享库路径	81
5.5.1	LD_LIBRARY_PATH	82
5.5.2	LD_DEBUG	82
5.6	小结	83
5.7	练习	83
第 6 章	深入 Unix 命令	84
6.1	命令的剖析	84
6.2	查找命令的相关信息	87
6.2.1	man	87
6.2.2	info	88
6.2.3	apropos	88
6.3	命令的修改	89
6.3.1	元字符	89
6.3.2	输入和输出重定向	90
6.3.3	管道	91
6.3.4	命令置换	92
6.4	操作文件和目录	93
6.4.1	ls	93
6.4.2	cd	94
6.5	常用的文件操作命令	94
6.5.1	cat	94
6.5.2	more/less	94
6.5.3	mv	95
6.5.4	cp	95
6.5.5	rm	95
6.5.6	touch	96
6.5.7	wc	96
6.6	文件所有权和权限	96
6.6.1	文件所有权	96
6.6.2	文件权限	97
6.6.3	umask	98

6.6.4	可执行文件	99
6.7	保持文件系统配额	99
6.8	小结	101
6.9	练习	101
第 7 章	用 vi 编辑文件	102
7.1	使用 vi	102
7.2	在文件中移动	104
7.3	搜索文件	108
7.4	退出并保存文件	109
7.5	编辑文件	110
7.5.1	删除字符	111
7.5.2	修改命令	113
7.5.3	高级命令	114
7.6	帮助	116
7.6.1	运行命令	117
7.6.2	替换文本	117
7.7	vi 的版本	120
7.8	小结	121
7.9	练习	121
第 8 章	高级工具	122
8.1	正则表达式和元字符	122
8.1.1	理解元字符	123
8.1.2	正则表达式	127
8.2	使用 SFTP 和 FTP	128
8.3	更高级的命令	132
8.3.1	grep	132
8.3.2	find	133
8.3.3	sort	134
8.3.4	tee	136
8.3.5	script	136
8.3.6	wc	136
8.4	小结	137
8.5	习题	137
第 9 章	高级 Unix 命令: Sed 和 AWK	138
9.1	sed	138
9.1.1	使用 -e 选项	140
9.1.2	sed 文件	140
9.1.3	sed 命令	142
9.2	AWK	143

9.2.1	用 AWK 提取数据	144
9.2.2	使用模式	145
9.3	利用 AWK 编程	146
9.4	小结	148
9.5	练习	148
第 10 章	作业控制和进程管理	149
10.1	进程	149
10.2	shell 脚本	150
10.3	正在运行的进程	151
10.3.1	ps 语法	152
10.3.2	进程状态	152
10.4	系统进程	153
10.5	进程属性	156
10.6	停止进程	156
10.6.1	进程树	158
10.6.2	僵死进程	159
10.7	top 命令	159
10.8	/proc 文件系统	161
10.9	SETUID 和 SETGID	162
10.10	shell 作业控制	163
10.11	小结	165
第 11 章	在指定时间运行程序	166
11.1	系统时钟	166
11.1.1	使用 date 检查和设置系统时钟	167
11.1.2	在 Linux 上利用 hwclock 同步时钟	167
11.1.3	利用 NTP 同步系统时钟	168
11.2	安排将来运行的命令	168
11.2.1	利用 cron 执行程序	169
11.2.2	使用 at 命令进行一次性执行	174
11.3	小结	177
11.4	练习	177
第 12 章	安全性	178
12.1	安全性的基础知识	178
12.1.1	资产价值保护	178
12.1.2	潜在的问题	179
12.2	保护 Unix 系统	180
12.2.1	口令的安全性	180
12.2.2	口令破译程序	181
12.3	限制管理访问	181

12.3.1	UID 0	181
12.3.2	根用户管理选项	182
12.3.3	设置 sudo	183
12.4	系统管理的预防性任务	185
12.4.1	删除不需要的账户	185
12.4.2	修补、限制或删除程序	186
12.4.3	禁用不需要的服务	186
12.4.4	监控并限制对服务的访问	187
12.4.5	实现内置防火墙	188
12.4.6	其他的安全程序	188
12.5	小结	188
12.6	练习	189
第 13 章	基本 shell 脚本编程	190
13.1	注释脚本	190
13.2	开始脚本编程	192
13.2.1	调用 shell	192
13.2.2	变量	193
13.2.3	从键盘读取输入	194
13.2.4	特殊变量	194
13.2.5	退出状态	195
13.3	流程控制	195
13.3.1	条件流程控制	195
13.3.2	迭代流程控制	201
13.4	选择脚本编程 shell	202
13.5	小结	202
13.6	练习	203
第 14 章	高级 shell 脚本编程	204
14.1	高级脚本编程的概念	204
14.1.1	输入和输出重定向	205
14.1.2	命令替换: 反引号和圆括号扩展	206
14.1.3	使用环境变量和 shell 变量	207
14.2	shell 函数	208
14.2.1	返回值	209
14.2.2	嵌套函数和递归	209
14.2.3	作用域	210
14.2.4	函数库	212
14.2.5	信号和陷阱	214
14.2.6	文件处理	214
14.2.7	数组	217
14.3	shell 的安全性	219

14.3.1	攻击可能来自何处	220
14.3.2	采取预防措施	220
14.3.3	受限 shell	220
14.4	系统管理	222
14.4.1	收集信息	222
14.4.2	执行任务	223
14.4.3	调试脚本	224
14.5	小结	225
14.6	练习	225
第 15 章	系统日志	226
15.1	日志文件	226
15.2	syslogd	226
15.2.1	syslog.conf	227
15.2.2	消息	230
15.2.3	日志记录器	231
15.3	轮循日志	231
15.4	监视系统日志	232
15.4.1	logwatch	232
15.4.2	swatch	234
15.5	小结	236
15.6	练习	236
第 16 章	Unix 网络互联	237
16.1	TCP/IP	237
16.1.1	TCP	237
16.1.2	IP	238
16.1.3	与 TCP/IP 一起使用的其他协议	238
16.1.4	网络地址、子网、子网掩码和 TCP/IP 路由选择	240
16.2	为 Unix 系统设置 TCP/IP 网络	243
16.2.1	TCP/IP 网络请求配置	243
16.2.2	动态设置	245
16.2.3	发送 TCP/IP 网络请求	246
16.2.4	回应 TCP/IP 网络请求	248
16.2.5	inetd	249
16.3	网络管理工具	251
16.3.1	通过 Traceroute 跟踪网络的性能	251
16.3.2	防火墙	252
16.3.3	例行检查网络延迟	253
16.4	小结	255
16.5	练习	255

第 1 章 Unix 基础

30 多年前,美国电话电报公司(AT&T)的贝尔实验室里的一群研究人员创建了 Unix 操作系统。经过随后 30 多年的持续发展, Unix 在许多地方都建立了自己的应用领域,从普遍存在的大型机到家用计算机,甚至到最小的嵌入式设备。本章将简要回顾 Unix 的历史,讨论目前所用的大部分 Unix 系统之间的一些差异,并介绍基础 Unix 操作系统的基本概念。

1.1 简史

就计算机而言, Unix 有很久远的历史。AT&T 公司的贝尔实验室在退出与通用电气(General Electric, G.E.)和麻省理工学院(MIT)的长期合作之后开发了 Unix, 该合作旨在创建一个用于 G.E.大型机的、称为 MULTICS(Multiplexed Operating and Computing System, 多操作和计算系统)的操作系统。1969 年, 贝尔实验室的研究人员创建了 Unix 的第一个版本(当时称为 UNICS, 或单操作和计算系统(Uniplexed Operating and Computing System)), 也正是由它演化为今天的通用 Unix 系统。

Unix 从面向最初的 PDP-7 小型机逐步演化到面向不同的机器结构, 并在大学里使用。为了扩大其使用范围, Unix 的源代码只需要很少的费用就可以获得。由于 Unix 在大学里得到认可, 因此使用它的学生陆续毕业并担任负责采购系统和软件的职位。当这些人开始为他们的公司购买系统时, 他们会考虑 Unix, 因为他们对它比较熟悉, 这样就进一步推广了 Unix 的应用。从 Unix 诞生的第一天起, 该操作系统的发展就一直引人注目, 因此现在它是许多大公司的计算机系统的支柱。

如今, Unix 不再是一个只取首字母的缩写词, 但是它来源于缩写词 UNICS。Unix 的开发人员和用户使用了大量的缩写来表示系统中的内容并用作命令。

1.2 Unix 的版本

早期可以获得的 Unix 是源代码而不是典型的二进制形式。这样, 其他人修改代码以满足自己的需要就会比较容易, 同时这些修改导致代码产生很多分叉, 也就意味着现在有许多不同的版本(也称为风格)。

源代码逐行说明程序或应用程序的操作, 描绘了程序的内部工作。有权限使用源代码使得理解程序内部发生的一切变得更简单, 并使修改程序更为容易。大多数商业程序都是以二进制的形式发布, 表示它们可以直接运行, 但是其内部的代码行是不可读的。

Unix 主要有两个可用的基础版本: AT&T System V 和 Berkley Software Distribution(BSD)。

绝大多数 Unix 版本都建立在这两个版本中的一个版本上。两者的主要区别在于可用的实用程序以及文件结构的实现不同。大多数 Unix 版本综合了两个基础版本的特性。例如，有些将 System V 版本的实用程序放在 `/usr/bin` 中，而将 BSD 版本的实用教程放在 `/usr/ucb/bin` 中，这样就可以选择想要的实用程序。这种安排说明 Unix 提供了以不同方式工作的灵活性。

Unix 系统的版本众多，用户可以从中任意选择，可以选择最符合自己需要或系统需求的版本。许多人认为能够进行选择是一种优势，但是也有些人认为这是一个弱点，因为这些存在细微差别的版本和风格导致了一些不兼容(例如在实现、命令、通信或方法上)。不存在 Unix 的“正确”版本，也没有哪个版本比其他的更官方一点；它们只是不同的实现而已。以 Linux 为例，它是 Unix 的一个变种。在 1991 年首次创建 Linux 时，就完全将它构建为一个免费的类 Unix 的系统，以代替可用的昂贵的商业 Unix 版本。表 1-1 是一些更为流行的 Unix 版本。

表 1-1

更流行的 Unix 版本	更流行的 Unix 版本
Sun Microsystem's Solaris Unix	Yellow Dog Linux(用于 Apple 系统)
IBM AIX	Santa Cruz Operations SCO OpenServer
Hewlett Packard HP-UX	SGI IRIX
Red Hat Enterprise Linux	FreeBSD
Fedora Core	OpenBSD
SUSE Linux	NetBSD
Debian GNU/Linux	OS/390 Unix
Mac OS X	Plan 9
KNOPPIX	

每种风格实现其 Unix 版本的方式都会稍有不同，但是即使在有些系统上命令的实现会发生变化，其核心命令及功能仍然遵循两个主要版本之一的原则。Unix 的多数版本使用 SVR4(System V)并添加 BSD 组件作为一个选项以最大化互操作性。对命令尤其如此，例如，在大多数系统中可用的 `ps` 命令(用于显示进程)有两个版本。`ps` 的一个版本可能位于 `/usr/bin/ps`(System V 版本)，而另一个版本可能位于 `/usr/ucb/bin` 内(BSD 版本)。命令的操作类似，但是以不同的方式给出输出或接受可选组件。

许多供应商都试图标准化 Unix 操作系统。其中最成功的尝试是非商业的电气和电子工程师协会(Institute for Electrical and Electronics Engineers, IEEE)的一个产品，即标准 1003(IEEE 1003)，也称为 POSIX(便携式操作系统界面, Portable Operating Systems Interface)标准。该标准已经向国际标准化组织(International Organization for Standardization)注册为 ISO/IEC 9945-1，可以在 <http://iso.org/iso/en/CombinedQueryResult.CombinedQueryResult?queryString=9945> 找到。POSIX 标准融合了统一 Unix 规范(Single Unix Specification, SUS)，变成一个适用于所有 Unix 版本的综合标准。它保留了原来的名字，仍称作 POSIX 标准。并不是所有 Unix 版本都严格按照字句来遵守 POSIX 标准，但是大多数都坚持标准中所描述的主要原则。

早期的 Unix 系统跟多数软件一样，主要是待售的商业产品，通常都必须付一定的费用才能获得运行该操作系统的权力。1984 年，工程师 Richard Stallman 开始着手 GNU 计划。该计

划致力于创建一个类 Unix 的、任何人都可以免费发布和使用的操作系统。他现在管理着免费软件基金会(Free Software Foundation, <http://gnu.org/fsf/fsf.html>), 他和他的支持者们创建的许多程序广泛应用于商业的和开放源码的 Unix 版本中。

GNU 代表 GNU's Not Unix, 是一个首字母递归的缩略词。GNU 计划想创建一个类似 Unix 的操作系统, 而不是一个 Unix 的派生系统(这意味着它是 Unix 的源码副本)。

1991 年, 芬兰研究生 Linus Torvalds 开始着手开发名为 Linux 的类 Unix 系统。Linux 实际上是一个内核(内核将在本章的后面进行讨论), 而大多数人都比较熟悉的部分——工具、shell 和文件系统——则由其他人创建(通常是 GNU 组织)。由于 Linux 计划具有发展动力, 它很快成长为 Unix 市场中的主要竞争者。许多人都是经由 Linux 转到 Unix 的, Linux 使台式机具备了过去要花费数千美元购买 Unix 才能获得的功能。Linux 的强大之处在于它开放使用许可证, 允许软件自由发布而不需要任何授权。对终端用户的惟一要求是对软件所做的任何修改都要与其他人共享, 从而使得软件以难以置信的速度成熟起来。发布 Linux 的许可证称为 GNU 公共许可证(GNU Public License, GPL), 可以在 <http://gnu.org/licenses/licenses.html> 获得。

另外一个流行的免费 Unix 版本是 BSD 家族的软件, 它们使用非常宽泛的 BSD 许可证(<http://opensource.org/licenses/bsd-license.php>)。该许可证允许对软件自由修改而不要求将软件源码提供给其他人。自 1994 年里程碑似的诉讼结束后, BSD Unix 开始自由发布并发展成 NetBSD, FreeBSD 和 OpenBSD, 并且它还构成 Darwin(Mac OS X 的基础)的底层技术。

自从 Microsoft Windows 操作系统飞速发展以来, Unix 操作系统就开始走下坡路, 这些可以免费获得的 Unix 派生系统给 Unix 操作系统注入了新的活力。另外, Apple 公司已经成为 Unix 系统最大的供应商。目前, Unix 在企业环境和终端用户台式机市场中都在向前发展。

1.3 操作系统组件

操作系统是用户和系统硬件之间的软件接口。不管所使用的操作系统是 Unix、DOS、Windows 还是 OS/2, 用户或程序员所做的一切都是以某种方式与硬件进行交互。在计算机发展的最初阶段, 用文本输出或一系列光点(lights)来表示系统请求的结果。Unix 最初是命令行界面(CLI)系统——不存在让系统更易于使用或者看起来更舒服的图形用户界面(GUI)。现在 Unix 有一些可以定制的用户界面, 它们是 Mac OS X Aqua、Linux 的 KDE 和 GNOME 界面, 这使得 Unix 系统真正可以用于普通用户的台式机。

下面我们将简要回顾一下构成 Unix 操作系统的各个组件: 内核、shell、文件系统和实用程序(应用程序)。

1.3.1 Unix 内核

内核是 Unix 系统的最底层。它提供了系统的核心功能并允许进程(程序)以一种有序的方式访问硬件。基本上, 内核控制进程、输入/输出设备、文件系统操作, 以及操作系统所需的任何其他关键功能。它还管理内存。这些都称为自治功能, 因为它们的运行不需要用户进程中的指令。内核支持系统以多用户(同时有多于一个的用户访问系统)、多任务(某个时刻有多个程序运行)模式运行。

内核是为特定的硬件构建的，它将运行在这些硬件上，因此为 Sun Sparc 机器构建的内核不能不经过修改就直接运行在 Intel 处理器上。由于内核处理的都是非常底层的任务，如访问硬件驱动或管理多任务，而且它没有友好的用户界面，因此通常不能由用户访问。

内核最重要的功能之一是简化了进程的创建和管理。进程是被执行的程序，在有些操作系统中也叫做工作或任务，这些程序都是由其所有者(人或系统)发起调用或执行的。进程的管理可能非常复杂，因为一个进程通常会调用其他进程，在 Unix 中称为分叉(forking)。进程还经常需要相互通信，收发允许其他动作执行的信息。内核在用户毫无感觉的情况下管理这一切。

内核还管理内存，它是任何系统的核心元素。它必须为所有进程提供足够的内存，并且有些进程还会需要大量内存。有时候一个进程需要的内存会超出可使用的内存大小，例如太多其他进程正在运行。这时候就要用到虚拟内存。当没有足够的物理内存时，系统通过把进程的一部分转移到硬盘上以设法容纳进程。当再次需要进程中被转移到硬盘上的那一部分时，再将其返回到物理内存中。这个过程称为页面调度(paging)，它使得系统即使在有限的物理内存的条件下也能具备多任务处理的能力。

虚拟内存的另一个方面称为交换(swap)，内核凭借这种机制识别出最不繁忙的进程或者是那些不需要立刻执行的进程。然后内核将整个进程移出 RAM，放入硬盘，直到下次再需要该进程时为止，此时进程可以从硬盘或者物理 RAM 上运行。两者的不同之处在于，页面调度只是将进程的一部分移入硬盘，而交换则将整个进程移入硬盘。Unix 中用作虚拟内存的硬盘分段称为交换空间(swap space)，在阅读本书时最好记住这个术语。交换空间耗尽将引起严重的问题，直至使系统失效，因此必须确保始终有足够的交换空间。无论何时发生交换，都要承受一定的性能下降，因为硬盘比物理 RAM 要慢一些。保证系统有足够的物理 RAM 就可以避免发生交换。

1.3.2 shell

shell 是一个命令行解释器，它使得用户能够与操作系统进行交互。shell 为系统提供了内核之上一层的功能，可以直接用它来管理和运行系统。所用的 shell 将会极大地影响您的工作方式。最初的 Unix shell 经过多年的发展已经分化出许多种不同的类型，每种类型中都有一些其创建者认为是其他 shell 中所缺乏的独有的特性。大多数系统中都有三种主要的 shell: Bourne shell(也叫作 sh)，C shell(csh)和 Korn shell(ksh)。shell 通过命令行以几乎独占的方式使用，命令行是用户与系统之间进行交互的一种基于文本的机制。

Bourne shell(也简称为 shell)是 Unix 的第一个 shell。它一直是在 Unix 系统上使用得最为广泛的 shell，提供了一种用于脚本编程的语言以及调用其他程序的基本用户功能。shell 适合于日常工作，特别适合于 shell 脚本编程，因为它的脚本便于移植(在其他 Unix 版本的 Bourne shell 中工作)。Bourne shell 惟一的问题在于，相对于一些更为现代的 shell，它的用户交互功能比较差。

C shell 是另外一个在 Unix 系统上普遍可用的流行的 shell。这个 shell 来自于加利福尼亚大学伯克利分校，其创建的目的是改进 Bourne shell 的一些缺点并使其类似于 C 语言(该语言正是建立 Unix 的基础)。控制作业的功能和为命令指定别名的功能(将在第 5 章讨论)使得该 shell 更易于与用户交互。C shell 在处理脚本编程时有一些早期的怪僻，因此通常认为用它创建 shell 脚本其健壮性不如 Bourne shell。人们最终修改了这些怪僻，但是 C shell 仍然有些细微的差异，这是由于提供该 shell 的实体(商业提供者或其他来源)给出了不同实现。

Korn shell 是由 David Korn 创建的，它克服了 Bourne shell 的用户交互问题，并解决了 C shell 的脚本编程怪僻这一缺点。Korn shell 添加了一些 Bourne shell 和 C shell 都没有的功能，综合

了各种 shell 的长处。Korn shell 惟一的缺点是它需要许可证，这样就使得它的使用没有另外两个 shell 普遍。

可用的 shell 远不止上面提到的这些。下面列出了一些在不同 Unix 系统中可用的 shell：

- sh(也称为 Bourne shell)
- PDKSH(Public Domain Korn shell)
- bash(Bourne Again Shell——Bourne shell 修补后的版本)
- Z shell
- TCSH(TENEX C shell)

Unix 的各个方面都有许多不同的实现，可以根据所提供的特性自由选择最符合需要的 shell。第 5 章将详细介绍几种 shell。

1.3.3 其他组件

其他 Unix 组件有文件系统和实用程序。文件系统使得用户能够以统一的方式查看、组织，以及保护存储设备上的文件和目录并与其进行交互。文件系统将在第 4 章深入讨论。

实用程序是一些应用程序，它们使得用户能够在系统上进行工作(不要与 shell 混淆)。这些实用程序包括用于 Internet 定位的 Web 浏览器、文字处理实用程序、e-mail 程序和本书中将要讨论的其他命令。

1.4 小结

本章简要地讨论了 Unix 的历史并介绍了 Unix 的一些不同版本。同时介绍了 Unix 的核心组件——内核、shell、文件系统和实用程序。

过去，人们认为 Unix 只适合于最熟悉计算机的用户以及想了解系统核心功能的人使用，没有考虑到审美学或者用户友好性。现在，Unix 已经发展成适合许多不同类型用户的需要，无论是严肃的企业环境还是计算机初学者的台式机。例如，许多 Unix 版本都有良好的台式机环境，并且，每一台正在出售的 Macintosh 计算机上都运行着一个最新版本的 Unix。

在第 2 章中，将从最基本的登录退出开始学习使用 Unix 系统。

第 2 章 起 步

本章将引导您与 Unix 操作系统进行交互。它介绍 Unix 的启动过程，说明如何登录到系统以及怎样正确关闭系统，并解释 shell 提供的功能。它还介绍了 man 命令，这是 Unix 的内置系统帮助工具。本章的内容是学习其他章节的基础。

2.1 系统启动

从电源关闭状态一直到操作系统完全可用所发生的一切称为启动过程(boot process)。用最简单的术语来说，启动过程由只读存储器(Read-Only Memory, ROM, NVRAM, 或固件)对某些程序的加载所组成，这些程序用于实际启动(开始)系统。该初始化步骤(通常称为自展)识别系统上可启动或开始的设备。一次只能从一个设备上启动或开始，但是，因为可将很多不同的设备识别为可引导的，所以，如果一个启动设备失效，就可以使用另外一个已经识别的设备。这些设备可以自动加载，或者系统会显示一个设备列表，用户可以从中进行选择。图 2-1 显示了 Intel 平台上 Solaris 引导系统中的一组启动设备。

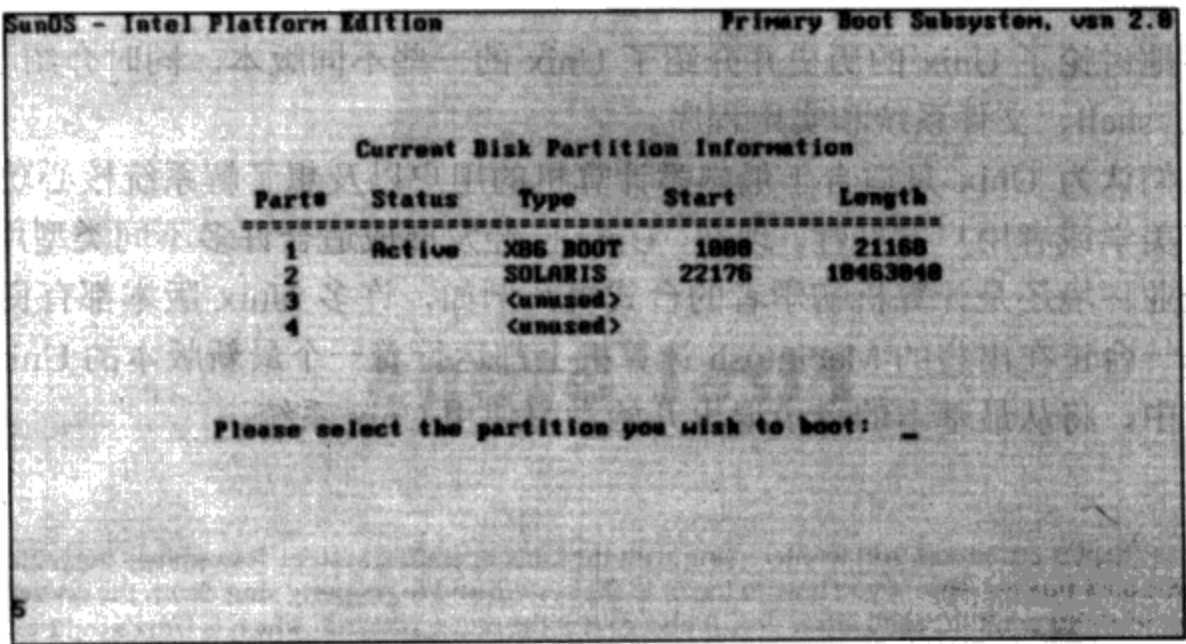


图 2-1

启动设备不一定是硬盘，因为系统可以从网络或可移除的存储设备如 CD-ROM 或软盘启动。启动设备只是保存了在何处加载操作系统的信息。引导程序只识别用于启动的硬件以及该硬件是否可用。

接着，引导程序将控制权转移给内核。至此还没有加载操作系统，对用户的生产来说，系统是不可用的。有些系统利用在屏幕上输出消息来显示启动过程，而有些系统则对用户隐藏了系统消息，用图形界面表示启动过程。图 2-2 显示了在 Solaris 启动过程中已识别的启动驱动器。

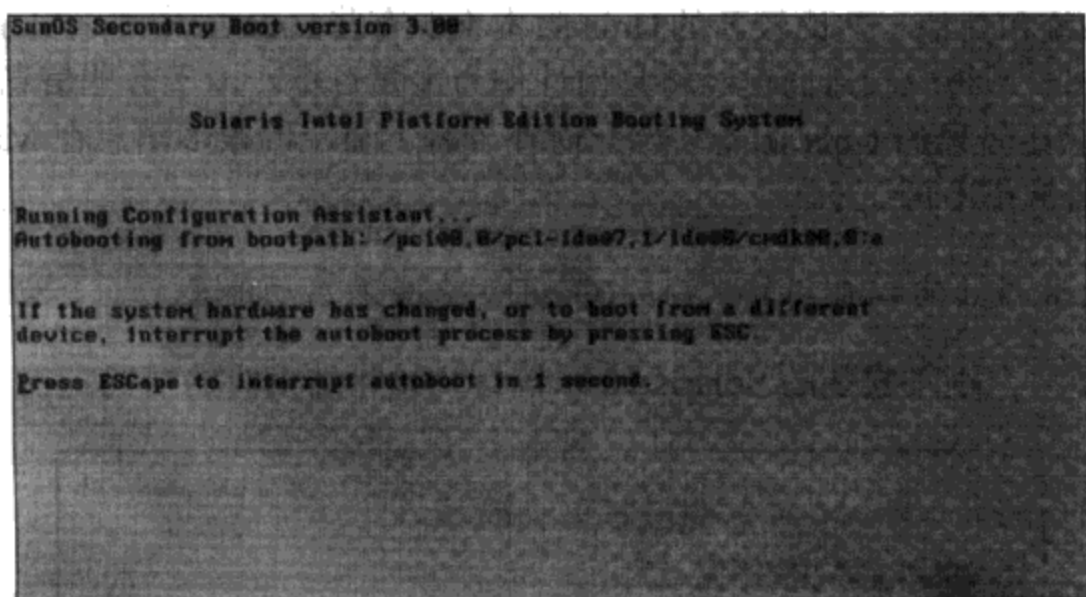


图 2-2

在最初的引导过程完成之后，引导程序开始加载 Unix 内核，内核通常位于系统的根分区。在大多数 Unix 系统上，内核称为 `unix`；而在 Linux 系统中，内核可能称为 `vmunix` 或 `vmlinuz`。它的位置根据 Unix 版本的不同而不同，如下列所示：

- AIX: `/unix`
- Linux: `/boot/vmlinuz`
- Solaris: `/kernel/unix`

这也只是内核的一部分不同位置，但是一般说来，在日常工作甚至开发过程中并不需要修改内核，除非是系统管理员，或者因为特殊的需求要从内核中添加或删除某些功能。

内核的初始化任务根据硬件和 Unix 版本的不同而不同，接下来是初始化阶段，在这个阶段中启动系统进程和脚本。`init` 进程是系统开始的第一个工作，它是其他所有进程的父进程。为了让系统运行，它必须一直处于运行状态。`init` 进程调用初始化脚本并完成系统相关的管理任务，例如启动 `sendmail`、`X` 或窗口服务器(提供图形用户界面)，等等。

`init` 进程查看初始化说明文件，通常将该文件称为 `/etc/inittab`。这个文件说明 `init` 进程应该如何解释不同的运行级别以及在每个运行级别中应该启动哪些脚本和进程。运行级别(run level)是一组进程(最基本的程序)或守护进程(始终都在运行的程序)。

图 2-3 显示了 Mac OS X 系统的初始化阶段。

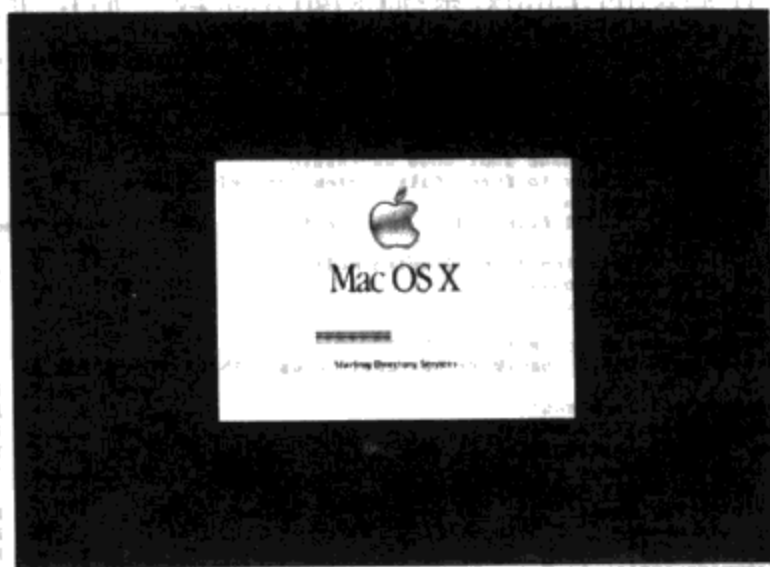


图 2-3

Mac OS X 系统和其他一些比较新的 Unix 版本不像有些 Unix 版本那么复杂, 因为随着 Unix 的发展, 不同 Unix 系统的制造商已经将易用性作为主要目标。由于有些信息对普通的终端用户并没有用, 所以比较早的 Unix 版本初始化时在屏幕上显示的很多消息在 Mac OS X 和用户友好的 Linux 中通常都不再显示。

在 Mac OS X 系统上可以使用转义序列(Cmd+v)来查看启动消息。

图 2-4 显示了一个刚刚安装的 Solaris 10 系统在初始化结束时的情况。

```

configuring IPv4 interfaces: pcn0.
add net default: gateway 192.168.1.1
Hostname: solaris
The system is coming up. Please wait.
checking ufs filesystems
/dev/rdisk/c0d0s7: is clean.
starting rpc services: rpcbind done.
Setting default IPv4 interface for multicast: add net 224.0/4: gateway solaris
syslog service starting.
syslogd: line 24: WARNING: loghost could not be resolved
Nov 20 20:21:07 solaris sendmail[239]: My unqualified host name (solaris) unknow
n: sleeping for retry
Nov 20 20:21:07 solaris sendmail[240]: My unqualified host name (solaris) unknow
n: sleeping for retry
volume management starting.
The system is ready.

solaris console login: _

```

图 2-4

起初, 这类信息看起来比较奇怪, 甚至令人担忧, 但是在脚本和日志中通常会包含对这些消息的解释, 当 Unix 知识越来越丰富时, 就可以据此追踪问题所在。例如, 第 10 行显示了 syslogd(系统日志守护进程, 在 15 章中讨论)中的一个错误: syslogd: line 24:WARNING:loghost could not be resolved。这看起来像是一个大麻烦, 但事实上只是一个小问题, 在/etc/hosts 中添加一个条目就可以解决。第 15 章将介绍更多关于这些消息的内容, 包括如何识别它们以及怎样排查问题。

图 2-5 显示了一个正在启动的 Linux 系统(初始化阶段之后), 同样也有一些可能会令人不安的消息, 如第 3 行显示的内容: Your system appears to have shut down unclearly。

```

[initializing USB controller (usb-uhci): [ OK ]
Mount USB filesystem [ OK ]
Your system appears to have shut down unclearly
Press Y within 2 seconds to force file system integrity check...y
Checking root filesystem
/dev/sda1: 114607/202240 files (0.5% non-contiguous), 507713/564275 blocks
[ OK ]
Remounting root filesystem in read-write mode: [ OK ]
Activating swap partitions: [ OK ]
Starting up RAID devices:
Checking filesystems
/dev/sda6: recovering journal
/dev/sda6: 725/105056 files (0.7% non-contiguous), 15207/371495 blocks
[ OK ]
Mounting local filesystems: [ OK ]
Checking loopback filesystems [ OK ]
Mounting loopback filesystems: [ OK ]
Loading keymap: us [ OK ]
Loading compose keys: compose latin inc [ OK ]
The BackSpace key sends: ^? [ OK ]
Enabling swap space: [ OK ]
Starting netprofile: [ OK ]
INIT: Entering runlevel: 5
Entering non-interactive startup
Checking for new hardware_

```

图 2-5

这些错误通常会自动修复，或者可以利用 `fsck` 命令来修改，这将在第 4 章中介绍。

启动屏幕中包含很多信息，但是当消息在屏幕上显示的时候，并没有必要逐条观察。可以利用 `dmesg` 命令把启动消息搜集起来，有空时再仔细阅读。要修改启动参数，必须修改系统只读存储器(Read-Only Memory, ROM)或者修改 Unix 操作系统的初始化脚本，本书后面将要讨论。

初始化阶段完成之后，系统开始运行并准备接收用户登录。如果是本地登录，将会在系统上看到登录提示符或者图形登录界面。

2.2 登录和退出 Unix

登录意味着作为需要使用资源的有效用户让 Unix 系统进行验证。在试图登录 Unix 系统时，通常会要求用户提供用户名和口令的组合以便通过验证，尽管登录还有更高的机制，如生物测定(例如视网膜扫描)或者每隔几秒钟就会修改口令组合的一次性标记。用户可以使用图形用户界面(GUI)或命令行进行登录。

2.2.1 GUI 登录

如果有一个键盘/鼠标以及一个监控器直接与 Unix 系统相连，就可以像用户登录本地系统那样登录 Unix。最初的登录界面有许多形式，从传统的只提供文本信息的命令行到包含图像的图形登录。来看几个例子。图 2-6 显示的是 Mandrakelinux 的命令行登录界面。

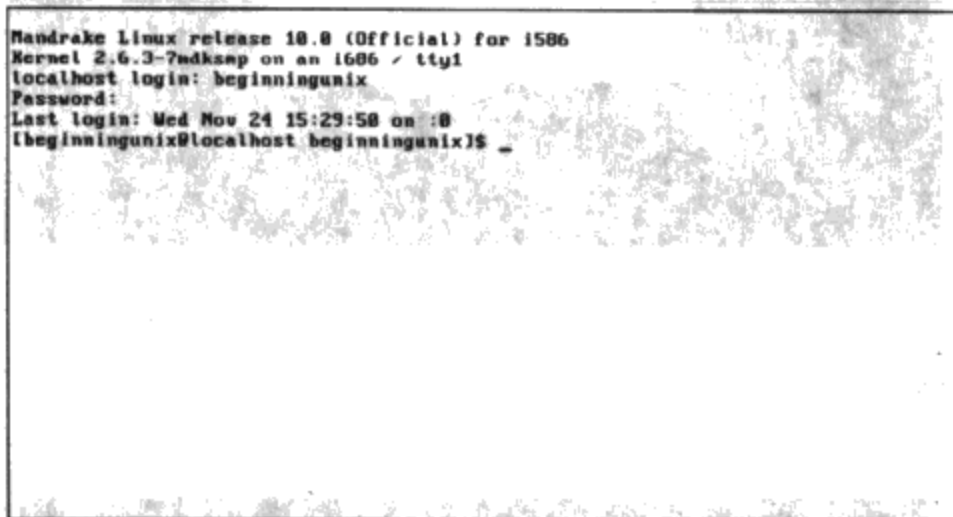


图 2-6

图 2-7 显示的是 Mandrakelinux 系统的图形登录界面。



图 2-7

图 2-8 显示的是 Solaris 10 系统的图形登录界面。

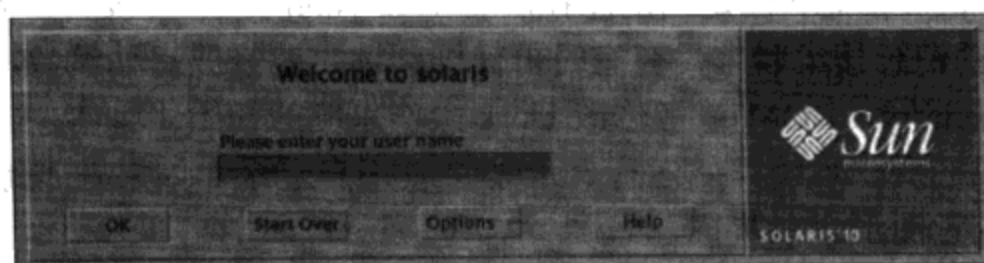


图 2-8

所提供的用户名和口令要与内部系统文件或数据库中的用户名和口令信息相对应，这些文件或数据库包含了一组有效的用户名和口令。图 2-9 显示的是 Mac OS X 系统的登录界面，要求用户选择以哪个用户名登录。

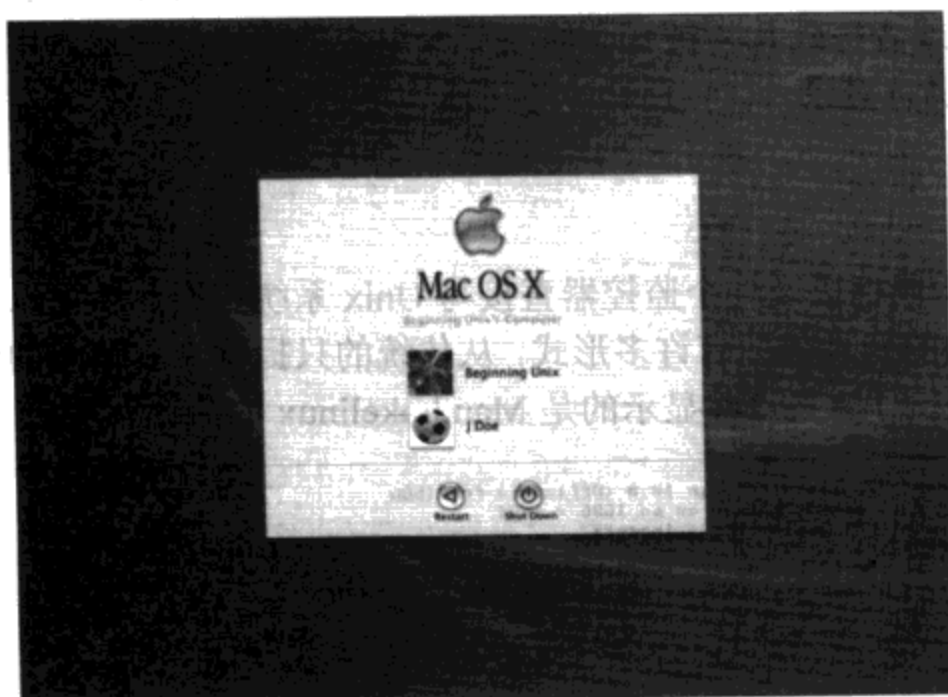


图 2-9

本例中，选择 Beginning Unix 之后，屏幕将会如图 2-10 所示，在这里必须输入一个有效的口令。



图 2-10

错误的口令通常会使系统产生一个文本或图形消息，说明用户所输入的口令是无效的。如果一个用户错误地输入口令超过三次或者其他指定的数字，大多数 Unix 系统就会冻结该账户或者为其设定一个时间延迟。这是出于安全考虑，因此一个人不可能轻易地连续输入不同的口令以试图登录其他人的账户。正确的口令将启动登录过程，如图 2-11 所示。

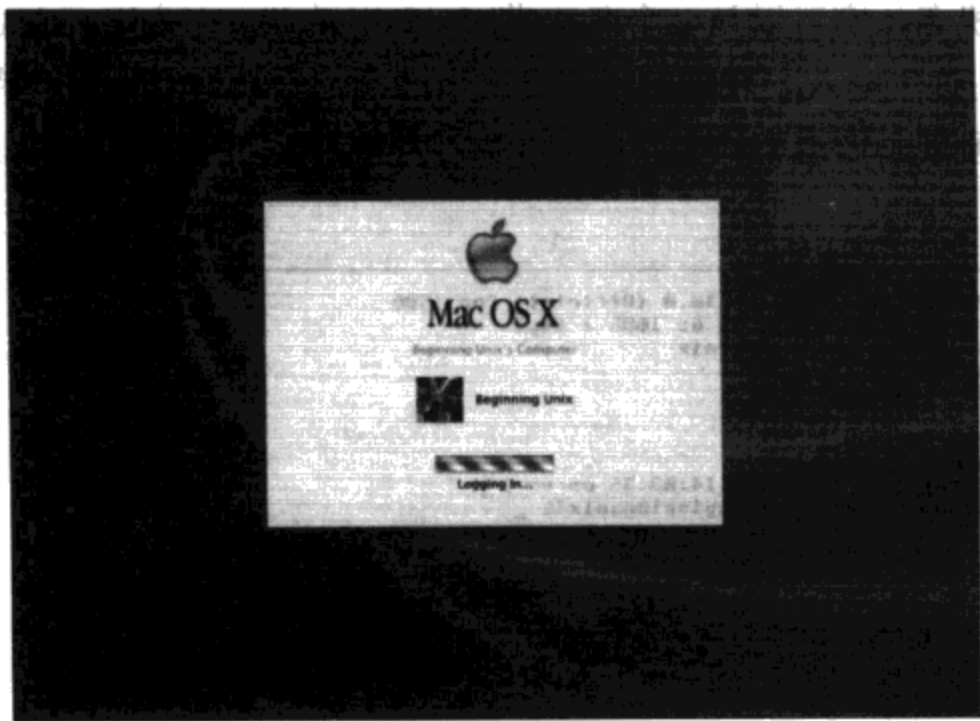


图 2-11

2.2.2 命令行登录

有时候，Unix 系统没有运行图形用户界面，所有工作都使用命令行来完成。在这种情况下，通常会看到一个标语消息，显示正在登录的机器类型，或者是系统管理员创建的消息。而有时候只能看到登录提示符。图 2-12 显示的是一个 Linux 系统的登录界面示例。

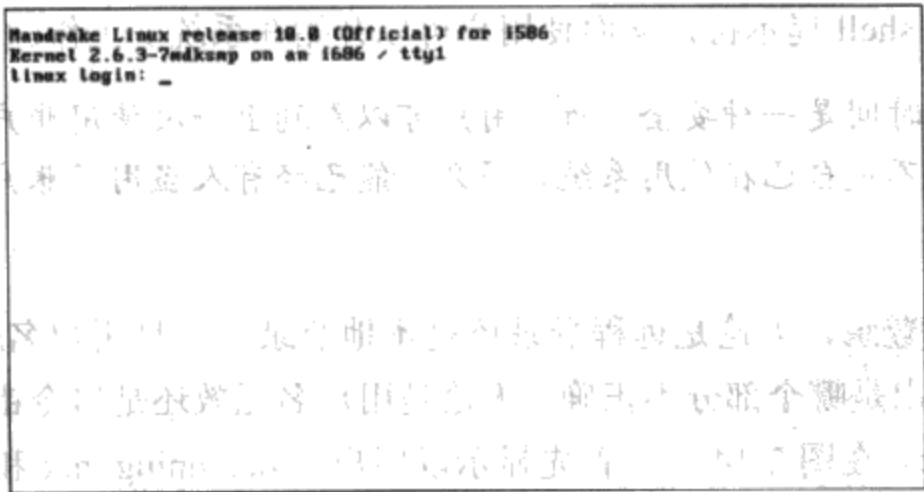


图 2-12

该屏幕上的标语消息是：

```
Mandrake Linux release 10.0 (official) for i586
Kernel 2.6.3-7ndksmp on an I 686
```

该标语信息的第 1 行指出这是一个 Linux 系统, Mandrake 10 发布版本。第 2 行指出系统正在运行的内核, 并指定电传打字机(teletype)编号(屏幕)。标语消息的内容因系统的不同而有所不同, 但是一般都能够明白地显示是什么信息。出于安全考虑, 在那些能够通过互联网(Internet)公开访问的系统上可以缺少这些信息(精确的系统说明使黑客攻击系统更为容易)。

第 3 行显示主机名, 它可以是一个名称(图 2-12 中是 linux)或 IP 地址(如 192.168.1.1), 然后是短语 login:。在这里输入要登录的用户名。请注意, 命令行登录不会提示可以使用的用户名有哪些, 因此必须事先知道用户名。图 2-13 首先显示了一次失败的登录, 然后是一次成功登录的一系列事件。

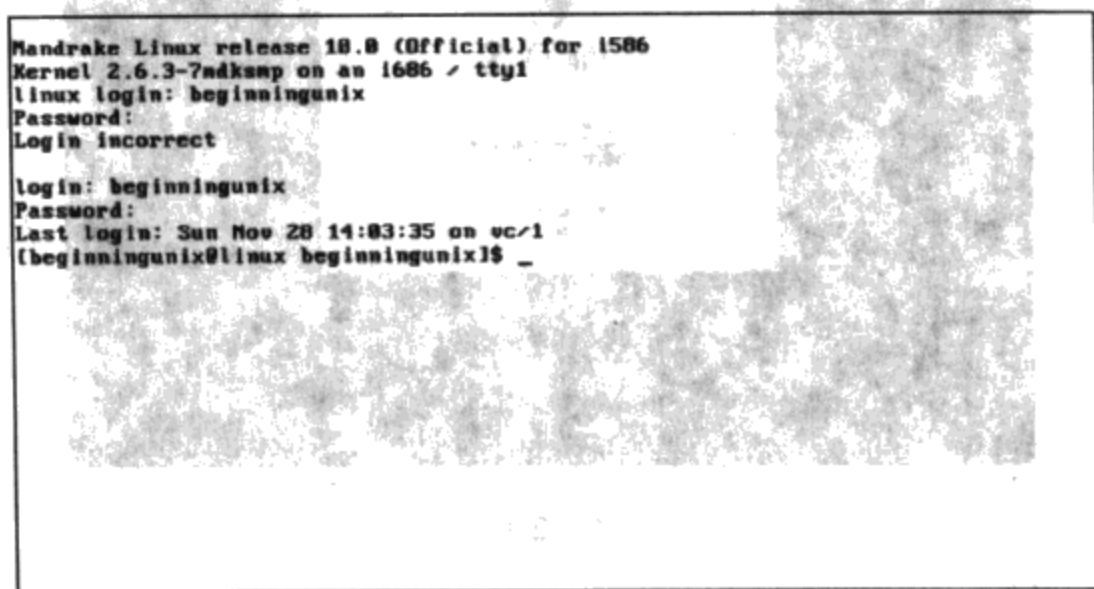


图 2-13

本例中, 用户输入 beginningunix 作为用户名并按下 Enter(或 Return)键。屏幕上出现输入口令请求(第 4 行)。用户为该账户输入一个错误的口令, 系统的响应为 Login incorrect 语句, 接着提示用户重新登录。这一次用户输入正确的用户名和口令。然后系统显示用户上一次登录的时间并显示命令行 shell 提示符, 从而使用户可以开始在系统上工作。

显示上一次登录时间是一种安全特性, 用户可以看到上一次使用账户的时间。如果注意到上一次登录的时间并不是自己在使用系统, 那么可能已经有人盗用了账户。要立即联系系统管理员。

如果使用命令行登录, 无论是远程登录还是本地登录, 一旦用户名/口令的组合无法通过验证, 系统并不会指出是哪个部分不正确。无论是用户名无效还是口令错误, 都会得到相同的消息: Login incorrect。在图 2-14 中, 首先显示以用户名 beginningunix 和一个错误的口令组合登陆, 接着以一个错误的用户名和 beginningunix 的口令组合登陆, 可以看到系统对两次登录尝试的响应是一样的: Login incorrect。这也是一种安全机制, 可以阻止恶意实体试图猜测系统上的用户名; 每个用户都必须有一个有效的用户名/口令组合才能登录。

不要忘记自己的用户名和口令, 因为在登录 Unix 时, 通常对两者都没有任何提示。

```

Mandrake Linux release 10.0 (Official) for i586
Kernel 2.6.3-7mdkmp on an i686 / tty1
linux login: beginningunix
Password:
Login incorrect

login: beginunix
Password:
Login incorrect

login: beginningunix
Password:
Last login: Sun Nov 28 14:21:19 on vc/1
[beginningunix@linux beginningunix]$ _

```

图 2-14

2.2.3 远程登录

Unix 支持网络互联，允许通过命令行或图形用户界面进行远程操作。远程登录时，通常要使用某种网络协议，例如 TCP/IP(在第 16 章中讨论)。

协议是一种用于在两个不同系统之间传送信息的标准。

表 2-1 是用于登录到远程 Unix 系统的一些最常用的方法。

表 2-1

命 令	说 明
ssh	交互式地登录一个 shell 以便执行多个功能，如运行命令。该方法使用加密技术来保护会话，因此用户名、口令以及与远程系统的所有通信都是加密的 [*] (对其他人不可读)
telnet	交互式地登录一个 shell 以便执行多个功能，如运行命令。因为该方法没有使用加密技术 [*] ，所以用户名、口令以及与远程系统的所有通信都是以普通文本的方式发送，很可能被网络上的其他人查看
sftp	登录以便在两个不同系统之间传送文件。该方法使用了加密技术。 [*] (sftp 将在第 8 章中讨论)
ftp	登录以便在两个不同系统之间传送文件。该方法没有使用加密技术。 [*] (ftp 将在第 8 章中讨论)

^{*}加密意味着文本是其他人无法理解的，意义是杂乱的。例如，如果对短语“this is my password”进行加密，那么通过网络查看该会话的任何其他人看到的可能是“14N!6x6*0|~dB/2”。

远程登录的方法远不止这些。例如，r 命令——rsh(远程 shell)、rcp(远程复制)和 rlogin(远程登录)——以前很盛行，但是因为它们几乎不提供安全机制，所以在现在的环境中一般不提倡使用它们。在功能上，rsh 和 rlogin 类似于 telnet，而 rcp 类似于 ftp。

为了能够远程登录，本地系统和远程系统必须相互连接并且允许经过通信路径进行访问(也就是说，没有防火墙或系统限制)。

大多数 Unix 系统都支持用于连接外部系统的协议/命令，但是在远程连接时，这些服务不一定可用(例如，远程系统可能不允许对系统的连接)。在这种情况下，为了确定远程连接的方法，必须联系 Unix 系统的系统管理员。

1. 使用 ssh

ssh(Secure SHell)和 telnet 两种方法都能够登录到远程系统并交互地运行命令；也就是说，即使不能够使用账户可用的所有命令，也可以使用其中的大多数，就像本地连接一样。要使用这些命令，至少需要如下信息：

```
command hostname
```

command 指示想要用于连接的协议(首选 ssh，如果该 shell 可用的话，因为它的会话是加密的)，而 hostname 指示想要连接的远程系统。hostname 可以是一个实际的名称(如 darwin)或者是一个 IP 地址(如 192.168.1.58)。要利用 ssh 命令从一个 Linux 系统连接到系统 darwin(IP 地址 192.168.1.58)，可以输入：

```
ssh darwin
```

或

```
ssh 192.168.1.58
```

IP 地址是计算机为了在系统之间传送信息而使用的一个数字名称。由于一长串数字难于记忆，因此可以用一个普通的名字来指代远程系统，当然，必须正确设置好这个系统。更多内容将在第 16 章中讨论。

在输入这两个命令中的任何一个之后，将会看到与通过命令行进行本地登录时相同类型的提示或信息。

例如，在图 2-15 中，运行命令 hostname，屏幕上就会显示运行该命令的系统的主机名。本例中，主机名是 DARWIN，它是一个 Mac OS X 系统。然后输入 ssh 命令：ssh 192.168.1.65(主机名为 linux 的 Linux 系统的 IP 地址)。

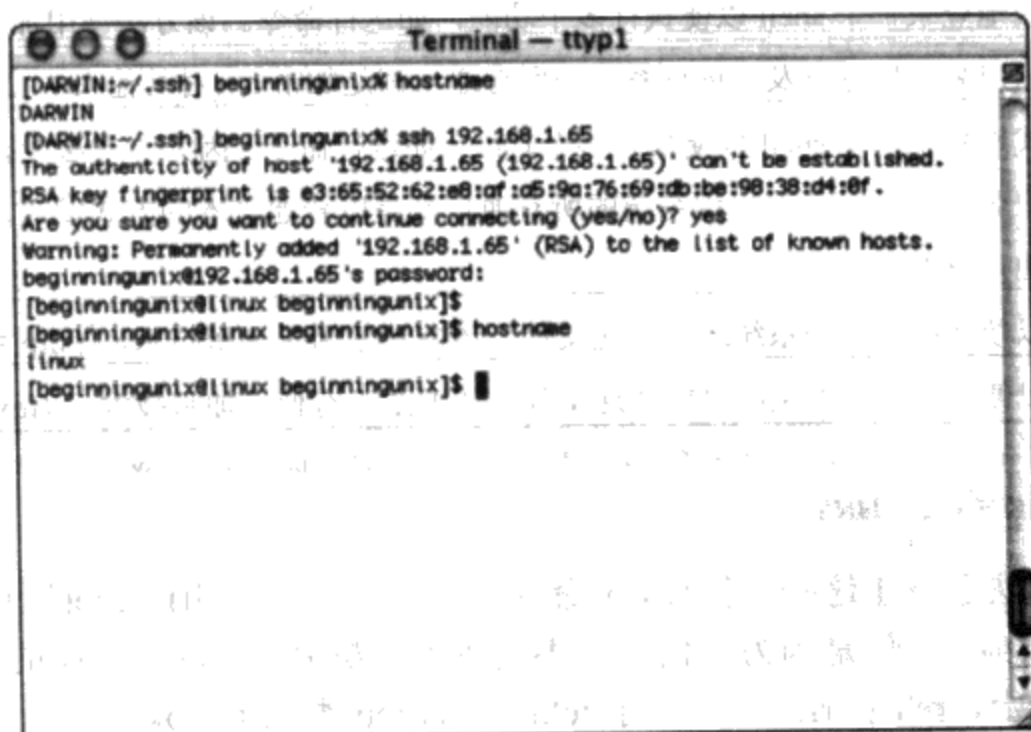


图 2-15

如果是第一次使用 ssh 连接远程系统，屏幕上将会提示用户接受远程系统的键值，或者标识符，如第 11 行所示(此标识符只在第一次连接系统时出现)。可以向远程系统管理员确认该键值。如果有人修改系统或者试图伪装成使用 ssh 的服务器(并试图窃取用户证书)，用户将会收到警告信息。

为用户账户输入口令后，就进入远程系统上的一个 shell。没有要求输入用户名的原因是因

为 ssh 命令发送在本地系统上登录时所使用的用户名, 除非指定不同的用户名。现在执行 hostname, 可以看出, 的确已远程登录到一个名为 linux 的机器上。

运行 ssh hostname 时, ssh 假定用户希望使用和本地登陆时使用的相同的用户名登录远程系统。由于不同的命名习惯, 用户名在不同系统上不一定总是相同, 因此在一个系统上是 jdoe, 在另一个系统上可能是 johnd, 而在第三个系统上可能是 jd1234。如果要用不同的用户名登录到一个远程系统, 可以使用下面的语法:

```
ssh username@hostname
```

如果已经以用户名 johnd 登录到系统 darwin, 而想以用户名 jdoe 远程登录到系统 192.168.1.65, 请输入:

```
ssh jdoe@192.168.1.65
```

请记住 hostname 可以是一个具体的名称(如 boardroom), 也可以是一个 IP 地址(如 192.168.1.58)。

2. telnet

ssh 为本地和远程服务器之间的通信加密, 增强了会话的安全性。但是, 由于系统限制或策略的原因, ssh 不一定总是可用。如果是这种情况, 通常就需要使用 telnet。telnet 提供相同的功能, 只是在本地和远程系统之间传送的数据没有加密。它是一个比较老的协议, 由于在许多主要的平台(包括 Microsoft Windows 和 DOS)上都默认可用, 因此即使在今天这种重视安全性的环境中依然获得广泛的使用。telnet 的使用与 ssh 非常相似, 只是有时候需要用户名和口令(telnet 并不总是假设以当前用户名登录远程服务器)。要从 darwin 机器上使用 telnet 登陆到 solaris 机器(192.168.1.60), 输入:

```
telnet solaris
```

或

```
telnet 192.168.1.60
```

如图 2-16 所示。

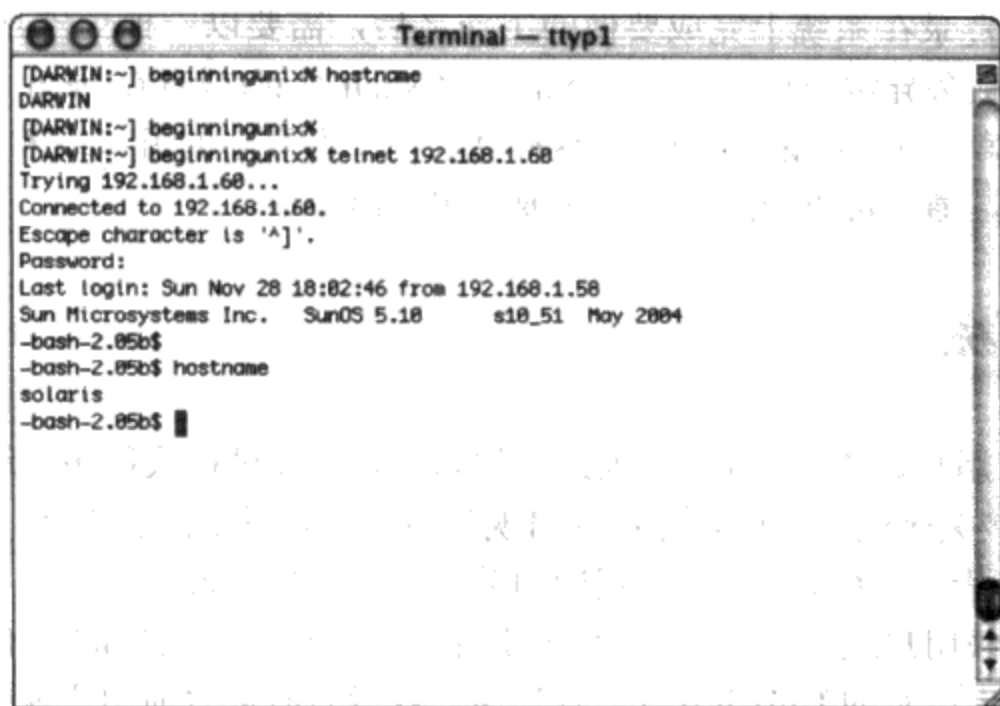


图 2-16

在图 2-16 中, 运行 `hostname` 以确定当前系统的主机名(DARWIN), 然后执行 `telnet 192.168.1.60` 命令。系统询问用户的口令, 输入口令后, 系统显示上一次登录时间和标语信息。再次运行 `hostname` 将获得预期的结果: `solaris`。现在用户可以运行命令了, 就好像他的监视器和键盘直接连接到 `solaris` 系统上一样。

2.2.4 shell

登录后, 进入由系统管理员预先设定的 `shell` (`shell` 已经在第 1 章中介绍过)。无论是显示命令行界面还是图形用户界面(GUI), 用户都可以访问 `shell`。图 2-17 显示的是一个典型的命令行界面。

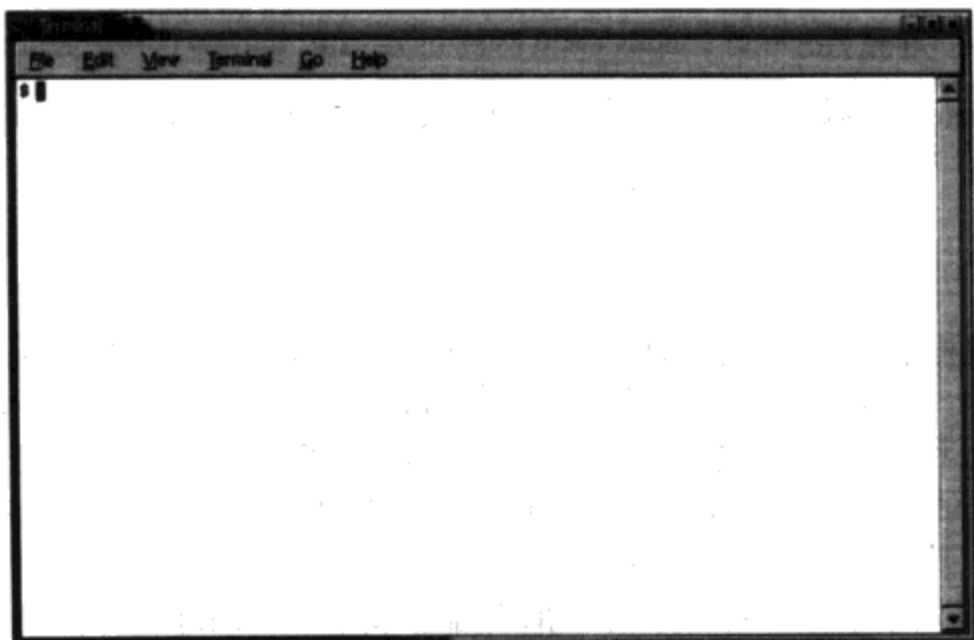


图 2-17

如果正在使用 GUI, 可以找到应用程序 `xterm` 或 `konsole`, 可以通过它们访问 `shell`。然后就可以运行本书中讨论的命令了。

2.2.5 退出

利用交互式的登录在系统上完成要做的工作之后, 需要以一种受控且有序的方式退出系统, 以防止进程或任务异常中止。用命令 `exit` 结束 `shell` 会话(或者在 `bash shell` 或 `c shell` 中使用命令 `logout`, 如第 5 章中讨论的那样)。这样就关闭了所登陆的窗口或者说完全结束了会话。如果正在使用 GUI, 通常可以通过一个按钮或按键退出会话。

2.3 关闭系统

Unix 是一个多用户、多任务系统, 因此通常总是有很多进程或程序在运行。因为需要同步文件系统, 所以直接关闭电源会使文件系统出现问题并影响系统的稳定性。即使没有用户登录, 系统上也总是有进程或任务在运行, 不正确的关机可能会导致很多问题。

通常需要以超级用户或根用户(Unix 系统上拥有最高特权的账户)的身份来关闭系统, 但是在一些独立的或个人所有的 Unix box 上, 管理用户或者有时候普通用户就能够这样做。在有些 GUI 上, 单击一个按钮就可以关闭系统。

通过命令行正确关闭 Unix 系统最一致的方式是使用表 2-2 所列命令中的一个。

表 2-2

命 令	功 能
halt	立即关闭系统
init 0	使用预定义的脚本切断系统的电源,以便在关机前同步数据并整理系统(不是在所有 Unix 系统上都可以使用这种方法)
init 6	通过将系统完全关闭,然后将它完全恢复来重新启动系统(不是在所有 Unix 系统上都可以使用这种方法)
poweroff	通过切断电源来关闭系统
reboot	重新启动系统
shutdown	关闭系统

首选的方法是使用 shutdown 命令,所有 Unix 系统上都可用使用这种方法。它使用系统提供的脚本进行正确的关闭,并且具有其他命令的大部分功能。一般情况下 halt 命令将立即关闭系统,而不执行推荐的关闭步骤,这会引起文件系统同步问题(可能有数据损坏或者更糟)。

关闭和重启系统的另一种方法是使用下面的命令:

```
shutdown -r
```

要关闭计算机,以便通过电源开关安全地切断系统的电源,可以使用下面的命令:

```
shutdown -h
```

要关闭系统而不损坏数据或避免系统不一致,最适当的方法是使用 shutdown 命令。

2.4 使用联机帮助页

Unix 命令总是有多个参数或选项,从而使得相同的命令具有不同类型的功能。因为任何人都不能记住所有的 Unix 命令和这些命令的所有选项,所以自从 Unix 的早期以来一直有联机帮助。Unix 的帮助文件称为联机帮助页(man pages)。联机帮助页(manual page)用一种任何用户都可读的标准格式提供联机文档,并且以一种统一的和合理的方式进行安装。可以简单地按照下面的语法使用该命令:

```
man command
```

如果想了解某条命令的更多信息,可以用这条命令名代替 command。例如,要查看命令 man 的联机帮助页,可以输入:

```
man man
```

此时将会看到与下面内容(来自 Linux 系统)类似的输出:

```
man (1)                                man (1)
NAME
```


man - format and display the on-line manual pages
manpath - determine user's search path for man pages

SYNOPSIS

```
man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
[-M pakhlist] [-P pager] [-S section_list] [section] name ...
```

DESCRIPTION

man formats and displays the on-line manual pages. If you specify *section*, **man** only looks in that section of the manual, *name* is normally the name of the manual page, which is typically the name of a command, function, or file. However, if *name* contains a slash (/) then **man** interprets it as a file specification, so that you can do **man ./foo.5** or even **man /cd/foo/bar.1.gz**.

See below for a description of where **man** looks for the manual page files.

OPTIONS

-C config_file

Specify the configuration file to use; the default is
/etc/man.config. (See **man.config(5)**.)

...

为节省篇幅，此处内容已省略。要查看完整的联机帮助页，请在命令行输入：**man man**。

...

SEE ALSO

apropos(1), **whafcis(1)**, **less(1)**, **groff(1)**, **man.conf(5)**.

BUGS

The **-t** option only works if a troff-like program is installed. If you see blinking \255 or <AD> instead of hyphens, put 'LESS-CHARSET=latin1' in your environment.

TIPS

If you add the line

```
(global-set-key [(f1)] (lambda () (interactive) (manual-entry (current-word) )))
```

to your **.emacs** file, then hitting **F1** will give you the man page for the library call at the current cursor position.

To get a plain text version of a man page, without backspaces and underscores, try

```
# man foo | col -b > foo.mantxt
```

September 2, 1995

man(1)

联机帮助页一般分为不同的部分，通常随着联机帮助页作者的喜好而变化。下面是一些比较常用的部分：

- **NAME**—— 命令的名称。
- **SYNOPSIS**—— 命令的常用参数。

- DESCRIPTION——命令的一般性描述以及它的作用。
- OPTIONS——描述命令所有的参数或选项。
- SEE ALSO——列出联机帮助页中与该命令直接相关或功能相近的其他命令。
- BUGS——解释命令或其输出中存在的任何已知的问题或缺陷。
- EXAMPLES(或 TIPS)——普通的用法示例，让读者知道如何使用这个命令。
- AUTHORS——联机帮助页/命令的作者。

用户并不总是知道要使用的命令，但是只要知道命令的一个要点，就可以利用-k 选项在联机帮助页中搜索，该选项在联机帮助页中寻找关键字。例如，如果需要修改文件的权限，但是忘了要使用什么命令，可以输入：

```
man -k permission
```

此时，将会得到一组命令，它们的关键字中都含有 permission。

实战

使用联机帮助页

(1) 使用联机帮助页，搜索选定的关键字，查看显示了哪些命令。如果一个关键字都想不起来，可以使用：

```
man -k shell
```

(2) 从搜索结果列表中选择一个命令，阅读它的联机帮助页。

工作原理

搜索结果显示了与关键字匹配的所有命令，然后就可以查看它们的联机帮助页以找到所需的命令。

当需要了解 Unix 系统中的命令或文件的相关信息时，联机帮助页是一个至关重要的资源和首要的研究手段。

2.5 小结

本章介绍了使用 Unix 系统的基础知识，从登录系统到关闭系统。另外还讨论了如何使用联机帮助页形式的联机帮助系统，随着掌握的 Unix 命令越来越多，它将使您成为一个自力更生的用户。

第 3 章 用 户 和 组

用户账户可以通过 shell、ftp 账户或者是其他方式提供对 Unix 系统的访问。要使用 Unix 系统提供的资源，需要一个有效的用户账户和资源权限(权限将在第 4 章中讨论)。把账户看作通行证，Unix 系统以此来验证用户的身份。

更多 Mac OS X 系统特有的关于用户和组的信息，请参考第 20 章。

本章讨论账户的基础知识以及在各种 Unix 系统上的账户，考察如何管理账户，研究组的用途以及组的工作原理。本章还介绍了 Unix 中与用户和组相关的其他信息。

3.1 账户基础知识

Unix 系统上主要有三种类型的账户：根用户(或超级用户)账户、系统账户以及普通用户账户。几乎所有账户都属于这些类型中的一种。

3.1.1 根账户

根账户的用户能够完全地、不受约束地控制系统，以至于可以运行命令来完全破坏系统。根用户(也称为根)能够在系统上进行任何操作，可以不受任何限制地访问、删除和修改文件。

Unix 方法学假定根用户知道自己想要做的事情，因此，他们执行的命令将完全破坏系统，Unix 也会服从。如果习惯于使用 Microsoft Windows 进行工作，那么它的管理员账户与 Unix 的根账户很相似，只是 Windows 通常会设法自我保护——如果试图格式化操作系统所在的硬盘，Windows 会阻止该操作，但是 Unix 会接受该命令并开始格式化，而不考虑这样做是否会导致自我毁灭。正是由于这条基本原则，人们通常只将根用于最重要的任务，而且只在必要的时候才使用它——并且非常谨慎。

3.1.2 系统账户

系统账户是对系统特定组件进行操作所需的那类账户。例如，它们包括邮件账户(用于电子邮件功能)和 sshd 账户(用于 ssh 功能)。系统账户通常由操作系统在安装过程中提供或者由软件制造商(包括内部开发商)提供。他们通常协助用户所需的服务或程序。

有多种不同类型的系统账户，其中一些在您的 Unix 系统上可能并不存在。例如，在 /etc/passwd 文件中可能找到的系统账户名有 adm、alias、apache、backup、bin、bind、daemon、ftp、guest、gdm、gopher、halt、identd、irc、kmem、listen、mail、mysql、named、noaccess、nobody、nobody4、ntp、root、rpc、rpcuser 和 sys。系统上的某些特殊功能通常需要用这些账户，对它们所做的任何修改都可能会给系统带来不良的影响。不要轻易修改它们，除非已经研究过它们的功能，并用各种不同的变化对系统做过测试。

3.1.3 用户账户

用户账户为用户和用户组提供对系统的交互式访问。普通用户通常都配以这类账户并且对关键系统文件和目录的访问权限是有限的。通常在一个账户名中希望使用 8 个或更少的字符，但是现在所有的 Unix 系统都不再有这个要求。然而，考虑到与其他 Unix 系统和服务的互操作性，最好是限制账户名在 8 个字符以内。

账户名和用户名的意思相同。

3.1.4 组账户

组账户增加了一种功能，这种功能可以将其他账户集中在一起组成一个逻辑排列，从而简化特权(权限)管理。Unix 权限作用于文件和目录上，并分别控制三类用户的权限：文件的所有者，也称为用户；指派给文件的组，也简称为组；在系统上拥有合法注册但既不是所有者也不属于组的人，也称为其他(others)。组的存在使得资源或文件的所有者能够授予一类用户访问文件的权力。例如，假设一家大约有 100 个员工的公司，该公司使用一台中央 Unix 服务器完成所有工作，包括生产、研究和支持。其中的三个员工组成公司的人力资源(HR)部门；他们通常会处理一些敏感的信息，包括工资、加薪和培训。HR 人员同样要将他们的信息存储在其他人在使用的服务器上，但是该部门的目录 Human_Resources 需要保护起来，从而阻止其他人查看这个目录的内容。为了使 HR 能够在它的文件上设置只允许 HR 全体员工访问的特殊权限，可以将这三个员工放入一个名为 hr 的组。然后就能够设置 Human_Resources 目录的权限，只允许这些成员查看和修改文件，而排除了不属于该组的所有人(其他任何人)。

组的一个强大之处在于，基于访问需求，一个账户可以属于多个组。例如，内部审计组的两个成员可能需要访问每个人的数据，但是他们的目录 Audit 需要保护起来以阻止其他任何账户的访问。要实现这一点，可以让他们属于所有组，同时还有一个专门的审计组，他们是该组仅有的成员。这种情况在本章后面讨论。

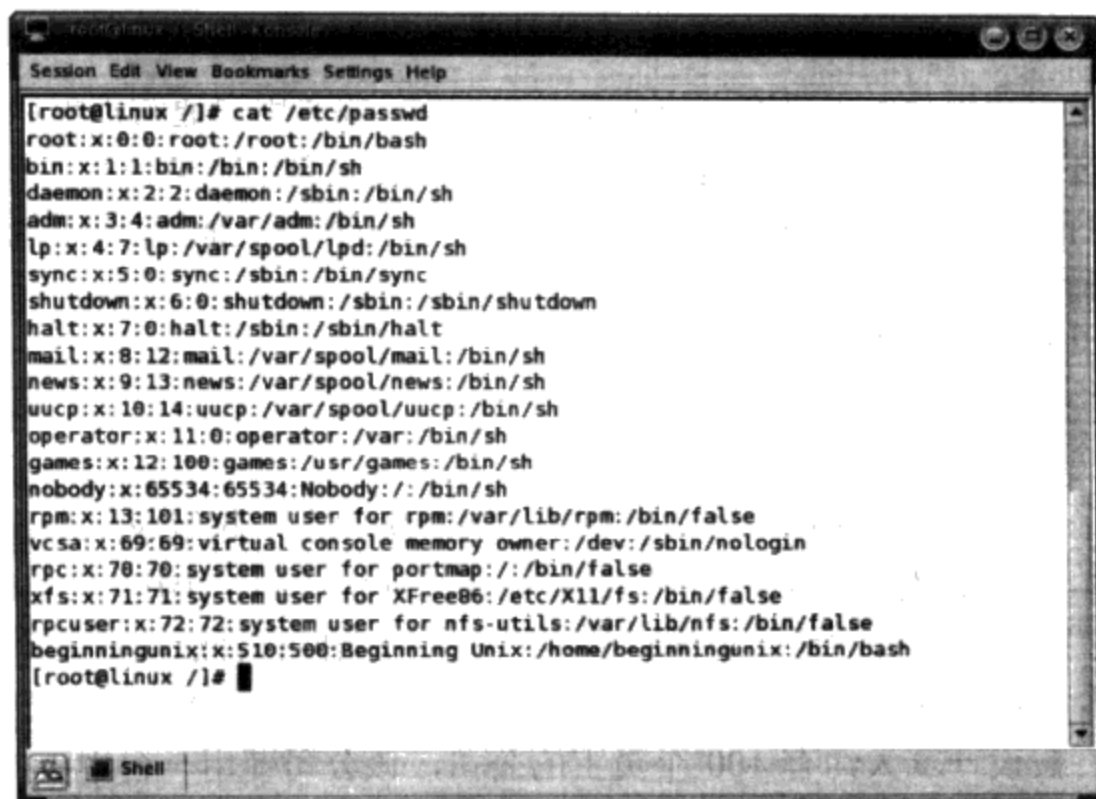
3.2 管理用户和组

用户管理是维持系统正常运行的基础。出于安全的考虑，管理权限应该只分配给需要对账户进行管理的少数用户。系统上有三个主要的用户管理文件：

- /etc/passwd —— 为系统识别已授权的账户。
- /etc/shadow —— 保存相应账户加密后的口令。大多数 Unix 系统都有这个文件。
- /etc/group —— 存放组账户的信息。

3.2.1 /etc/passwd

第一个也是最重要的管理文件是/etc/passwd。该文件保存了 Unix 系统上与账户相关的大部分信息。几乎在系统上拥有账户的任何人都可以查看这个文件，但是只有根用户才能修改它。图 3-1 显示一个来自 Linux 系统的/etc/passwd 示例文件。



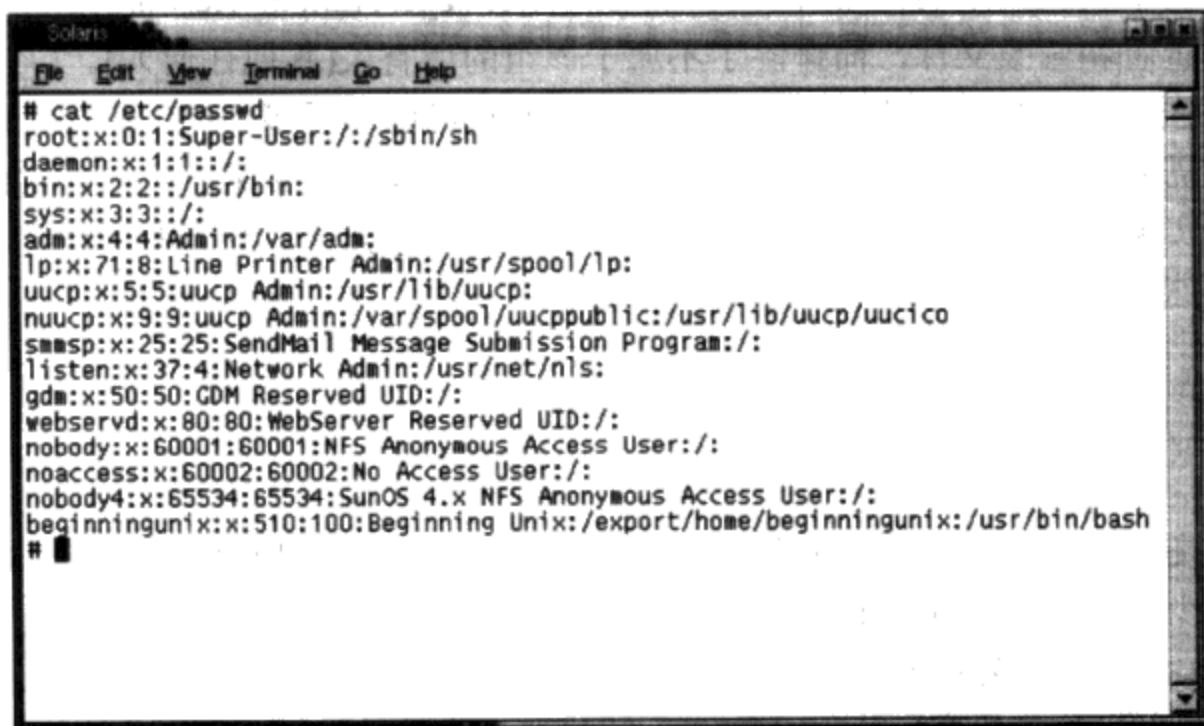
```

[root@linux ~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/sh
daemon:x:2:2:daemon:/sbin:/bin/sh
adm:x:3:4:adm:/var/adm:/bin/sh
lp:x:4:7:lp:/var/spool/lpd:/bin/sh
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/bin/sh
news:x:9:13:news:/var/spool/news:/bin/sh
uucp:x:10:14:uucp:/var/spool/uucp:/bin/sh
operator:x:11:0:operator:/var:/bin/sh
games:x:12:100:games:/usr/games:/bin/sh
nobody:x:65534:65534:Nobody:/:/bin/sh
rpm:x:13:101:system user for rpm:/var/lib/rpm:/bin/false
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
rpc:x:70:70:system user for portmap:/:/bin/false
xfs:x:71:71:system user for XFree86:/etc/X11/fs:/bin/false
rpcuser:x:72:72:system user for nfs-utils:/var/lib/nfs:/bin/false
beginningunix:x:510:500:Beginning Unix:/home/beginningunix:/bin/bash
[root@linux ~]#

```

图 3-1

图 3-2 显示一个来自 Solaris 10 系统的 `/etc/passwd` 文件。它与图 3-1 所示的文件几乎一样，因为在各种 Unix 系统中这个文件的格式都是相同的。



```

# cat /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
sys:x:3:3:/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
nuucp:x:9:9:uucp Admin:/var/spool/uucppublic:/usr/lib/uucp/uucico
smmsp:x:25:25:SendMail Message Submission Program:/:
listen:x:37:4:Network Admin:/usr/net/nls:
gdm:x:50:50:GDM Reserved UID:/:
webserverd:x:80:80:WebServer Reserved UID:/:
nobody:x:60001:60001:NFS Anonymous Access User:/:
noaccess:x:60002:60002:No Access User:/:
nobody4:x:65534:65534:SunOS 4.x NFS Anonymous Access User:/:
beginningunix:x:510:100:Beginning Unix:/export/home/beginningunix:/usr/bin/bash
#

```

图 3-2

观察文件中的任意一行(图 3-3 中的示例使用的是图 3-1 所示文件结尾处的 `beginningunix` 行)，可以看出每一行都由冒号分成了 7 个独立的部分——称为字段。虽然有些字段可以为空，但是文件中的每个条目都必须具备全部的 7 个字段。图 3-3 标出了每个字段的位置。

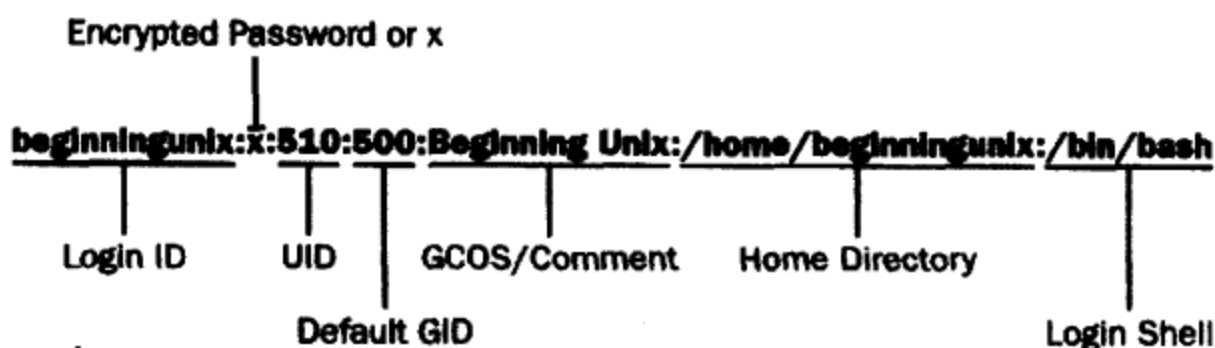


图 3-3

下面根据图 3-3 中的示例，给出文件条目中各字段的说明：

(1) Login ID(用户名)，用户登录系统时输入的账户。用户名应该是惟一的，因此应该避免重名，否则将会导致混淆并产生严重的权限问题。用户名通常是由管理员分配的。因为用户必须使用自己的用户名登录系统，所以选择用户名时，要折衷考虑复杂性和易用性。

`beginningunix: x: 510:500: Beginning Unix: /home /beginningunix: /bin/bash`

(2) 加密的口令或 x。如果使用隐式口令，该字段就只含有一个 x。在 Unix 早期，口令字段中含有已加密的用户口令，但是随着机器越来越强大，破坏和发现口令变得更加容易，因此将口令转移到一个名为 `/etc/shadow` 的单独文件中。只允许特定的账户查看该文件。有些 Unix 版本仍然在 `/etc/passwd` 文件中包含加密的口令，但是一般不赞成这种做法。管理员通常会为用户分配一个初始密码。

`beginningunix: x: 510:500: Beginning Unix: /home /beginningunix: /bin/bash`

(3) UID(用户 ID 号)，系统通过它识别账户。这是 Unix 系统表示用户的方式(而不是使用用户名)。用户一般只通过账户名与系统进行交互，但是 Unix 系统用一个号码(UID)来代表用户。每个账户都分配了一个 UID，范围一般是从 0 到 65535，其中 0~99 保留为系统 ID(根用户，也就是超级用户，总是 0)。并不是所有系统都有 65535 的限制(有些系统允许更大的数字)。UID 并不一定必须是惟一的，但是当两个用户共享一个 UID 时，日志和权限就会变得很混乱，因此允许用户共享 UID 并不是一种好的做法(可以通过组来实现以用户共享 UID 的方式查找的功能)。通常由管理员分配账户名和 UID。

`beginningunix: x: 510:500: Beginning Unix: /home /beginningunix: /bin/bash`

用户在系统中实际是由 UID 标识的。可以把根账户的名字改成 admin，但是因为与该账户相关联的 UID 是 0，所以系统仍然将它识别为超级用户。也可以把 0 UID 分配给另一个用户，那么这个账户将具有超级用户的权限(这种分配会产生安全问题，极不提倡)。

(4) 默认 UID(组 UID)——账户所属的首要的或默认的组。这并不会限制一个账户所能从属的组的总数；它只是识别用户登录后通常从属的组。这个号码不需要是惟一的，因为许多用户可以共享同一个组而不会对系统产生不良影响。号码较小的组通常是系统账户组。

`beginningunix: x: 510:500: Beginning Unix: /home /beginningunix: /bin/bash`

(5) GCOS(或注释)，这个字段保存账户的相关信息，如用户全名、电话或办公室号码以及任何其他可读信息。该字段几乎能够包含任何需要的内容(冒号除外，它表示字段的结束)。大

多数组织都利用它为账户添加一些联系信息,已备不时之需。系统上的任何用户都能够查看该文件(和字段)中的所有内容,因此不要提供敏感的信息,例如信用卡号码或社会保险号码。该字段可以为空而不会产生不良的影响(“空白”字段在中间,一边有一个冒号)。

```
beginningunix: x: 510:500: Beginning Unix: /home /beginningunix: /bin/bash
```

有趣的是,GCOS 字段的名称起源于通用电气公司综合操作系统(General Electric Comprehensive Operating System, GECOS),或通用综合操作系统。该字段最初用于保存某些服务的 GCOS 标识,这些服务在 GECOS 系统以外的其他系统上运行(GECOS 是通用电气公司自己的操作系统)。现在 GCOS 已经不常用,但是提到该字段的时候仍然使用这个术语。

(6) 账户启动的位置、或主目录(用于存储个人文件)。它可以是任何有效的目录(通常是 /home,但并不总是这样),用户拥有该目录的所有权限(读取、写入和执行)。该目录通常为相关联的账户所拥有。不要将 /tmp 分配给任何账户作为主目录,因为这样会产生严重的安全隐患。

```
beginningunix: x: 510:500: Beginning Unix: /home /beginningunix: /bin/bash
```

(7) 用户的登录 shell。它必须是一个有效的 shell(通常列在 /etc/shells 文件中),否则用户将不能交互式地登录。通常在 /etc/shells 文件中标识出所有有效的 shell。如果在第 7 个字段中标识的 shell 不存在(例如一个拼写错误的条目),那么用户将不能够交互式地登录。所以在手动编辑该字段时,必须非常仔细。

```
beginningunix: x: 510:500: Beginning Unix: /home /beginningunix: /bin/bash
```

3.2.2 /etc/shadow

/etc/shadow 文件包含已加密的本地用户的口令记录以及所有口令的期限(说明口令过期的时间)或限制。图 3-4 显示一个来自 Linux 系统的 /etc/shadow 示例文件。

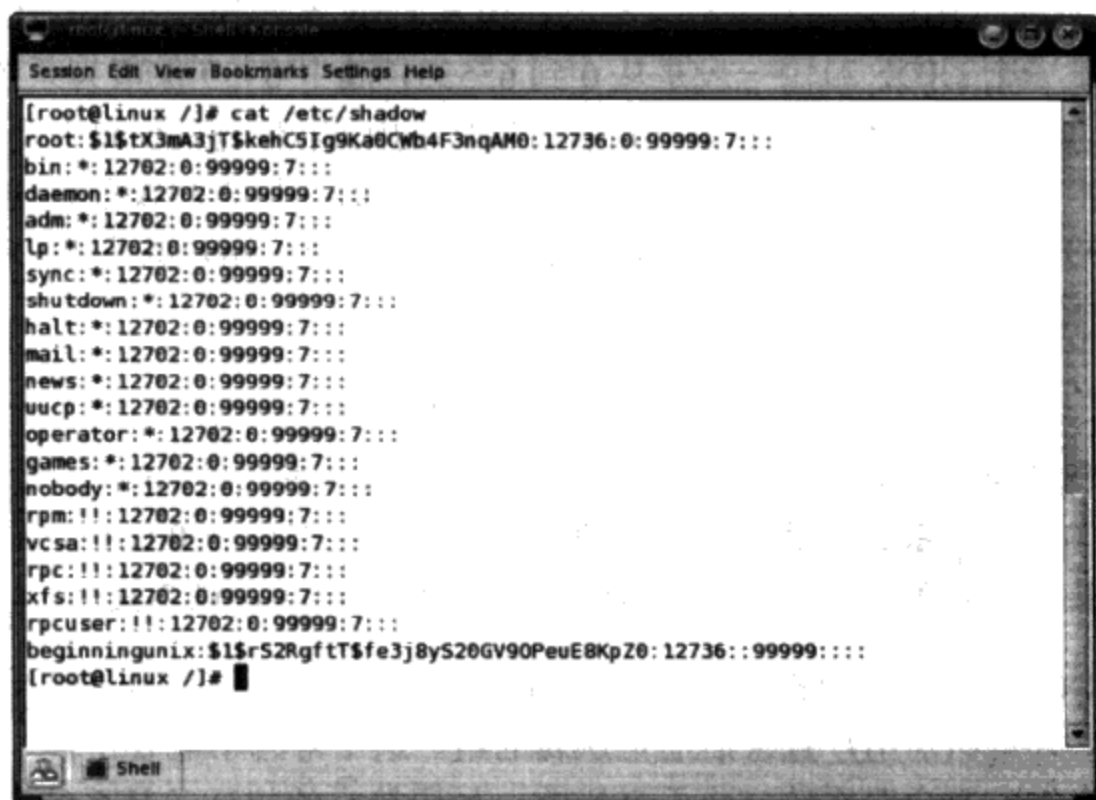


图 3-4

图 3-5 显示一个来自 Solaris 10 Unix 系统的 /etc/shadow 示例。其字段内容与图 3-4 中所示的有细微差别,但是都有 9 个字段。

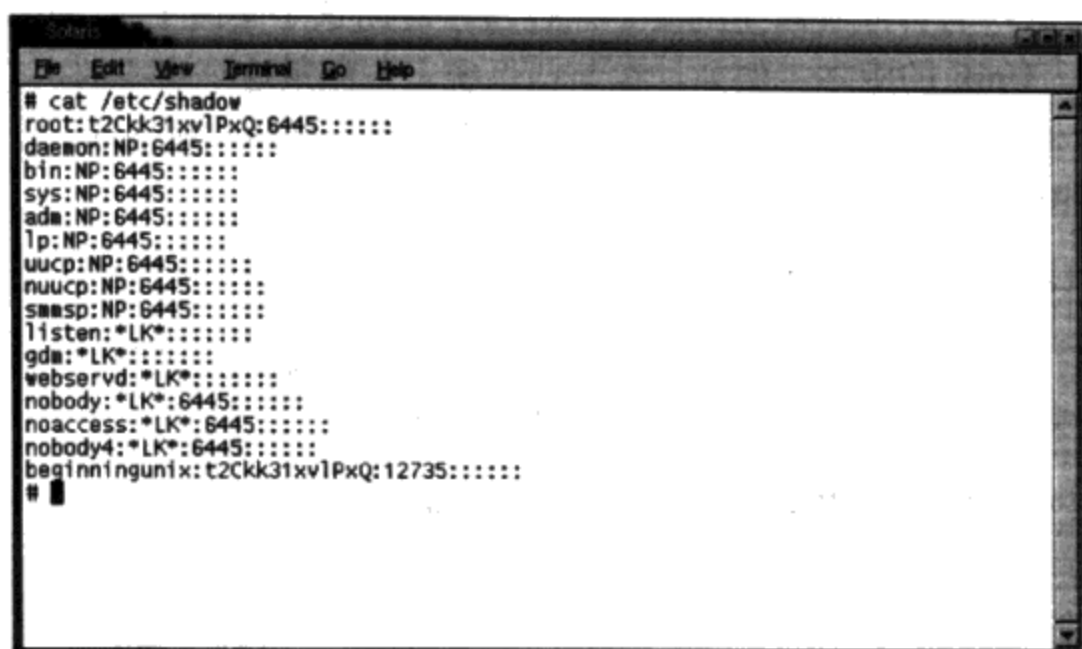


图 3-5

图 3-6 使用图 3-4 中的最后一行指出由冒号分隔的各个字段。



图 3-6

下面根据图 3-6 中的示例，给出文件条目中各字段的说明：

(1) 登录 ID(用户名或账户名)。该信息与账户的/etc/passwd 文件条目的第一个字段对应。

beginningunix:\$1\$cth3s70B\$Sol7rv9u.UyKtEyZ0HP.V.:12736::9999::::

(2) 加密后的口令(它可以有 13 个字符或更多字符，这依赖于 Unix 的实现)。由于只有根用户能够读取这个文件，相对于将口令放在任何用户都能读取的/etc/passwd 文件中，口令获得了更多的保护。如果这个字段是空的，系统就不会要求用户输入口令——这是一种非常危险的

情况，因为只需要账户名会危及系统的安全。还可以利用这个字段锁定一个账户(禁止任何人使用它)，这取决于 Unix 的版本。例如，在某些系统上，如果这个字段的内容是 NP，那么这意味着用户不能登录到该账户，而必须用他自己的账户登录然后利用 `sudo`(本章后面讨论)。还可以用一个 `a*`(在 Solaris 用 `*LK*`)表示已将账户锁定。

```
beginningunix:$1$c3h3s70B$Sol7rv9u.UyKtEyZ0HP.V.:12736::9999:::
```

(3) 自 1970 年 1 月 1 日以来，直到口令被修改时为止的天数。该字段与其他字段一起使用以决定账户和口令是否依然有效，以及口令是否需要更新。

```
beginningunix:$1$c3h3s70B$Sol7rv9u.UyKtEyZ0HP.V.:12736::9999:::
```

人们将 1970 年 1 月 1 号称为新纪元。Unix 的创建者选择这个日期作为系统日期的开始时间。

(4) 用户能够再次修改其口令之前所必须经过的最少天数。该字段使得系统管理员能够阻止用户自从上次修改口令之后过快地再次进行修改，如果黑客破解了某个账户的口令，这种方法可以减少他们修改该口令的机会。该字段也可以用于管理功能，如系统之间的口令传输。

本例中，该字段为空：

```
beginningunix:$1$c3h3s70B$Sol7rv9u.UyKtEyZ0HP.V.:12736::9999:::
```

(5) 口令需要修改之前保持有效的最大天数。管理员使用这个字段来执行口令修改策略，以降低恶意实体使用暴力破解(不断地尝试口令)破译密码的可能性。因为这种破解方法会耗费大量的时间，这与口令的优劣有关。

```
beginningunix:$1$c3h3s70B$Sol7rv9u.UyKtEyZ0HP.V.:12736::9999:::
```

(6) 口令到期前警告用户的天数。给每个用户发警告，通知他口令将要到期，使其有机会在过期之前选一个方便的时间修改口令，这是一种很好的处理方法。如果用户没有在给定时间内修改口令，那么他将无法登录系统，直到系统管理员介入为止。

本例中，字段 6~9 都为空，跟大多数系统中的情况一样。

```
beginningunix:$1$c3h3s70B$Sol7rv9u.UyKtEyZ0HP.V.:12736::9999:::
```

(7) 随各种 Unix 实现的不同而不同，但是通常表示口令不能用之前账户可以连续不活动的天数，或者是从口令期满到账户不能用之间的天数。

(8) 自 1970 年 1 月 1 日以来的天数，直到账户期满时为止。这在创建时间有限的账户时很有用(例如为那些有固定的聘用和合同终止日期的临时雇员创建账户)。

(9) 保留字段以备将来使用。

3.2.3 /etc/group

`/etc/group` 文件包含每个账户的组信息。图 3-7 显示一个来自 Linux 系统的 `/etc/group` 示例文件。

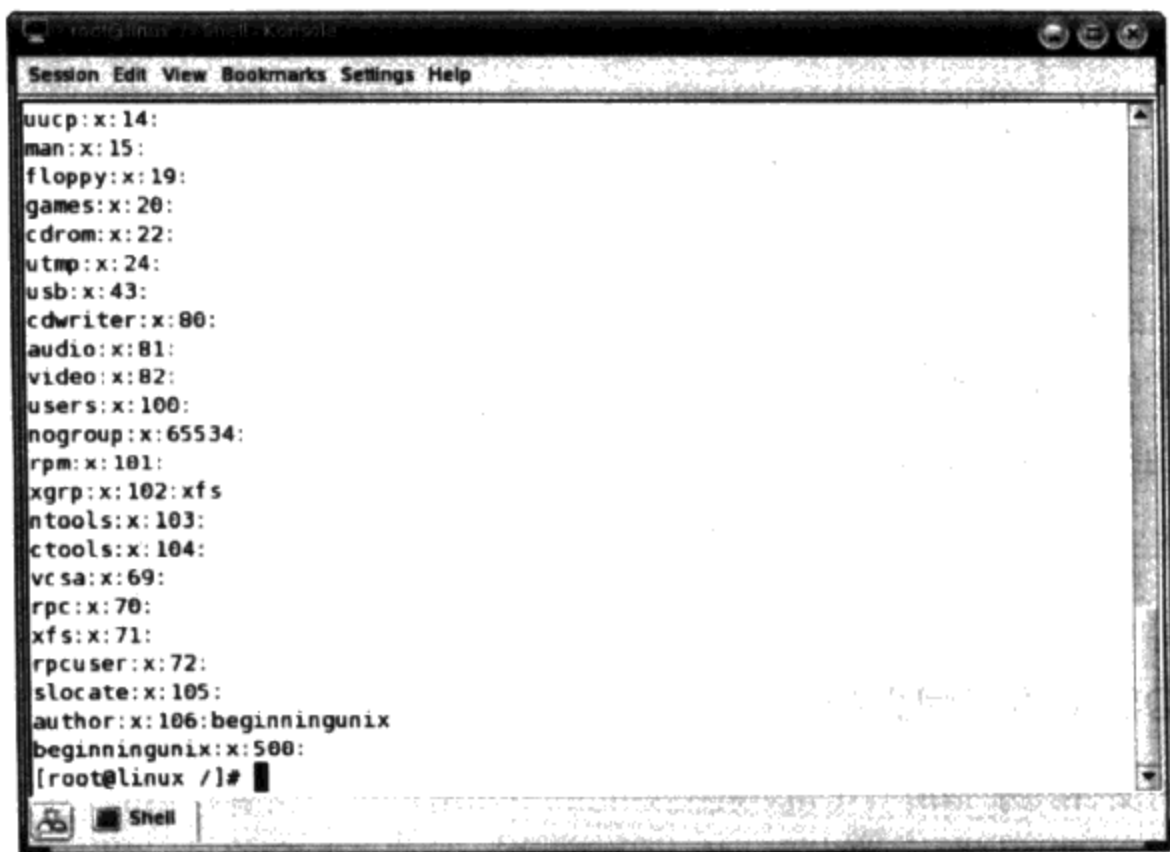


图 3-7

图 3-8 显示 Solaris 10 系统上的相同文件。

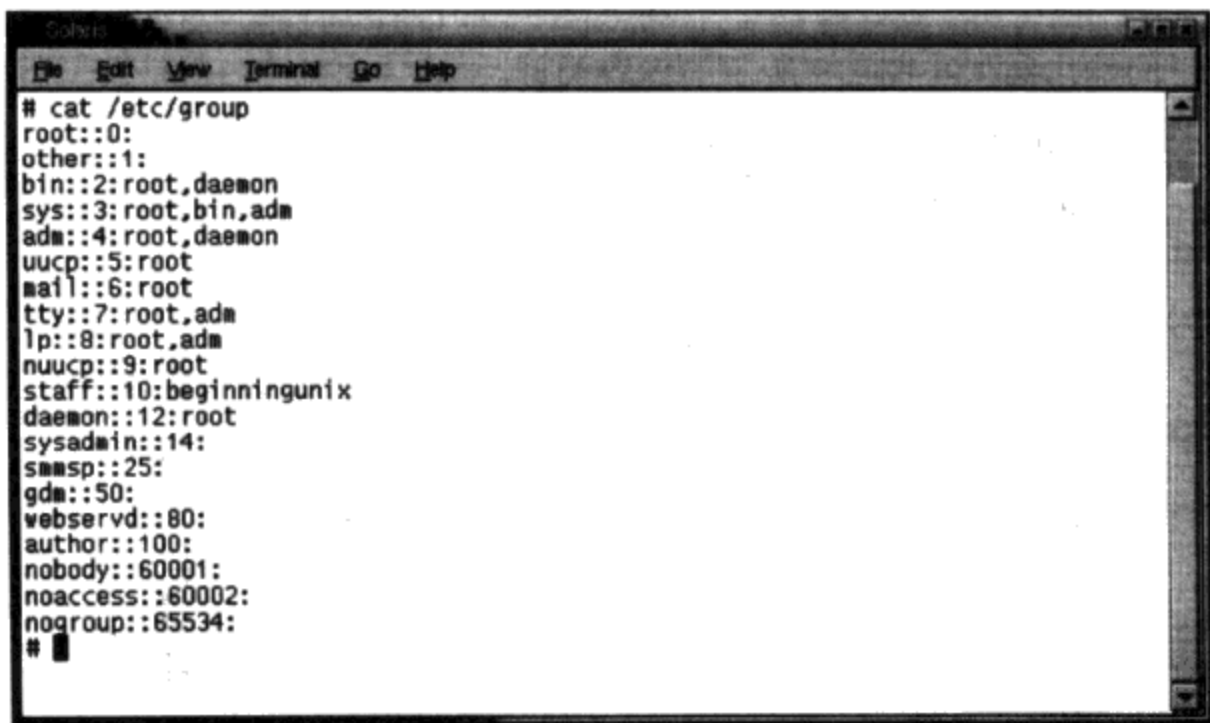


图 3-8

这些文件具有相同的格式。下面以图 3-7 的倒数第二行为例，描述由冒号分隔的 4 个字段：

(1) 组名，用户通过它来识别组。本例中，组名与账户名相同。

beginningunix:x:500:

(2) 使用该组的口令。在大多数系统上，这个字段都为空(没有口令)，但是它可以包含已加密的组口令或者只有一个 x，这表示隐式口令。在这里也存在/etc/passwd 文件中的安全性问题，这也是有些系统使用组阴影文件的原因。组阴影文件通常位于/etc/gshadow；关于这个文件及其格式的更多信息可以参考供应商的文档。

```
beginningunix:x:500:
```

(3) 组 ID(GID)。该号码标识系统中的组。它是 Unix 查看组的方式(类似于/etc/passwd 中的 UID)。

```
beginningunix:x:500:
```

(4) 属于组的账户列表, 账户之间用冒号分隔。图 3-7 中示例行前面的那一行显示了账户 beginningunix 还属于 author 组:

```
author:x:106:beginningunix
```

因此可以根据该信息相应地设置权限(第 4 章中讨论)。这个字段可以为空, 如本例所示。

```
beginningunix:x:500:
```

3.2.4 Mac OS X 的不同之处

前面介绍的文件几乎是所有 Unix 系统上的主要的用户管理文件。Mac OS X 是一个例外。它也含有/etc/passwd, /etc/shadow 和/etc/group 文件, 但是这些文件只能用于单用户模式(在第 20 章中讨论)。信息的主要存储位置是 NetInfo 数据库, 可以用 niutil 命令查看和修改这个数据库。下面是查看一组当前数据库的命令:

```
niutil -list . /
```

该命令产生的输出结果与图 3-9 所示的内容类似。

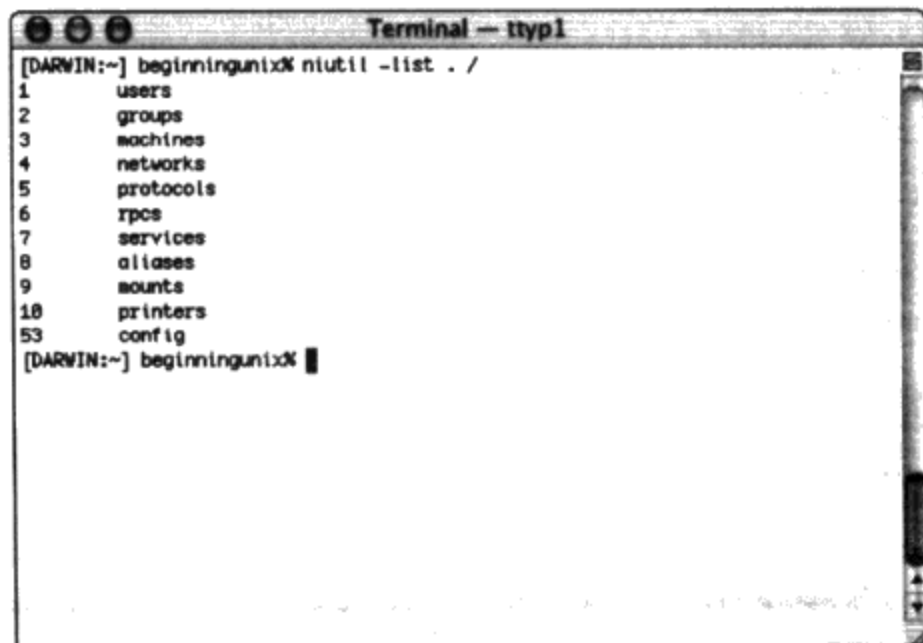


图 3-9

要查看其中某个数据库, 例如当前用户的清单, 可以输入如下命令:

```
niutil -read . /groups
```

该命令的输出显示在图 3-10 中。

关于 Mac OS X 上的 NetInfo 数据库的更多信息, 请参考第 20 章。

Mac OS X 的图形用户界面(GUI)使得管理用户账户变得很容易, 因为它从创建之初就考虑

了易用性。这相对于其他一些 Unix 系统是一个优势。

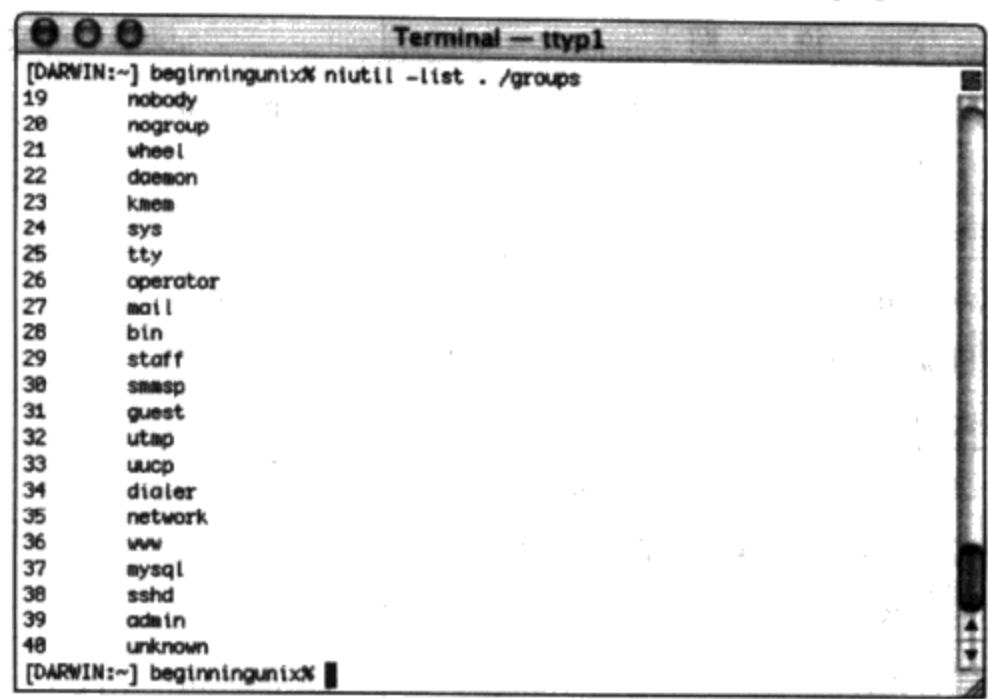


图 3-10

3.3 管理账户和组

有些 Unix 系统使用不同的命令或命令结构(命令的选项),但是在命令行创建、修改和删除账户和组的命令是相当标准的。表 3-1 列出在大多数 Unix 系统上可用的命令及其说明。

表 3-1

命 令	说 明
useradd	向系统中添加账户
usermod	修改账户属性
userdel	从系统中删除账户
groupadd	向系统中添加组
groupmod	修改组的属性
groupdel	从系统中删除组

要手动添加或删除一个账户(不使用上述命令),必须:

- 修改/etc/passwd 以添加或删除账户行。
- 修改/etc/shadow 以添加或删除账户行。
- 修改/etc/group 以添加或删除账户引用。
- 添加或删除账户的主目录(如果没有共享,这个目录默认情况下不是共享的)。

使用表 3-1 中的命令以避免这些步骤。这样还可以降低引入输入错误的危险,如果这些文件中有输入错误,会使系统不可用。要运行这些命令,必须以根用户(超级用户)的身份登录。

3.3.1 账户管理

可以使用 useradd 命令在命令行添加一个用户。表 3-2 描述了 useradd 命令的一些选项。

表 3-2

选 项	说 明	受影响的文件和字段
-c	GCOS 的注释或注释字段(如果注释中有空格要使用引号)	/etc/passwd; 字段 5
-d	账户的主目录	/etc/passwd; 字段 6
-e	以 yyyy-mm-dd 或 mm/dd/yy 格式表示的账户的终止日期, 其格式与 Unix 版本有关(这个日期后账户将失效)	/etc/shadow; 字段 8
-f	在账户不能用之前, 账户可以不活动的天数; 或者是从口令到期, 一直到账户不能使用之间的天数	/etc/shadow; 字段 7
-g	初始组(默认组)	/etc/passwd; 字段 4
-G	一系列用冒号隔开的、辅助的或次要的组, 用户属于这些组	/etc/group; 在命令行中所识别的组的字段 4
-m	如果主目录不存在, 创建主目录	没有用到
-s	用户交互式登录的 shell	/etc/passwd; 字段 7
-u	分配用户 ID(是惟一的, 除非使用-o 选项, 它允许相同的 UID)。UID 0~99 通常保留给系统账户	/etc/passwd; 字段 3

该命令的结构是:

```
useradd -c comment -d home directory -e expiration date -f inactive days -g
primary
(default) group -G secondary groups -m -s shell -u user id accountname
```

最后一项是账户名。它是必选的。它是/etc/passwd 文件中的字段 1。

这里有一个示例, 它为用户创建一个名为 **unixnewbie** 的账户, 用户的真实姓名为 **Jane Doe**。**Jane** 需要使用这个账户, 直到 2006 年 7 月 6 日为止。她的首要组是 **users**, 次要组是 **authors**。她已经请求将 **Bourne shell** 作为默认的 shell。她不能确定自己是否会使用这个系统, 因此如果她在 60 天内没有使用这个账户, 就使该账户不能用。创建这个账户的 **useradd** 命令是:

```
useradd -c "Jane Doe" -d /home/unixnewbie -e 040406 -f 60 -g users -G authors -m -s
/bin/ksh -u 1000 unixnewbie
```

运行这个命令之后, 必须使用 **passwd accountname** 命令为该账户设置一个口令。为了给 **Jane Doe** 的账户创建口令, 根用户将输入:

```
passwd unixnewbie
```

新账户所有者应该立刻修改该口令。

一个新的临时员工 Sarah Torvalds 今天加入公司(5/01/05)。Sarah 的经理要求为 Sarah 创建一个账户。Sarah 加入公司来协助年终的人力资源工作，因此她需要访问默认的用户组以及 hr 组。她与公司的合约自开始之日(创建账户的同一天)起 120 天后结束。创建的标准用户有一个为期 10 天的不活动账户(inactive account)，默认情况下属于 employee 组，并使用 c shell。使用名和姓来创建用户名(账户名不超过 8 个字符，以便与其他 Unix 系统兼容)。因为不能访问任何图形工具，所以需要使用 useradd 命令为 Sarah 创建一个账户。首先以根用户身份登录，然后运行下面的命令：

```
# useradd -c "Sarah Torvalds" -d /home/storvald -e 05/01/05 -f 30 -g employees -G
hr -m -s /bin/csh -u 1005 storvald
# passwd storvald
Changing password for user storvald.
New UNIX password:
Retype UNIX password:
passwd: all authentication tokens updated successfully.
#
```

工作原理

useradd 命令修改/etc/passwd、/etc/shadow 和/etc/group 文件并创建一个主目录。相对于手动编辑这三个文件并创建主目录，想想看这样做方便了多少！由于文件格式是标准的并且使用起来非常容易，因此 useradd 工作得很快。还可以用这个命令创建脚本，从而使添加账户的过程更为简单。

可以使用-D 选项为 useradd 的一些选项分配默认值，从而简化命令的使用。可以参考 useradd 的联机帮助页以获取更多信息。

可以使用 usermod 命令从命令行对已有账户进行修改(而不是修改系统文件)。它使用的参数与 useradd 命令相同，不过增加了参数 -l，这个参数允许修改账户名。例如，要把 Sarah Torvalds 的账户名改成 saraht 并提供相应的主目录，可以执行如下命令：

```
usermod -d /home/saraht -m -l saraht storvald
```

该命令修改 Sarah Torvalds 的当前账户(storvald)，并创建新的主目录/home/saraht(-d/home/saraht -m)和新的账户名 saraht(-l saraht)。选项-m 创建以前不存在的主目录。

使用 userdel 命令非常容易，因此必须谨慎使用，否则可能会带来危险。该命令只有一个可用的参数或选项：-r，用于删除账户的主目录和邮件池(邮件文件)。下面是删除 saraht 账户的方法：

```
userdel -r sarath
```

如果想保留她的主目录以便进行备份，应省略-r 选项。可以在需要的时候再删除主目录。

useradd, usermod 和 userdel 命令在大部分 Unix 系统(Solaris, Linux, BSD 等)中的操作都类似，不过 Mac OS X 系统除外。如果希望在 Mac OS X 系统中利用命令行来修改账户，则需要使用带有-create、-createprop 和-appendprop 参数的 niutil。niutil 是 Mac OS X 系统特有的一

个命令。如果正在使用 Mac OS X 系统并且需要在命令行添加用户，则可以参考联机帮助页以获得更多信息。

3.3.2 组管理

大多数 Unix 系统都是利用 `groupadd`、`groupmod` 和 `groupdel` 命令完成组管理的。`groupadd` 的语法为：

```
groupadd -g group_id group_name
```

例如，要给金融部门创建一个名为 `finance_2` 的新组并分配一个惟一的 GID 535，可以利用：

```
groupadd -g 535 finance_2
```

这个命令在 `/etc/group` 文件中产生恰当的条目。

要修改一个组，可以使用 `groupmod` 命令，其语法为：

```
groupmod -n new_modified_group_name old_group_name
```

要将 `finance_2` 的组名改成 `financial`，输入：

```
groupmod -n financial finance_2
```

`groupmod` 命令也可以使用 `-g` 选项来修改组的 GID。下面是将 `financial` 组的 GID 改为 545 的方法：

```
groupmod -g 545 financial
```

要删除一个已存在的组，只需要 `groupdel` 命令和组名。删除 `financial` 组的命令是：

```
groupdel financial
```

这样做只删除了组，并没有删除与这个组相关联的任何文件(这些文件的所有者依然可以访问它们)。

3.3.3 使用图形用户界面工具进行用户管理

在各种 Unix 系统上有许多可用的图形用户界面(GUI)工具。虽然由于篇幅所限，本书并没有深入介绍它们，但是应该知道它们的存在。对新管理员来说，使用 GUI 工具可以使管理变得更简单。但是在最初使用 GUI 工具之前，有必要很好地理解命令行界面工具。下面我们一起来探讨几个 GUI 工具，关于这些可用工具的更多信息可以参考自己拥有的相应文档。

Mac OS X

Mac OS X 的用户管理工具非常简单。要使用这些工具，单击屏幕右上角的 Apple 图标并选择 `System Preference`(系统属性)。然后在标注了 `System` 的部分中选择 `Account`(左下)。图 3-11 显示了所出现的屏幕。

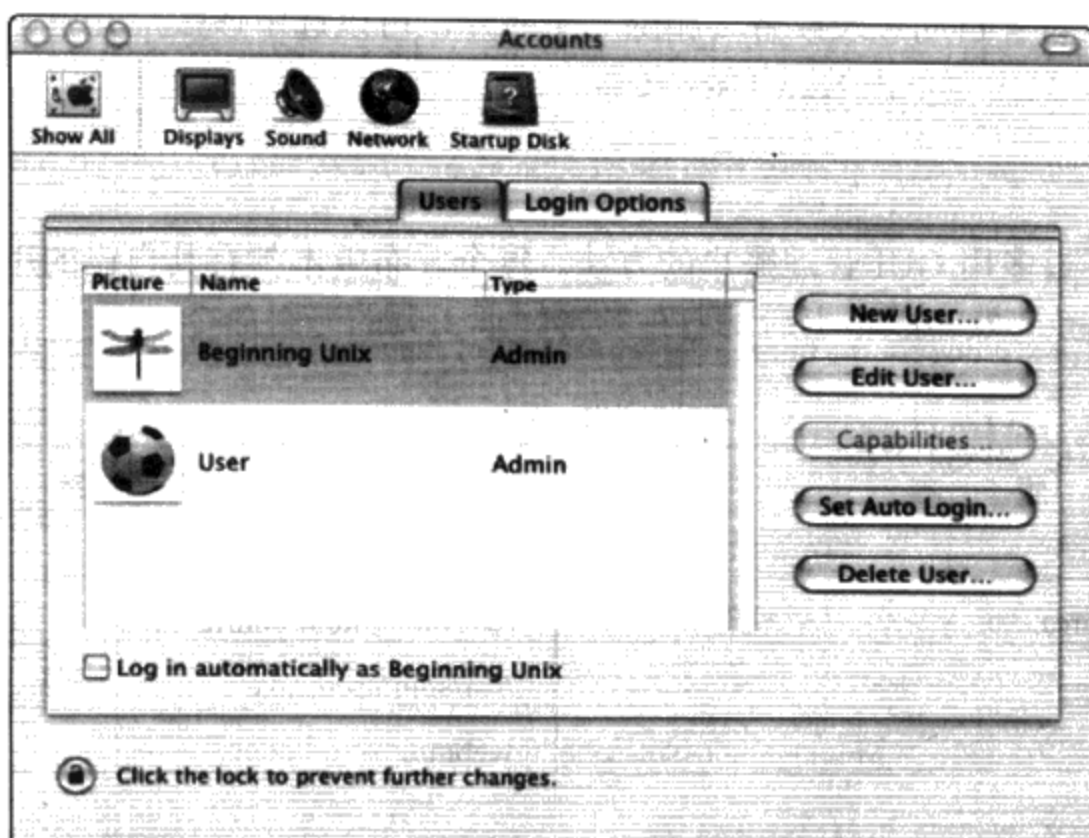


图 3-11

在图 3-11 中可以看见两个账户——User 和 Beginning Unix(两个都是 admin 类型的账户,意味着它们可以在系统上运行系统管理员命令)。可以在这个窗口上添加、编辑或修改,以及删除账户。也可以设置账户在启动后自动登录。

要编辑一个已有的账户,只需要选中该账户并单击 Edit User 按钮即可。图 3-12 显示的是一个账户界面窗口。

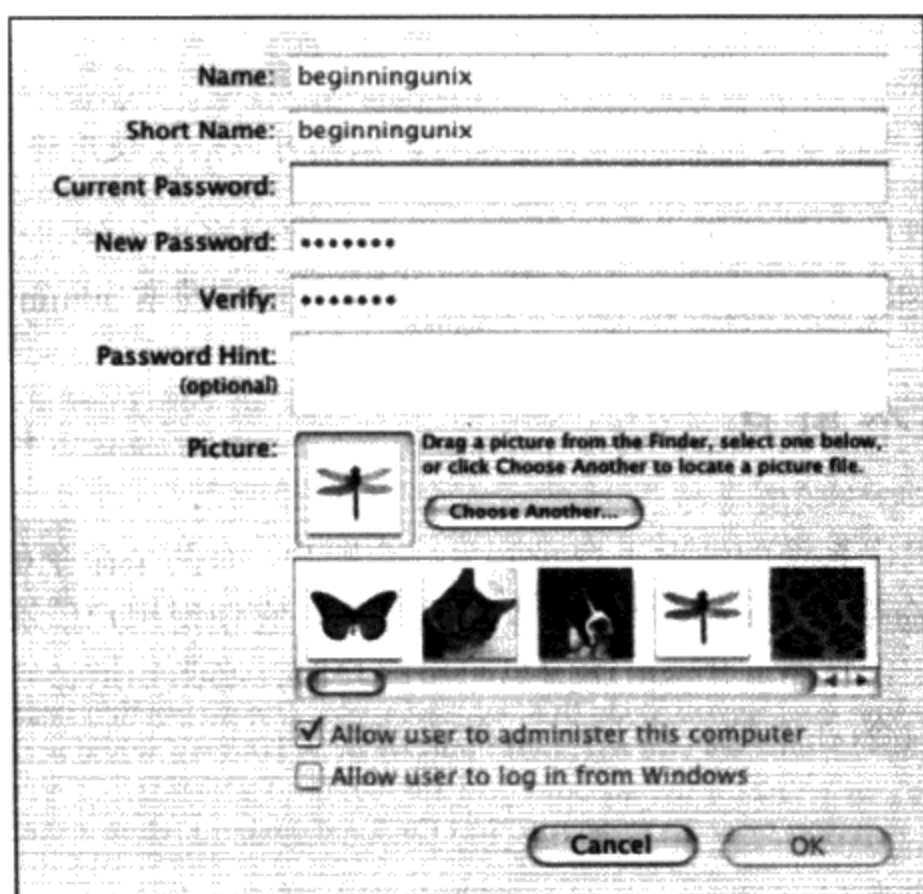


图 3-12

要为一个账户设置自动登录或者要删除一个账户，可以选中该账户并单击相应的按钮。要创建一个新用户，只需要单击 New User 按钮。

Linux

Linux 提供了很多用 GUI 管理账户的方法。每个发布版本都有其自己的用户管理方法。表 3-3 是在主要的 Linux 发布版本上启动各种图形管理工具的一组命令。

表 3-3

版 本	命 令
SUSE	/sbin/yast2
Red Hat(Fedora Core)	/usr/bin/system-config-users
Mandrakelinux	/usr/sbin/userdrake
All	webmin

webmin 命令能够对用户和其他类型的管理任务进行远程管理，它通常提供图形界面。可以从 <http://webmin.com> 获得这个工具，它可以工作在大多数 Unix 系统，包括 Solaris 的系统。在 Unix 的大多数版本上，webmin 并不是默认安装的。

Linux 工具的功能各异，但是一般都能够提供与命令行等价的所有功能。

Solaris

Solaris 提供了一个叫做 admintool 的工具，可以利用它对账户和组进行细化管理。要使用 admintool，可以在命令行输入如下命令：

```
admintool &
```

该命令后面的&符号将命令进程放到后台执行，从而可以继续使用终端窗口进行其他操作。

它有很多特性，既能够管理用户也能够管理设备。要更多地了解这个工具所具有的能力，可以访问 Sun Microsystems 公司的 Web 站点(www.sun.com)并搜索 admintool。

3.4 变成另一个用户

有时候需要在不退出系统的情况下登录到另一个账户。有两个命令可以做到这一点：su 和 sudo。su(switch user)命令在 Unix 的所有版本上都可用。它使得用户能够在访问另一个账户的同时保持现有的登录。在试图利用 su 访问另一个账户时，必须知道该账户的口令，除非是根用户，那样就不需要口令(在本地系统上)。su 的语法如下：

```
su accountname
```

例如，如果已经作为 jdoe 登录，并且想作为 jsmith 登录，则输入：

```
su jsmith
```

使用 `su` 时, 继续使用自己的环境变量和配置文件(在第 5 章中讨论)。如果想使用新账户的用户环境, 可以在 `su` 和账户名之间加一个破折号(-):

```
su - jsmith
```

系统要求输入所切换账户的口令, 但如果是根用户, 则会立即登录到账户而无需输入口令。如果只输入 `su` 命令而没有输入账户名(有没有-都可以), 表示用户试图登录到根账户, 系统会要求输入根的口令。许多人认为 `su` 代表超级用户, 因为单独运行 `su` 命令会切换到根账户或者超级用户账户。当不再需要使用 `su` 切换的账户时, 输入 `exit`, 就会返回到原来的账户(和环境, 如果可用的话)。

`sudo(supereuser do)`命令使得超级用户或根管理员能够执行可由其他用户运行的命令。它并不是在所有 Unix 系统上都可用, 但是可以从 <http://courtesan.com/sudo/> 下载。下面是该命令的语法:

```
sudo command to run
```

要列出可以使用 `sudo` 运行的所有命令, 输入:

```
sudo -l
```

在运行任何命令之前, 系统通常会要求用户输入口令以便验证用户证书。

在 Mac OS X 系统上, 使用 `su` 命令作为根用户登录并不容易, 但是可以使用 `sudo` 命令获得同样的功能, 输入命令:

```
sudo /bin/sh
```

该命令使用户进入根 shell, 这与不带参数单独运行 `su` 等价(对其他 Unix 版本也有效)。

3.5 与用户和组相关的命令

许多命令可以提供重要的用户和组信息以帮助管理账户和系统。例如, 命令 `who` 标识出当前登录在系统上的用户。要使用该命令, 只需要在命令行输入 `who`。其输出与图 3-13 类似。

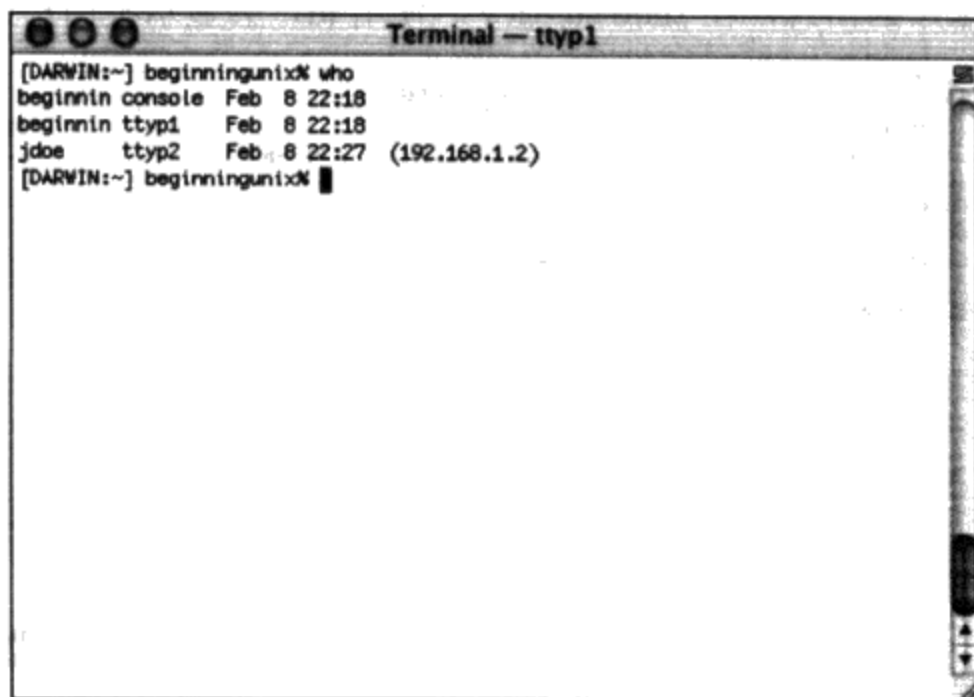


图 3-13

输出有 4 列: 登录名、终端、登录时间、远程主机名或 X 显示。图 3-13 中, 有 3 个用户登录: 两个 `beginnin`(其中一个实际上是 `beginningunix`, 由于输出空间所限, 名字被截断)和一

个 jdoe。console 是终端(屏幕显示), tty1 和 tty2 也是终端(识别用户所在终端的设备)。下一个字段记录每个用户登录的日期和时间, 可以看出 jdoe 来自一个远程连接(192.168.1.2)。

有时候会登录不同的机器或者切换用户太多次以至于不能确定自己当前是什么用户。要解决这个问题, 可以使用 whoami 或 who am i 命令。这两个命令看起来几乎一样, 但是空格号使得它们产生很大的差别。whoami 显示当前作为什么用户登录, 而 who am i 显示最初作为什么用户登录到系统。图 3-14 显示了两个命令的示例。

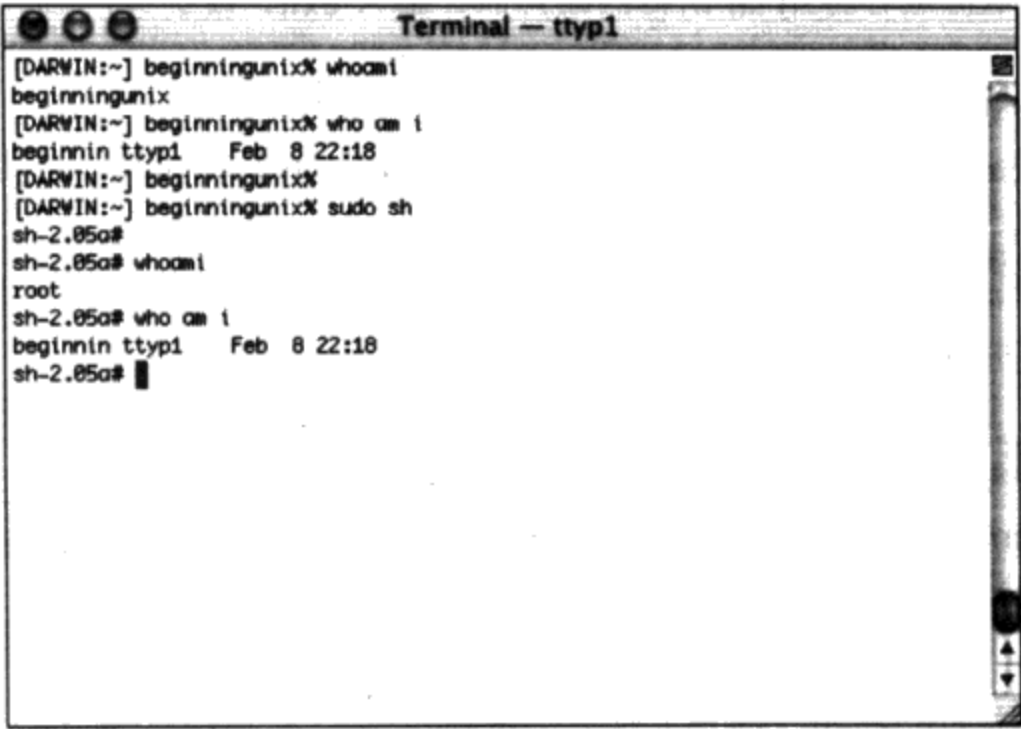


图 3-14

第一次运行 whoami 命令时, 显示用户为 beginningunix, 也就是登录到这个系统的用户。随后的 who am i 命令也显示用户是 beginningunix, 以及其他使用 who 命令获得的信息。sudo sh 命令使 beginningunix 进入根 shell, 接下来的 whoami 显示用户为根用户, 如图 3-14 所示。

命令 id 显示登录的用户以及用户的组的相关信息。图 3-15 中是使用 id 的示例, 用 id 命令显示 beginningunix 账户的信息, 在 sudo sh 命令后使用 id, 会显示根用户的信息。



图 3-15

uid=之后和 gid=之前的内容是用户 ID 信息, 而 gid=之后的内容表示账户所属的主要和次要(附加)的组。

命令 `groups` 能够显示用户或账户所属的组。单独运行时, 它显示当前登录用户的组。将账户名作为参数, `groups` 会显示该用户的组。图 3-16 显示了这两种情况的示例。

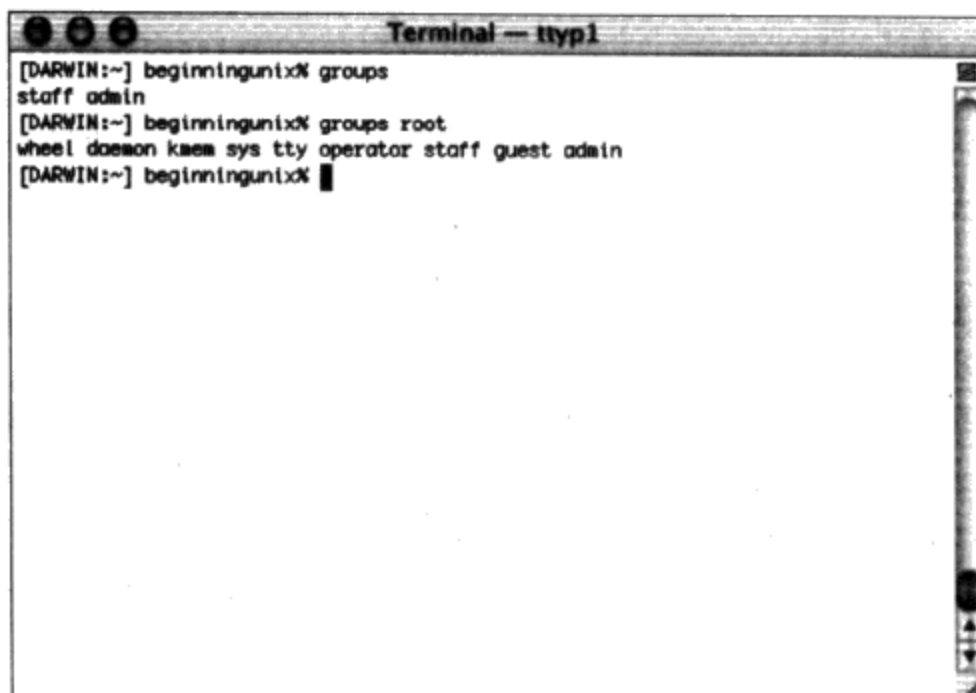


图 3-16

所有这些命令将帮助您管理账户和系统。

3.6 小结

本章学习了账户和组的概念以及如何管理它们。探讨了切换用户以及其他重要的管理命令, 这些内容有助于构建 Unix 知识基础, 以便进入后续章节的学习。既然已经理解了账户和组, 那么就可以利用掌握的新知识来解答下面的练习了。

3.7 练习

- (1) 与用户和组的管理有关的三个主要文件是什么? 各自的功能又是什么?
- (2) Jane Doe 是一名新合同工, 她将于 2005 年 5 月 1 日加入公司, 在一个信息技术 (Information Technology) 项目中工作, 同时还协助财政年度结束时的一些人力资源工作。Jane 的经理已经要求为 Jane 创建一个账户, 该账户需要访问 `employees`、`info_tech_1`、`info_tech_2` 和 `info_tech_5` 这几个组。她与公司的合同自开始之日起 31 天后结束。利用 `useradd` 命令创建该账户。下面是需要用到的一些其他信息: 创建的标准用户有一个为期 10 天的不活动账户, 并使用 Korn shell。利用名和姓的组合来创建用户名(账户名)(账户名最多为 8 个字符)。在该公司中, 所有账户都在 `/etc/passwd` 文件中有对应的员工姓名, 全职员工姓名前标注 E, 合同工姓名前标注 C, 标注和员工姓名之间有一个空格。系统把 `/export/home` 目录作为主目录。分配给 Jane 的账户的 `userid` 为 1000, 因为所创建的最近的账户 UID 是 999。假定已经作为根用户登录系统。

第 4 章 文 件 系 统

文件系统是 Unix 的一个组件，它能够让用户查看、组织以及保护存储设备上的文件和目录并与其进行交互。Unix 中有几种不同类型的文件系统：面向磁盘的、面向网络的、专用的或虚拟的。

- 面向磁盘的(或本地的)文件系统——位于硬盘、CD-ROM、DVD ROM、USB 驱动或其他设备上的实际可访问的文件系统。示例有 UFS(Unix 文件系统, Unix File System)、FAT(文件分配表(File Allocation Table), 通常是 Windows 和 DOS 系统)、NTFS(新技术文件系统(New Technology File System), 通常是 Windows NT、2000 和 XP 系统)、UDF(通用磁盘格式(Universal Disk Format), 通常是 DVD)、HFS+(分级文件系统(Hierarchical File System), 如 Mac OS X)、ISO9660(通常是 CD-ROM)和 EXT2(扩展文件系统 2, Extended Filesystem 2)。
- 面向网络的(或基于网络的)文件系统——可以远程访问的文件系统。这些文件系统通常在服务器端是面向磁盘的, 客户机通过网络远程访问数据。示例有网络文件系统(NFS, Network File System)、Samba(SMB/CIFS)、AFP(Apple 文件归档协议, Apple Filing Protocol)和 WebDAV。
- 专用的或虚拟的文件系统——通常是没有实际驻留在磁盘上的文件系统, 如 TMPFS(临时文件系统)、PROCFS(进程文件系统, Process File System)和 LOOPBACKFS(回送文件系统, Loopback File System)。

本章将深入讨论面向磁盘的文件系统, 并简要介绍面向网络的和专用的文件系统。Mac OS X 系统的用户应该谨记, 虽然 Mac OS X 与传统 Unix 系统的文件系统设计截然不同, 但是本章提到的所有实用程序依然适用于 Mac OS X 系统。另外, Unix 是区分大小写的操作系统, 而 Mac OS X 系统是不区分大小写的/保留大小写的操作系统。这一差别的重要性将在本章后面进行讨论。

4.1 文件系统基础

文件系统是多个文件的逻辑集合, 它位于分区或磁盘上。分区是信息的容器, 如果需要的话, 它可以包含整个硬盘。例如, 可以整个吃掉一个苹果派, 也可以切成片, 这类似于对硬盘或其他物理存储设备的操作方式。一片苹果派类似于磁盘的一个分区, 而完整的苹果派表示将整个磁盘作为一个分区。当然还有更高级的含义, 但是就本章而言, 只是把整个硬盘或硬盘的系统划分看作分区。

一个分区通常只包含一个文件系统, 例如一个分区包含根(/)文件系统而另外一个分区包含 /home 文件系统。每个分区上包含一个文件系统, 这种组织方式允许对不同的文件系统进行逻辑维护和管理。这些分区对用户是不可见的。在 Unix 系统上, 用户可以轻易地在任意数量的文件系统之间进行切换, 甚至不知道自己已经从一个文件系统转移到了另外一个文件系统。

在 Unix 中, 任何软硬件都被视为文件, 包括物理驱动器, 如 DVD-ROM、USB 驱动、软驱等。通过使用文件系统, Unix 能够在处理资源时保持一致, 并且用户在与系统进行交互时也

能够采用一致的方式。从而就很容易理解为什么文件系统是 Unix 操作系统不可或缺的一部分。

Unix 使用分层结构来组织文件，提供了一种自顶向下的方法来查找信息，该方法以一种有组织的方式逐层向下搜索以定位所需的内容。这与档案柜的工作原理类似。档案柜本身可以保存所有信息——换句话说，它是文件系统的基础。例如，要找到某个员工的雇佣信息，需要定位正确的档案柜、柜中正确的抽屉、抽屉中正确的文件夹、和文件夹中信息的正确页码。

在 Unix 中，每个文件和目录都是从根目录开始的，根目录通常表示成/(不要将这个目录与第 3 章中讨论的名为根的用户账户相混淆)。其他所有的文件和目录都起源于根目录。根目录通常包含一组通用目录(请参阅本章的“根的基本目录”一节)，这些目录中又包含了子目录，依此类推。要在 Unix 中找到指定的信息，需要定位正确的目录、正确的子目录以及正确的文件。

4.1.1 目录结构

Unix 使用分层的文件系统结构，它很像一棵倒置的树，将根(/)作为文件系统的基础，其他所有目录都从这里开始。绝大多数 Unix 系统都使用图 4-1 中所示的目录。表 4-1 中列出了这些目录的说明(关于 Mac OS X 系统的目录结构，请参阅第 20 章)。并不是 Unix 的每个版本都含有列出的所有目录，而且该列表列出来的内容也不是非常完整，因为 Unix 供应商可以添加他们自己的目录。

根据供应商及其客户的需要，每个供应商的 Unix 系统实现它们自己的目录结构。没有哪个系统具有与其他系统完全一样的目录结构，但是它们通常都具有本章所描述的目录并且遵循相关约定。

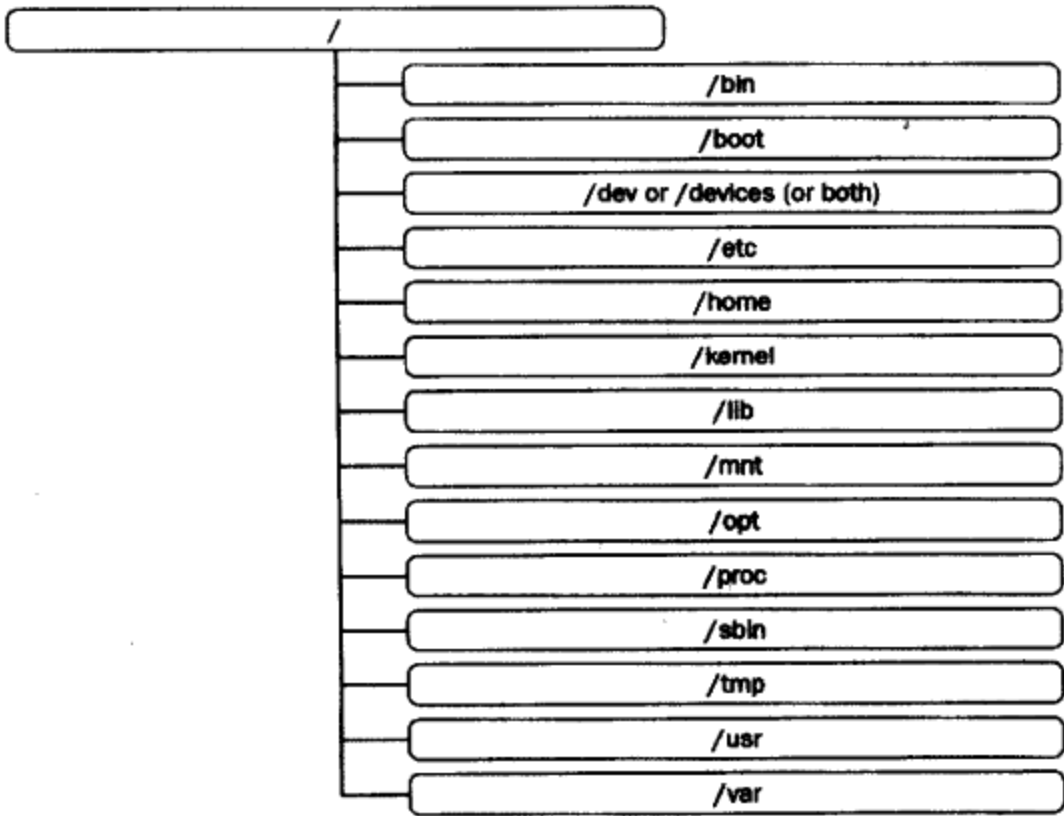


图 4-1

本质上，总是从根目录出发来查找任何其他的目录或文件。如果前面要查找的雇佣信息存储在 Unix 计算机上，就可以在/home/hr/A_J/John_Doe 找到指定的信息(假设该员工的名字是 John Doe)。这里，/是根目录，home 是根的子目录，hr 是 home 的子目录，A_J 是 hr 的子目录，而所要查找的文件 John_Doe 位于目录 A_J 中。

4.1.2 根的基本目录

除了记住根目录是文件系统的基础之外，还应该知道在大多数 Unix 系统上通常存在着一些核心目录。这些目录有其特殊的作用，通常保存相同类型的信息以便定位文件。表 4-1 中列出了在主要的 Unix 版本上存在的目录。

表 4-1

目 录	说 明
/	根目录中只包含文件结构的顶层所需要的那些目录(或者是那些已经安装在其中的目录)。根目录下不必要的子目录会搅乱系统，使管理更加困难，而且根据系统的版本，可能会占用分配给/的空间
bin	通常包含二进制(可执行的)文件，这些文件对系统的使用非常关键。它通常还会包含基本系统程序，如 vi(用于编辑文件)、passwd(用于修改口令)和 sh(Bourne shell)
boot	包含用于启动系统的文件
dev devices	系统上可能只存在其中一个目录或者两个目录都存在。它们包含设备文件，通常包括 cdrom(CD-ROM 驱动器)、eth0(以太网接口)和 fd0(软驱)。(在不同的 Unix 系统中，设备的命名通常是不同的)
etc	包含系统配置文件，如 passwd(保存用户账户的信息，不要与/bin/passwd 相混淆)；hosts(包含与主机解析有关的信息)和 shadow(包含加密后的口令)
export	通常包含远程文件系统(物理系统以外的那些系统)，例如为了节约空间并将主目录集中起来，从另外一个系统导出的主目录
home	包含用户和其他账户(例如，在/etc/passwd 中指定的账户)的主目录
kernel	包含内核文件
lib	包含共享的库文件，有时候还包含与内核相关的其他文件
mnt	用于安装其他的临时文件系统，例如分别用于 CD-ROM 驱动器和软驱的 cdrom 和 floppy
proc	包含所有标志为文件的进程，它们是通过进程号或者其他的系统动态信息进行标志的
sbin	包含二进制(可执行的)文件，通常用于系统管理。示例有 fdisk(用于划分物理磁盘)和 ifconfig(用于配置网络接口)
tmp	保存某些临时文件，这些文件在两次系统启动之间使用(有些 Unix 系统在两次启动之间并不删除 tmp 目录的内容)
usr	可以用于各种目的，或者可以被许多用户使用(如用于联机帮助页)。它可以包含管理命令、共享文件、库文件以及其他内容
var	通常包含长度可变的文件，例如日志和打印文件、以及数据量可变的任何其他类型的文件。以日志文件(通常位于/var/log)为例，它可以非常小，也可以非常大，这依赖于系统配置

实际 Unix 系统中的目录可能更多，也可能更少，或者包含了所有这些目录，但是通常都会含有其中的 5 个到 6 个，再加上一些子目录，这些子目录随实现的不同而变化。

4.2 路径和大小写

在继续阅读本书之前，还有另外两个重要的概念需要了解：路径(绝对路径和相对路径)和区分大小写。

每个文件都有一个绝对路径和一个相对路径。绝对路径是指文件在文件系统中的准确位置，如/etc/passwd。相对路径是指相对于用户当前位置的一个文件或目录的位置。例如，如果用户在/etc目录下，则/etc/passwd的相对路径是passwd，因为它和用户在相同的目录下。这类似于家庭住址。如果是将自己的地址告诉街坊邻居，可能会说自己住在某条街过去两条街。这就是相对地址——相对于邻居而言。如果是将自己的地址告诉其他国家的某个人，就必须说的更加详细，如美国蒙大纳州的某个城镇中的某条街的某号以及邮政编码，这就是绝对地址。如果从已知位置调用文件或目录，那么使用相对位置是个不错的选择，但是绝对路径总是一个更为安全的选择，因为它指定了准确位置。

Unix是区分大小写的操作系统。这意味着文件或目录名的大小写是有区别的。在DOS或Microsoft Windows系统中，可以随意输入文件名而不用考虑它的大小写。而在Unix中，必须知道文件或目录名的大小写，因为real_file、Real_file和REAL_FILE是三个不同的文件名。但是，为了方便用户使用，Unix的文件名按惯例都是小写(系统生成的文件更是如此)。Mac OS X是不区分大小写/保留大小写的文件系统。这意味着在Mac OS X系统上，将一个文件命名为real_file、Real_file或REAL_FILE是没有区别的，但是同时只能存在其中的一个文件名。尽管Mac OS X不区分大小写，但是它保留了大小写输入。在Mac OS X计算机和其他的Unix系统之间交换文件时，应该牢记这一点。

4.3 文件系统导航

既然已经了解了文件系统的基础知识，那么就可以开始导航到所需要的文件了。表4-2中列出了用于导航系统的命令。

表 4-2

命 令	说 明
cat	Concatenate: 显示一个文件
cd	Change directory: 转到指定的目录
cp	Copy: 把一个文件/目录复制到指定位置
file	识别文件类型(二进制、文本、等)
find	查找文件/目录
head	显示文件的开始部分
less	从开头或结尾开始浏览整个文件
ls	List: 显示指定目录的内容
mkdir	Make directory: 创建指定的目录
more	从头到尾浏览一个文件

(续表)

目 录	说 明
mv	Move: 移动文件/目录的位置或者重命名一个文件/目录
pwd	Print working directory: 显示用户的当前目录
rm	Remove: 删除文件
rmdir	Remove directory: 删除目录
tail	显示文件的结尾部分
touch	创建一个空文件或者修改一个现有文件的属性
whereis	显示文件的位置
which	如果文件位于用户的 PATH 内, 则显示文件位置

下面我们将更进一步地研究其中的一些命令。

4.3.1 pwd

所研究的第一个命令是 `pwd`，它显示用户在文件系统中的当前位置。用户知道自己在文件系统中的位置这一点非常重要，因为，当用户以为自己是在某个目录下，而实际上却是在另外一个目录下时，他运行的某些命令可能会给系统造成严重的损害。`pwd` 命令没有参数；只需要在命令行输入 `pwd` 即可。相应的输出与图 4-2 中所示的类似。



图 4-2

4.3.2 cd

`cd(change directory)`命令使得用户能够随意改变自己在文件系统中的位置。不带参数单独使用该命令时，用户将返回到自己的主目录。要转到另外一个目录，需要将该目录的名字作为参数：

```
cd directory
```


例如，如果输入 `cd /etc`，则将转到 `/etc` 目录(可以使用 `pwd` 命令来确认新位置)。只要用户具有进入相应目录的权限，`cd` 命令就可以让用户到达指定的位置。要进入 `/var/adm` 目录，可以使用下面的命令：

```
cd /var/adm
```

用户登录到 Unix 系统之后通常会从某个目录中开始自己的工作，这个目录称为主(home)目录(如果存在管理错误或者所指定的主目录有问题，也可以从其他的目录中开始)。用户通常要管理自己主目录中的内容(文件或目录)，用户的主目录在 `/etc/passwd` 文件中定义并且存储用户的文件。在许多命令中都可以使用 `~`(颚化符号)来表示主目录。例如，`cd ~` 将使用户返回到自己的主目录，而 `ls ~` 列出用户主目录中的内容。

请记住，在 Unix 中，每个目录都是一个文件，包括当前目录和当前目录之前的(或者之上的)目录。每个目录中都有两个文件，分别是 `.`(当前目录)和 `..`(上一层目录)。例如，如果目前在 `/usr/openwin/share/etc/workspace/patterns` 下，并且希望转到 `/usr/openwin/share/etc/workspace` 下，可以简单地输入 `cd ..` 来代替比较长的 `cd /usr/openwin/share/etc/workspace`。这个约定在脚本编程中应用得很多，第 13 章和第 14 章将介绍这方面的内容。

4.3.3 which 和 whereis

`which` 和 `whereis` 命令帮助查找那些已经知道名字但是不知道位置的文件。将文件名作为参数，`which` 只在用户的 `PATH` 所指定的文件中查找(`PATH` 是一个环境变量，它包含一组目录，可执行文件可能位于这些目录中；第 5 章中讨论这个环境变量)。例如，如果正在使用 `ls` 命令，并希望知道所用的 `ls` 命令在文件系统中的位置(大多数 Unix 系统都含有 `ls` 的 BSD 版本和 System V 版本)，就可以使用命令 `which ls`。它将显示用户 `PATH` 中的 `ls` 命令的实例。`whereis` 命令将在系统为它定义的所有目录中定位要查找的命令，而不是只搜索用户的 `PATH`。如果为 `which` 或 `whereis` 提供的参数在文件系统上并不存在，那么用户会收到一个 `command not found` 类型的错误消息。

图 4-3 显示的是 `which` 和 `whereis` 命令的输出示例，它们以 `vi` 命令作为参数(`vi` 是一个编辑器，在第 7 章中讨论)。

`which` 命令只显示了 `/usr/bin/vi`，因为在用户的 `PATH` 中，`/usr/bin` 在 `/usr/ucb` 之前(参见图 4-3 中 `echo $PATH` 命令的输出)。`whereis` 命令的输出显示了 `vi` 命令的所有位置，这些位置都包含在系统定义的一组标准目录中。

`echo` 命令显示所提供参数的内容。当 `echo` 与一个已经定义的系统变量一起使用时，它将显示该变量的含义。

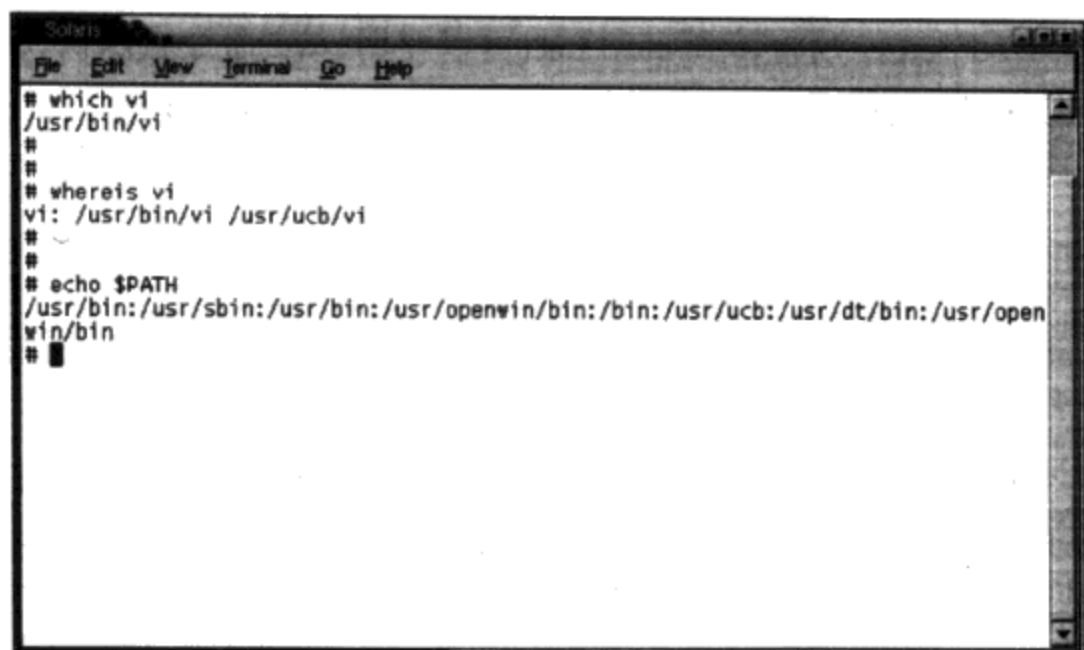


图 4-3

4.3.4 find

在 Unix 中, 还可以使用 `find` 命令来定位文件, 虽然该命令可能占用大量的资源(使系统响应变慢)。该命令的语法如下:

```
find pathname options
```

例如, 如果希望找到 `lostfile` 文件, 并且认为它位于 `/usr/share` 目录下, 可以使用下面的命令:

```
find /usr/share -name lostfile -print
```

`find` 命令有多个选项, 关于它的众多用法, 可以参考联机帮助页以获得更多的说明。

4.3.5 file

找到一个文件后, 通常希望对它执行一些操作。首先要确定它的文件类型(如二进制或者文本), 这时候就需要用到 `file` 命令。该命令的语法如下:

```
file filename
```

命令的输出将显示该文件是二进制的文件、文本文件、目录文件、设备文件或者是 Unix 中任何其他类型的文件。据此可以确定是否可以使用本章后面讨论的方法来查看该文件。例如, 由于字符的编码方式不同, 利用 `more` 命令将不能很好地显示一个二进制文件或目录文件的内容。

4.3.6 ls

`ls` 命令能够列出用户有权访问的任何目录的内容。单独运行 `ls` 命令将列出当前目录的内容。要显示任何其他目录的内容, 请使用 `ls path`。例如, `ls /usr/bin` 显示 `/usr/bin` 目录下的文件和目录。使用 `ls -l` 命令将显示目录内容的相关扩展信息。图 4-4 首先显示了 `ls` 的输出, 接着显示了 `ls -l` 的输出。

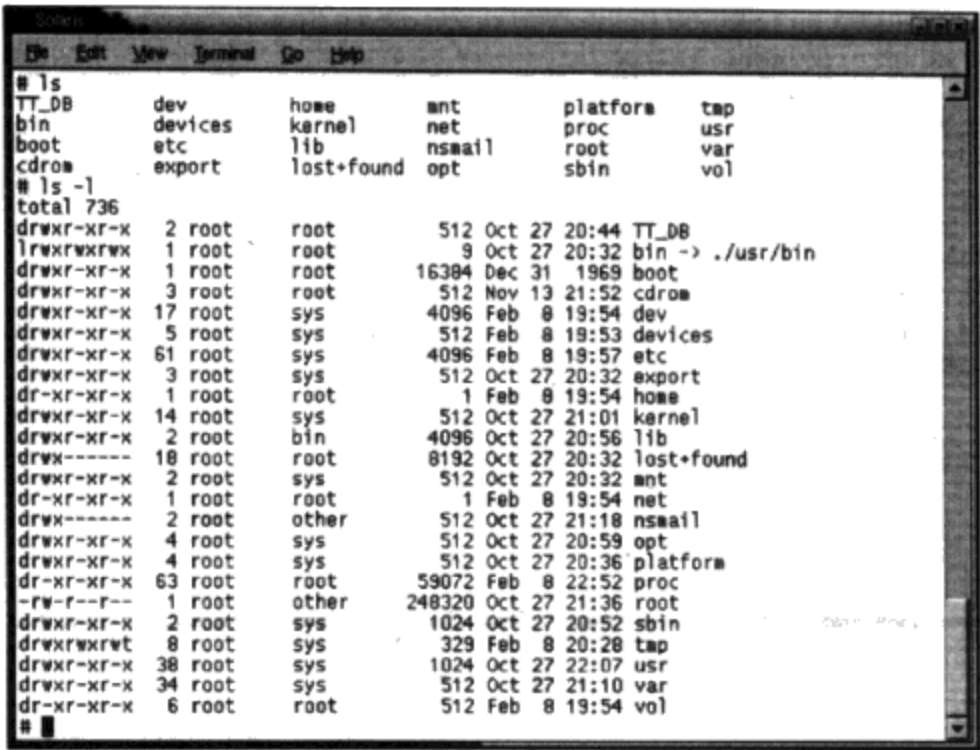


图 4-4

表 4-3 描述了扩展信息所包含的内容。这里以图 4-4 中的某一行为例：

drwxr-xr-x 61 root sys 3584 Nov 3 19:20 etc

表 4-3

ls -l 的输出	说 明
drwxr-xr-x	文件的类型以及与文件相关的权限
61	文件的链接数目(在“文件类型”一节中讨论)
root	文件的所有者(在第 3 章中讨论)
sys	文件所有者从属的组(在第 3 章中讨论)
3584	文件的大小(以字符计算)
Nov 3	上一次修改文件或目录的时间
19:20	
etc	文件或目录的名字

如果由于某些原因导致 ls 命令不可用，则可以使用 echo 命令来显示文件。简单地使用 echo directory*来查看一个目录的内容。例如，要查看 /目录的内容，可以使用 echo /*，它的输出与不带参数单独运行 ls 命令时的输出类似。要显示隐藏文件(下一段讨论)，使用 echo /*.*命令。必须使用元字符 *。（元字符将在第 7 章和第 8 章中讨论）。

利用 ls 命令的-a 选项可以显示所有的文件和目录，包括隐藏的文件和目录。在文件名的前面加一个 .(句号)可以隐藏该文件或目录。标准 ls 命令的输出是不包含隐藏文件的。隐藏文件或目录的一个合理的原因是为了减少运行 ls 命令时所显示的内容的数量。图 4-5 首先显示了一个目录下运行 ls 的输出，接着是运行 ls -a 的输出，从而可以看出其中的差别。

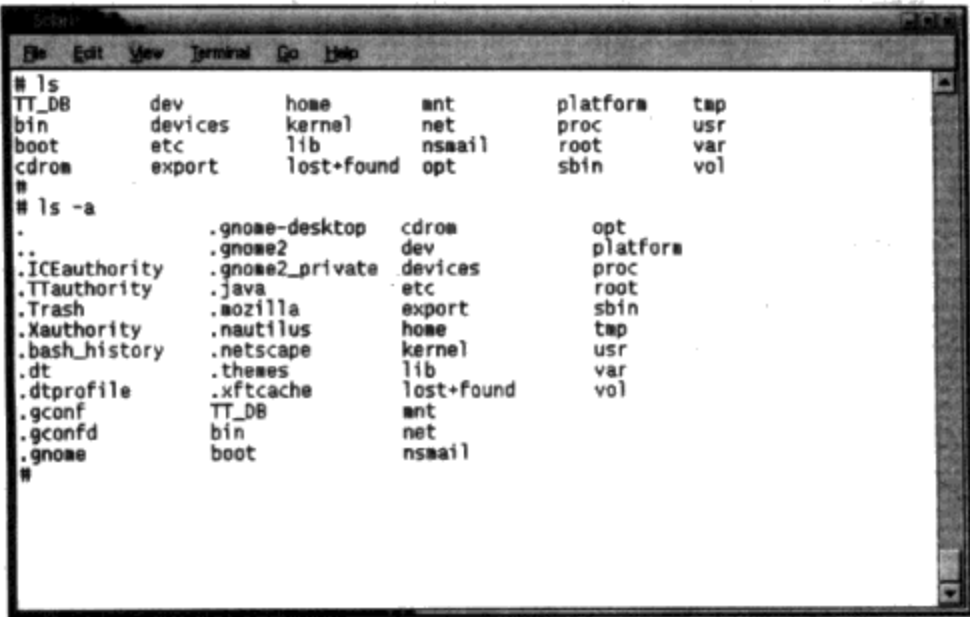


图 4-5

当一个非根用户试图列出某个目录的内容时，可能会出现禁止访问的错误，这是在使用 ls 命令时最常遇到的问题之一。这个错误通常是由于权限不足(本章后面讨论)导致的。

4.4 文件类型

在 ls -l 示例中(图 4-4)，每个文件行都以 d、- 或 l 开始。这些字符指示了所列文件的类型。还有其他的文件类型(这些文件类型用 ls -l 命令输出条目的第一个字符表示，如表 4-4 所示)，但是这三种是最普通的。

表 4-4

文件类型	说 明
-	普通文件，如 ASCII 文本文件、二进制可执行文件或硬链接(链接将在下一节中讨论)
b	块设备文件 (块输入/输出设备文件，用于从一个设备发送数据或者向它传送数据，有多种设备，例如物理硬盘)
c	字符设备文件(原始输入/输出设备文件，用于从一个设备发送数据或者向它传送数据，有多种设备，例如物理硬盘)
d	目录文件(包含一组其他文件和/或目录的文件，这些目录是该目录的子目录)
l	符号链接文件(下一节讨论)
p	命名管道(一种进程间通信的机制)
s	套接字(用于进程间通信)

4.5 链接

Unix 中的链接类似于 Microsoft Windows 中的快捷方式。要理解链接，需要了解 inode。在 Unix 中，每个文件都有一个相关联的数字，称为 inode。Unix 不是使用文件名来引用文件；它使用 inode。在一个分区中，inode 是惟一的，因此，如果两个完全不相关的文件位于不同的分

区内，它们可以有相同的 inode。这与驱动器的许可证号码(inode)非常类似，许可证号码在一个国家(地区)内是惟一的。在另一个国家中可以有相同的驱动器许可证号码，这两个号码根据国家来区分。

链接在许多方面都非常有用，例如可以为命令、程序或文件取别名，使其具有一个更为普通的名字。还可以使用链接为一个文件创建“副本”，这个副本不会占用存储空间，因为它指向相同的内容。

有两种类型的链接：硬链接和软链接(也称为符号链接)。硬链接不能跨越文件系统(物理文件系统如硬盘驱动器)，并且用于链接的文件与原文件完全一样。在 inode 引用中，所链接的文件与链接名具有相同的 inode 值，也正是因为这个原因，我们不能在不同的文件系统之间进行硬链接。对硬链接的文件所做的修改会反映到由硬链接产生的文件上，反之亦然。要创建一个硬链接，可以使用下面的命令：

```
ln file_name link_name
```

硬链接与原文件具有相同的 inode，这一点使用 `ls -li` 命令可以看出。

软(符号)链接能够跨越文件系统，甚至可以跨越不同的计算机系统。它具有惟一的 inode 值，如果删除该链接，其原文件依然存在。要创建一个符号链接，使用命令：

```
ln -s file_name link_name
```

回头来看图 4-4，观察 `ls -l` 输出中的第 2 个文件，它的文件类型位置上是一个 l，并且其文件名后面有一个 ->。这表示它是一个链接。名为 bin 的目录在 -> 之后显示 `./usr/bin`。这意味着该目录实际位于 `/usr/bin` 中。通常使用链接来实现以下功能：使文件查找更容易，为其他文件创建方便的快捷方式，对文件进行分组，以及使用其他的名字调用文件或目录。虽然目录通常会在 `ls -l` 输出的第一列中显示类型为 d，但是链接到目录的文件则显示类型为 l，因为它并不是真正的目录，而是指向目录的链接。

创建软链接时，总是使用绝对路径而不是相对路径以最大化可移植性(在大多数类型的 Unix 系统上，可移植性能够用脚本来实现)。

无论用户以哪种方式引用文件(通过 `hard_link`、`soft_link` 或者原文件)，都可以查看对任意链接或原文件所做的修改。在移动或者删除一个存在链接的文件时，必须认清系统上的所有软链接，因为这种操作可能会使它们断开。例如，假定在某个目录中有 `sales_forecasts.txt` 文件，该文件包含年度销售预测。如果希望来自不同文件系统的其他用户能够通过软链接查看该文件，则可以在一个共享目录中创建一个软链接，称为 `steves_sales_forecasts.txt`，从而其他人可以方便地定位和访问它。还可以在自己的主目录中创建硬链接 `my_sales_forecasts_2005.txt`(假定在相同的文件系统中)，从而可以在需要的时候方便地引用该文件。如果修改原文件(`sales_forecasts.txt`)的名字，硬链接文件(`my_sales_forecasts_2005.txt`)仍将指向正确的文件，因为硬链接使用 inode 作为引用，当文件名改变时，inode 并没有改变。但是，软链接文件 `steves_sales_forecasts.txt` 将不再指向正确的位置，因为软链接使用文件名作为引用。如果修改任意链接(软的或硬的)的名字，它们将仍然指向正确的位置，因为原文件没有改变。

关于修改具有链接的文件，还有最后一点要说明：如果删除了存在链接的原文件(`sales_forecasts.txt`)，然后用相同的文件名(`sales_forecasts.txt`)重新创建一个文件，该文件含有不

同的数据，那么硬链接将不再有效，因为文件的 inode 已经改变了，但是软链接将依然有效，因为它只引用文件名。

实战

创建链接

链接的概念最初理解起来可能比较困难，但是尝试使用它们就可以弄清楚链接是什么。

- (1) 使用 `cd` 命令定位主目录：

```
$ cd ~
```

- (2) 使用 `touch` 命令创建一个名为 `original_file` 的文件：

```
touch original_file
```

该文件是即将对其进行硬链接和软(符号)链接的基本文件。

- (3) 运行 `ls -l` 命令来查看刚才创建的文件，其输出与下面的内容类似：

```
$ ls -l
-rw-r--r--  1 username  usergroup   0 Jan 16 16:19 original_file
$
```

请注意，对该文件的链接数目是 1(第 2 列)，意味着这是对该 inode 的惟一链接(文件的大小是 0——Jan 前面的一列——意味着这个文件不包含数据)。

- (4) 使用 `ln` 命令对 `original_file` 创建一个硬链接，该链接命名为 `hard_link`：

```
$ ln original_file hard_link
$
```

如果试图在不同文件系统上的文件之间创建一个硬链接，将会收到一个错误信息，类似于 `ln : /hard_link is on a different file system.`

- (5) 再次运行 `ls -l`。将会在类似输出中看到如下两个文件(除了主目录文件中的其他文件以外)：

```
-rw-r--r--  2 username  usergroup   0 Jan 16 16:19 hard_link
-rw-r--r--  2 username  usergroup   0 Jan 16 16:19 original_file
```

请注意对该文件的链接数目是 2(第 2 列)，意味着这是对 inode(原始文件)的两个链接之一。两个文件的上一次修改日期(Jan 16 16:19)也是一样的，即使没有修改 `original_file`。

- (6) 运行 `ls -li` 命令来显示文件的 inode 值。将会看到这两个文件的 inode 值是一样的。

```
$ ls -li
116 hard_link
116 original_file
$
```

- (7) 现在使用 `ln -s` 命令来为 `original_file` 创建一个软链接，称为 `soft_link`：

```
$ ln -s original_file soft_link
```

- (8) 再次使用 `ls -l` 命令显示文件。其输出与下面的内容类似：

```
$ ls -l
```

```
-rw-r--r--  2  username  usergroup  0  Jan 16 16:19 hard_link
-rw-r--r--  2  username  usergroup  0  Jan 16 16:19 original_file
-rw-r--r--  1  username  usergroup 13  Jan 16 16:30 soft_link -> original_file
$
```

请注意，相对于 `original_file` 和 `hard_link`，`soft_link` 显示了一个不同的链接数目(1)和修改时间(16:30)。它还包含额外的输出显示了链接的指向(`->original_file`)，因为它链接到另外一个文件系统，而不是直接链接到相同的 `inode`。

- (9) 使用 `ls -i` 命令来查看所有文件的 `inode` 值。可以看出 `soft_link` 的 `inode` 与 `original_file` 和 `hard_link` 的 `inode` 不同。

```
$ ls -i
116 hard_link
116 original_file
129 soft_link
$
```

- (10) 使用 `cat` 命令来查看每个文件的内容，确认这些文件中没有包含任何文本或数据：

```
$ cat original_file
$ cat hard_link
$ cat soft_link
$
```

- (11) 要了解原文件的改变是如何影响链接文件的，可以使用 `echo` 命令和输出重定向给 `original_file` 中添加一行 “This text goes to the original_file”。

```
$ echo "This text goes to the original_file">>>original_file
```

该命令回显所输入的文本，然后将输出添加(>>)到文件 `original_file` 的末尾。因为该文件中没有其他数据，所以添加命令将新行放在 `original_file` 的开头。

- (12) 运行 `ls -l` 命令来查看原文件和链接文件在文件大小方面的变化，尽管我们只给 `original_file` 文件添加了数据。

```
$ ls -l
-rw-r--r--  2  username  usergroup  36 Jan 16 16:52 hard_link
-rw-r--r--  2  username  usergroup  36 Jan 16 16:52 original_file
-rw-r--r--  1  username  usergroup  13 Jan 16 16:30 soft_link -> original_file
$
```

可以看出 `hard_link` 和 `original_file` 的大小以及修改时间都发生了变化，但是 `soft_link` 文件保持不变。如果用 `cat` 命令来查看文件的内容，可以发现三个文件具有完全相同的内容：

```
$ cat original_file
This text goes to the original_file
$ cat hard_link
This text goes to the original_file
$ cat soft_link
This text goes to the original_file
$
```

- (13) 要了解硬链接文件的变化对原始文件的影响，可以使用 `echo` 命令和输出重定向给 `hard_link` 中添加一行文本 “This text goes to the hard_link file”。

```
$ echo "This text goes to the hard_link file">>hard_link
```

- (14) 运行 `ls -l` 并观察其输出。`original_file` 和 `hard_link` 的修改时间和大小都发生了改变，但是 `soft_link` 没有变化。

```
$ ls -l
-rw-r--r--  2 user-name  usergroup  73 Jan 16 17:11 hard_link
-rw-r--r--  2 username   usergroup  73 Jan 16 17:11 original_file
-rw-r--r--  1 useame     usergroup  13 Jan 16 16:30 soft_link ->
original__file
```

- (15) 现在再次使用 `cat` 命令来显示文件的内容。注意每个文件的变化：

```
$ cat original_file
This text goes to the original_file
This text goes to the hard_link file
$ cat hard_link
This text goes to the original_file
This text goes to the hard_link file
$ cat soft_link
This text goes to the original_file
This text goes to the hard_link file
$
```

如果使用 `echo` 命令给 `soft_link` 文件添加一行文本，将会修改原始文件，从而也更新了 `hard_link` 文件。

- (16) 使用 `echo` 命令和输出重定向给 `soft_link` 文件中添加一行 “This text goes to the soft_link file”：

```
$ echo "This text goes to the soft_link file">> soft_link
```

- (17) 使用 `cat` 命令来显示文件的内容，将会得到如下输出：

```
$ cat original_file
This text goes to the original_file
This text goes to the hard_link file
This text goes to the soft_link file
# cat hard_link
This text goes to the original_file
This text goes to the hard_link file
This text goes to the soft_link file
$ cat soft_link
This text goes to the original_file
This text goes to the hard_link file
This text goes to the soft_link file
$
```

工作原理

这些链接都引用相同的文件，但是它们在系统上出现的方式不同。软链接和硬链接都指向

同一个文件，并且编辑它们都将修改原文件的内容。软链接和硬链接之间的主要区别在于，在删除原文件时，它们的处理方式不同；当链接和所链接的文件位于不同的文件系统上时，它们的使用方式不同。

4.6 文件和目录权限

文件的权限是 Unix 系统安全的第一道防线。Unix 权限的基本类型有读、写和执行权限，在表 4-5 中描述了这些权限。

表 4-5

权 限	应用于目录	应用于任何其他类型的文件
读(r)	授予读取目录或子目录内容的权限	授予查看文件的权限
写(w)	授予创建、修改或删除文件或子目录的权限	授予写入权限，允许一个经过授权的实体修改文件，例如向文本文件中添加文本或者删除文件
执行(x)	授予进入目录的权限	允许用户“运行”程序
-	无权限	无权限

下面是 `ls -l` 命令的输出示例，其中包括一个文件和一个目录：

```
$ ls -l /home/mikec
-rwxr-xr-- 1 mikec  users  1024      Mov 2 00:10  myfile
drwxr-xr--- 1 mikec  users   1024     Nov 2 00:10  mydir
```

每个文件的权限都是用左起第 2 到第 10 个字符来记录(记住第 1 个字符表示文件类型)。权限分成 3 组，每组有 3 个字符，组中的每个位置对应一个指定的权限，其顺序为：读、写、执行。前 3 个字符(2~4)表示文件所有者的权限(本例中是 `mikec`)。第 2 组的 3 个字符(5~7)表示文件所属组的权限(示例中为 `users`)。最后一组的 3 个字符(8~10)表示其他任何人的权限(Unix 中称为“其他”)。表 4-6 详细说明了 `ls -l` 输出示例中的 `myfile` 的权限。

表 4-6

字 符	应 用 于	定 义
<code>rwX</code> (字符 2—4)	文件的所有者(在 Unix 中称为用户)	文件的所有者(<code>mikec</code>)对文件具有读取(或者查看)、写入和执行的权限
<code>r-x</code> (字符 5—7)	文件所从属的组	该文件所属的组(用户群)中的用户能够读取文件，并且，如果文件中含有可执行组件(命令等)，也可以执行该文件。该组没有写入权限——请注意—字符表示无权限
<code>r--</code> (字符 8—10)	其他任何人(其他)	有效登录到系统的其他任何人只能够读取文件——没有写入和执行权限(<code>--</code>)

字符-是一个占位符，它提供了适当的分隔以利于阅读。如果用户 `sallyb` 属于该用户组并且希望查看 `myfile`，那么她可以这么做，因为该组对这个文件具有读取和执行权限。如果她不是

所有者也不属于该用户组，那么只有当“其他”组(“其他任何人”)具有读取权限时，她才能够查看这个文件。本例中，其他任何人具有读取权限，所以 sallyb 能够查看该文件。

目录权限稍有不同，如本节开始处的表 4-5 所示。读取权限允许读目录和子目录的内容；写入权限能够创建、修改以及删除文件和目录；执行权限允许进入目录。

4.7 修改权限

要修改文件或目录权限，可以使用 `chmod(change mode)` 命令。该命令有两种使用方式：符号模式和绝对模式。使用 `chmod` 的绝对模式修改权限需要用到权限的数字表示，这种表示方法更加有效，而且系统也是用这种方法查看权限的。使用 `chmod` 的符号模式修改权限则使用所熟悉的 `rwX` 格式，对大多数新用户来说，这种方式更容易理解。

4.7.1 以符号模式使用 chmod

对初学者来说，修改文件或目录权限最简单的方法是使用符号模式。文件权限的第一个集合(`ls -l` 命令输出的第 2—4 个字符)用字母 `u` 来表示，代表用户；第二个集合(字符 5—7)用 `g` 来表示，代表组；最后一个集合(字符 8—10)用 `o` 来表示，代表其他任何人(其他)。还可以使用 `-a` 选项同时对所有 3 个组进行授权或者删除其权限。

通过使用表 4-7 中的操作符，可以利用符号权限来添加、删除或指定权限集合。示例文件 `testfile` 的原始权限为 `rwXrwxr--`。

表 4-7

chmod 操作符	含 义	示 例	结 果
+	为一个文件或目录添加指定的权限	<code>chmod o+wx testfile</code>	为 <code>testfile</code> 文件的其他用户(权限字符集 9~10)添加写入和执行权限
—	从一个文件或目录中删除指定的权限	<code>chmod u-x testfile</code>	删除文件所有者执行 <code>testfile</code> 的权限(<code>u</code> 表示用户或所有者)
=	设置指定的权限	<code>chmod g=r-x testfile</code>	为组设置 <code>testfile</code> 的读取和执行权限(不能写入)

下面是使用 `testfile` 文件的示例。在 `testfile` 文件上运行 `ls -l` 命令，显示文件的权限为 `rwXrwxr--`：

```
$ ls -l
-rwxrwxr-- 1 toms  users  1024      Nov 2 00:10  testfile
```

然后在 `testfile` 文件上依次运行表 4-7 中的每一个 `chmod` 命令示例，再次执行 `ls -l` 命令，

就可以看出权限的变化;

```
$ chmod o+wx testfile
$ ls -l
-rwxrwxrwx 1 toms users 1024 Nov 2 00:10 testfile
$ chmod u-x testfile
$ ls -l
-rw-rwxrwx 1 toms users 1024 Nov 2 00:11 testfile
$ chmod g=r-x testfile
$ ls -l
--rw-r-xrwx 1 toms users 1024 Nov 2 00:12 testfile
$
```

可以将这些命令合并在一行中执行, 如下所示:

```
$ chomd o+wx,u-x,g=r-x testfile
```

4.7.2 以绝对模式使用 chmod

利用 chmod 命令修改权限的第二种方式是用一个数字来表示文件权限的每一个集合。每种权限都分配了一个值, 如表 4-8 所示, 把每个权限集合中的数字相加就得到表示该集合的数字。

表 4-8

数 字	八进制的权限表示	权 限 引 用
0	无权限	---
1	执行权限	--X
2	写入权限	-W-
3	执行和写入权限: 1(执行)+2(写入)=3	-WX
4	读取权限	r--
5	读取和执行权限: 4(读取)+1(执行)=5	r-X
6	读取和写入权限: 4(读取)+2(写入)=6	rw-
7	所有权限: 4(读取)+2(写入)+1(执行)=7	rwX

每个集合的数字组合在一起就构成了文件的权限。例如, 如果文件所有者(用户)具有读取(4)、写入(2)和执行(1)权限(4+2+1=7), 那么组具有读取(4)权限, 其他任何人无权限(0), 则文件的权限是 740。如果希望按照上面的示例修改 myfile 文件的权限, 可以使用下面的命令:

```
chmod 740 myfile
```

它遵循 chmod 命令的语法: chmod permission filename。

要将 testfile 文件的权限(刚才用 chmod 符号权限将其变成-rw-r-xrwx)恢复到原始状态, 可以运行如下命令:

```
$ chmod 774 testfile
```

然后运行 ls. -l 来验证:

```
$ ls -l /home/toms
-rwxrwxr-- 1 toms  users  1024      Nov 2 00:10  testfile
```

在 `chmod` 命令中, 如果用 043 代替 774, 新的权限将是:

```
$ ls -l /home/toms
----r---wx 1 toms  users  1024      Nov 2 00:10  testfile
```

权限是一个复杂的概念, 对系统的安全极为重要。第 12 章将更加深入地讨论权限的安全含义。

4.8 查看文件

在对文件系统进行搜索并找到所需的文件之后, 用户可能会希望查看文件的内容。在 Unix 中, 有许多方法可以实现这一点, 从使用交互式的编辑器(如 `vi`, 将在第 7 章中讨论)到使用本节介绍的一些命令。这里所讨论的命令能够让用户快速地查看一个文件, 然后继续其他的操作, 而不需要打开一个单独的程序。这些命令还有其他的功能, 但是, 就本章而言, 重点放在它们的文件查看功能上。

要把文件内容输出到当前的终端屏幕上以查看文件, 可以使用命令 `cat filename`。但是, 将该命令应用于长文件时可能会出现問題, 因为 `cat` 本身只是简单地将文件内容一股脑地显示在屏幕上, 没有输出暂停——用户不得不非常快速地进行阅读。`more` 命令可以解决这个问题。它的运行方式与 `cat` 一样, 但是必须按下空格键或者方向键才会继续显示文件的内容, 从而使得用户能够每次查看一屏的输出。使用 `more` 命令, 还可以通过按下 `Enter` 键每次前进一行。`less` 命令更为强大, 因为用户可以使用 `vi` 移动键(第 7 章中讨论)或者方向键上下移动文件。用户必须按下 `q` 键以退出文件查看。下面是使用 `more` 和 `less` 命令的示例:

```
more /etc/syslog.conf
less /etc/syslog.conf
```

`head` 和 `tail` 命令也很有意义, 因为它们能够查看一个文件的开头(head)和结尾(tail)。它们的使用方法是:

```
head /etc/syslog.conf
tail /etc/syslog.conf
```

默认情况下, 这些命令只显示一个文件的前 10 行或者后 10 行。如果希望查看更多的行, 或者不想看到那么多行, 可以使用 `-n x` 参数来指定, 用希望查看的行数来代替 `x`。下面两个示例分别查看一个文件的前 15 行和后 15 行。

```
head -n 15 /etc/syslog.conf
tail -n 15 /etc/syslog.conf
```

`tail` 命令的一个重要选项是 `-f`(表示 follow)。该选项不断地扫描输入文件, 而不是简单地显示指定的行。例如, 当有事件发生时, 要实时地查看 `/var/log/syslog` 文件(该文件是许多 Unix 系统中的系统日志文件)的内容, 可以运行:

```
tail -f /var/log/syslog
```

当日志写入/var/log/syslog 文件中时，输出将显示该文件的内容，直到用户按下 Ctrl+C 组合键以停止循环为止。这对于监视文件内容的变化非常有用，特别是对不断增长的日志文件。

4.9 创建、修改和删除文件

要在文件系统中复制某个文件，可以使用 cp 命令。下例是把文件/etc/skel/cool_file 复制到另外一个位置的方法：

```
cp /etc/skel/cool_file /home/danl/cooll
```

必须具有适当的权限才能复制、移动或者修改文件。要完成复制，通常至少要对源文件(要复制的文件)具有读取权限，对目标目录和/或文件具有写入权限。

该命令在/home/danl 目录下创建了/etc/skel/cool_file 的一个副本，名称是 cooll。

cp 命令在复制文件时很有用，但是要把文件从一个位置转移到另外一个位置而不复制，则需要使用 mv(move)命令，它的语法与 cp 命令类似。例如，下例是将/etc/skel/cool_file 文件从它的原始位置转移到/home/danl，并且重命名为 cooll 的方法。

```
mv /etc/skel/cool_file /home/danl/cooll
```

mv 命令还可以只修改文件或目录的名字。例如，要把/home/danl/cooll 文件的名称改成/home/danl/login_script，可以执行如下命令：

```
mv /home/danl/cooll /home/danl/login_script
```

mv 命令也可以作用于目录，因此可以将整个目录从一个位置转移到另外一个位置。如果将 danl 的用户名改成 danl12，可以使用下面的命令来修改主目录名：

```
mv /home/danl /home/danl12
```

要创建一个空文件，可以使用 touch 命令。如果在一个现有文件上使用 touch 命令，将更新它的最后修改时间，但是如果将该命令与一个新的文件名一起使用，则创建一个空文件。该命令的语法是：

```
touch filename
```

为进行测试或者其他目的而需要创建空文件时，这个命令很有用。

4.9.1 删除文件

当然，有时候会希望完全删除一个文件。这时可以使用 rm(remove)命令。下面是从系统中删除/etc/skel/cool_file 文件的方法：

```
rm /etc/skel/cool_file
```

rm 命令具有非常强大的选项，其中最主要的两个是 -f 和 -r。-f 选项使 rm 命令强行删除一个文件，而不会询问是否真的想删除这个文件；该选项使得 rm 命令只是执行删除的操作而不

会产生任何输出。`-r` 选项使得 `rm` 命令能够进入到所指定目录的任意子目录中去, 这时 `rm` 命令的参数是一个目录名。如果用 `rm` 命令指定删除某个文件, 则 `rm` 命令将不会进入到任何目录中 (只有当指定删除某个目录时, 它才会进入该目录)。

输入 `rm` 命令的参数时要非常仔细, 特别是使用 `-f` (强制, *force*) 和 `-r` (递归, *recursive*, 或进入子目录) 选项时, 因为这个命令可能会删除或破坏系统。例如, 如果已经作为根用户登录, 则希望删除文件 `/tmp/remove_file`, 并且输入:

```
rm -rf / tmp/remove_file
```

`/` 和 `tmp` 之间意外的空格将导致该命令删除 `/` 文件系统, 从而删除整个系统。单独使用 `rm` 命令而不带任何选项是不能删除目录的, 但是具有 `-rf` 选项的 `rm` 命令能够删除目录及其子目录。使用 `rm` 命令之前, 要确定自己在目录结构中的准确位置 (使用 `pwd` 命令), 如果有可能的话, 最好使用绝对路径以确保自己准确地知道所要删除的内容。这里有一个示例: 假定文件系统已满, 需要立刻在系统上腾出空间。在 `/var/log/archives` 目录下运行 `ls -l` 命令 (当然已经对所有内容进行了备份) 并决定删除系统上所有额外的日志。假设已切换到根目录, 却误以为 `ls -l` 命令是在这个目录下运行的, 并运行下面的命令:

```
rm -rf *
```

该命令证明了您处在一个错误的终端窗口中——当前位于 `/` 目录下。如果已经作为根用户登录, 将会完全删除所有的系统文件, 导致系统不可用, 因为所运行的命令将在系统上递归地删除 `(-r)` 所有文件 `(*)` 而不会给出任何提示 `(-f)`。

4.9.2 创建和删除目录

`mkdir` 和 `rmdir` 命令是专用于处理目录的。`mkdir` 创建一个新目录以存储文件和其他目录。它的语法是 `mkdir directory_name`。例如, 要创建一个名为 `testdir` 的目录, 可以使用命令:

```
mkdir testdir
```

`testdir` 命令保存在当前的工作目录下。如果希望将它放在一个不同的目录下, 就需要使用绝对路径。下面是在 `/tmp` 目录下创建目录 `testdir` 的方法:

```
mkdir /tmp/testdir
```

要删除一个目录, 使用语法 `rmdir directory_name`。要删除上例中创建的 `testdir` 目录, 可以使用命令:

```
rm -r /tmp/testdir
```

与 `rm` 命令一样, 如果不知道自己在文件系统的位置并且以根用户的身份运行命令, 就可能造成重大的损失, 虽然由于 `rmdir` 命令自身的局限性, 后果不会非常严重。

`rmdir` 命令只是完全删除空目录, 从而提供了一些安全措施以防止意外地删除含有文件和其他目录的目录。

4.10 基本的文件系统管理

与所有的存储媒介一样，文件系统能够耗尽存储空间，如果缺乏适当的管理将会产生严重的问题。管理分区空间的第一种方法是使用 df(disk free)命令(分区的概念已经在本章的前面讨论过)。命令 df -k(disk free)显示磁盘空间的使用情况，以千字节为单位，如图 4-6 所示。

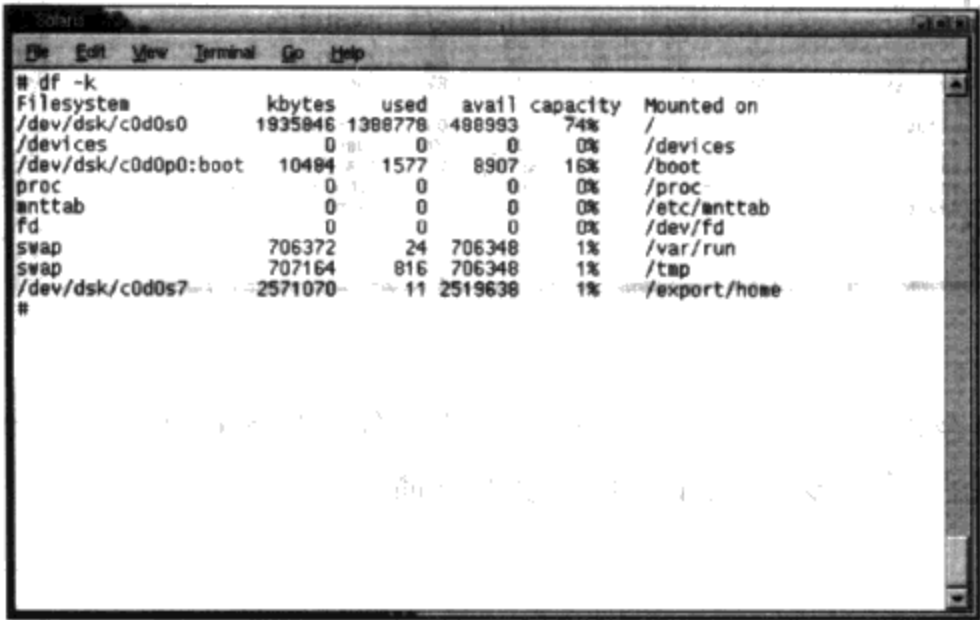


图 4-6

有一些目录，如 proc 和 fd，在 kbytes、used 和 avail 列显示为 0，在 capacity 列显示为 0%。这些是专用的(或者虚拟的)文件系统，尽管它们位于磁盘上的/目录下，但是它们本身并不占用磁盘空间。df -k 的输出通常在所有的 Unix 系统上都是一样的。表 4-9 中列出了它通常包含的内容。

表 4-9

列	说 明
Filesystem	物理文件系统(fdX(X=floppy drive number)=软驱，/dev/dsk/c0t0d0s0 表示磁盘驱动器上的一个分区，等)
kbytes	存储媒介上可用空间的总千字节数
used	已用空间的总千字节数(由文件占用)
avail	还可以使用的总千字节数
capacity	文件所占用的总空间的百分比
mounted on	文件系统的安装位置。在图 4-6 中，/(根)文件系统安装在/dev/dsk/c0d0s0 上，并且所分配的总空间中只剩下 26%可用。安装将在本章后面讨论

追踪 avail 和 capacity 列很重要，因为我们不希望自己的/(根)或者/tmp 分区被填满，否则将会导致严重的问题。文件系统的每一个部分都是它自己的实体；/文件系统位于自己单独的物理分区(或设备)上，/export/home(在 Unix 的 Sun Solaris 版本上，它通常作为用户的主目录)也一样。即使/export/home 的空间已经耗尽，根和其他分区的空间并不会跟着耗尽，因为它们都是它们自己的实体(下一节讨论)。

可以使用 -h(人类可读的，human readable)选项来显示输出，该选项用一种更易于理解的符号来表示空间的大小，如图 4-7 中的下半部分所示(-h 选项并不是在所有的 Unix 版本中都可用)。

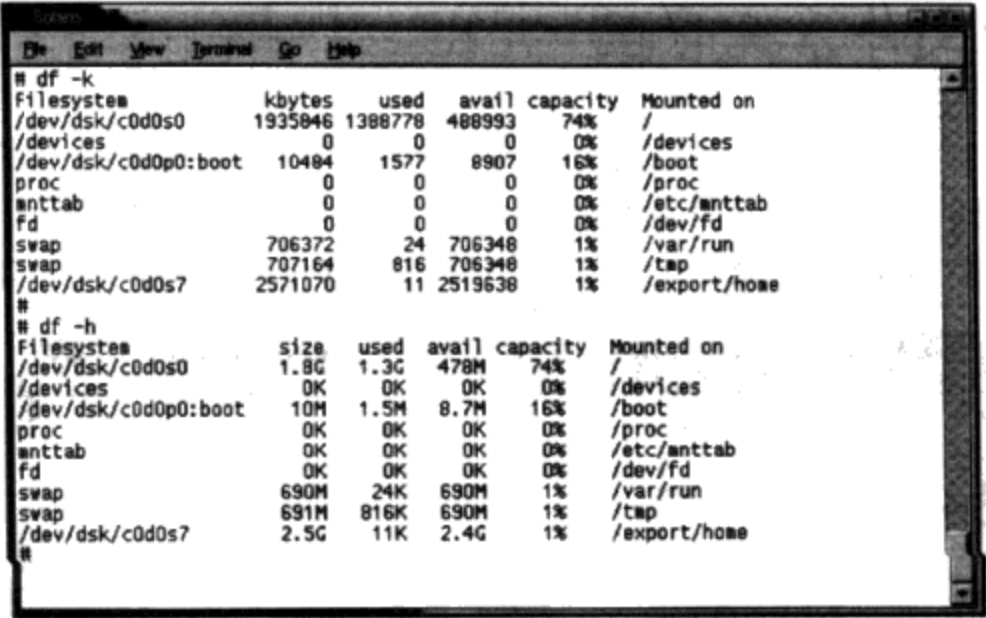


图 4-7

du(disk usage)命令能够通过指定目录来显示某个分区目录的磁盘空间使用情况。如果希望确定某个特殊的目录占用了多少空间，可以使用该命令。

```
$ du /etc
10    /etc/cron.d
126   /etc/default
6     /etc/dfs
...
```

-h 选项使得输出更容易理解:

```
$ du -h /etc
5k    /etc/cron.d
63k   /etc/default
3k    /etc/dfs
...
```

另外一个应该熟悉的命令是 fsck(file system check)。Unix 通常使用一个超级块来追踪文件系统，该超级块中包括文件系统的大小、可用的空闲块以及其他相关信息。当系统非正常关机(例如断电的时候系统仍然处于多用户模式)或者系统发生损坏时，将在这个超级块中产生错误。这些差错包括：系统的代码块标记为空闲(意味着可以写入)，而实际上它们正在使用(这将引起严重的数据破坏)；inode 大小错误；以及其他的管理问题。这些错误将导致超级块不一致，需要对它们进行修复。fsck 命令尝试修复它们。由于该命令会潜在地产生一些灾难性的问题，运行它的时候要极为小心，因此需要参考它的联机帮助页以获得更多的信息。

4.11 使文件系统可访问

文件系统必须先安装，然后系统才能使用它。启动时系统将安装根目录和在/etc/fstab 或/etc/dfstab 中命名的任何其他文件(或者是任何确定安装处理方式的文件，这些文件随着 Unix 版本的不同而不同)以便于系统使用。这些安装通常不需要用户干预，并且对终端用户通常是不可见的。安装一个文件系统意味着让系统使用这个文件系统。例如，如果希望在 Unix 中使用 CD-ROM 驱动器(cdrom)，系统必须把它安装到根文件系统从而可以定位该驱动器。可以通

过/mnt 目录或者/cdrom 目录把 CD-ROM 驱动器添加到根文件系统中。在有些系统上(例如 Mac OS X), 该操作自动完成; 而在有些系统上, 必须执行相应的命令才能使文件系统可用。观察图 4-8 中的示例文件。

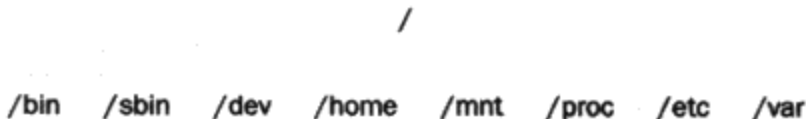


图 4-8

根据 Unix 的惯例, 通常在/mnt 目录安装临时文件系统(如 CD-ROM 驱动器、远程网络驱动器和软驱)。一个名为 fstab 的文件(这个名字随 Unix 的版本的不同而不同)标识了所有应该在启动过程中安装的不同文件系统。例如, 如果希望使用一个 CD-ROM 驱动器, 那么需要告诉 Unix 已经在主文件系统中添加了新的文件系统, 因为在使用 Unix 的过程中, 可能会用到多个 CD-ROM 驱动器。通常不希望将可移除设备作为启动安装过程的一部分, 因为如果删除了该设备会引起问题。用户需要为 Unix 标识这个新的文件系统, 从而使 Unix 能够了解如何与该媒介进行互操作, 同时用户还需要指出它在文件系统的位置(哪个目录)。可以使用本章后面介绍的 mount 命令来添加 CD-ROM 驱动器, 而且可以通过传递的选项根据需要使用文件系统。安装了该文件系统之后, 新的文件系统层次结构中将在/mnt 目录下包含 cdrom, 如图 4-9 所示:



图 4-9

要查看系统上当前安装的(可以使用的)文件系统, 使用如下命令:

mount

得到的输出与图 4-10 中所示类似。



图 4-10

mount 命令的输出通常分成几列，表 4-10 中分别对其进行了说明。

表 4-10

列	说 明
安装的设备	实际的设备名称：例如，这是一个 SCSI CD-ROM
安装点	新目录在文件系统中的位置
文件系统的类型	文件系统类型——例如 msdos、hfs 和 iso9660——告诉系统如何与存储媒介进行交互
安装选项	与文件系统的类型相关联的安装选项
转储选项	用于备份系统
fsck 选项	启动过程中用于识别文件系统检查的次序

例如，观察图 4-10 中 mount 命令输出的最后一行：

```
/dev/ide/host1/bus1/target0/lun0/cd on /mnt/cdrom type iso9660
(rp,nosuid,nodev,umask=0,iocharset=iso8859-1,cbdepage=850,user=beginningunix)
```

安装的设备是 /dev/ide/host1/bus1/target0/lun0/cd——这是实际的名称；它是一个 SCSI CD-ROM 设备。安装点是 /mnt/cdrom，文件系统的类型是 iso9660，这是一个 CD-ROM。该设备的选项有 ro(只读，read only)，这意味着不能对 CD-ROM 写入数据；nosuid(没有设置用户 ID 或组 ID)；nodev(没有为该文件系统解释字符或块专有设备)；umask；iocharset(用于将八位字符转换成十六位字符)；codepage(用于转换 FAT 和 VFAT 文件名)；和 user(用户名，本例中是 beginningunix)，他可以根据需要安装文件系统。该设备没有转储和 fsck 选项。

文件系统可以自动安装也可以手动安装，这依赖于所使用的 Unix 系统，因此用户可能并不需要手动安装它们。可以运行 mount 命令来查看已经安装的文件系统，如果输出列表中不存在需要的文件系统，则它没有自动安装(假定从启动到运行 mount 命令这段时间内没有人安装了该文件系统)。如果需要安装一个文件系统，可以使用 mount 命令，其语法如下：

```
mount -t file_system_type device_to_mount directory_to_mount_to
```

例如，如果希望将一个 CD-ROM 安装到目录 /mnt/cdrom，可以输入：

```
mount -t iso9660 /dev/cdrom/mnt/cdrom
```

这条语句假定 CD-ROM 设备叫做 /dev/cdrom 并且希望把它安装到 /mnt/cdrom。参考 mount 联机帮助页以获得更多特定的信息或者在命令行输入 mount -h 以获得帮助信息。安装完成后，可以使用 cd 命令从安装点来定位新的可用文件系统。

要从系统中卸载(删除)文件系统，使用 umount(请注意拼写：只有一个 n)命令，通过在命令中指定安装点或设备来完成卸载。例如，要卸载 cdrom，可以使用如下命令：

```
umount /dev/cdrom
```

mount 命令使得用户能够访问自己的文件系统，但是在大多数的现代 Unix 系统上，自动安装功能使得该进程对用户是不可见的，并且不需要用户的任何干预。

4.12 小结

文件系统的概念非常重要，因为在 Unix 系统中存储和使用应用程序都需要用到它们。本章讨论了导航文件系统的方法、文件的不同类型、权限的基本工作原理、快速查看文件的方法和使用目录的方法。本章还讨论了基本的文件系统管理以及用户使用文件系统的方法。

4.13 练习

编写一个命令，设置文件 `samplefile` 的文件权限，使得文件所有者具有读取、写入和执行权限，组和其他人具有读取和执行权限，请使用绝对模式(也称为八进制模式)来实现。

第 5 章 定制工作环境

一个环境变量控制 Unix 环境的一个特定方面。也就是说，环境变量将影响用户使用计算机时的视觉和感觉，以及用户可能从来不会注意到的许多底层的操作。可以使用环境变量来修改 Unix 环境的几乎每一个方面。

本章将非常详细地解释环境变量的概念并介绍一些很常用的环境变量。其中着重介绍 shell，它是一个程序，可以把用户的按键转化成操作系统能够识别并接受的命令。目前有许多可用的 Unix shell，这些数量众多的 shell 多多少少会导致一些混淆。使用的时候，请比较各种 shell 之间的区别，选择最适合自己的 shell。

5.1 环境变量

Unix 的灵活性是令人难以置信的。这一特点亦好亦坏，取决于用户所要实现的目的。对大多数用户来说，学习 Unix 的过程都是很艰难的。但是，如果用户在深入学习 Unix 之前能够花费一定的时间来定制自己的环境，就可以避免很多麻烦！

5.1.1 PS1 变量

可以在很大程度上修改 Unix 的行为，这取决于对某个特殊环境变量所赋的值。例如，环境变量 PS1 控制顶层命令提示符(command prompt)，或者游标前的字符串。当用户打开一个终端窗口或者登录到机器的控制台之后，就可以看见该提示符。提示符几乎能够包含用户所希望的任何内容，只要使用适当的值来定义这个环境变量。

下面的示例假定用户正在使用 Bourne 或者 bash shell 环境，默认安装时通常如此。如果执行命令：

```
PS1=">"
```

那么该 shell 中的顶层命令提示符将显示为：

```
>
```

游标在字符>之后。很简单，不是吗？如果执行下面的命令又会怎样呢？

```
PS1="I am ready to do your badding , Bob! "
```

不难猜到，得到的提示符类似于：

```
I am ready to do your badding , Bob!
```

其后紧跟着游标。尽管这类提示符很有趣，但是用户很快就会感到无聊。比较有用的提示符含有用户工作目录的相关信息，也就是在整个文件系统中的当前位置。当用户试图确定某个

特定文件的路径时，这些信息就会非常关键(要了解更多关于路径的知识，请参阅本章后面的“路径”一节)。

实战

配置 bash 提示符

在 bash shell 中,使用 PS1 环境变量可以有很多种方法来配置用户的提示符(虽然格式不同,但是在其他 shell 中也使用 PS1 变量)。一个有用的配置是在提示符中显示工作目录。要设置 PS1 的值使其显示工作目录，可以执行如下命令：

```
PS1=" [\u@\h \w]\$"
```

工作原理

该命令的结果是在提示符中显示用户的用户名、机器的名称(主机名)和工作目录(包含用户名是很有用的，因为这样就能够判断是以自己的用户名登录的，还是作为另外一个用户登录的，特别是如果您是惟一的系统管理员时)。下面是一个提示符示例：

```
[dave@linux1 /etc]$
```

有相当多的转义序列可以作为 PS1 的值参，尽量只使用最关键的转义序列以免提示符过长。

表 5-1

转 义 序 列	功 能
\t	当前的时间，表示成 HH:MM:SS
\d	当前的日期，表示成星期、月、日期(或日)
\n	换行
\s	当前的 shell 环境
\W	工作目录
\w	工作目录的完整路径
\u	当前用户的用户名
\h	当前机器的主机名
\#	当前命令的命令号。每输入一个新的命令，命令号就相应增加
\\$	如果有效的 UID 是 0(也就是说，作为根用户登录)，则提示符以字符#结束；否则，使用\$

5.1.2 其他环境变量

shell 有自己的环境变量。另外，如果深入研究 shell 脚本编程，就能够创建自己的环境变量以实现脚本的特殊功能。例如，可以创建一组变量，这组变量基于各个用户使用磁盘空间的情况来定义他们的提示符。这不是一个标准的 shell 变量，但是它适用于多种环境，如果有一个年龄比较小的用户经常下载大量的音乐或视频文件，把硬盘空间塞得满满的！对于这种情况，使用 shell 环境变量就特别有用。

要了解 shell 脚本——用于自动化各种 shell 功能的文本文件——的更多知识，请参阅第 13 章和第 14 章。

基于所选的 shell 环境，环境变量也发生相应的变化。虽然大多数 shell 都含有实现相同功能的变量，但是这些变量的名字在不同的 shell 中稍有差别，或者名字相同的变量在不同环境中的语法是不一样的。在本章后面的“配置 shell”一节中将会介绍一些独特的环境变量。

有些用户使用 Unix 已多年，但是几乎没有修改过环境变量。还有些用户比较痴迷于这方面，直到把每个可能的变量都做修改并且配置到最好的程度，他们才会满意。绝大多数用户都介于两者之间，他们配置喜欢的文本编辑器或邮件客户程序，可能会修改提示符，也会做一些其他的小改动以创建一个更加熟悉和舒适的计算机环境。

5.2 路径

正如第 4 章所述，Unix 系统上的每个元素都被视为文件。换句话说，操作系统用处理文件的方式来处理所有的命令和可执行程序。对有些用户来说，这个概念比较难以理解，特别是那些从纯粹的 GUI 环境如 Microsoft Windows 中转过来的用户。但是，从长远的观点来看，这种处理方式使得管理 Unix 系统更为简单。

根目录用字符/来表示。

关于 Unix 和它处理文件的方式，需要记住的是每个文件的位置都是惟一的，不管这个文件是命令、程序还是静态文档。这个位置就叫做完整路径名(full path name)，它指定文件在整个文件系统中的惟一位置。例如，命令 ls(用于列出文件)通常的完整路径名是/bin/ls。这意味着 ls 命令通常存储在/bin 目录下，该目录属于根目录下面的第一层目录。完整路径名把特定的文件树压缩成一行，但是也可以以分级的形式来表示路径名：

```
/
bin
ls
```

使用完整路径名有其自身的优点。特别地，它是了解文件系统并记住某个文件的特定位置的一种很好方法。但是，在日常应用中，完整文件名相当冗长，特别是如果正在使用的程序或文档在嵌套目录中存储得很深时。假设正在编写一个新程序，该程序的可执行文件位于主目录的一个子目录中。要执行这个文件，必须输入类似于下面的完整路径名：

```
/users/home/susan/Myprog/prog
```

在编写和调试程序的过程中，可能会数百次地输入该命令。这样就会按大量的键和消耗大量的体力，更不要说会增大输入错误的概率了。

完整路径名的另外一个问题是，用户可能希望使用某个特定的程序，但是并不知道它在文件系统中的具体位置。例如，最古老的 old-school Unix 仍然使用一个直接称为 mail 的电子邮件程序。这个程序可能位于/bin/mail 或者/usr/mail，甚至可能位于某个完全不标准的目录下，这取决于系统管理员构造文件系统的方式。如果刚刚安装了一个新风格的 Unix 并且使用默认设置，则安装程序可能会将 mail 放在一个意想不到的地方。最终结果是用户要花费大量的时间和精力来搜索文件系统，以便能够获得正确的完整路径名并最终运行电子邮件程序。

5.2.1 PATH 环境变量

上述这些问题有一个通用的解决方法。PATH 环境变量包含一组目录，可执行程序可能位于这些目录中。如果 PATH 变量的值中含有某个目录，那么调用该目录中的可执行文件时就不需要输入目录名。例如，如果邮件程序存储在 /usr/bin 中，并且 /usr/bin 是 PATH 的值的一部分，那么就可以简单地在命令提示符之后输入 mail 来调用程序，而不需要使用完整的路径 /usr/bin/mail。

在 Unix 术语中，调用一个程序就是运行它。

PATH 变量的值通常在一个配置文件(例如 /etc/profile)中设置，具有全局意义。大多数标准的 Unix 系统会在配置文件中设置一些特定的通用目录。另外，许多大型软件包(例如 Mozilla Web 浏览器)在安装过程中会自动将自己的目录添加到 PATH 变量中，以便系统能够很容易地找到它们。

用户可以向 PATH 变量中添加自己的值。当用户调用一个 shell 环境时，系统执行 shell 的配置文件——包括全局的和用户特有的。用户希望添加到 PATH 变量中的任何其他值也可以添加到 shell 配置文件中。

在 Bourne 和 bash shell 中，向 PATH 变量中添加值的格式为：

```
PATH=$PATH:new value
```

因此，如果一个名为 joe 的用户希望将自己的主目录 /home/joe 添加到路径中，可以采用下面的方式来实现：

```
PATH=$PATH:/home/joe
```

可以一次添加多个目录，用冒号将这些目录隔开：

```
PATH=$PATH: /home/joe: /home/joe//myprog:/home/joe/ myprog/bin
```

另外，用户可以将命令：

```
export PATH
```

添加到初始化文件中。这将使得变量的新值在这个 shell 以外的其他 shell 中也可用。这对于使用多个 shell 或者图形界面的用户很有用。

尽管大多数用户不需要考虑将目录名添加到 PATH 值中的次序，但是有些时候这个次序是很重要的。例如，假定有两个程序，它们位于不同的目录下但是具有相同的名字。如果通过在命令提示符之后输入文件名来调用程序，那么 shell 将依次在 PATH 目录中查找文件。只要 shell 发现匹配的程序，就会启动该程序，不管它是不是想要调用的程序。要想利用文件名优先调用另一个程序，需要在提示符后输入完整的路径名。

PATH 语句开始处的美元符号(\$)提示 shell，将新目录或值添加到 PATH 的当前值中，而不是取代它。例如，如果执行命令 PATH=PATH:/usr/sbin，那么 PATH 的值将包含相对目录 PATH 和完整路径名 /usr/sbin。如果执行的命令是 PATH=\$PATH:/usr/sbin，那么将把 /usr/sbin 目录添加到已包含在 PATH 的值中的目录内。

如果最近修改过 PATH 的值，并且突然不能够使用简单的命令名来调用程序了，那么可能是在执行命令的时候遗漏了一个\$。需要进行测试并为 PATH 重新配置正确的值。

5.2.2 相对路径和绝对路径

当使用完整路径名进行工作时，了解相对路径和绝对路径之间的差别是非常重要的。在 Unix 环境中，所有路径都相对于根目录(/)来命名。因此，就像以前所解释的，/bin 是/的子目录，/bin/appdir 是/bin 的子目录，从而使得 appdir 成为一个第 3 级目录。完整路径名/bin/appdir 还是一个绝对路径名，因为它含有树型结构中的所有元素，从/到最终目的地。

但是，在文件系统中变换目录时，不需要每次都使用绝对路径名。用户只需要知道目的地和出发点——用户的当前工作目录。如果用户执行了本章前面介绍的命令，那么就可以很方便地在提示符中显示当前的工作目录。

假定当前的工作目录是/bin。要使用 cd(修改目录)命令转移到 appdir 子目录，可以使用绝对路径，如下所示：

```
cd /bin/appdir
```

但是目前已经位于/bin 目录下，为什么要增加额外的按键呢？只需要执行命令：

```
cd appdir
```

如果给出的目录名不是绝对路径，就假定它是相对于当前目录的。例如，如果 appdir 目录包含一个名为 dir1 的子目录，就可以使用下面的命令从/bin 转移到该目录：

```
cd appdir/dir1
```

请注意相对路径前没有斜杠。

不管路径的级数是多少，一样可以使用相对路径。如果工作目录是 dir1，则相对路径：

```
dir2/dir3/dir4
```

仍然意味着：

```
/bin/appdir/dir1/dir2/dir3/dir4
```

对文件系统的研究越深入，相对路径就会越有用。

5.2.3 切换文件系统

除了使用绝对路径名和相对路径名之外，Unix 还将一些简化符号用于常用的目录功能。这些约定使得用户更容易理解自己在整个文件系统中的位置。

- 用一个圆点(.)表示当前目录。如果希望指定当前目录中的某个文件(例如，它是一个可执行文件，但是不在 PATH 中)，可以使用 ./myfile 来引用它。
- 类似地，用一对圆点(..)来表示当前目录的父目录。如果当前目录是/bin，执行命令 cd ..，将返回到根目录。
- 用符号(~)表示用户的主目录。不管用户位于文件系统中的哪个位置，命令 ls ~都将产生一个文件列表，这些文件存储在用户主目录中。

5.3 选择 shell

虽然许多 Unix 系统可能看起来都一样，但是事实上存在着细微的差别，这些差别将影响最终用户使用系统的方式。一种最基本的配置是选择 shell 环境(shell environment)。shell 是一个程序，它位于用户和操作系统的内核之间。当用户执行命令或者在提示符后进行输入时，就会与 shell 进行交互。接着，shell 将用户的命令和按键转化成内核能够理解的内容。内核做出响应，然后 shell 将输出提交给用户。

典型的 Unix 系统会默认安装几个 shell，这些 shell 对用户都是可用的。一般说来，shell 选择关系到个人的喜好。当系统管理员创建一个新的用户账户时，为该账户分配一个默认的 shell。如果无法接受默认的 shell，用户可以在稍后对其进行修改，但是绝大多数用户都会一直使用默认的 shell，而不管它是不是最适合自己的环境。如果您是自己的系统管理员，那么选择 shell 很简单，只需要在创建自己的账户时设置所希望的 shell，就像第 3 章中所描述的那样。

用户为什么会希望修改自己的 shell 环境呢？或许是因为不喜欢某个特定 shell 中的提示符风格。用户可能具有使用某种编程语言的丰富经验，希望选择一个以类似的方式组织命令的环境。用户也可能着迷于 shell 脚本编程的自动控制能力，希望选择一个 shell 能使这个过程尽可能的容易。用户可以修改 shell，就像换鞋一样，可以使用一个 shell 来做某种工作而使用另外一个来实现不同的目的。修改 shell 非常容易，无论是临时修改还是永久修改。

5.3.1 临时修改 shell

为单个用户会话，或者甚至只为两三条的命令修改 shell 环境是很容易的，然后返回到正常选择的 shell。要实现这一点，只需要简单地将所需 shell 的名字作为命令执行就可以。例如，如果默认的 shell 是 bash，而希望使用 tcsh，就可以简单地在 shell 提示符后输入：

```
tcsh
```

然而，如果存储 tcsh 的目录不是 PATH 环境变量值的一部分，就需要使用完整路径名，通常是：

```
/bin/tcsh
```

就这么简单！当用户准备返回到自己的默认 shell(本例中是 bash)中时，只需要在 shell 提示符后输入：

```
exit
```

要体验新的 shell 或者为了特定的目的在 shell 之间进行切换，这是最好的方式。即使在新的 shell 环境中退出系统，再次登录时仍将使用默认 shell。

惟一的不足之处在于，每一次用户希望在新的会话中使用新 shell 时都必须执行正确的命令。如果将 bash 设置为默认 shell，却发现大部分工作都是在 tcsh 中完成的，那么最好修改默认的 shell 环境，下一节介绍了如何实现这一点。但是，对那些在一个会话中使用多个 shell 的用户而言，只在需要的时候调用特定的 shell 是一种比较快的管理环境的方法。

5.3.2 修改默认的 shell

如果用户有一个偏爱的 shell，而它却不是用户账户上的默认 shell，那么用户有两种选择：使用前一节介绍的方法，每次登录都修改 shell，或者永久修改默认的 shell。对后者而言，用户修改控制 shell 选择的变量的值，从而在每次登录时都会调用新的 shell。这种状态将一直有效，直到用户再次修改默认的 shell 时为止。

要修改默认的 shell，使用 chsh，它是修改 shell 的命令。在命令提示符后输入：

```
chsh
```

系统将会提示用户输入口令和新 shell 的名称。输入这些信息后，就将新的 shell 设置为默认的 shell。要在新旧版本之间切换 shell，还需要输入 shell 名作为一个单独的命令。

chsh 并不是在每个系统上都有效。可以使用 -e 或 -s 标记来尝试使用 passwd 程序，即 passwd -e 或者 passwd -s。如果这两种调用也失效，那么需要联系系统管理员以寻求帮助。

5.3.3 各种 shell

虽然大多数用户都明白必须有一个 shell 环境，但是很多人不知道该如何选择。有许多种 Unix shell，有一些使用得很广泛，而有一些只对它们的创建者有意义，只有它们的创建者才会使用它们。每一种 shell 都有自己的优缺点，这使得用户难以确定哪种 shell 最适用于自己的特殊环境。

不用紧张！大多数 Unix 用户都只会使用少数几种 shell，并将其用于不同的目的。把用得最顺手的 shell 作为主 shell，但是用户可能希望使用不同的 shell 来编程，甚至使用一个专门的 shell 来玩游戏或者进行其他的娱乐活动。本节将讨论最流行的 Unix shell，介绍它们的基础知识以及用户希望尝试每个 shell 的原因。

1. Bourne Shell

无论使用哪一个 Unix 版本，都可能在系统中的某个目录下找到 Bourne shell 的一个版本。Bourne 是最早的 Unix shell，这么多年以来，它几乎没有发生变化。事实上，Web 上最流行的 Bourne 手册是 1978 年最初为 Bourne 4.3 版所编写的手册。最初的 Bourne 文档和当前的 Bourne 文档之间的惟一主要区别是较新的文件是 HTML 格式。

可以在站点 <http://steve-parker.org/sh/bourne.shtml> 上找到这个手册，它是由 Steve Bourne 亲自编写的。

根据每个人的观点，用户可能认为 Bourne shell 是一个优雅且可靠的环境，也可能认为它在方便性和易用性上存在着严重的缺陷。无论用户在日常工作中是否选择使用 Bourne，都应该花费一定的时间来熟悉它的基础知识。当机器中可用的资源有限时——例如只能使用急救盘——Bourne 可能是惟一可用的 shell 环境。因此，在出现这种危机之前，最好知道如何使用它。

许多系统管理员喜欢将 Bourne 用于 shell 脚本编程。跟所有的 shell 一样，Bourne 既是用户环境也是命令语言，要自动化大量常用的管理例程而无需构建复杂的程序并学习新的编程语言，shell 脚本编程是一种简单的方法。很多书籍以及网络上都介绍了大量的 Bourne shell 脚本编程的内容。虽然具有更多特性的新 shell 使得用户与系统的交互更容易，但是 Bourne 依然在

shell 脚本编程领域内占有一席之地。

使用 `sh` 命令来调用 Bourne。默认的 Bourne 提示符以 `$` 符号结束，其后是游标；如果作为根用户登录，那么将看到 `#` 而不是 `$`。Bourne shell 启动时解析两个配置文件：

- `/etc/profile` —— 一个全局配置文件，适用于 Bourne 家族的 shell。
- `.profile` —— 存储在用户自己的主目录下的一个文件，含有用户希望的特殊配置，以指定自己账户中的 shell 行为。这个文件也适用于 Bourne 家族的 shell。

在本章后面的“配置 shell”一节中将学习如何使用这些文件。

2. ash

有些系统可能运行 `ash` shell 以取代正宗的 Bourne。`ash` 最初是由 NetBSD 开发小组创建的，它是一个小尺寸的 Bourne 克隆版本，所占用的磁盘和内存空间比 Bourne 少很多。它对于内存较少或者硬盘较小的机器特别有用。虽然它缺少一些标准的 Bourne 性能，例如最近所执行命令的历史记录文件，但是它是一个功能完整的 shell 环境。如果正在运行一个只有干骨架式的 (bare-bones) Unix box，它没有大量的空间来安装各种软件，特别是如果正在运行 BSD 版本时，就可以考虑安装 `ash`。如果所使用的操作系统中没有包含 `ash`，则可以从许多联机软件库中下载 `ash` 程序包或者源代码。

3. Bourne Again Shell

尽管 Bourne shell 几乎存在于世界上的每一个 Unix 系统中，但是它不一定是硬盘上最友好的 shell。启动 `bash`，它是一个类似 Bourne 的 shell 而不是 Bourne 的克隆版本。`bash` shell 最初是为 GNU 操作系统编写的 shell 环境，GNU 操作系统是一个只含有免版税代码的类似 Unix 操作系统(因此从技术上讲，不能将它称为一个 Unix 版本——事实上，GNU 代表“GNU's Not Unix”)。为了满足 GNU 项目的哲学要求，开发人员构建了一个模拟 `sh` 的脚本编程功能的 shell，即 Bourne Again Shell，或者 `bash`。

起初人们打算将 `bash` 作为 Bourne 的一个简易替代，但是 `bash` 经过随后几年的持续发展，目前已经具有了许多特性——从用户界面工具到更加先进的脚本编程语言——这在最初的 Bourne shell 中是不存在的。

大多数 Bourne shell 脚本都可以不用改动直接在 `bash` 下运行。然而，由于 `bash` 自带的语言做过修改，因此反过来并不总是可以。要想在纯粹的 Bourne 环境中运行 `bash` 脚本，必须对它做一些改动。

`bash` shell 极为流行，在那些最初通过 Bourne shell 学习脚本编程、但是希望使用更好的用户界面工具的用户中尤其如此，并且它已经移植到了几乎所有的 Unix 版本中。用户的 Unix 机器中很可能含有 `bash` shell，甚至可能就是默认的 shell 环境。即使不希望进行太多的 shell 脚本编程，`bash` shell 也值得一试。这个 shell 的众多特性对任何水平的用户都具有相当大的吸引力：

- `bash` 提供环境变量以配置用户使用 shell 的每个方面。
- 命令历史记录使得用户能够回滚到前面的命令，这便于使用重复的命令，也便于确定问题的根源。
- `bash` 提供内置的算术功能，包括命令行上的整数算术命令和用于 shell 脚本编程的算术表达式。
- 通配符表达式帮助用户查找程序的准确名称，或者帮助查看系统上所有文件名中含有特定字符串的文件。还可以将通配符与命令一起使用，以便在多个文件上执行某个操作。

- 命令行编辑使得用户能够使用 Emacs 和 vi(文本编辑器)命令在提示符后编辑文件，而不需要打开这个文件。
- Bash 具有一组内置命令，它们大大简化了机器的管理，其中许多都可以在命令行和脚本中使用(请参考联机帮助页以获得更多的内容，在命令行提示符后执行 `man bash` 命令可以找到它的联机帮助页)。

4. Korn Shell

Korn shell 是 Bourne shell 家族的另一个成员，用 `ksh` 命令来调用它。这个 shell 也很流行，其受欢迎程度不亚于 `bash`。但是，两个 shell 之间的界线有点模糊，因为 `bash` 结合了 Korn shell 的许多特性，包括算术功能和可配置成类似于 Emacs 或 vi 的命令行编辑功能。

如果打算进行大量的 shell 编程，Korn shell 或许是个不错的选择。它有选择地包含了一组很好的编程特性，包括为 shell 脚本构建菜单的能力以及数组变量(array variable)的使用。数组变量是在一个变量中通过索引引用的多个值。高级算术运算和数组的结合意味着 Korn shell 的用户能够创建功能非常强大的 shell 脚本。事实上，最新版本的 `ksh` 所具有的功能在许多流行的编程语言中也能找到，如 Tcl、Icon、AWK 和 Perl，但是 `ksh` 具有更友好的界面。

默认安装时，机器上可能没有 Korn shell。可以从 AT&T 公司的网站下载最新的版本，网址是 <http://research.att.com/sw/download>。可以从 <http://kornshell.com> 获得 FAQ 和安装帮助，还有示例脚本以及与 Tcl 或 Motif 脚本一起使用的扩充知识。

对有些用户来说，程序的开放源码特性是选择软件的一个关键因素。尽管 AT&T 公司现在发布了开发源码的 Korn shell，但是它以前并不总是开放源码的。如果用户愿意使用 `ksh` 的克隆版本(这种版本是一个真正开放源码的 shell 环境)，那么可以考虑 `pdksh`，即公有域(Public Domain)Korn shell。遗憾的是，由于 `pdksh` 是一个无偿的计划，它没有与 Korn shell 一样的开发进度表，因此相对于目前的 `ksh` 版本，现在的 `pdksh` 没有 `ksh` 那样丰富的特性。尽管如此，如果用户确实想使用一个百分之百开放源码的机器，不妨试一下 `pdksh`。可以从站点 <http://web.cs.mun.ca/~michael/pdksh/> 上获得更多内容。

5. Z Shell

Z shell(`zsh`)出现得比较晚，大约是 20 世纪 90 年代初期才出现的。然而，在计算机科学领域内，14 年是一个相当长的时间，`zsh` 已经是一个成熟的 shell 环境，目前发展到 4.2.0 版本。在所有的与 Bourne 有关的 shell 中，Z shell 最像 Korn shell。事实上，Korn 的用户会发现转移到 Z shell 之后不会有任何不同。据说，Z shell 与几乎所有目前主要的 shell 环境都有相似之处。配置选项多得让人无从选择，如果愿意投入时间，用户或许能够让 Z shell 以自己希望的任何特殊的方式进行工作。

尽管 Z shell 可以完成 C shell 中的很多工作，但是要知道它是在 Bourne 基础上构建的，因此趋向于 sh 的语法和过程。

Z shell 具有各种各样的特性和选项。如果某个地方的一个用户打算在 shell 中做点事情，那么他很可能是在 Z shell 下。显然，Z shell 的优点是它能够做很多事情。它的缺点在用户身上一——用户可能不希望使用具有如此多选项和功能的 shell。可以将 `zsh` 看作个无所不能的汽车、缝纫机或者是在市场上呼声甚高的 PDA。如果每天花费百分之九十的时间赶 10 公里同样的路程去上班，您不会需要任何奇特的越野功能。然而，如果花费百分之九十的时间将孩子带到乡下游玩，您可能会希望尽可能地取消汽车的各种限制。即使只想进行越野旅行，也可能希望汽车

具有尽可能高的功率。Z shell 就是这样的一个工具。

根据操作系统软件包的年代不同，系统中可能没有包含最新版本的 Z shell，在 2004 年 8 月发布了 4.2 版。可以访问 <http://zsh.org> 并选择最方便的镜像站点。所有的镜像都提供最新的 Z shell 软件包，以及文档和重要联机文档的链接，例如 FAQ 和用户指南。

6. C Shell

您精通 C 语言吗？是否需要一个理解 C 风格命令的 shell 环境？那么 C shell——csh 和 tcsh——可能是合适的 shell。在初级用户阶段，C shell 与那些和 Bourne shell 有关的 shell 之间没有太大的差别。但是，一旦开始操作环境变量和 shell 脚本编程，很快就会看出两个 shell 之间的差别。

C shell 中使用的编程语法派生于 C 语言。毫不意外，C 的拥护者通常也是 C shell 的拥护者。如果习惯于使用 C 语法，那么这些 shell 就会很简单，尤其是在编写 shell 脚本时。如果对 C 语言不熟悉，那么当 C 语法不同于传统的 sh 风格的命令时，用户可能会非常困惑。例如，基于 Bourne 的 shell 使用下面的语法定义一个环境变量：

```
VARIABLE = value
```

而在 C shell 中，使用下面的命令来定义环境变量：

```
setenv VARIABLE value
```

因此，Bourne 风格的命令 `EDITOR="pico"` 在 C shell 中可能会表示成 `setenv EDITOR pico`。

我们不止一次地收到过用户的求助信，他们不能让自己的 shell 正常地工作。大多数情况下，这都是因为用户不知何故进入了另外一个家族的 shell，也许是通过调用 shell 或者是由于系统管理员指定了默认软件。如果试图执行 Bourne 风格的命令但是不能执行，那么用户很可能是在一个 C 类型的 shell 中。反过来也一样；如果试图使用 setenv 设置一个变量但是没有成功，那么用户可能处在 Bourne 类型的环境下。要进入希望使用的 shell，可以通过适当的命令来调用它，如果系统上不存在这个 shell，可以请系统管理员安装相应的 shell 软件包。

基本的 C shell，csh，具有多种基本功能，包括档名完成(filename completion)、通配符替换和一些有用的管理工具，如工作控制和历史替换。tcsh 是 csh 的一个增强版本，提供了更多的灵活性。在 tcsh 中，用户能够找到可配置的命令行编辑、自动补齐(command completion)、极好的目录解析方法以及许多该 shell 特有的环境变量。在 tcsh 中，还可以使用键盘上的方向键在命令历史中前后移动，这个功能在许多比较现代的 shell 中很普通，但是在比较老的 shell 环境中是默认不可用的。

Unix 的默认安装中可能带有其中一个或全部两个的 C shell。如果需要下载或更新 C shell，可以访问相应发布版本的下载站点以获得 C 软件包和源代码。还可以登录 <http://tcsh.org> 查找 tcsh 软件包和文档。

7. Perl shell

由于 Perl 是一种非常流行的编程语言，因此一些技术出色的 Perl 拥护者试图构建 Perl shell。

这项工作正在进行当中，但目前还没有开发出能够替代上述任何一种 shell 的可靠的 shell。然而，如果喜欢使用 Perl，那么可能会希望自己的机器上拥有基于 Perl 的 shell，以便利用 Perl 的解释器和正则表达式规则。有两种正在积极开发中的基于 Perl 的 shell：

- psh——Perl shell psh 目前的版本是 1.8。可以从站点 <http://gregorpurdy.com/gregor/psh/> 上下载最新的版本。
- Zoidberg——一个模块化的基于 Perl 的 shell，还处于试验阶段。目前它还没有推出 1.0 版本，所以是真正的 beta 软件。但是，将 Perl 模块的便利性与正规的 shell 环境相结合，这一概念很具有诱惑力。可以从站点 <http://zoidberg.student.utwente.nl/index.plp> 上了解更多的相关内容。

8. 可用的 shell

并不是每个系统上都安装了所有的 shell。大多数系统通常至少具有下列 Bourne shell 版本中的一个：真正的 Bourne shell(sh)、GNU 的 Bourne Again Shell(bash)或者一般的公有域 Bourne 克隆版本(ash)。无论在机器上安装哪一个 Bourne 版本，都可能是链接到名称 sh，可以将这个名称作为命令来调用一个工作原理类似 Bourne 的 shell 环境。

另外，许多 Unix 的默认安装中还包含某些风格的 C shell：要么是 csh(最早的 C shell)，要么是 tcsh(GNU 开发的一个具有相似工作原理的 C shell)。搜索/bin 目录和其他的类似目录以查看是否有其他 shell，如 Korn shell ksh。甚至还可能发现更为少见的 shell，如 Z shell 和 Perl shell。在许多比较老的系统上，用户不可能找到这些 shell，但是可以下载软件包来尝试使用一下。

9. 用于娱乐的 shell

希望在 shell 环境中消磨一些时间吗？可以考虑 game shells 的 odd subculture，这些 shell 是为了模拟流行的基于文本的 Unix 游戏，例如探险游戏或者多用户地牢游戏(MUD)而编写的。这些 shell 并不是为实际的工作设计的，但是常用的 Unix 命令将触发一些游戏角色的特定动作。

为模拟探险游戏，人们编写了 3 种 shell：advshell、asvsh 和 nadvsh。advshell 是最早的一个，它是一个 sh 脚本。advsh 是用 C 语言编写的，而 nadvsh(New Adventure Shell)是为 bash、ksh 和 zsh 编写的。这些 shell 不需要额外的库，一般都能良好地运行。要获得更多信息，可以查看相关主页：

- advshell 和 advsh: <http://ifarchive.org/if-archive/shells/>
- nadvsh: <http://nadvsh.sourceforge.net/>

或许您称不上是一个探险游戏的玩家，或者您已经花费了太多时间来玩自己喜欢的打打杀杀的 MUD 游戏以学习新的技巧。Mud-shell 可能正是您所需要的！这个 shell 是用 Perl 编写的，实际上可以将其理解为一个正规的 shell 环境，虽然处在其中会感到非常刺激。可以从站点 <http://xirium.com/tech/mud-shell> 获得更多信息。

5.4 配置 shell

选择一个 shell 环境之后，可能希望对它进行配置。shell 配置有 4 个主要元素，可以使用 4 者的任意组合，以使得自己的环境最为理想：

- 运行控制文件
- 环境变量
- 别名
- 选项

在这一节中，将学习更多关于这些元素的内容，并给出一些示例，演示它们的用法。

5.4.1 运行控制文件

只要启动 shell 就会执行运行控制文件，无论是登录到自己的账户时还是在提示符后执行命令以调用 shell 时。当 shell 启动时，它解析所有可应用的运行控制文件。shell 所检查的第一个运行控制文件是一个全局配置文件。取决于 shell 的不同，可能会使用不止一个全局配置文件，其中一个文件定义全局设置，一个控制登录 shell，还可能有另外一个定义子 shell 进程。当完成全局配置文件的解析之后，shell 接着解析存储在用户账户中的任何已有的个人配置文件。与全局配置一样，可能具有不止一层个人配置文件。

表 5-2 显示了用于流行的 Unix 版本的各种运行控制文件。该表中的文件既有全局运行控制文件也有个人运行控制文件。

表 5-2

shell	配 置 文 件	文件的用途
bash	/etc/bashrc	全局配置
	/etc/profile	用于登录 shell 的全局配置
	~/.bash_profile	用于登录 shell 的用户个人配置
	~/.bashrc	用于所有子 shell 的用户个人配置
	~/.profile	用户的个人配置，用于所有的登录 shell。如果 ~/.bash_profile 不存在，就读取这个文件
sh&ash	/etc/profile	全局配置
	~/.profile	用户的个人配置
ksh&pdksh	/etc/profile	全局配置
	~/.profile	用户的个人配置
zsh	/etc/zshenv	第一个全局配置文件
	/etc/zprofile	全局的登录 shell 配置
	/etc/zshrc	第二个全局配置文件
	/etc/zlogin	全局的登录 shell 配置
	/etc/zlogout	全局的清理文件
	\$ZDOTDIR/.zshenv	用户的第一个个人配置文件
	\$ZDOTDIR/.zprofile	用户的个人登录 shell 配置
	\$ZDOTDIR/.zshrc	用户的第二个个人配置文件
	\$ZDOTDIR/.zlogin	用户的个人登录 shell 配置
	\$ZDOTDIR/.zlogout	用户的个人清除文件

(续表)

shell	配置文件	文件的用途
psh	~/.pshrc	用户的配置文件(在 Perl shell 下没有全局配置文件)
csh&tcsh	/etc/csh..cshrc	全局配置文件
	~/.csh.login	用户的个人登录配置

Unix 的不同版本调用 shell 初始化文件的方式是不同的。本文中的示例与用户系统中的方法不一定完全匹配。如果遇到问题，请参考本地的 shell 文档。

对 Z shell 感兴趣的读者请仔细阅读文档。\$ZDOTDIR 是一个变量，它控制用户个人文件的位置。如果没有定义该变量，就使用用户的主目录。

1. 全局配置文件

在初始登录之后，当 shell 环境开始运行时，如果默认的 shell 需要配置文件，那么它所解析的第一个配置文件是全局配置文件。/etc/profile 是 Bourne 家族 shell 的一个配置文件，它包含全局设置，这些设置会影响机器上的每个账户。这个文件通常控制文件的导出方式、默认的终端类型、以及通知特定用户有新电子邮件的消息。/etc/profile 还用于创建 umask，即默认的权限设置，在创建每个文件的时候应用于新文件。

在一个配置好的系统上，只有根用户能够修改/etc/profile。请记住，该文件中的每一个设置都影响 Bourne 家族 shell 的所有用户，并且该文件是在特定用户的个人首选项文件之前调用的。最好严格控制/etc/profile 文件中的内容，从而使登录不需要太多的时间。/etc/profile 文件在各种 Unix 版本下的运行惊人地相似，即使它包含个人配置。下面给出一些来自不同的 Unix 风格的/etc/profile 示例，以便说明其中的相同和不同之处。

ash 的/etc/profile

ash 的/etc/profile 文件很基础，没有定义许多全局设置：

```
PATH=" /bin: /usr/bin: /sbin: /usr/sbin: /usr/local/bin: /usr/XHR6/bin"
exec 'set -o vi'
ulimit -c 0
if [ 'id -gn' = 'id -un' -a 'id -u' -gt 14 ]; then
umask 002
else
umask 022
fi
USER='id -un'
PS1="#"
LOGNAME=$USER
HISTSIZE=1000
HISTFILE="$HOME/.history"
EDITOR=mp
INPUTRC=/etc/inputrc
TERM=linux
NNTPSERVER="news.comcast.net"
# GS_FONTPATH="/usr/X11R6/lib/X11/fonts/Type1"
```

```
eXport PATH PS1 USER LOGNAME HISTSIZE INPUTRC EDITOR TERM NNTPSERVER
```

请注意 PATH 中的由冒号分隔的目录值，它们只包含常用的二进制目录。

Red Hat Linux 的/etc/profile

下面的/etc/profile 来自 Red Hat Linux 机器，它将 bash 作为默认 shell。

```
# /etc/profile

# System wide environment and startup programs
# Functions and aliases go in /etc/bashrc

if ! echo $PATH | /bin/grep -q "/usr/XHR6/bin" ; then
    PATH="$PATH:/usr/X11R6/bin"
fi

ulimit -S -c 1000000 > /dev/null 2>&1
if [ 'id -gn' = "id -un" -a 'id -u -gt 14 ] ; then
    umask 002
else
    umask 022
fi

USER='id -un'
LOGNAME=$USER
MAIL="/var/spool/mail/$USER"

HOSTNAME'/bin/hostname'
HISTSIZE=1000

if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ] ; then
    INPUTRC=/etc/inputrc
fi

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC

for i in /etc/profile.d/*.sh ; do
    if [ -x $i ] ; then
        . $i
    fi
done

unset I
```

从第 4 行可以看出，别名和其他功能存放在/etc/bashrc 文件中，这是 bash 所特有的另一个全局配置文件。这个文件的结构类似于一个 shell 脚本，带有包含 if-then 编程结构的内容。但是，它仍然是一个相对简单的全局配置文件。

Unix 中通用的/etc/profile

如果在许多 Unix 版本上工作，并且不喜欢记住每种风格的命令行环境之间的差别，可以在 <http://bluehaze.com.au/unix/eprofile> 上查看通用的/etc/profile。它创建了一个通用的命令行环境，这个环境在每一个主要的商业 Unix 和 Linux(如 Solaris、SunOS、HP-UX、Linux 和 Irix) 中都能使用。这一点令人印象非常深刻！

/etc/profile 文件本身长达 9 页，因此本书不包含它的内容。然而，即使只是为了研究这个文件的思想，也应该好好查看一下它。它的注释非常完备，每一个代码块都有准确的解释，几乎没有无关的内容。简而言之，这个文件对/etc/profile 及其众多的用途都是一个极好的说明。请仔细查看该文件。

/etc/bashrc

如果决定运行 bash(Bourne Again SHell)，可能会希望配置一个附加的全局运行控制文件，即/etc/bashrc(或者/etc/bash.bashrc，取决于所用的 Unix 版本)。和/etc/profile 一样，/etc/bashrc 定义 bash 用户会话的独有特征。它控制系统上所有 bash shell 用户的默认配置，因此应该设置权限，只允许管理员编辑该文件。

取决于正在讨论的 Unix 风格和已在/etc/profile 文件中设置的控制信息，/etc/bashrc 文件将会有很大的差别。许多管理员将主要的配置存储在/etc/profile 中，而保持/etc/bashrc 文件的简短性，只在其中保存 bash 特有的内容。例如，下面是来自 BSD 安装的/etc/bashrc 文件：

```
# System-wide .bashrc file for interactive bash(1) shells.
PS1='\h:\w \u\$ '
# Make bash check it's window size after a process completes
shopt -s checkwinsize
```

这个文件只有两个功能：定义 bash 提示符和检查窗口大小。系统上关于 shell 环境的其他配置都由/etc/profile 文件控制，因此，无论使用哪个 shell，都会用到这些控制信息。

在 Red Hat Linux 系统上，/etc/bashrc 文件看起来稍微有些不同：

```
# /etc/bashrc
# System wide functions and aliases
# Environment stuff goes in /etc/profile

# by default, we want this to get set.
# Even for non-interactive, non-login shells.
if [ "'id -gn'" = "'id -un'" -a 'id -u' -gt 99 ]; then
    umask 002
else
    umask 022
fi

# Here's where I ask you to put the "umask 022" line to override previous

# are we an interactive shell?
if [ "$PS1" ]; then
    if [ -x /usr/bin/tput ]; then
        if [ "x'tput kbs'" != "x°" ]; then # We can't do this with "dumb" terminal
            stty erase 'tput kbs'
```

```

        elif [ -x /usr/bin/wc ]; then
            if [ "'tput kbs|wc -c '" -gt 0 ]; then # We can't do this with "dumb"
terminal
                stty erase 'tput kbs'
            fi
        fi
    fi
case $TERM in
    xterm*)
        if [ -e /etc/sysconfig/bash-prompt-xterm ]; then
            PROMPT_COMMAND=/etc/sysconfig/bash-prompt-xterm
        else
            PROMPT_COMMAND='echo -ne
"\033]0;${USER}@${HOSTNAME%%.*}:${PWD/$HOME/~}\007"'
        fi
        ;;
    screen)
        prompt_command='echo -ne
\033_${USER}@${HOSTNAME%%.*}:${PWD/$HOME/~}\033\'
        ;;
    *)
        [ -e /etc/sysconfig/bash-prompt-default ] &&
            PROMPT_COMMAND=/etc/sysconfig/bash-proxnpt-default
        ;;
esac
[ "$ps1" = "\\s-\\v\\\$ " ] && PS1="[\\u@\\h \\W]\\\$ "

if [ "x$SHLVL" != "x1" ]; then # We're not a login shell
    for i in /etc/profile.d/*.sh; do
        if [ -r "$i" ]; then
            . $i
        fi
    done
fi
fi
# vim:ts=4:sw=4

```

与/etc/bashrc 的第一个版本一样，这个文件只用于 bash 特有的功能，并且这个文件包含了注释行，指引读者在/etc/profile 文件中查看通用的环境设置。特别地，该文件主要关注这个特定 shell 中的环境类型，以便为用户提供适当的提示符和功能。对于交互式的和非交互式的(就是说，把它作为一个哑终端来调用)shell，这个文件的响应是不同的。

2. 个人运行控制文件

shell 在触发之后还会解析个人运行控制文件。但是，并不是每个用户都拥有个人运行控制文件，这个文件可以为用户的 shell 环境定义额外的配置。例如，如果要查找某个在本节开始处的表 5-2 中定义的个人运行控制文件，但是没有发现该文件，那么这个文件可能不存在。这些文件只有在需要用到时才创建。个人运行控制文件的常用设置中包括 PATH 变量中的更多记录项、首选终端设置以及颜色、字体大小、别名等等。

5.4.2 环境变量

正如本章开头所说明的那样，环境变量能够用于配置所特定 shell 的行为的几乎每一个方面。无论是手动配置为数众多的变量还是使用图形配置工具来定义变量，用户都可能拥有很多与自己的账户有关的变量设置。

可以使用 `set` 命令查看在系统上任意 shell 中所定义的变量。只需在命令提示符后输入 `set` 并按下 `Enter` 键，其输出将列出系统上当前定义的所有变量及其相应的值。如果某个特定的变量没有在输出结果中列出来，那只是因为还没有对其进行定义。给这个变量赋一个值，则下次运行 `set` 时，该变量就会出现在输出结果中。

`set` 命令的输出显示了在当前 shell 中定义的所有环境变量的值：来自 `/etc/profile` 文件的值、来自个人 `~/.profile` 配置文件中的值、用户和其他用户手动定义的所有变量以及程序运行中定义的所有变量。

为了说明 Unix shell 所使用的不同类型的变量，下面的内容显示了在 3 种 shell 中执行 `set` 命令的结果，这 3 种 shell 分别是：`tcsh`、`bash` 和 `zsh`。观察每个命令的输出，可以看出这些 shell 之间的差异以及它们在环境变量中的配置重点。

1. tcsh 环境变量

从输出结果可以看出，下面的 `tcsh` 会话运行在一个 Mac OS X 机器上。Mac OS X 是基于 FreeBSD 的，在 Terminal 窗口中获得的 Unix 体验足以让您感到惊奇！这里为 `tcsh` 设置的变量非常有限，`PATH` 的值很短，提示符很常用，而且输出中没有显示很多不常见的变量。

```
% set
addsuffix
argv      ()
cwd        /Users/joe
dirstack   /Users/joe
echo_style bsd
edit
gid        20
group      staff
history    100
home       /Users/joe
killring   30
loginsh
owd
path       (/bin /sbin /usr/bin /usr/sbin)
prompt     [%m:%c3] %n%#
prompt2    %R?
prompt3    CORRECT>%R (y|n|e|a)?
promptchars %#
shell      /bin/tcsh
shlvl      1
status     0
```

```
tcsh      6.12.00
term      xterm-color
tty       ttypl
uid       501
vuser     joe
Version tcsh 6.12.00 (Astron) 2002-07-23 (powerpc-apple-darwin)
options 8b,nls,dl,al,kan,sm,rh,color,dspm,filec
```

在 tcsh 下也可以使用 `setenv` 命令定义环境变量。

2. bash 环境变量

下面的内容是在相同的机器上为 `bash` 设置的一些环境变量。请注意这个输出比较长，说明在这个安装中，`bash` 具有更多特有的变量(对于这些示例所显示的 `tcsh` 和 `bash` 的输出，`/etc/profile` 文件都是一样的)。

```
$ set
BASH=/bin/bash
BASH_VERSINFO=([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="powerpc-apple-darwin7.0")
BASH_VERSION='2.05b.0(1)-release'
COLUMNS=80
DIRSTACK=()
EUID=501
GROUP=staff
GROUPS=()
HISTFILE=/Users/joe/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/Users/joe
HOST=Joseph-Merlinos-Computer.local
HOSTNAME=Joseph-Merlinos-Computer.local
HOSTTYPE=powermac
IFS=$' \t\n'
LINES=24
LOGNAME=joe
MACHTYPE=powerpc
MAILCHECK=60
```

3. zsh 环境变量

最后，在一台 Red Hat Linux 机器上定义的 Z shell 环境变量是：

```
$ set
BASH=/bin/bash
BASH_ENV=/home/joe/.bashrc
BASH_VERSINFO=([0]="2" [1]="04" [2]="21" [3]="1" [4]="release" [5]="i386-redhatlinux-gnu")
BASH_VERSION='2.04.21(1)-release'
COLORS=/etc/DIR_COLORS
COLUMNS=80
DIRSTACK=()
EUID=501
```

```

GROUPS=()
HISTFILE=/home/joe/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/home/Joe
HOSTNAME=surimi.nigiri.org
HOSTTYPE=i386
IFS='
'
INPUTRC=/etc/inputrc
LANG=en_US
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LINES=24

```

完整的 Z shell 输出比 tcsh 或 bash 的输出要长得多。就像前面所提到的，Z shell 是一个特性繁多的 shell 环境，可配置性非常强。其灵活性表现在所列变量的长度上。

5.4.3 别名

定制工作环境的一个很好的方法是使用别名(alias)。别名只不过是使用一个术语取代另一个术语的一种方式。毫无意外，创建别名使用的命令是 `alias`。

例如，如果经常出现输入错误，那么用户可能会希望在删除文件之前能获得一个确认提示信息。使用命令 `rm -i` 能够做到这一点。但是，如果只使用 `rm` 命令而没有 `-i` 标记，那么可能会删除需要保存的文件，可以创建别名来避免这个问题。将 `rm` 命令作为字符串 `rm-i` 的别名。这样，通过输入通常的 `rm` 命令就可以调用新功能了。

要创建一个别名，可以使用如下格式：

```
alias rm = "rm -i"
```

一般的语法是：

```
ailas command = string
```

如果从命令行来实现这个功能，那么别名只能在当前的 shell 只使用。也就是说，如果启动一个新的 shell 或者退出原来的 shell，就将失去这个别名。要让别名始终可用，需要将命令放在用户的个人 shell 配置文件中。

C shell 版本中的别名功能与 Bourne 版本中的不同。例如，在 C shell 中，上述的别名命令将变成 `alias rm 'rm -i'`，这里使用单引号而去掉了等号。如果在设置别名时遇到困难，可以参考 shell 文档。

5.4.4 选项

并不是所有的 shell 功能都需要始终运行。如果机器的内存有限，可能会希望关闭一些可选的 shell 元素以加快系统运行的速度。不同的 shell 具有不同的选项，但是通过执行带有附加选项的 shell 命令可以调用所有的选项，其格式为：

```
shellname -option
```

表 5-3 有选择地列出了几个流行的 shell 中的一些选项。通过参考 shell 的联机帮助页(通过在命令提示符后输入 **man shellname** 可以获得), 用户可以了解到更多与所选 shell 的特定的可用选项有关的内容。

表 5-3

shell	选 项	功 能
bash	-norc	在 shell 启动或者用户登录时, 将不会读取.bashrc 文件
	-rcfile <i>filename</i>	在启动时, 读取这个指定的文件而不是.bashrc
	-nolineediting	在这个会话过程中, 命令行编辑功能将不会起作用
	-posix	只有符合 POSIX 标准的功能才会在这个会话过程中运行
csh/tcsh	-e	如果命令出错, 就退出 shell
	-f	启动 shell 而不读取.cshrc 或.tcshrc
	-s	只接受来自标准输入的命令
	-t	执行单个命令, 然后退出 shell
ksh	-n	解析命令(读取并检查语法错误), 但是不执行
	-f	禁用文件名匹配替换(或称为“globbing”)
	-r	在受限模式下启动 shell, 这种模式不允许切换工作目录、输出重定向或者修改基本的变量值
	-C	防止使用重定向操作符>覆盖已有的文件
zsh	-c	为 zsh 命令设置一个参数, 这个参数作为执行的第一条命令, 而不是等到该 shell 启动之后再输入第一条命令
	-v	冗长的选项(在大多数 shell 中都可以找到), 在处理过程中, 将命令打印到标准输出上

5.5 动态共享库路径

作为一个程序员, 总是需要使用一些标准库, 它们是若干年前由其他人构建的。这些库包含标准函数, 程序员在设计自己的代码时, 可以使用这些已经成为标准的函数。在大多数 Unix 系统上, 这些库是基本安装的一部分。事实上, 许多 Unix 系统本身的代码也依赖于库函数。

由于这种情况, 用户必须能够定位已经安装在系统上的库, 并且让自己的库对那些以其他账户运行的程序也是可用的。在基于 GNU 的系统上存在一些配置文件, 这些文件中含有一组目录, 可以从中搜索需要的库, 其功能类似于 PATH 变量, 只是指定的目录仅与库有关。绝大多数情况下(只要是在 GNU/Linux 系统上), 这些文件位于/etc/ld.so.conf。而在 FreeBSD 系统上, 它们可能在/var/run/ld.so.hints 或/var/run/ld-elf.so/hints 下。

虽然 Apple 公司的 Mac OS X 系统是基于 FreeBSD 的, 但是它并不使用这些文件。

5.5.1 LD_LIBRARY_PATH

如果用户希望在一个非标准的位置存放库文件，那么 LD_LIBRARY_PATH 是最适合使用的环境变量。它的工作原理非常类似于 PATH，其值是一组用冒号分隔的目录。在读取各种配置文件之前，先计算它的值；如果在多个目录下具有同名的库，那么 LD_LIBRARY_PATH 中的目录的优先级将高于配置文件中指定目录的优先级。

LD_LIBRARY_PATH 变量并不是在所有的 Unix 系统上都有效。它工作在起源于 GNU 的系统上以及 FreeBSD 上，但是在 HP-UX 上，这个功能是由 SHLIB_PATH 变量来实现的。在 AIX 上，对应的变量是 LIBPATH。

许多评论员认为 LD_LIBRARY_PATH 变量及其类似的变量是有害的，至少大多数情况是这样。其原因通常是考虑到安全性问题，并且，如果处理不当，LD_LIBRARY_PATH 确实会引起混淆。在决定是否使用这个变量之前，如果想了解这个争论的大概内容，可以访问 www.visi.com/~barr/ldpath.html。

尽管存在合理的安全性考虑，但是这个变量仍然有其正当的用途。使用 LD_LIBRARY_PATH 的最一般的原因是测试。假定用户正在处理一个叫做 myprog 的程序，这个程序使用 libmyprog 库。用户拥有一个正在使用的 mylib 版本，它安装在 /usr/lib/libmyprog.so.3.，但是用户希望使用库的新版本来测试程序。用户可能将新库安装在 /usr/lib/test/libmyprog.so.3.，并且令 LD_LIBRARY_PATH 的值中含有 /usr/lib/test/。这样，当用户运行程序时，将加载库的新版本而不是旧版本。完成测试之后，可以将该变量重置为空值，为库分配一个新的版本号，并使用链接到新版的指令重新编译程序。

5.5.2 LD_DEBUG

另一个有用的环境变量是 LD_DEBUG。这个变量使得 C 加载器打印出冗长的信息，这些信息显示 C 加载器正在执行的操作。这个变量只有几个可选的值，通过把变量的值设置为 help 然后运行一个程序，就能够查看这些值。程序员会发现 files 和 libs 选项特别有用，因为它们显示了在调试过程中访问的是哪一个库。例如：

```
$ export LD_DEBUG=help
$ ls
Valid options for the LD_DEBUG environment variable are:
```

bindings	display information about symbol binding
files	display processing of files and libraries
help	display this help message and exit
libs	display library search paths
reloc	display relocation processing
statistics	display relocation statistics
symbols	display symbol table processing
versions	display version dependencies

To direct the debugging output into a file instead of standard output a filename can be specified using the LD_DEBUG_OUTPUT environment variable.

```
$
```


5.6 小结

在安装完一个 Unix 版本并且一切都步入正轨之后，就需要配置安装以适应自己的需要。Unix 操作系统具有难以置信的灵活性，并且用户使用系统的每一个方面几乎都是可以配置的。一个基本的用户配置涉及到选择 shell 环境，它担任用户输入和操作系统内核之间的翻译工作。选择一个 shell，然后使用多个工具对其进行配置以符合自己的习惯。

在配置用户环境以满足自己的喜好这一过程中，其基本步骤是：

- 查看系统中已经安装的 shell，并决定一个 shell 是否够用或者是否需要下载并安装新的软件包。
- 阅读新 shell 的文档。shell 通常都会带有非常详尽的文档资料，从基本的联机帮助页到当地计算机书店中厚厚的书籍，应有尽有。通常情况下，初学阶段使用联机帮助页和一些 Web 搜索就已足够，但是，如果对详细的配置或 shell 脚本编程感兴趣，可以深入阅读本书或其他针对所选 shell 的书籍。
- 查看在安装 shell 时默认建立的配置文件。/etc/profile 文件是否满足个性化要求？请记住，要编辑全局配置文件，需要作为根用户登录。
- 配置提示符，使它能够提供信息并易于理解。

5.7 练习

- (1) 编辑 shell 提示符以显示日期和用户 ID。
- (2) 已经在机器上安装了一个新程序，但是在输入命令时，这个程序并没有运行。最可能的问题是什么？
- (3) 假定要永久保留在练习 1 中所做的修改。应该如何实现？

第 6 章 深入 Unix 命令

回忆一下您用过的第一台计算机。是 Macintosh 吗？是运行某个 Windows 版本的 PC 机吗？如果开始使用计算机的时候，图形界面已经普及，那么您可能会觉得 Unix 有点令人迷惑。然而，如果能够记得曾经在 Commodore 64 或 Amiga 机器上编写过小型 BASIC 程序，那么您可能会认为 Unix 是一个受欢迎的回退，回退到显示器只显示命令提示符的日子里。

很久以前，不存在计算机桌面、窗口、图标或鼠标这类东西，只有基于文本的终端和命令行。事实上，Unix 是第一个独立于机器的操作系统——在 Unix 之前，每台机器都需要它自己的个性化操作系统以处理其特有的硬件机制。可以想象，在大规模生产计算机之前的日子里，开发一个公共的操作系统有多么困难！自 Unix 早期以来，它就主要使用命令行界面：一个简单的提示符，其后跟着游标，只使用文本。在这种环境中，只有一种模式可以与机器进行交互，就是通过命令。

当然，现在 Unix 系统提供了图形界面。在有些 Unix 版本下，图形界面是默认的用户环境。但是，在 Unix 机器上，始终都能以纯文本的模式进行工作。事实上，如果机器出现故障或损伤，那么纯文本环境可能是惟一可用的环境。

命令是可执行的程序。它们有时候是单独的程序，而有时候是 shell 的内置功能。无论哪种情况，命令都使得用户能够要求机器执行某种操作。这些操作可能很简单，比如列出目录的内容，也可能很复杂，例如运行一长串脚本，这些脚本设置机器运行时的基本参数。虽然在本书的前几章中已经使用过一些命令，但是本章将研究 Unix 命令的概念，并介绍经常会用到的一些基本管理命令。本章还会介绍组合命令的一些方法，从而使得输出更加复杂和有趣。在后面的一些章节中，可能会再次遇到某些命令，这样就有更多的机会试用它们。

6.1 命令的剖析

Unix 命令可以分成两部分：命令本身和追加给它的参数。大多数命令名与该命令所调用的操作在字面上是一样的。例如，ls 命令用于列出目录的内容。如果在命令提示符后输入 ls，那么工作目录(就是用户当前所在的目录)的内容将显示在屏幕上：

```
$ ls
Game Mail resume.txt
```

在命令行后单独执行命令时，所发生的一切称为命令的默认行为。然而，命令实际上比默认行为所暗示的要强大得多。通过使用参数，用户能够影响命令的行为或输出。参数添加了额外的信息，这些信息将修改命令执行的方式。考虑如下命令：

```
ls /etc
```

这里，命令本身是 `ls`。附加的 `/etc` 部分是一个参数，它修改命令的目录，从而使该命令显示 `/etc` 目录的内容。在 Linux 系统上，其输出类似于下面的内容：

```
$ ls /etc
a2ps.cfg          gshadow          makedev.d        rc6.d
a2ps-site.cfg     gshadow-         man.config        rc.d
adjtime           gtk              mc.global         re.local
alchemist         host.conf        mesa.conf         rc.sysinit
aliases           HOSTNAME         mime-magic        redhat-release
aliases.db        hosts            mime-magic.dat    resolv.conf
anacrontab        hosts.allow      mime.types        rmt
at.deny           hosts.deny       minicom.users     rpc
auto.master       hotplug          modules.conf      rpm
auto.misc         identd.conf      modules.conf      samba
bashrc            im               modules.devfs     screenrc
CORBA             im_palette.pal   motd              securetty
cron.d            im_palette-small.pal mtab              security
cron.daily        im_palette-tiny.pal nmh               sendmail.cf
cron.hourly       imrc             nscd.conf         sendmail.cf.rpmsave
cron.monthly      info-dir         nsswitch.conf     services
crontab           init.d           openldap          sgml
cron.weekly       initlog.conf     opt               shadow
csh.cshrc         inittab          pam.d             shadow-
csh.login         inputrc          paper.config      shells
default           ioctl.save       passwd            skel
devfsd.conf       iproute2         passwd-           smrsh
dhcpc             isdn             pbm2ppa.conf      snmp
dhcpcd           issue            pine.conf         sound
DIR_COLORS       issue.net        pine.conf.fixed   ssh
dumpdates        krb5.conf        pnm2ppa.conf      sudoers
esd.conf          ldap.conf        ppp              sysconfig
exports          ld.so.cache      printcap          sysctl.conf
fdprm            ld.so.conf       printcap.local    syslog.conf
filesystems      lilo.conf        profile           termcap
fstab            localtime        profile.d         updatedb.conf
fstab.REVOKE     login.defs       protocols         vfontcap
ftpaccess        logrotate.conf   pwdb.conf         wgetrc
ftpconversions   logrotate.d      rc                X11
ftpgroups        lpd.conf         rc0.d             xinetd.conf
ftphosts         lpd.perms        rc1.d             xinetd.d
ftpusers         ltrace.conf      rc2.d             yp.conf
gnome            lynx.cfg         rc3.d             ypserv.conf
gpm-root.conf    mail             rc4.d
group            mailcap          rc5.d
croup-           mail.re
```

利用这个参数，用户 `ned` 能够查看 `/etc` 的内容而不需要离开主目录(注意代码块第一行中的命令提示符，它指出了工作目录)。

第 5 章解释了在命令提示符中显示工作目录的方法。

这类参数称为目标(target)，因为它们为命令提供了所要处理的目标位置。还有其他类型的参数，称为选项、开关或标记。特定的命令通常具有自己独特的参数，关于这些参数的信息，可以在该命令的联机帮助页中找到。要了解更多关于联机帮助页的内容，可以参阅本章的下一节，“查找命令的相关信息”，或者是本书的第2章。

参数能够影响命令的输出格式以及它的操作。例如，如果将上述命令改写成：

```
ls -l /etc
```

那么 ls 命令将以长格式来显示/etc 的目录列表，这种格式将提供所列文件的其他相关信息。下面是 ls -l/etc 命令的部分输出：

```
$ ls -l /etc
total 1780
-rw-r--r-- 1 root root 15221 Feb 28 2001 a2ps.cfg
-rw-r--r-- 1 root root 2561 Feb 28 2001 a2ps-site.cfg
-rw-r--r-- 1 root root 47 Dec 28 2001 adjtime
drwxr-xr-x 4 root root 4096 Oct 1 2001 alchemist
-rw-r--r-- 1 root root 1048 Mar 3 2001 aliases
-rw-r--r-- 1 root root 12288 Sep 8 2003 aliases.db
-rw-r--r-- 1 root root 370 Apr 3 2001 anacrontab
-rw----- 1 root root 1 Apr 4 2001 at.deny
-rw-r--r-- 1 root root 210 Mar 3 2001 auto.master
-rw-r--r-- 1 root root 574 Mar 3 2001 auto.misc
-rw-r--r-- 1 root root 823 Feb 28 2001 bashrc
drwxr-xr-x 3 root root 4096 Apr 7 2001 CORBA
drwxr-xr-x 2 root root 4096 Mar 8 2001 cron.d
drwxr-xr-x 2 root root 4096 Oct 1 2001 cron.daily
drwxr-xr-x 2 root root 4096 Oct 1 2001 cron.hourly
drwxr-xr-x 2 root root 4096 Oct 1 2001 cron.monthly
-rw-r--r-- 1 root root 255 Feb 27 2001 crontab
drwxr-xr-x 2 root root 4096 Oct 1 2001 cron.weekly
-rw-r--r-- 1 root root 380 Jul 25 2000 csh.cshrc
-rw-r--r-- 1 root root 517 Mar 27 2001 csh.login
drwxr-x--- 2 root root 4096 Oct 1 2001 default
```

注意参数-l 提供了哪些附加信息？这个详细的输出显示了许多有用的数据，包括每个文件的权限、所有者和创建日期。

标记通常——但是不一定——位于一个破折号或者连字号之后。例如 tar 命令，它的标记前就没有破折号。可以同时使用多个标记，关于多个标记的排列方式，通常有一定的灵活性。需要注意的是，不要使用太多的参数，以免输出难以理解。

除了同时接受多个参数以外，有些命令要求多个目标。考虑 cp 命令，该命令产生一个特定文件的副本。要使用这个命令，必须定义所要复制的文件以及副本文件。例如，命令

```
cp /etc/profile /home/joe/peofile
```

创建/etc/profile 文件的一个副本，称为/home/joe/profile。在这种格式中，/etc/profile 称为源文件(source)，/home/joe/profile 称为目标文件(destination)。并不是所有的命令都要求一个源文件和一个目标文件，在有些程序中，省去其中一个可能导致命令假定一个源文件或者目标文件。在使用一个命令之前，必须了解该命令的默认行为。

尝试这个练习以了解 Unix 命令语法的灵活性。使用前面介绍的 ls 命令，添加 -a 选项以便在目录列表中包含隐藏文件；隐藏文件是那些文件名以点号开头的文件，例如 .bashrc。

(1) 在命令提示符后执行命令：

```
ls -l -a /etc
```

(2) 执行命令：

```
ls -la /etc
```

(3) 比较两个命令的输出。它们的结果是一样的。

工作原理

无论一个命令的参数是合并的还是独立添加的，该命令都产生相同的输出。

6.2 查找命令的相关信息

由于 Unix 命令非常复杂，最好在机器上安装一个信息源以帮助自己解决命令语法和参数的难题。Unix 提供了几种方法来访问命令的相关信息，从简单地指定相关文件到冗长的信息文件。这一节中，将学习最重要的 3 个命令，以了解机器上与程序相关的更多信息。

如果机器上没有安装专门的联机帮助页，可以在站点 <http://unixhelp.ed.ac.uk/alphabetical> 上的集合中找到它。

6.2.1 man

正如第 2 章中所学到的，Unix 联机帮助页是了解任何特定命令的最佳方法。使用命令 `man commandname` 可以找到特定的联机帮助页。联机帮助页可能比较简短，例如：

```
$ man apropos
apropos(1)                                apropos(1)

NAME
    apropos - search the whatis database for strings

SYNOPSIS
    apropos keyword ...

DESCRIPTION
    apropos searches a set of database files containing short
    descriptions of system commands for keywords and displays
    the result on the standard output.

SEE ALSO
    whatis(1), man(1).
```

Jan 15, 1991

1

(END)

其他的联机帮助页可能会一屏接一屏地显示。但是无论联机帮助页有多长，它们都遵循本例所显示的基本格式。联机帮助页提供了名称、函数、语法以及用于特定命令的可选参数。

6.2.2 info

有些程序员在他们的软件包或者程序中包含一组额外的帮助文档，称为信息帮助页(info page)。可以使用 `info` 命令来访问这些帮助页，例如：

```
$ info info
File: info.info, Node: Top, Next: Getting Started, Up: (dir)

Info: An Introduction
*****

Info is a program for reading documentation, which you are using now.

To learn how to use Info, type the command 'h'. It brings you to a
programmed instruction sequence.

* Menu:

* Getting Started::          Getting started using an Info reader.
* Advanced Info::           Advanced commands within Info.
* Creating an Info File::    How to make your own Info file.

---zz-Info: (info.info.gz)Top, 16 lines--All-----
Welcome to Info version 4.0. Type C-h for help, m for menu item.
```

如果系统上没有安装信息帮助页，那么将不能使用 `info` 命令。

通常在 GNU 软件或者自由软件基金会所开发的软件中能够找到信息帮助页。几乎所有的程序员都会为他们的程序创建基本的联机帮助页，但是并不是每个程序员都会创建更详细的信息帮助页。虽然如此，仍然可以试一试 `info` 命令，看看希望了解的程序是否存在信息帮助页。

要知道 `info` 通常是调用 Emacs 文本编辑器来显示信息帮助页的。如果对 Emacs 不熟悉，那么在页面中定位信息时，可能会感到相当困惑。可以输入 `C x C c` 以退出 Emacs。如果这个命令没有用，可以按 `Q` 键以退出信息浏览器。

6.2.3 apropos

当用户认为已经输入了正确的命令名，但是所要查找的特定文件或联机帮助页并没有出现时，应该如何找到它们呢？`apropos` 命令或许能帮上忙。该命令使用关键字来查找相关文件。它的语法很简单——只需要执行如下命令：

```
apropos keyword
```

下面是另一个示例：

```
$ apropos gzip
gunzip [gzip]      (1) - compress or expand files
gzip               (1) - compress or expand files
zcat [gzip]        (1) - compress or expand files
zforce             (1) - force a '.gz' extension on all gzip files
```

`apropos` 输出没有说明文件在文件系统中的位置，但是在开始搜索特定的软件包或命令之

前，确认它们是否已经安装在机器上是一个不错的方法。要找到其中某个软件包，可以使用 `whereis` 命令，例如 `whereis gunzip`。将会看到类似于下面的输出：

```
$ whereis gunzip
/usr/gunzip  /usr/bin/gunzip  /usr/share/man/man1/gunzip.1.gz
```

如果 `apropos` 命令不能工作，则可能需要执行命令 `catman -w`。当这个命令完成之后，可以试着再次使用 `apropos` 命令。

6.3 命令的修改

Unix 命令并不局限于在提示符之后输入命令名时所执行的功能。可以使用许多不同的工具来增强或改变命令的功能，或者使用这些工具管理命令的输出。本节将研究一些最流行的修改 Unix 命令的方法。

6.3.1 元字符

使用命令行界面的一个比较有趣的方面在于，能够使用特定的称为元字符(metacharacter)的字符来改变命令的行为。这些字符并不是命令本身的一部分，但它们是 shell 的特性，能够让用户创建复杂的操作。

这里所给出的语法都基于 Bourne 和起源于 Bourne 的 shell。本节描述的大多数特性在 Unix 中都是很基本的，但是在不同 shell 中的实现可能不一样。如果在使用元字符时遇到困难，可以参考 shell 的文档以确认正确的语法。

最流行的元字符称为通配符。它们是专用字符，能够用于同时匹配多个文件，从而增大一次性找到想要的文件名或目标的可能性。有 3 种最常用的通配符：

- `?` 匹配文件名中的任何一个字符。
- `*` 匹配文件名中的一个或多个字符。
- `[]` 匹配包含在 `[]` 符号内的某个字符。

可以在一个命令中综合使用这些通配符以定位多个文件，或者在不记得文件的全名时找到该文件。例如，假定工作目录中含有文件：

```
date help1 help2 help3 myprog.f myprog.o
```

可以向 `ls` 命令的目标参数中添加通配符，以便在单次搜索中找到其中的任意文件或者所有这些文件。表 6-1 给出了可能显示的各种文件组合，这取决于所用的通配符。

表 6-1

参数+通配符	所匹配的文件
help?	help1 help2 help3
myprog.[fo]	myprog.f myprog.o
*	date help1 help2 help3 myprog.f myprog.o
*.f	myprog.f
help*	help1 help2 help3

当查找某个特定类型的文件时，通配符也很有用。许多工作在异构环境(即，具有 Unix、Windows 和 Macintosh 机器的网络)中的用户在网络上的多台机器之间传输由 Microsoft Office 创建的文件。要使用通配符来指定工作目录中的所有 Word 文件，可以使用如下命令：

```
ls *.doc
```

工作在异构环境中的其他人在所有平台上的各种文本编辑器和字处理器中都使用纯文本文件作为易读的公共语言。如果工作目录中的所有文本文件都使用后缀 .txt 来标记，那么可以使用下面的命令列出这些文件：

```
ls *.txt
```

如果通常使用 .txt 和 .text 作为文本文件的后缀，那么通配符的作用就更大了。执行如下命令：

```
ls *.t[ex]*
```

其输出将列出目录中的所有文本文件，而不论是使用哪个后缀来命名文件的。

如果通配符搜索中的第一个和最后一个字符都使用星号，那么将得到比期望更多的结果，因为这个搜索还将定位临时文件和系统文件。如果得到的结果过多，以至于不能恰当地处理它们，那么可以精化通配符并重新执行搜索。

元字符是正则表达式的一个重要方面。在第 8 章中将学到更多关于该内容的知识。

6.3.2 输入和输出重定向

为了执行，每个命令都需要一个输入源和一个输出目的地。将这些属性编入命令中，作为默认行为，称为命令的标准输入和标准输出。在绝大多数情况下，标准输入是键盘，标准输出是屏幕——特别地，如果正在使用图形界面，则是 Terminal 窗口。

在有些情况下，标准输入和输出要单独定义。如果自己的机器是这种情况，用户很可能已经知道应如何处理。如果坐在一台不熟悉的机器面前，而且它没有键盘或者没有按照您的期望，将输出显示到显示器上，那么可以找到机器的管理员并请教输入输出设置。用户可能正试图在一台非常重要的机器上工作，这台机器已经为每个人设置了输入和/或输出设备，以防止不速之客破坏机器中的数据。

尽管永久修改标准输入和输出是可能的，然而这通常不是一个好主意。但是，用户可以为单独的操作修改输入和输出。这称为重定向(redirection)，它是一个很好的方法，能够将一系列任务以流线型的方式运行。利用重定向，用户可以迫使某个特定的命令从源文件而不是键盘提取输入，或者将输出放到显示器以外的某个地方。例如，用户可能希望设置一个程序，不需要从键盘调用就可以运行它，然后将它的输出转储到一个文本文件，这样在空闲时就可以仔细阅读该文件的内容。

输入和输出重定向使用字符<和>来定义临时输入和输出源。假定用户希望使用 ls 命令列出某个目录的内容，但是希望将输出存放到文本文件中而不是将其打印到屏幕上。要这样做，可以创建一个名为 lsoutput 的文件，然后执行如下命令：

```
ls > lsoutput
```

ls 命令的输出通常会显示到屏幕上，但是字符>提取 ls 的输出，并将它写到 lsoutput 文件中(如果指定的文件不存在，那么>运算符将创建该文件)。

在重定向输出时需要非常小心。例如，如果上述命令执行了两次，那么第二个命令将覆盖 lsoutput 文件的内容，并破坏先前的数据。如果希望保存先前的数据，则可以使用>>操作符，它将新数据追加到文件的末尾。如果指定的文件不存在或者为空，那么>>的操作与>一样。基于用户的输入技巧，养成使用>>的习惯可能会更安全。

与>重定向命令的输出一样，字符<用于修改输入。该功能通常用于构建一个命令链，从而将命令 A 的输出作为命令 B 的输入。它是一种自动化管理功能的好方法。例如，假定用户希望将文件 terms 中包含的一组文件按字母顺序排列，那么可以将 sort 命令和输入重定向操作符<结合在一起使用，如下所示：

```
sort < terms
```

本例中，sort 命令将从 terms 文件而不是标准输入中提取它的输入。

输入和输出重定向还可以结合在一起使用。例如，命令

```
sort <term> terms-alpha
```

将 terms 文件中的条目排序，然后将排序后的输出写入一个名为 terms-alpha 的文件中。可以看出，输入和输出重定向的复杂性取决于操作符的基本逻辑。

对程序员来说，输出重定向操作符是一个便利的工具，它能够用于重定向由命令产生的错误信息。通过向操作符中添加一个数字 2(在起源于 Bourne 的 shell 中)就可以做到这一点。

Bourne shell 使用 0、1 和 2 作为输出重定向的文件描述符。那些对 C 编程比较熟悉的读者应该知道，0 表示 stdin(standard input)，1 表示 stdout(standard output)，2 表示 stderr(standard error)。

例如，如果正在调试一个程序 myprog，并且希望查看程序运行时产生的错误信息，可以在工作目录内执行如下命令：

```
myprog 2> errfile
```

在命令提示符后输入程序名来运行该程序，然后在文本编辑器中打开 errfile 文件以查看 myprog 产生的错误信息。如果 errfile 不存在，这个程序就没有产生错误。

6.3.3 管道

管道(pipe)是一个操作符，它把输入和输出重定向结合在一起，从而将一个命令的输出立即作为另一个命令的输入。管道用垂直线字符(|)表示，该字符通常是一个移位字符，位于键盘上 Return 或 Enter 键附近的某个位置。假定用户希望列出某个目录的内容，但是目录的列表太长了，以至于许多记录项在能够看清楚之前就已经滚出了屏幕。管道提供了一种简单的方式，可以每次只显示一页输出，使用下面的命令实现：

```
$ ls -l /etc | more
total 1780
-rw-r--r-- 1 root root      15221 Feb 28 2001 a2ps.cfg
-rw-r--r-- 1 root root      2561 Feb 28 2001 a2ps-site.cfg
```

```

-rw-r--r-- 1 root root      47 Dec 28 2001 adjtime
drwxr-xr-x 4 root root    4096 Get  1 2001 alchemist
-rw-r--r-- 1 root root    1048 Mar  3 2001 aliases
-rw-r--r-- 1 root root  12288 Sep  8 2003 aliases.db
-rw-r--r-- 1 root root     370 Apr  3 2001 anacrontab
-rw----- 1 root root      1 Apr  4 2001 at.deny
-rw-r--r-- 1 root root     210 Mar  3 2001 auto.master
-rw-x--x-- 1 root root     574 Mar  3 2001 auto.misc
-rw-r--r-- 1 root root     823 Feb 28 2001 bashrc
drwxr-xr-x 3 root root    4096 Apr  7 2001 CORBA
drwxr-xr-x 2 root root    4096 Mar  8 2001 cron.d
drwxr-xr-x 2 root root    4096 Oct  1 2001 cron.daily
drwxr-xr-x 2 root root    4096 Oct  1 2001 cron.hourly
drwxr-xr-x 2 root root    4096 Oct  1 2001 cron.monthly
-rw-r--r-- 1 root root     255 Feb 27 2001 crontab
drwxr-xr-x 2 root root    4096 Oct  1 2001 cron.weekly
-rw-r--r-- 1 root root     380 Jul 25 2000 csh.cshrc
-rw-r--r-- 1 root root     517 Mar 27 2001 csh.login
drwxr-x--- 2 root root    4096 Oct  1 2001 default
--More--

```

这里显示的内容与本章前面所看到的 `ls -l /etc` 的输出是一样的,但是这次只显示了一屏的输出文件,因为输出通过管道发送给 `more` 命令,从而受控于 `more` 命令(在本章后面的“常用的文件操作命令”一节中将学到更多关于 `more` 命令的知识)。

管道、重定向和本节中的所有其他特性几乎可以无限制地组合在一起,以创建复杂的命令链。例如,命令

```
sort < terms > terms-alpha | mail fred
```

将执行前面描述的排序操作,然后将 `terms-alpha` 文件的内容邮寄给用户 `fred`。对这些组合的惟一限制是用户的独创性。

6.3.4 命令置换

命令置换也是一种将一个命令的输出作为另一个命令的参数的方法。相对于简单地将输出通过管道传给第二个命令,命令置换是一种更为复杂的操作,但是可以用它来创建比较复杂的命令字符串。考虑如下命令:

```
ls $(pwd)
```

这里,第一个运行的命令是 `pwd` 命令,它输出工作目录的名称。然后将它的值作为参数发送给 `ls` 命令。该命令将输出当前工作目录的目录列表。

细心的读者将会注意到,这个示例与单独的 `ls` 命令具有相同的效果,但是它是一个有用的示例,可以阐明命令置换的工作方式。

的确,这个命令与下面的命令具有相同的效果:

```
pwd | ls
```

输出是一样的，但是其幕后的执行方式有一些差别。使用\$运算符的结构比较特殊，因为圆括号内的命令将在子 shell 中执行——也就是说，系统将产生一个新的 shell 实例，计算命令的值，关闭子 shell 并将结果返回到最初的 shell 中。

如果设置了特殊的环境条件，这些条件可能不会转移到子 shell 中(例如手动设置的 PATH 值)，那么子 shell 可能不会继承这些条件。在这种情况下，命令将失效。

除了使用\$()结构以外，还可以使用反引号。例如

```
ls `pwd`
```

实现与 ls \$(pwd)完全相同的功能。另一种方法是使用花括号，如下所示：

```
ls ${pwd}
```

两者的不同之处在于花括号中的命令在当前 shell 中执行，而不需要产生子 shell 进程(花括号并不是在所有的 Unix 版本中都有效)。

6.4 操作文件和目录

最常用的 Unix 命令是那些管理文件和目录的命令。当用户处理自己的工作时，可能会在一天中数百次地执行其中的某个命令。本节将介绍两个最流行的文件管理命令以及与之相关的各种参数。

6.4.1 ls

本章的前面已经介绍了使用 ls 命令以列出目录的内容。ls 的输出可以比较简短，也可能极为冗长和复杂。两种结果都很有价值，这取决于用户所需的目录信息的信息类型。ls 命令的语法是：

```
ls [options] [directory]
```

如果没有指定目录，那么 ls 假定用户希望了解当前工作目录的内容。不过，用户能够使用 ls 命令获得系统上任意目录的信息，只要该目录的权限允许用户读取其中的内容(权限将在本章的后面讨论)。

ls 命令提供了许多参数，可以利用它们来定制输出以提供需要的信息。表 6-2 显示了一些最常用的选项。如果这些不是所需的参数，则可以使用 man ls 命令查阅 ls 的联机帮助页。

表 6-2

参 数	功 能
-l	用长格式列出目录的内容，显示每个文件的大小、权限和其他数据
-t	依据时间标记(上一次的修改时间)列出目录内容
-a	列出目录的所有内容，包括文件名以字符 . 开头的隐藏文件
-i	列出目录内容，包括 inode 或磁盘索引号
-R	列出目录内容，包括所有的子目录以及它们的内容

6.4.2 cd

cd 命令是经常用到的一个命令。它用于切换文件系统。其语法是：

```
cd directory
```

如果执行该命令而没有指定目录，cd 将自动切换到用户的主目录。当用户一直在文件系统中切换，并且在另一个目录结构中嵌套得很深时，这个命令尤其有用；只需要在命令提示符后输入 cd 就可以立即返回到熟悉的环境。

6.5 常用的文件操作命令

找到需要的文件后，就可以对其进行许多操作。本节将介绍几个用于操作单个文件的命令，不论它们是程序、文档或者是 Unix 操作系统视为文件的其他元素。

可用的命令取决于所用的 Unix 版本以及系统管理员所做的安装选择。

6.5.1 cat

如果用户希望查看特定文件的内容，那么 cat 命令是最容易的方法，它将该文件的内容打印到标准输出，通常是显示器。该命令的语法如下：

```
cat [options] file(s)
```

如果简单地执行 `cat filename` 的命令，那么输出内容将显示到屏幕上，若文件的内容超出屏幕的长度，将滚动显示。如果文件太长，不能够单屏查看，可以使用管道将输出传给 `more` 或 `less` 命令(下一节讨论)。表 6-3 显示了 cat 命令的许多选项。

表 6-3

参 数	功 能
-n	输出的行数
-E	在每一行的结尾显示一个\$字符
-s	将连续的空行合并成一个空行
-t	将非打印字符 tab 显示成^I
-v	显示所有的非打印字符

cat 命令的一个很有用的用法是将多个文件连接成一个更长的新文件，从而更易于同时读取这些文件的内容。可以使用下面的命令来实现：

```
cat file1 file2 file3 >> newfile
```

请注意这个命令中使用了重定向操作符>>。

6.5.2 more/less

more 和 less 命令实质上是一样的。它们用于将命令输出或文件内容分隔成可以单屏显示的部分，从而使用户能够更容易地阅读其中的内容。more 和 less 命令都能够在文件中向前移动，但是只有 less 可以用于后退。这两个命令具有相同的语法：


```
more filename
less filename
```

当用户看完当前屏幕中的输出之后，可以按空格键切换到下一屏。如果使用 less 命令查看输出或文件，则可以使用 B 键返回到上一屏。

在使用 more 或 less 命令时，如果已经发现所要寻找的信息，但是整个文件还没有滚动完，则可以简单地按下 Q 键。这样就将返回到命令提示符。

6.5.3 mv

mv 命令用于将文件从一个位置转移到另一个位置。它的语法是：

```
mv old new
```

其中 old 是当前的文件名，用于将这个文件转移到由 new 定义的位置。如果 new 的值只是一个新的文件名，那么重命名该文件并将它保留在当前目录中。如果 new 的值是一个新的目录位置，那么将文件转移到新位置，仍然使用现有的文件名。如果 old 的值是一个目录，那么整个目录和目录的内容都将转移到 new 所指定的位置。

如果 new 是已有的文件或目录，那么它的内容将被 old 的内容覆盖，这取决于系统设置。因此，在重用文件或目录名时一定要小心。

6.5.4 cp

与 mv 命令类似，cp 命令用于创建新文件或者将文件的内容转移到另一个位置。但是，与 mv 不同的是，cp 将源文件完好地保存在原始位置。cp 的语法是：

```
cp file1 file2
```

其中 file1 是源文件，file2 是目标文件。如果使用已有文件的名字作为目标文件的值，那么 cp 将用 file1 的内容覆盖这个已有文件的内容。

6.5.5 rm

rm 命令用于删除文件。它使用如下语法：

```
rm [options] filename
```

这个命令相当具有破坏性，执行该命令时一定要谨慎，尤其是使用通配符时。例如，命令 rm conf* 将删除所有以字符串 conf 开头的文件，无论用户是否希望删除这些文件。表 6-4 列出了 rm 常用的一些选项。

表 6-4

参 数	功 能
-i	使用交互模式，提示用户确认每个删除
-r	使用递归模式，删除所有子目录以及它们包含的文件
-f	使用强制模式，忽略所有警告(非常危险)

用户应该知道，将某些选项合并使用——尤其是-r和-f标记——是很危险的。如果从根目录执行命令 `rm -rf *.*` ，那么将删除文件系统中的每个文件，如果从主目录执行这个命令，则将删除主目录中的所有文件。千万不要执行这个操作。

6.5.6 touch

`touch` 命令用于更新指定文件中的时间标记。时间标记显示了上一次修改或访问文件的时间。该命令使用如下语法：

```
touch filename
```

如果作为参数的文件名不存在，那么 `touch` 将使用这个文件名创建一个空文件。

6.5.7 wc

使用 `wc` 命令来确定特定文件的长度。`wc` 的语法如下：

```
wc [options] filename
```

默认情况下，输出显示以单词为单位的长度。表 6-5 显示了 `wc` 的一些可用选项。

表 6-5

参 数	功 能
-c	显示指定文件中的字符(字节)数
-l	显示指定文件中的行数
-L	显示指定文件中的最长一行的长度

6.6 文件所有权和权限

Unix 的一个显著特征是，它从一开始就被设计成为多用户系统。相对地，其他操作系统只是近年来才创建了真正的单机上的多用户功能。由于 Unix 的多用户设计，它必须使用相应的机制，从而使得用户能够管理他们自己的文件，但是无权访问其他用户的文件。这些机制称为文件所有权和文件权限。

6.6.1 文件所有权

任何 Unix 用户都能够拥有文件。一般情况下，用户所拥有的文件是由用户自己创建的文件，或者是以用户的身份执行一些操作时所创建的文件。当然也有例外，那就是超级用户，也称为根。超级用户能够使用 `chown` 命令修改任何文件的所有权，无论他是否创建了该文件。例如，如果超级用户执行命令

```
chown jane /home/bill/billsfile
```

那么文件 `/home/bill/billsfile` 的所有权将转移给用户 `jane`。发生这种情况时，难道 Bill 不会感到吃惊吗？

1. 用户名与 UID

目前为止，我们已经熟悉了用户名的概念，就是登录 Unix 机器时所使用的名字。这个名字由系统管理员(或者是用户自己，如果用户是自己的系统管理员)分配给用户。除了用户名之外，每个用户还有一个用数字表示的 ID 号，称为用户 ID 或者 UID，系统通过它来识别用户。虽然在创建账户时可以指定该数字，但是通常情况下，这些数字都是自动分配的。这个数字本身是任意的，尽管许多系统都要求普通用户的 UID 数字要大于 500。

超级用户的 UID 总是为 0。

对于登录以外的其他用途而言，用户名和 UID 基本上是同义的。例如，如果 Jane 的 UID 是 503，那么可以简单地将命令

```
chown jane /home/bill/billsfile
```

看作是

```
chown 503 /home/bill/billsfile
```

这里假设 Jane 的 UID 是 503。对于一般的用途，用户不需要知道自己的 UID，但是有的场合会用到它。

2. 组

除了 UID 之外，用户还至少拥有一个组 ID，或者 GID。跟 UID 一样，操作系统使用 GID 而不是组名来管理组。每个用户至少属于一个组，也可以属于多个组。组包含的用户共享用于某项操作的某种权限。用户所从属组的组名可以与用户的用户名相同。超级用户可以将用户添加到其他组中，这取决于用户对某个文件或目录的访问需求。

组也可以拥有文件，并且可以将文件所有权从一个组转移到另一个组。要进行这样的操作，可以使用 `chgroup(change group)` 命令：

```
chgroup groupname filename
```

与 `chown` 一样，可以将组名替换为 GID。

6.6.2 文件权限

文件权限的概念与文件所有权的概念是相关的。作为文件的所有者，用户有权决定哪些用户可以访问文件，以及其他用户能够具有哪种访问权限。对 Unix 初学者而言，文件权限有点令人混淆，但是它们是维持机器的安全性和可靠性的重要角色，因此本节将回顾在第 4 章中介绍的有关权限的内容。

应该尽可能的限制权限。只要合法的用户能够以希望的方式使用文件，就不要授予他其他的访问权限。

有 3 种文件权限：

- 读取(能够查看文件)
- 写入(能够编辑文件)

- 执行(能够将文件作为程序执行)

同样地，有 3 类用户，能够将权限应用于这 3 类用户：

- 用户(特定文件的所有者)
- 组(文件所从属的组)
- 所有人(所有用户和组)

当用户为特定的文件定义权限时，必须为每类用户都分配一种权限。如果用户正在编写一个程序，可能希望授予自己读取、写入和执行权限。如果用户在一个团队中工作，就可能希望系统管理员为这个团队创建一个组。然后用户就可以给这个团队授予读取和执行权限，从而使他们能够测试该用户的程序并提出意见，但是不能编辑这个文件。由于这个团队没有写入权限，因此必须通过该用户才能进行修改。

如果用户打算创建一个组，从而为某个特定的用户集授予读取和执行权限，那么一定要确保所有不在该组中的用户没有同样的权限。

1. 读取权限列表的方法

如何得知文件的权限呢？很简单，只需要使用 `ls -l` 命令，然后权限信息就会作为目录列表的一部分显示出来。例如，下面是本章前面所显示的 `/etc` 目录列表的一部分(从左往右，各个列分别显示了文件权限、UID、用户名、组名、文件大小、时间标记和文件名)：

<code>-rw-r--r--</code>	<code>1</code>	<code>root</code>	<code>root</code>	<code>1048</code>	<code>MAR</code>	<code>3</code>	<code>2001</code>	<code>aliases</code>
<code>-rw-r--r--</code>	<code>1</code>	<code>root</code>	<code>root</code>	<code>12288</code>	<code>SEP</code>	<code>8</code>	<code>2003</code>	<code>aliases</code>
<code>-rw-r--r--</code>	<code>1</code>	<code>root</code>	<code>root</code>	<code>370</code>	<code>APR</code>	<code>3</code>	<code>2001</code>	<code>anacrontab</code>
<code>-rw-----</code>	<code>1</code>	<code>root</code>	<code>root</code>	<code>1</code>	<code>APR</code>	<code>4</code>	<code>2001</code>	<code>at.deny</code>

左边的字符串显示了每个文件的所有权限信息。权限信息由 9 个字符组成，从左边起第二个字符开始。前 3 个字符表示用户的权限，接下来的 3 个字符表示组的权限，最后 3 个字符表示所有其他人的权限。

`-rwxrwx---`

上面的表达式说明用户和组具有读取、写入和执行权限。字符串

`-rw-rw-r--`

说明用户和组具有读取和写入权限，而所有其他用户只具有读取权限。

2. 修改权限

文件的权限只能由文件的所有者或超级用户修改。使用 `chmod`(表示“change mode”)命令修改权限。`chmod` 有两种使用方法：符号模式或绝对模式。这些方法已经在第 4 章中介绍过。

6.6.3 umask

创建文件时，该文件具有默认的权限设置。在大多数系统上，用户对自己所创建的任何文件都具有读取和写入权限，而用户所在的组或其他用户没有权限。与 Unix 中的所有其他行为一样，该行为是可配置的。

默认的权限方案是由 `umask` 命令控制的。跟 `chmod` 一样，`umask` 将一个数值作为它的参数。

然而，与 `chmod` 不同的是，`umask` 指定的值表示的是拒绝(denied)的权限。也就是说，`umask` 授予所指定权限以外的权限。

为了更具体地进行说明，假定用户希望授予所有新创建文件的权限模式是 644，根据第 4 章中的介绍，就是授予所有者读取和写入权限，组和其他人具有读取权限。只要从 666 中减去这个数字，就得到 `umask` 的正确参数，这里 666 是文件默认的八进制基数：

```
umask 022
```

这个命令将导致所有新创建的文件都具有权限模式 644。`umask` 的作用只局限于调用该命令的 shell 内。要使这个作用持续有效，应将该命令保存到 `.profile` 文件中。

6.6.4 可执行文件

如前所述，程序是可执行的文件。这个概念看起来很浅显，但是要让文件可执行，用户真正需要做的是，将文件的执行权限授予用户希望能够运行这个程序的任何人。不要认为这很简单，经常有人在 Unix 机器上创建了一个程序，却不理解它为什么不运行。让文件具有执行权限是这个过程中的关键步骤。

首先，使用 `ls -l` 命令查看文件的当前权限。如果对该文件不具有执行权限，则可以使用 `chmod` 命令修改权限。但是，请记住，用户必须是文件的所有者才可以修改文件的权限。如果不是所有者，那么用户必须找到文件的所有者并请他执行 `chmod` 命令以授予自己权限。

如果具有超级用户的访问权限，那么可以强行修改权限，而不需要求助于文件的所有者。然而，这样做显然很不友好。如果必须做出修改，那么最好是告诉这些用户，您已经修改了他们的文件的权限。

如果仍然不能调用可执行文件，那么用户可能遇到了不熟悉 Unix 的程序员经常要面对的一个问题——错误的 `PATH` 值。第 5 章中已经介绍过 `PATH` 环境变量的概念。如果含有可执行文件的目录没有包含在 `PATH` 变量的值中，那么用户必须给出可执行文件的完整路径名才能运行该文件。如果用户希望除了自己之外，其他用户也可以使用该程序，一个比较好的办法是将可执行文件转移到全局 `PATH` 变量中的某个目录下，全局 `PATH` 变量由系统管理员定义。解决了 `PATH` 问题之后，用户只需要在提示符后输入程序名就可以运行该程序了。

6.7 保持文件系统配额

磁盘空间问题是一个经常困扰 Unix 管理员的问题。任何拥有计算机的人，无论他使用计算机的时间有多长，都应该知道文件会迅速增加，以至于耗尽所有的可用磁盘空间。不难猜想，在多用户系统上，这个问题出现的概率会增大许多倍。当系统上具有多个用户时，每个用户都倾向于占用更多空间，这样就会导致磁盘空间占用问题的畸形发展。应该如何处理这个问题呢？

幸运的是，在大多数 Unix 系统上都有内置的解决方法：磁盘配额。配额限制了分配给每个用户的磁盘空间总量。大多数 Unix 系统都具有某类可用的配额系统，但是它们在具体实现上存在差别。如果机器上默认的配额系统不够有效，用户就可以购买足够强大的商业软件以应对众多频繁进行文件交换的用户。

实现配额的第一步是为文件系统激活配额。在大多数系统上,这需要修改文件系统控制文件(通常是/etc/fstab)。不同的系统在该文件中使用不同的格式来激活配额。例如,在 Linux 下,/etc/fstab 文件可能类似于:

```

LABEL=/          /          ext3          defaults          1    1
LABEL=/boot      /boot      ext3          defaults          1    2
none            /dev/pts   devpts       gid=5,mode=620    0    0
LABEL=/home      /home      ext3          defaults,usrquota,grpquota 1 2
none            /proc      proc          defaults 0 0
none            /dev/shm   tmpfs        defaults 0 0
/dev/hda2        swap       swap         defaults 0 0
/dev/cdrom       /mnt/cdrom udf,iso9660  noauto,owner,kudzu,ro 0 0
/dev/fd0         /mnt/floppy auto         noauto,owner,kudzu 0 0

```

添加到/home 文件系统那一行中的 `usrquota` 和 `grpquota` 选项说明配额只对这些目录有效。不具有这些选项的目录没有激活配额。

其他 Unix 风格具有不同的格式,可以参考系统文档以了解正确的语法。

在为/etc/fstab 文件中适当的目录行添加了配额选项,或者根据系统需要完成了其他配置之后,必须重新安装文件系统。如果用户的系统是个人计算机,只有用户自己(或者少数几个人)在使用,那么可以简单地重启机器。如果是在一个比较大的网络中激活配额,那么用户可能需要手动卸载(`umount` 命令)然后重新安装(`mount` 命令)文件系统。

如果系统上有多个用户,那么您可能希望在一个通信量比较低的时间执行该操作,例如在深夜。您还需要把任何已经登录的用户踢出系统。

到目前为止,已经在给定的目录中激活了配额。要让配额开始工作,需要在文件系统的顶层目录下创建一个配额文件。该文件定义了配额必须遵循的限制。与/etc/fstab 文件一样,Unix 的不同风格使用不同的方法来实现这一点。在 Linux 系统下,可以使用 `quotacheck` 命令:

```
quotacheck -acug /home
```

这个命令执行 4 个单独的参数:

- `-a` —— 检查/etc/mtab 文件中所有已经安装的本地文件系统,查看是否激活了配额。
- `-c` —— 为每个激活配额的文件系统创建配额文件。
- `-u` —— 为用户配额执行 `quotacheck` 检查。
- `-g` —— 为组配额执行 `quotacheck` 检查。

在其他的系统下,用户可能必须手动创建配额文件。通常的创建过程类似于:

```

cd /home
touch quotas
chmod 600 quotas

```

这段程序创建了 `quotas` 文件,并规定只有超级用户才有写入权限。虽然任何用户都能够在他拥有权限的文件上执行 `chmod` 命令,但是只有超级用户才能够通过对文件执行 `chmod` 命令,使其具有根用户级别的权限。应该只允许根用户设置系统上的配额。

创建了配额文件之后,使用 `edquota` 命令设置配额。例如,要为用户 `jeff` 编辑配额,可以

执行如下命令：

```
edquota jeff
```

该命令为用户 jeff 打开了一个配额文件。在文本编辑器(关于文本编辑器的更多知识，请参阅第 7 章)中打开该文件，它类似于下面的内容：

```
Disk quotas for user jeff (uid 506)
Filesystem  blocks      soft      hard    inodes      soft      hard
/dev/hda3   440436          0          0    37418          0          0
```

block 列显示了 Jeff 的账户中的数据量，inodes 列显示了各个文件的 inode 值。将希望授予用户的软限制和硬限制来代替“soft”列和“hard”列下面的“0”。用户可以超出软(soft)限制，但是会收到警告；不能超出硬(hard)限制。退出编辑器，用户配额的设置就完成了。

一个块有 512 个字节的数据。在设置配额之前，一定要计算清楚，这样才能给用户分配正确的磁盘空间量。

6.8 小结

无论用户是否使用图形界面完成大部分的工作，总是能够从命令行管理 Unix 机器。命令调用程序来修改用户使用计算机的各个方面，使得用户能够操作文档和可执行代码，并且执行管理任务。本章介绍了几组命令：

- 查找信息的命令
- 以非标准的方法引导输入和输出的命令
- 帮助用户操作文件系统的命令
- 修改和移动已有文件的命令
- 定义和修改文件所有权和权限的命令
- 设置和执行磁盘配额的命令

6.9 练习

创建一个命令，列出/home 目录下以字母 k 开头的每个子目录的内容，计算找到的文件总数并将这个数字保存在文件 k-users-files 中。

第 7 章 用 vi 编辑文件

在 Unix 中，有许多编辑文件的方法，包括早期的基于行的文本编辑器，例如 ed(EDitor)和 ex(EXtended)，以及面向屏幕的文本编辑器 Emacs(Editor MACroS)和 vi(VIsual editor)。行编辑器只允许用户逐行编辑文件；屏幕编辑器使得用户能够结合上下文编辑多行文本。通常将 vi 看作 Unix 编辑器的标准，因为：

- 它通常在任何 Unix 系统中都是可用的。一旦学会了使用 vi 的方法，即使用户转移到另一个 Unix 系统(这个系统允许更快地恢复或调整文件)上工作，也不需要重新学习该编辑器的使用方法。
- 在所有的 Unix 版本上(从 Linux 到 Mac OS X 再到 Sun Solaris，等等)，vi 的实现都非常类似。
- 它需要的资源很少。
- 相对于 ed 或 ex，它具有更友好的用户界面。

vi 读作“vee eye”。有一个流行的笑话，是说一个新手(通常是不熟悉 Unix 或计算机的人)问一个比较有经验的 Unix 老手，Seven(VII)编辑器将什么时候问世，因为 Six(VI)编辑器似乎已经过时了。

几乎每一个使用 Unix 的人都将会遇到需要了解 vi 用法的时候，特别是在资源有限的情况下(没有提供图形用户界面[GUI]或者只有最少的可用资源)。使用 vi 对新用户来说是相当困难的，特别是对那些习惯于使用面向图形的字处理器的用户。vi 是一个纯文本编辑器，即使将它和一个文件格式化程序，如 LaTeX(<http://directory.fsf.org/text/doc/LaTeX.html>)结合在一起使用，它本身也不能产生各种奇妙的格式。

本章将向读者介绍使用 vi 的方法，从基本的用法到更高深的文本操作功能。

7.1 使用 vi

刚开始使用 vi 时，它看起来可能是一个冷冰冰的、没有什么帮助的环境，但是一旦用户熟悉了它的基本用法，就会发现它其实是很友好的。有几种启动 vi(“打开一个实例”)的方法，它们都是在控制窗口中输入命令，如表 7-1 所示。

表 7-1

命 令	说 明	结 果
vi	不使用文件名参数启动 vi	vi 启动一个空的面板。在退出之前，必须把所做的编辑保存到一个新文件中
vi filename	1. 使用已有文件名作为参数 2. 使用新文件名作为参数(直到在 vi 中保存了该文件之后，它才存在)	在 vi 中打开已有的文件。保存该文件时将更新文件中修改过的内容 保存这个文件时，将使用参数中指定的文件名来创建一个新文件

(续表)

命 令	说 明	结 果
vi -R filename 或 view filename	以只读模式打开文件	文件是只读的(不能保存修改); 这是在文件上练习使用 vi 命令的很好方法

图 7-1 显示了用命令 vi filename(vi testfile)启动编辑器的方法。



图 7-1

无论何时启动 vi——单独使用和已有文件名一起使用或者和新文件名一起使用——都以命令模式开始。图 7-2 显示了在 vi 中打开的一个新文件。

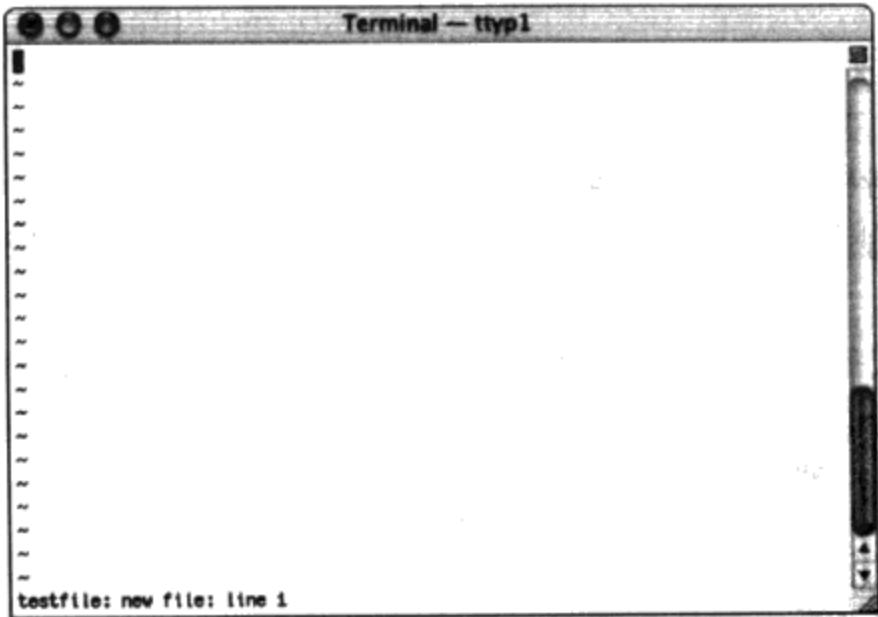


图 7-2

读者将会注意到，在游标之后的每一行上都有一个颚化符号(~)。颚化符号表示未使用的行(没有换行符或者任何类型的文本)。如果某行不是以颚化符号开始，并且仍然是空白的，那么一定存在空格、制表符、换行符或者其他一些不可见的字符。如图 7-2 所示，大多数 vi 版本几乎都没有什么信息(有些版本会提供最小限度的信息或提示，告诉用户接下来的操作)；惟一的信息位于屏幕底部的状态行内，该行显示了文件名和游标所在的行。与 Unix 中的许多内容一

样, vi 所提供的这条信息会随着系统的不同而变化。用户的 vi 实现中可能包含更多或者更少的信息。

图 7-3 显示了使用已有文件(/etc/services)作为参数启动 vi 的方法。



图 7-3

当用户试图用 vi 打开文件时, 如果收到拒绝访问错误或者只读指示符, 那么可能是因为该文件超出了用户的访问权限, 也可能是因为用户以只读模式启动了 vi。

对用户而言, 非文本文件, 如二进制文件在 vi 中会出现问题, 例如显示乱码——输出不可读。如果查看文件时获得意外的结果(例如不可读), 会简单地输入: q! 以退出文件编辑, 而不要对文件做任何改动。

vi 中有两种模式:

- 命令模式——使用户能够执行管理任务, 如保存文件、执行命令、移动游标、剪切(拖动)并粘贴多行或多个单词, 以及进行查找和替换。在这种模式下, 所输入的任何内容——甚至单个字符——都被解释成命令。
- 插入模式——使用户能够向文件中插入文本。在这种模式下键入的所有内容都被解释成输入。

vi 总是以命令模式启动。要输入文本, 必须进入插入模式。那么如何进入这种模式呢? 只需要简单地输入 i。还可以通过其他的方法进入插入模式, 将在本章后面进行介绍。要退出插入模式, 可以按 **Esc** 键, 然后用户就返回到命令模式了。

如果不能确定目前所处的模式, 可以按 **Esc** 键两次, 然后用户就会进入命令模式。事实上, 为了安全起见, 用户可以每次都按两下 **Esc** 键以进入命令模式。

7.2 在文件中移动

要在文件中移动而不影响文本内容, 用户必须处于命令模式下(按 **Esc** 两次)。最基本的是字符之间的移动。表 7-2 列出了一些命令, 用户可以使用这些命令一次移动一个字符。

表 7-2

命 令	结 果
k	上移一行
j	下移一行
h	左移一格
l	右移一格

用户可以使用大多数键盘上都有的标准方向键在文件中移动,但是建议用户了解表 7-2 中指定的键,因为并不是所有使用 vi 的平台上都带有方向键。

如果用户希望在文件中移动多个字符或者多行,可以将希望移动的数目放在移动键之前。例如,如果希望在文件中下移 10 行,则可以输入 **10j**(在命令模式下),然后光标将从当前行开始下移 10 行。下面是另一个示例;假定文件中存在如下文本行:

The quick brown fox jumped over the lazy dog.

如果光标位于该句中的第一个字符 T 处,输入 **10l**,光标将右移 10 个字符(包括空格)——到达 brown 中的 b 处。如果光标位于 fox 中的 f 上,输入 **6h**,光标也将移动到 brown 中的 b 处,向左移动 6 个字符。

vi 区别大小写,因此用户在使用命令时要特别注意大小写。

在 vi 中,还可以通过许多其他的方法在文件中移动。请记住,必须在命令模式下(按 **Esc** 两次)进行移动。表 7-3 列出了其中一些方法。

表 7-3

命 令	说 明	结 果
0	零	将光标放在一行的开始
\$	美元符号	将光标放在一行的结尾
w	小写 w	将光标移动到下一个单词(单词是一组由空格、tab 或标点符号分隔开的字符)
b	小写 b	将光标移动到前一个单词
(左圆括号	将光标放在当前句子的开始处(句子由后面跟两个空格的标点符号标识)
)	右圆括号	将光标放在下一句的开始处
Ctrl+F	同时按下 Ctrl 和 F 键	向前滚动一屏
Ctrl+B	同时按下 Ctrl 和 B 键	向后滚动一屏
G	大写 G	将光标放在文件的最后一行上(还可以使用 xG 来表示光标所要移动到的行号;例如,4G 表示将光标移动到第 4 行)
:x	冒号后跟一个数字(用数字来代替 x)	将光标放在 x 所指定的行上(例如,:4 表示将把光标移动到文件的第 4 行)

看一些使用这些键在文件中移动的示例。图 7-4 显示了一个在 Mac OS X 系统上的 vi 会话中打开的文件(/etc/syslog.conf)。

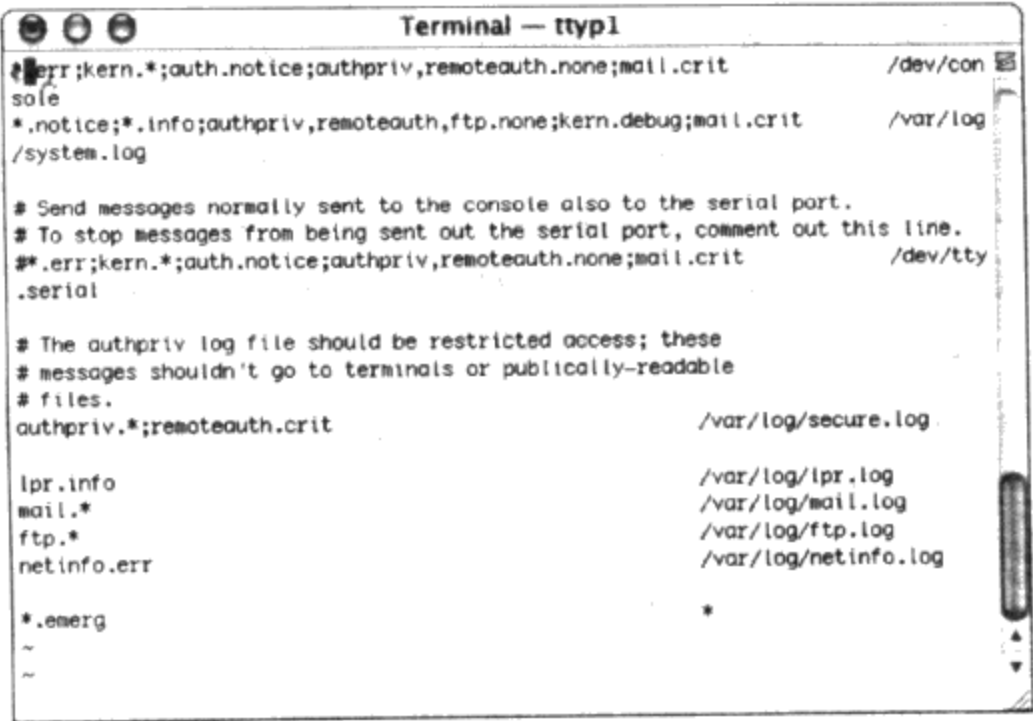


图 7-4

在/etc/syslog.conf 文件中，*表示出现零到多次，第 8 章中将给出解释。与在大多数 Unix 文件中一样，一行开始处的#符号表示注释，系统不会读取该行的内容，只是将它提供给用户以便于查看文件。

在这个文件中，光标位于第 1 行的第 2 个字符(.)上。由于终端窗口的大小不同，新行从什么地方开始可能会令人混淆。本例中，单词“sole”单独占了一行，看起来好像是一个新行，但是事实上，下一个新行从“sole”后面的星号(*)开始；当用户使用行移动命令时，要紧记这一点。当用户在 vi 中打开行标号时，就会更容易分辨每一行的开始。通过在命令模式下使用:set nu 选项可以做到这一点(如图 7-5 所示)。:set nu 命令为每一个新行编号(包括空行)——编号并不是实际文件的一部分，但是它帮助用户区分每一行。删除行编号的命令是:set nonu。

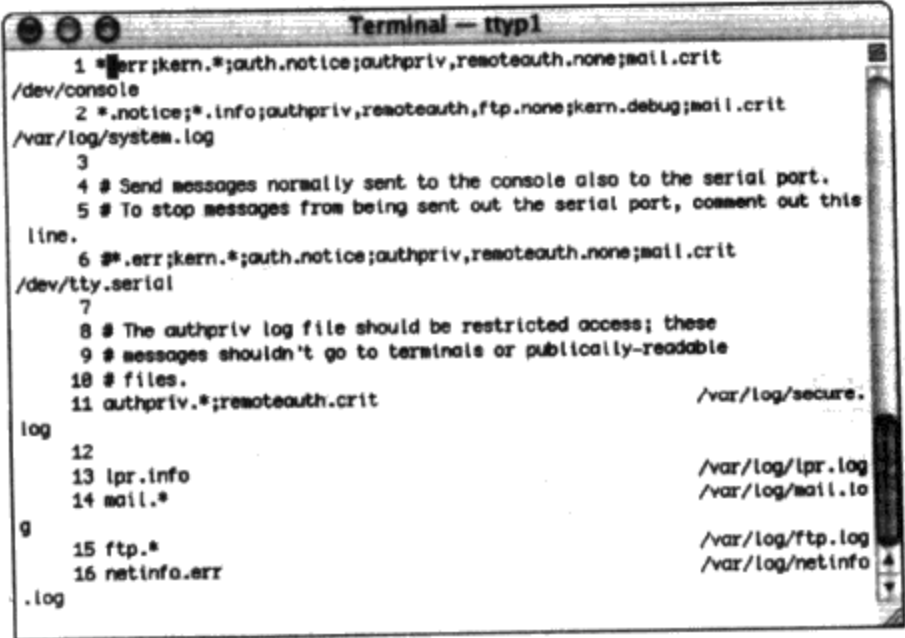


图 7-5

当用户处于插入模式下时,按 **Enter** 键开始一个新行。如果用户在按 **Enter** 键之前,连续输入了 5000 个单词,那么将这 5000 个单词看成一行。

如果在命令模式下,在图 7-5 所示的文件中输入了 **S**,那么游标将移动到第 1 行的末尾,假定单词 `/dev/console` 后面没有空格,那么游标将移动到控制台中的字符 **e** 上。如果在命令模式下输入 **0**(零),那么游标将移动到第 1 行的 *****(星号)上。既然已经返回到行首,那么输入 **w**,游标将移动到 `err` 中的字符 **e** 上。游标移动到这个位置是因为命令 **w** 和 **b**(小写)控制游标逐个单词地移动,将每个标点符号都看作是一个单词。大写的 **W** 和 **B** 实现相同的功能,只是不计算标点符号。如果输入 **W**,游标将移动到下一个单词,因此它将移动到第 1 行的 `/dev/console` 中的第 1 个 **/** 上。还可以在这些命令前加一个数字,从而将游标移动相应数目的位置。例如,如果希望前移 6 个文本块,可以输入 **6w**。

请记住,vi 中的所有命令都是区分大小写的。用错误的大小写格式输入命令键将会导致意外的输出。

要确定游标的当前位置,可以使用组合键 **Ctrl+G**。游标的行号以及字符位置将显示在屏幕底部的状态行中,如图 7-6 所示。如果用户已经浏览了文件,希望返回到光标所在的行,那么这个命令就会派上用场。

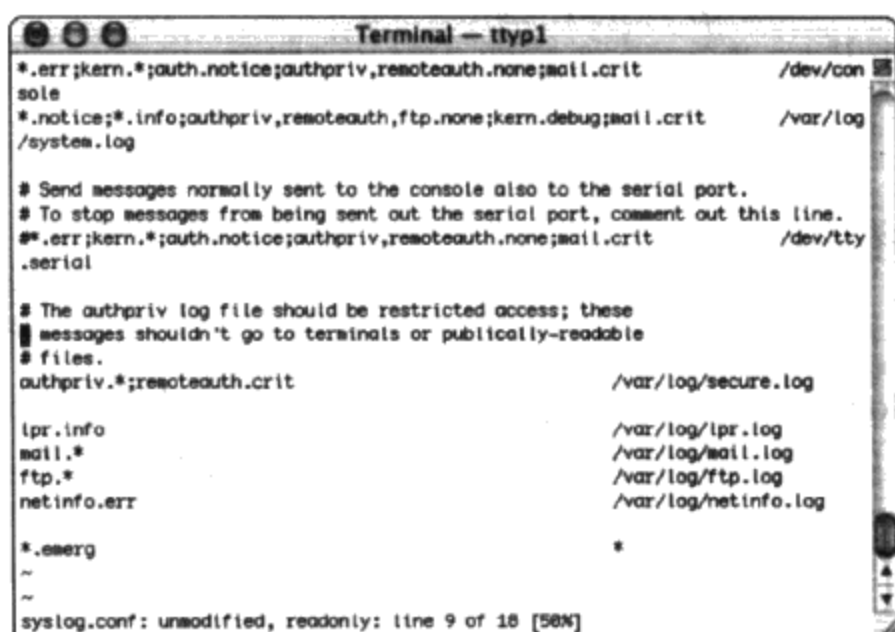


图 7-6

图 7-6 中状态行的内容是:

```
syslog.conf: unmodified, readonly: line 9 of 18 [50%]
```

它说明当前正在查看的是 `syslog.conf` 文件,用户还没有以任何方式修改过(包括空格)该文件。该文件以只读模式打开,这意味着基于当前的文件权限,该文件不能以同样的名字保存,也不能覆盖或修改。游标位于第 9 行上,文件总共有 18 行。游标位于文件的 50% 的标记处。

如果打开的文件不在当前目录中,那么 vi 的状态条将显示这个文件的完整路径。例如,如果本例中的文件是在 `/home` 下而不是在 `/etc`(保存文件的位置)下打开的,那么文件名将显示完整路径 `/etc/syslog.conf`。

用户可以使用 `line-numberG` 命令返回到指定的行。例如,如果用户位于文件的末尾,并希望返回到图 7-6 中所示的游标所在的行,那么可以在命令模式下输入 **9G**。

Ctrl+G 组合将显示不同的输出,这取决于运行 vi 的平台。例如,图 7-7 显示了在 Linux 系统上使用这个命令(Ctrl+G)的结果。

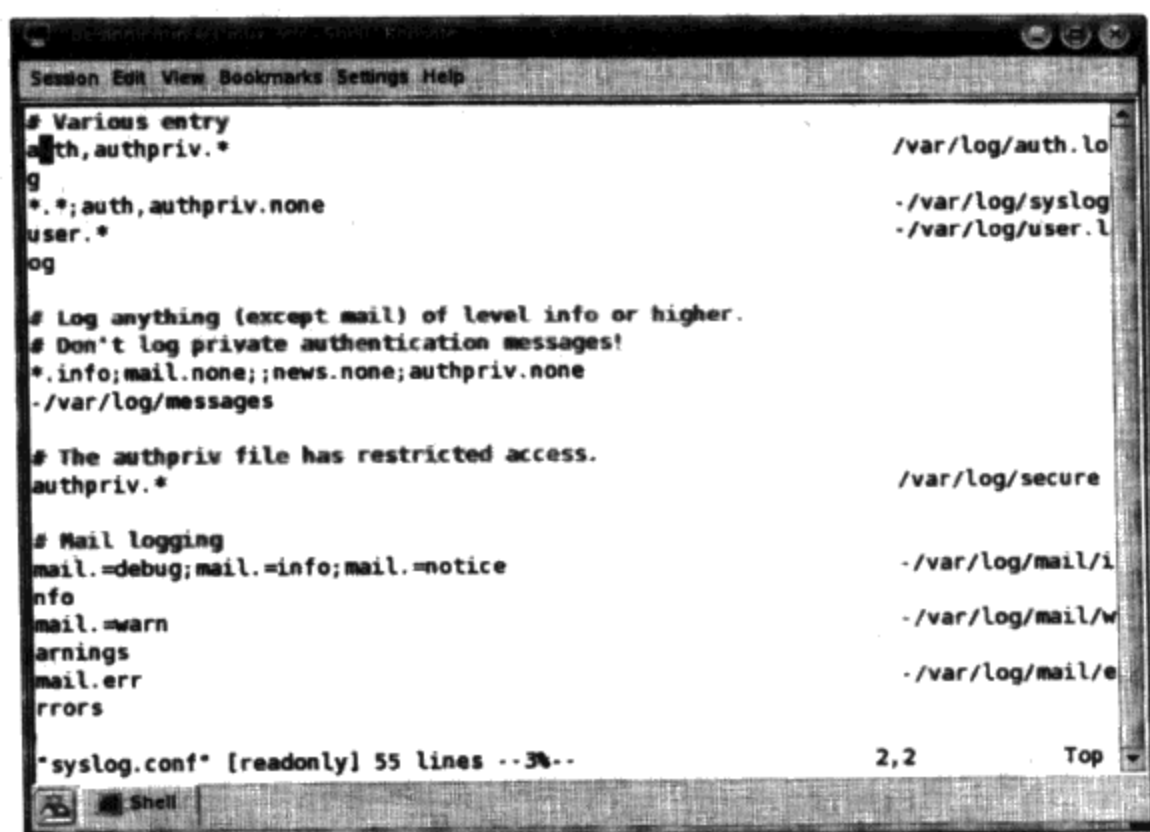


图 7-7

图 7-7 底部的状态行是:

```
"syslog.conf" [readonly] 55 lines --3%--      2,2      Top
```

该行显示文件 syslog.conf 是以只读模式打开的,总共有 55 行。光标位于文件中 3%的位置,在第 2 行第 2 个字符上(2, 2), (如果文件不是在它自己的目录中打开的,那么将显示文件的完整路径)。

7.3 搜索文件

有些时候,用户希望找到文件中的某个特定部分。在这种情况下,用户可以执行简单的搜索。要进行搜索,用户可以按两次 **Esc** 以进入命令模式,并输入/, 其后紧跟希望搜索的字符,然后按 **Enter** 或 **Return** 键。例如,要搜索字符串 end, 可以输入/end 然后按 **Enter** 或 **Return** 键。该搜索将找到相应的字符串,无论它位于单词的哪个位置,因此,如果用户正在搜索单词 end, 可能不会找到这个单词,因为光标将移动到文件中与搜索内容匹配的实例处,可能是 end 本身,但是也可能是 suspend, depend, endeavor, fender 等等。如果在搜索字符串中输入空格,那么也会将它考虑在内,因此输入/ end 可能找到一个以 end 开头的单词。如果希望从光标的当前位置向前(向文件的开头)搜索,可以输入?(问号)来代替/。

如果正在搜索的字符存在多个实例,那么用户可以在每个实例的后面输入一个小写 n, 从而沿同一个方向搜索(向前或者向后),或者输入一个大写 N, 以便沿着当前搜索的相反方向进行搜索。

7.4 退出并保存文件

在完成对文件的读取或编辑之后，用户将希望退出 vi。最常用的退出命令是：`:q!`和`:wq`。要退出 vi 而不保存所做的修改，可以进入命令模式并输入`:q!`。如果用户希望保存文件并退出 vi，可以进入命令模式并输入`:wq`。表 7-4 列出了用于退出和保存文件的主要命令(请记住按两次 **Esc** 以确保在命令模式下)。

表 7-4

命 令	说 明	结 果
<code>:q</code>	冒号和小写 q	退出 vi 编辑器。如果对文件进行了修改，那么 vi 要么询问用户是否希望不进行保存就直接退出，要么通知用户存在修改，不能退出
<code>:w</code>	冒号和小写 w	写(保存)当前文件。如果用户正在编辑一个已有文件并且没有适当的权限对该文件进行写入，那么用户将不能保存这个文件，并且会收到一个错误消息(如果以只读模式打开该文件会得到相同的结果)
<code>:wq</code>	冒号与小写 w 和 q	对文件进行写入，然后退出(具有与 <code>:w</code> 一样的权限问题)
<code>:q!</code> 或 <code>:w!</code> 或 <code>:wq!</code>	在上述命令的后面加惊叹号	惊叹号(!)告诉 vi 直接执行命令，而不提供任何保护措施，例如，询问用户是否希望覆盖已有文件，或者询问用户是否希望在退出之前保存文件。如果用户对文件没有写入权限，并且使用 <code>:wq!</code> 命令，那么 vi 将写入失败，只能成功退出，用户将失去自己所做的编辑
<code>ZZ</code>	两个大写的 Z	保存文件并退出(与 <code>:wq</code> 相同)
<code>:x</code>	冒号和小写 x	保存文件并退出
<code>:w filename</code>	冒号，小写 w，空格，文件名(用实际的文件名代替 <i>filename</i>)	将文件保存为 <i>filename</i> 。该命令将修改后的文件保存为另一个文件，这意味着，对于那些用户没有写入权限的文件，用户也可以保存对它们的修改。用户将新文件保存到一个自己具有写入权限的目录下，例如用户的主目录下。如果用户启动 vi 时没有使用文件名作为参数，那么必须使用这个命令，否则用户会丢失文件中的所有数据
<code>:e!</code>	冒号和小写 e，后跟惊叹号	打开文件的上一次成功写入的版本(没有保存任何当前的修改)。这是一个很好的方法，可以恢复自从上一次写入以来所发生的多个错误，而不需要退出并重启 vi

如果用户使用:w filename 将自己的编辑保存到另一个文件中，该文件不同于用户原先进行编辑的文件，那么用户应该知道自己仍然停留在原始文件中，而不是在保存了自己工作的新文件中。例如，如果用户打开/etc/syslog.conf 并且进行修改，但是不希望在复查之前保存所有的修改，那么用户可以使用:w /tmp/syslog.conf 命令将改变后的文件保存到/tmp/syslog.conf。如果用户继续对打开的文件进行更多的修改，并且使用:wq!命令保存文件，那么所有的修改都将保存到/etc/syslog.conf 中。

vi 还允许用户使用重定向字符>>将创建的文件追加到另一个文件的末尾。例如，如果用户希望将自己的当前文件追加到/tmp/testfile2 中，只需要执行:w>>testfile2，当前文件的内容就将添加到 testfile2 文件的末尾。要使这个命令能够生效，testfile2 文件必须已经存在。

7.5 编辑文件

前面几节介绍的都是只能在命令模式下运行的命令，这种模式不允许用户编辑自己的文件。要对文件进行编辑，用户需要进入插入模式，就像本章前面所提到的那样。从命令模式进入插入模式有许多可用的方法，如表 7-5 所示。

表 7-5

命 令	说 明	结 果
i	小写 i	在当前的游标位置之前插入文本
I	大写 I	在当前行的开始处插入文本
a	小写 a	在当前的游标位置之后插入文本
A	大写 A	在当前行的结尾处插入文本
o	小写 o	在游标位置的下面为文本条目创建一个新行
O	大写 O	在游标位置的上面为文本条目创建一个新行

vi 中的大部分命令都可以在前面加上希望动作执行的次数。例如，命令 2O 在游标位置的上面创建两个新行。

在命令模式下输入这些命令，从而开始向 vi 中输入文本的过程。简单地输入一个小写字符 i 将使用户进入插入模式，在游标位置之前输入文本。利用下面的语句(本节始终使用这些语句)来观察一些示例：

```
Sentence one.  
The quick brown fox jumps over the lazy dog.  
Sentence three.
```

如果游标位于第 2 句的单词 brown 中的字符 r 上，并且在命令模式下输入字符 i，那么用户接下来输入的任何字符都将放在 brown 中的 b 和 r 之间。假定用户在进入插入模式之后输入了字符 newtext。那么示例语句将变成：

```
The quick bnewtextrown fox jumps over the lazy dog.
```

如果在命令模式下输入 a 而不是 i，那么文本插入将从 brown 中的 r 之后 o 之前开始，示

例语句将变成:

```
Sentence one.
The quick brnewtextown fox jumps over the lazy dog.
Sentence three.
```

如果使用 o 命令从相同的起始点进入同一个文件, 那么结果将是:

```
Sentence one.
The quick brown fox jumps over the lazy dog.
newtext
Sentence three.
```

从相同的起始点使用 O 命令将得到:

```
Sentence one.
newtext
The quick brown fox jumps over the lazy dog.
Sentence three.
```

使用 I 或 A 将允许用户分别在当前行的开始(在“The”之前)或结尾(在“dog.”之后)插入文本。用户会一直保持在插入模式下, 直到他按下 **Esc** 键为止。

7.5.1 删除字符

在 vi 中删除字符、行或单词需要使用它自己的命令集。表 7-6 列出了最常用的命令。

表 7-6

命 令	说 明	结 果
x	小写 x	删除游标所在位置的字符(如果命令前有数字 n, 那么将删除从游标所在位置开始的后 n 个字符)
X	大写 X	删除游标位置前面的字符(如果命令前有数字 n, 那么将删除从游标所在位置开始的前 n 个字符)
dw	小写 dw	从当前的游标位置开始删除, 一直到下一个单词(或者是多个单词, 如果命令前有数字)为止
D	大写 D	从游标位置开始删除, 一直到当前行结束
dd	小写 dd	删除游标所在的行

利用相同的示例语句, 游标仍然在单词 brown 中的 r 上:

- x 命令删除字符 r。

```
Sentence one.
The quick bown fox jumps over the lazy dog.
Sentence three.
```

- X 命令删除字符 b:

```
Sentence one.
```

The quick brown fox jumps over the lazy dog.
Sentence three.

- **dw** 命令从游标所在的字符开始一直删除到下一个单词的开头:

Sentence one.
The quick brown fox jumps over the lazy dog.
Sentence three.

- **D** 命令从游标所在的字符开始一直删除到行尾(直到遇到新行为止):

Sentence one.
The quick b
Sentence three.

- **dd** 命令删除整行, 只剩下:

Sentence one.
Sentence three.

在这些命令的前面都可以添加动作发生的次数。例如, 使用原始示例, 从游标位于 **brown** 中的 **r** 上开始, 命令 **2x** 删除两个字符:

Sentence one.
The quick brown fox jumps over the lazy dog.
Sentence three.

执行 **2dw** 命令的结果是:

Sentence one.
The quick jumps over the lazy dog.
Sentence three.

2D 命令从游标所在的 **r** 开始, 一直删除到下一行的结尾, 因此只剩下:

Sentence one.
The quick b

在 **vi** 的各种实现中, 有些选项的功能稍有不同。如果本章中的某个命令没有得到预期的结果, 请参考 **vi** 的联机帮助页。

2dd 命令只留下第一行:

Sentence one.

在 **vi** 中, 可以将命令结合起来构成复杂的操作。例如, 命令序列 **ddO** 将删除当前行, 然后再为用户的新文本开辟一个新行。在示例语句中, 游标位于 **brown** 中的 **r** 上, 在命令模式下输入 **ddO**, 然后输入 **Now is the time for all good men to come to the aid of the party**。得到的结果是:

Sentence one.
Now is the time for all good men to come to the aid of the party.
Sentence three.

这两个独立的命令一起运行, 删除游标所在的行(**dd**)并开辟一个新行(**O**)。

7.5.2 修改命令

用户还能够在 vi 中修改字符、单词或行，而不用删除它们。表 7-7 中给出了一些相关的命令。

表 7-7

命 令	说 明	结 果
cc	两个小写 c	删除当前行的内容，保留用户输入的文本
cw	小写 c 和小写 w	修改游标所在的单词，修改范围是从游标位置一直到单词结束。该命令还使得用户进入插入模式。如果用户希望修改整个单词，那么必须把游标放在单词的第一个字符上面
r	小写 r	替换游标所在的字符。完成替换之后，vi 将返回到命令模式
R	大写 R	覆盖从游标当前所在的字符开始的多个字符。必须按 Esc 键停止覆盖
s	小写 s	用输入字符替换当前字符。完成之后，用户仍然处于插入模式下
S	大写 S	删除游标所在的行并用新文本替换。输入新文本后，vi 仍然保持在插入模式下

来看这些命令是如何工作的。再次使用下面的示例语句：

```
Sentence one.  
The quick brown fox jumps over the lazy dog.  
Sentence three.
```

cc 命令得到与命令序列 ddO 相同的结果。在示例语句中，游标位于 brown 中的 r 上，在命令模式下输入 cc，然后输入 **Now is the time for all good men to come to the aid of the party**。得到的结果是：

```
Sentence one.  
Now is the time for all good men to come to the aid of the party.  
Sentence three.
```

cw 命令与命令序列 dwi 一样，只是该命令在完成删除之后会保留一个空白字符。假定游标仍然位于 brown 中的 b 上，下面的文本行分别是两个命令的结果，可以据此进行比较：

```
The quick b fox jumps over the lazy dog.  
The quick bfox jumps over the lazy dog.
```

其中第一行使用 cw，第二行使用 dwi。cw 命令使用户进入插入模式，从而用户可以在相应的点对文本进行修改。例如，如果用户输入 **cw lue**，那么 fox 的颜色将变成 blue：

```
Sentence one.  
The quick blue fox jumps over the lazy dog.  
Sentence three.
```

r 命令使得用户能够用接下来输入的任何单个字符替换游标所在的字符。在命令模式下，如果在示例语句中输入 **r**，然后输入 **5**，那么结果将是：

The quick b5own fox jumps over the lazy dog.

R 命令允许用户使用所输入的新文本来替换多个字符(从游标所在的字符开始)——输入的新字符替换一个原始字符。输入 **R**，然后输入 **5551234**，得到的结果是：

The quick b555 1234 jumps over the lazy dog.

小写 **s** 的作用与 **R** 一样，只是它删除游标所在的字符然后开始覆盖。

7.5.3 高级命令

vi 中有一些高级命令，它们可以简化日常的编辑工作，使得 **vi** 的效率更高。表 7-8 列出了一些最有用的命令。

表 7-8

命 令	说 明	结 果
J	大写 J	将当前行与它下面的行合并到一起
yy	两个小写 y	拖动(复制)当前行
yw	小写 y 和小写 w	拖动(复制)当前单词，拖动的范围是从游标所在的字符一直到单词的结尾
p	小写 p	将拖动的文本放在游标的后面
P	大写 P	将拖动的文本放在游标的前面

使用 **J** 命令将两行合并到一起。利用示例语句(并且游标位于 **r** 上)，**J** 的结果是：

Sentence one.
The quick brown fox jumps over the lazy dog. Sentence three.

在 **J** 命令前添加一个数字可能会也可能不会合并多行，这取决于用户的 **vi** 实现。

在编辑比较大的文件时，拖动命令是最有用的命令之一。拖动类似于图形编辑器中的复制命令。当游标位于 **brown** 中的 **r** 上时，输入 **yy**，将把整行复制到内存中。如果用户希望将拖动文本放在某一行上，可以将游标移动到该行的前一行上，并输入 **p**。所复制的行将会出现在当前行之后的下一行上。例如，当游标位于 **brown** 中的 **r** 上时运行 **yy** 命令。然后将游标移动到“Sentence three”这一行上，并输入 **p**，得到的结果是：

Sentence one.
The quick brown fox jumps over the lazy dog.
Sentence three.
The quick brown fox jumps over the lazy dog.

在拖动和粘贴时，如果输入删除命令，那么用户将丢失在拖动缓冲区中保存的内容，因为缓冲区内将充满所有已删除的内容。用户可以使用移动或插入命令，这样不会丢失拖动缓冲区的内容。该行为在有些 Unix 版本上可能不同，特别是在 Solaris 的版本上。

yw 命令只复制单词的一部分，从游标所在的字符到单词的结尾，用户可以指定所要复制的单词的个数。例如，如果使用命令 **2yw**，然后将游标移动到“Sentence one”中的 **S** 上，并输入 **p** 命令，用户将得到如下输出：

```
Srown fox entence one.  
The quick brown fox jumps over the lazy dog.  
Sentence three.
```

使用 **P** 命令将得到下面的内容：

```
rown fox Sentence one.  
The quick brown fox jumps over the lazy dog.  
Sentence three.
```

这些都是非常强大的命令。适当练习使用这些命令以适应它们的输出形式。

实战

使用 vi 中的命令

创建一个新文件，输入一些文本并尝试一些学过的命令。

(1) 打开/tmp 目录下名为 **beginning_unix_testfile** 的新文件：

```
vi /tmp/beginning_unix_testfile
```

(2) 目前处在命令模式下。通过输入 **i** 切换到插入模式。

(3) 输入下面的内容，在每行的末尾都必须按 **Enter** 键：

```
The quick brown fox jumps over the lazy dog.  
Sentence two.  
Sentence three.  
Sentence four.  
Vi will never become vii.  
Sentence six.
```

(4) 按下 **Esc** 键返回到命令模式。

(5) 输入 **1G** 使游标移动到第一行，然后输入 **4l**，从而将游标放在 **quick** 中的 **q** 上。

(6) 输入 **cw**，该命令将删除单词 **quick** 并切换到插入模式。

(7) 输入单词 **slow**，然后按下 **Esc** 键。文件现在的内容是：

```
The slow brown fox jumps over the lazy dog.  
Sentence two.  
Sentence three.  
Sentence four.  
Vi will never become vii.  
Sentence six.
```

(8) 输入 **2j** 使游标向下移动两行。游标将位于第 3 行上的 **Sentence** 中的最后一个 **e** 上。

(9) 输入 **r**，然后输入 **E**。该句将变成：

```
SentencE three.
```

(10) 输入 **k** 一次使游标上移一行。输入 **2yy** 复制两行。

(11) 输入 **4j** 使光标移动到最后一行，然后输入 **p** 粘贴缓冲区内的文本。得到的结果是：

```
The slow brown fox jumps over the lazy dog.  
Sentence two.  
Sentence three.  
Sentence four.  
Vi will never become vii.  
Sentence six.  
Sentence two.  
SentenceE three.
```

(12) 按下 **Esc** 键，然后输入 **q!** 退出该文件(没有保存输出)。

工作原理

这个练习给用户提供了一个机会，让用户尝试使用本章中迄今为止所学到的一些命令。成功使用 **vi** 有以下几个要点：

- 要使用命令，用户必须在命令模式下(任何时候都可以按两次 **Esc** 键以确保在命令模式下)。
- 用户必须注意所有命令大小写的正确性。
- 要输入文本，用户必须在插入模式下。

用户可能已经注意到，**vi** 中的命令相当多，在学习使用 **vi** 时，本章中众多的命令表格是很好的参考资料。

7.6 帮助

在 **vi** 中使用的命令很多，因此，如果用户不能记住所有的命令或者它们的用途，也是很正常的。在命令模式下，用户总是可以在命令行(控制台)输入 **man vi** 以获得详尽的联机帮助页，包括有关不同 **vi** 模式的信息以及其他的相关信息。

如果用户犯了严重的错误并希望退出文件，只要输入 **:q!**，这样将直接从文件中退出而不进行保存。另一种方法是使用 **:w filename**，该命令用另外的名字保存这个文件——以防用户希望查看该文件以确定所犯的错误。在这种情况下，原始文件中没有写入任何修改。

如果用户已经做了许多改动，又不希望丢失某些先前的编辑，可以使用撤销命令 **u** 和 **U**。**u** 命令撤销上一次的编辑，例如，如果用户不小心使用了 **3000dd** 命令，以至于删除了 3000 行，那么可以在命令模型下输入 **u**，所删除的 3000 行将回到原来的位置。**U** 命令恢复当前行，因此，如果用户意外地删除了 “The quick”，可以在命令模式下输入 **U**，当前行将恢复到原始的 “The quick brown fox jumps over the lazy dog.”。

vi 的有些版本允许无限制地撤销命令，而有些版本只在缓冲区中保留最后一次的修改，如果有其他命令，如另一个拖动或删除命令需要用到缓冲区，则缓冲区内的原有内容将被覆盖。有多种方法可以保存缓冲区中的内容，不过这超出了本章的范围。请参考 **vi** 的联机帮助页以获得更多关于缓冲区操作的信息。

另一个常见的问题是编辑时在文件中显示的系统信息。可以想象，这些信息在用户编辑文本时出现会引起混淆。例如，假定用户正在编辑图 7-8 中所示的/etc/syslog.conf 文件。

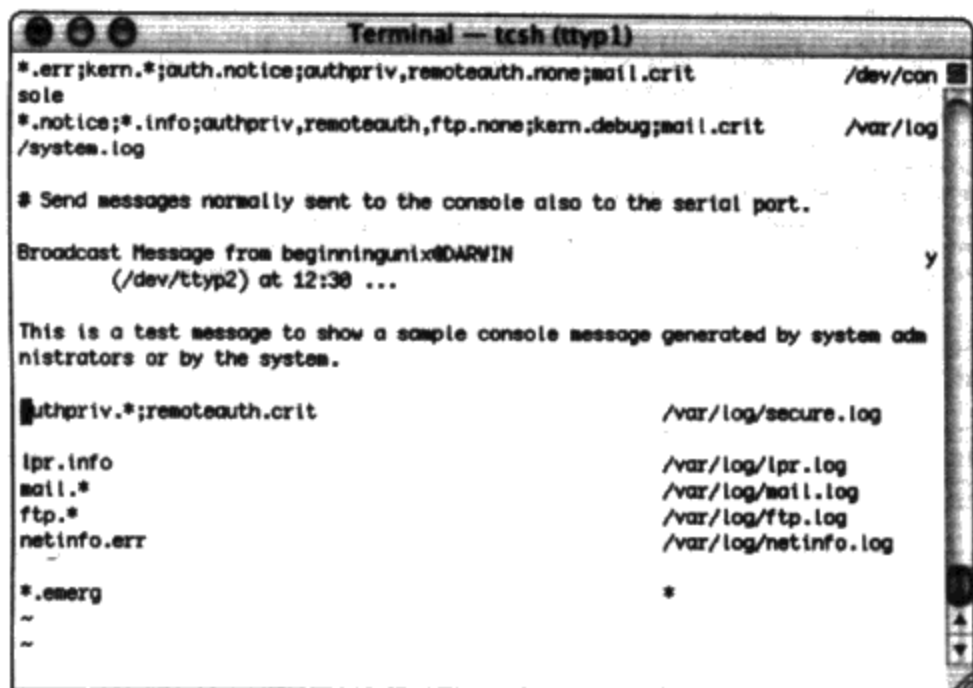


图 7-8

看到“Broadcast Message...”这一行了吗？它不是文件的一部分——它是编辑过程中突然出现的系统消息。可以使用组合键 **Ctrl+L** 来删除这些消息并返回到最初的文本。

7.6.1 运行命令

vi 具有在编辑器中运行命令的能力。要运行某个命令，只需要进入命令模式并输入 **!:command**。例如，在用某个文件名保存文件之前，如果用户希望检查该文件名是否存在，可以输入 **!:ls**，就能够在屏幕上看到 ls 的输出。按任意键(或者命令的转义序列)，用户将返回到 vi 会话。当用户不希望中断 vi 会话，但是需要命令的输出时，这个功能就会很有用。

注意，如果授予某人使用 vi 的根用户能力(例如通过 sudo)，那么即使指定了文件，那个人也能够利用 **!** 序列来运行命令，这将带来严重的安全隐患。千万不要使用 sudo、RBAC 等命令授予用户运行根或任何其他交互式编辑器的能力。

7.6.2 替换文本

vi 的替换命令(**:s/**)使得用户能够快速替换文件中的多个单词或多组单词。该命令的语法是：**:s/characters to be replaced/what to replace with/**。例如，要用“spelled correctly”替换“misspelled”，可以使用命令：

```
:s/misspelled/spelled correctly/
```

当然，事情不一定像看起来那么简单。例如，图 7-9 中所示的文件含有 84 个“misspelled”。

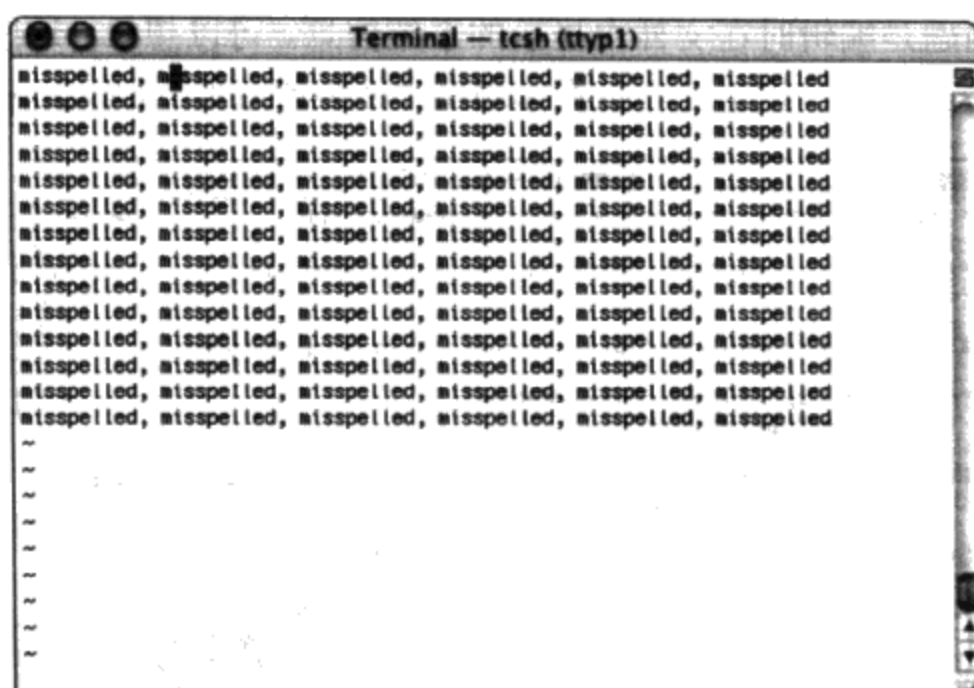


图 7-9

请注意光标位于第 1 行上。输入命令:`s/misspelled/spelled correctly/`, 将得到如图 7-10 中所示的结果。

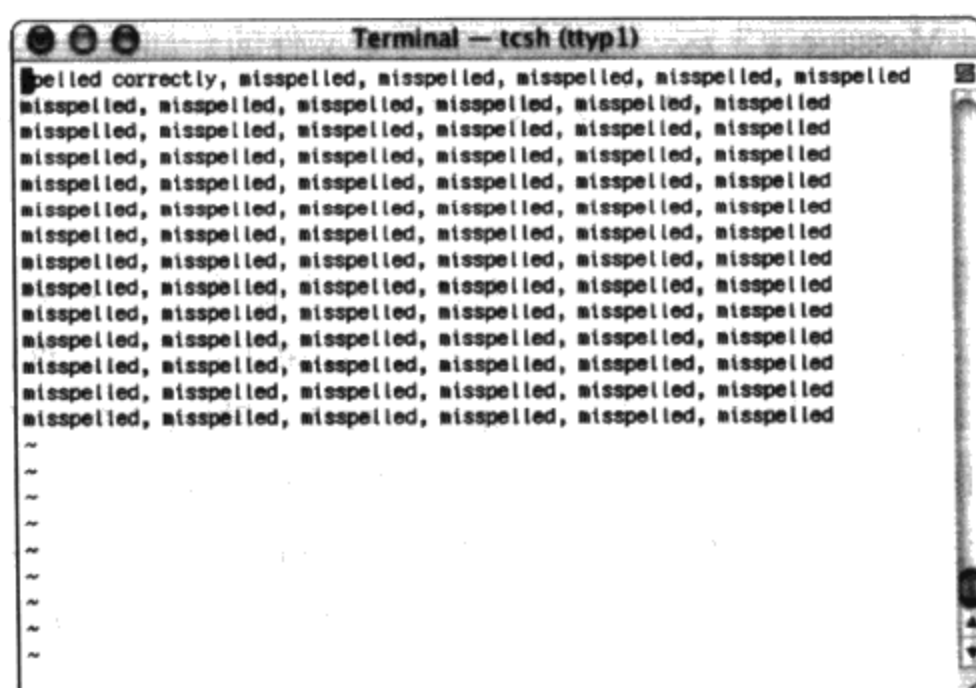


图 7-10

这并不完全是用户所希望的结果, 对吗? 该命令只修改了第一个 “misspelled”, 因为置换命令的工作原理是: 只修改在光标所在行上发现的第一个实例。要修改这一行上的所有实例, 需运行命令:

```
:s/misspelled/spelled correctly/g
```

`g` 代表 `globally`, 在 Unix 中, 它意味着光标所在的行。该命令的结果是修改了光标所在行上的所有实例, 如图 7-11 所示。

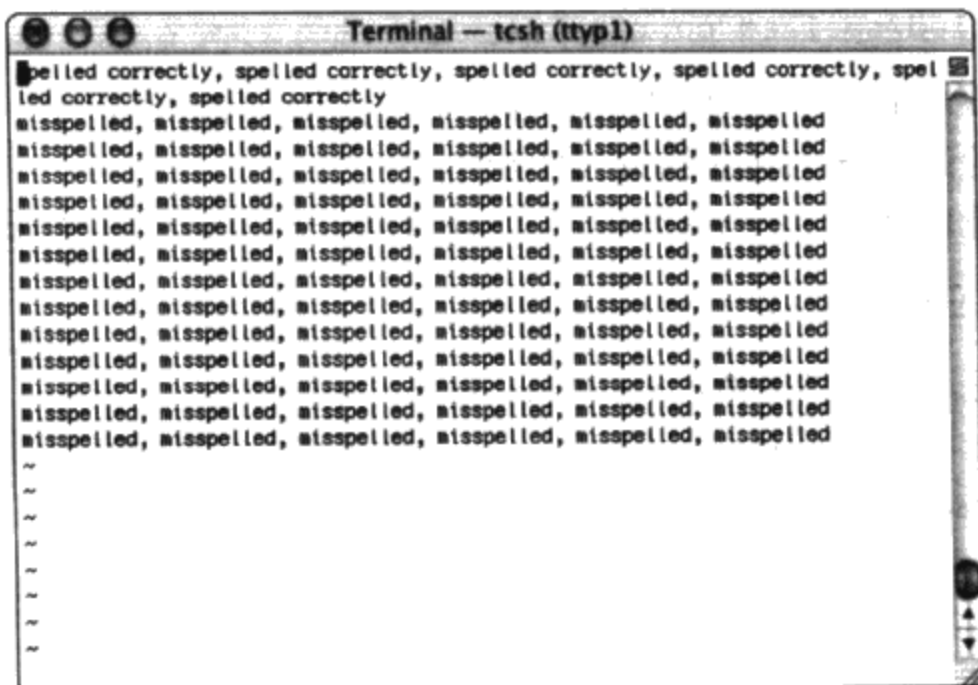


图 7-11

如果用户希望修改连续多行上的拼写，可以在下面的语法中指定行号：

`x,ys//characters to be replaced/what to replace with/`

其中 `x` 是所修改的第一行的行号，`y` 是最后一行的行号。要用 “spelled correctly” 替换第 2 到第 8 行中的所有 “misspelled”，可以输入下面的命令：

```
:2,8s/misspelled/spelled correctly/
```

结果显示在图 7-12 中(开启了 `.set nu` 选项以显示行号，行号并没有和文件一起保存)。

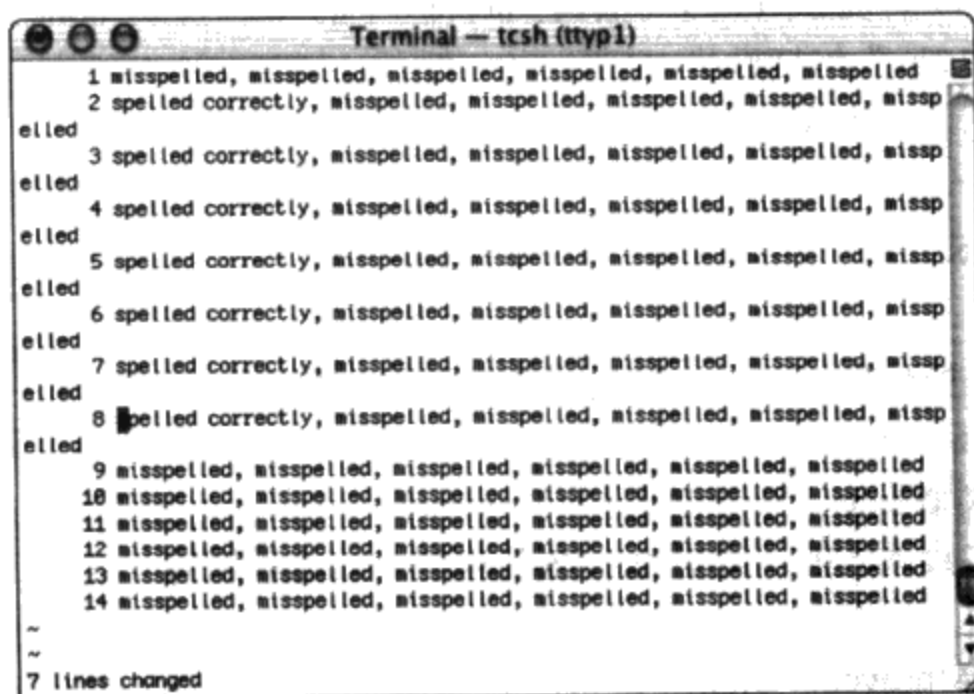


图 7-12

这不是预期的结果，对吗？只替换了每一行的第一个 “misspelled”。要将第 2 到第 8 行的每一个 “misspelled” 都改成 “spelled correctly”，需要在命令的末尾添加一个 `g`，如果所示：

```
:2,8s/misspelled/spelled correctly/g
```

要替换整个文件中的所有 “misspelled”，需要使用下面的命令：

```
:1,$s/misspelled/spelled correctly/g
```

这个命令将替换从第一行(1)到文件末尾(\$)的每个实例。请记住，命令的末尾需要添加 **g**，否则只修改每一行的第一个“**misspelled**”的实例。

如果用户不希望盲目地接受所有替换，可以在命令序列的最后添加确认命令 **c**：

```
:1,$s/misspelled/spelled correctly/gc
```

该命令的结果是，用户必须输入 **yes** 或 **no** 来确认每次修改。

元字符——Unix 中具有特殊意义的字符——能够用来指定某些序列。例如，*****表示零个或多个字符。要用“**essential**”替换所有的“**particular**”和“**particularly**”，使用下面的命令：

```
:1,$s/particular*/essential/g
```

元字符将在第 8 章中深入讨论。

****(反斜线)说明其后的元字符是具有实际意义的字符。要用 **hello** 替换所有的*****，可以使用下面的命令：

```
:1,$s/\*/hello/g
```

本节所要讨论的最后一组表达式是**<**和**>**，它们分别匹配单词开头和结尾的字符。要找到所有在单词的开头(例如“**theater**”)而不是其他任何地方(例如“**lathe**”或“**scathed**”)出现的“**the**”，并将它们替换成“**none**”，可以输入：

```
:1,$s/\<the /none/g
```

该命令用单词“**none**”替换“**the**”，将实例“**theater**”中的“**the**”换成“**none**”(将变成“**noneater**”)，而令“**bathed**”和“**lathe**”维持原状。

以上是一些可用的基本表达式；第 8 章将介绍另外一些表达式。

7.7 vi 的版本

很多编辑器都是模仿最初的 **vi**，因此大多数人根本不再使用它，而是在使用另外的版本。例如，大多数 Unix 系统运行 **Vim**，它是改进的 **vi**，但是大多数用户都没有认识到这一点，因为 **Vim** 的安装运行类似于 **vi**，只是设置了一个 **.vimrc** 文件。下面的列表描述了许多目前可用的 **vi** 版本，但是在大多数系统上，最初的 **vi** 依然可用。学习原始的 **vi** 是有用的，以防机器上没有安装其他的版本。

- **Vim**——**Vi IMproved**。Unix 中最常用的 **vi** 版本。它具有 **vi** 的所有特性，并添加了一些重要的改进，包括语法高亮显示和多级撤销。如果用户正在一台现代化的机器上使用 **vi**，那么很可能是在使用 **Vim**。可以从 <http://vim.org> 上获得更多信息。
- **Elvis**——**vi** 的升级版，具有一些额外的特性。可以从 <http://elvis.vi-editor.org> 找到更多信息。
- **Vile**——表示 **Vi Like Emacs**。它试图将 **vi** 和 **Emacs** 的优点结合在一起。可以从 <http://dickey.his.com/vile/vile.html> 获得更多信息。

- Nvi——vi 的 BSD 版本。可以从 www.bostic.com/vi 获得更多信息。
- 在 vi 爱好者的主页 <http://thomer.com/vi/vi.html> 上列出了 vi 的许多其他版本。

7.8 小结

本章从头开始介绍了 vi 的用法，包括：

- 在文件中使用键(命令)进行移动
- 搜索单词或文本串
- 保存文件和退出 vi
- 编辑文件(插入模式)
- 在文件中搜索和替换文本
- 在搜索和替换中使用元字符

到目前为止，读者已经学习了在 vi 中编辑文件的基础知识。在进入第 8 章学习正则表达式之前，可以通过做下面的练习来巩固有关 vi 的新知识。

7.9 练习

- (1) 在使用 vi 时，用户如何能够完全确认自己所处的模式？
- (2) 在命令模式下，如何在文件中搜索单词“computer”？怎样沿相反的方向继续这个搜索？
- (3) 如何复制 5 行文本，然后将它们插入到下移 10 行的位置？
- (4) 如何将整个文件中的单词“person”替换成“human”，即使同一行中存在多个“person”？

第 8 章 高级工具

在第 6 章中，我们学习了基本的 Unix 命令以及它们的工作原理。现在我们将基于这些知识来扩充自己的命令库。本章介绍高级命令和正则表达式(用于匹配特定模式字符串的公式)。正则表达式很重要，而且刚开始学习它们的时候会具有相当的难度。但是不要担心，因为本章为用户的学习打下了一个很好的基础。

8.1 正则表达式和元字符

正则表达式是具有一定句法的集合或短语，表示某类文本或字符串。正则表达式使得用户能够利用比较少的预定义字符集合来表示多种字符组合。它们通常含有元字符——代表一组字符或命令的字符。

讨论正则表达式和元字符一定要有示例，因此创建一个名为/tmp/testfile 文件，本章中的示例都将使用该文件。以下是输入到该文件中的内容(注意大小写和标点符号)：

```
Juliet Capulet
The model identifier is DEn5c89zt.
Sarcastic was what he was.
No, he was just sarcastic.
Simplicity
The quick brown fox jumps over the lazy dog
It's a Cello? Not a Violin?
This character is (*) is the splat in Unix.
activity
apricot
capulet
cat
celebration
corporation
cot
cut
cutting
dc9tg4
eclectic
housecat
persnickety
The punctuation and capitalization is important in this example.
simplicity
undiscriminating
Two made up words below:
c?t
C?*.t
```

```
cccot
cccccot
```

保存该文件，本章将一直使用这个文件。准备好研究元字符了吗？

8.1.1 理解元字符

元字符在减少和命令一起使用的文本数量以及用最少的字符集表示多组文本方面是非常有用的。下面的列表描述了一些比较常用的元字符。每种情况中显示的结果是将示例搜索在文件 `testfile` 上运行时得到的输出。结果列表中的匹配字符是黑体字，以便于查看匹配的内容。

如果使用过 Microsoft Windows 或 MS-DOS 操作系统，那么可能会对通配符比较熟悉，通配符有些类似于元字符。但是不要将通配符(用在 shell 中)与元字符混淆，因为系统对它们的解释是不同的，如果试图将它们交换使用，就会产生意想不到的结果。

- `.` ——说明：点号或句号。代表一个字符。

示例：寻找含有字母 `c` 和字母 `t`，并且两个字母之间只有一个字符的所有实例：

```
c.t
```

在文件 `testfile` 上运行的结果为：

Simplicity	cut	simplicity
apricot	cutting	c? t
cat	dc9tg4	cccot
cot	house cat	cccccot

- `[]` ——说明：方括号。其结果将匹配括号内的任意一个字符。

示例：寻找含有字母 `c` 和字母 `t`，并且两个字母之间有且只有一个包含在方括号内的字符的所有实例：

```
c[aeiou]t
```

在文件 `testfile` 上运行的结果为：

Simplicity	cot	simplicity
apricot	cut	ccot
cat	house cat	ccccot

- `*` ——说明：星号。代表零个或多个其他字符。

示例：寻找含有字母 `c` 和字母 `t`，并且两个字母之间存在零个或多个字符的所有实例：

```
c*t
```

在文件 `testfile` 上运行的结果为：

```
Juliet Capulet
The model identifier is DEn5c89zt.
```

Sar**castic**, was what he was.
 No, he was just sar**castic**.
 Simp**l**icity
 The qu**ic**k br**o**wn f**o**x j**u**mps o**ve**r the lazy dog
 It's a **C**ello? **N**ot a Violin?
 This **ch**aracter is (*) is the splat in Unix.
 act**i**vity
 apr**i**cot
 cap**u**let
 cat
 cele**br**ation
 corp**o**ration
 cot
 cut
 cut**t**ing
 dc**9**tg4
 ec**l**ectic (also ec**l**ectic; same word so only one instance shows)
 house**cat**
 persn**i**ckety
 The punct**u**ation and **cap**italization is important in this example.
 simp**l**icity
 undisc**r**iminating
 c?t
 c?***.**t
 ccc**o**t
 ccc**o**ccot

- `[^insert_character(s)]` —— 说明：包含一个脱字符号的方括号。不要匹配脱字符号之后的任何字符。

示例：寻找含有字母 c 和字母 t，并且两个字母之间不存在方括号中的任何字符的所有实例：

`c[^aeiou]t`

在文件 testfile 上运行的结果为：

dc**9**tg4
 c?t

- `^insert_character` —— 说明：只有当搜索序列位于行首时才进行匹配。

示例：寻找在行首含有字符串 ca 的所有实例：

`^ca`

在文件 testfile 上运行的结果为：

cap**l**uet
 cat

如果没有^，那么输出将包含所有的 ca 实例，无论 ca 位于一行中的哪个位置：


```
Sarcastic was what he was.  
No, he was just sarcastic.  
capulet  
cat  
housecat  
The punctuation and capitalization is important in this example.
```

- **^[insert_character(s)]** ——说明：脱字符号后跟一个方括号。匹配括号内的任意一个字符；匹配行首的序列。

示例：寻找字母 c 位于行首，后跟任意一个括号内的字符，然后是字符 t 的所有实例：

```
^c[aeiou]t
```

在文件 testfile 上运行的结果为：

```
cat  
cot  
cut  
cuttling
```

如果语法中没有^，那么输出将包含所有的 c[aeiou]t 实例，无论它位于一行中的哪个位置：

```
Simplicity  
apricot  
cat  
cot  
cut  
cuttling  
housecat  
simplicity  
cccot  
cccccot
```

- **\$** ——说明：美元符号。只匹配在行尾出现的实例。

示例：寻找在行尾出现的字符 c 和字符 t，两者之间不存在字符或者存在任意的字符组合：

```
c*t$
```

在文件 testfile 上运行的结果为：

Ca pulet	ca t	c?t
DEn5 C 89 z t	co t	c?*t
apricot	cu t	cccot
ca pulet	house ca t	ccccot

- **** ——说明：反斜线。删除紧跟其后的字符的特殊意义，因此?将具有实际意义，而不再是元字符。

示例：寻找字符 **c**、字面量 **?** 和字符 **t** 的所有实例：

`c\?t`

在文件 `testfile` 上运行的结果为：

`c?t`

- **?** —— 说明：问号。代表零个或一个字符(不要与 ***** 混淆，它匹配零个、一个或多个字符)。并不是在 Unix 的所有程序中都可用。

示例：寻找含有字符 **c** 和字符 **t**，并且两个字符之间有零个或一个字符的所有实例：

`c?t`

在文件 `testfile` 上运行的结果为：

```
Simplicity
eclectic
activity
housecat
apricot
The punctuation and capitalization is important in this example.
cat
simplicity
cot
c?t
cut
cccot
cutting
cccccot
dc9tg4
```

- **[a-z]** —— 说明：括号中含有所有的小写字母。匹配任意一个字母。
示例：匹配含有字母 **c** 和字母 **t**，并且两者之间存在着 **a** 到 **z** 中的某个字母的所有实例：

`c[a-z]t`

在文件 `testfile` 上运行的结果为：

Simplicity	cut	simplicity
apricot	cutting	cccot
cat	housecat	cccccot
cot		

- **[0-9]** —— 说明：匹配数字 0—9 中的某个数字。
示例：寻找含有字母 **c** 和字母 **t**，并且两者之间存在 0 到 9 中的任意一个数字的所有实例：

`c[0-9]t`

在文件 testfile 上运行的结果为：

```
dc9tg4
```

- **[d-m7-9]** ——说明：匹配 d 到 m 之间的某个字符或者 7 到 9 之间的某一个数字。

示例：寻找含有字母 c 和字母 t，并且两者之间存在字母 c 到 t 或者数字 0 到 4 中的任意一个字母或数字的所有实例：

```
c[c-t0-4]t
```

在文件 testfile 上运行的结果为：

Simplicity	dc9tg4
apricot	cccot
cot	cccccot
simplicity	

以上所列的元字符是比较常用的，并且通常在大多数命令中都是可用的。并不是所有的元字符或正则表达式都适用于每个程序，有时候在某个程序中只支持元字符的一个子集。请阅读所用命令的联机帮助页以确定该命令所支持的元字符。

元字符和正则表达式通常和以下命令一起使用(当然还有其他的命令)：

awk	fgrep
ed	less
emacs	more
expr	sed
grep	vi
egrep	

8.1.2 正则表达式

正则表达式可能极为简单，也可能非常复杂，这取决于使用它们的程序以及用户所要查找的内容。正则表达式包含元字符或普通字符。正则表达式是用于匹配某些内容的语法，元字符使得用户能够利用比较小的预定义字符子集来表示更复杂的普通字符组。下面的这个正则表达式非常简单，它将匹配括号中的任意单字符。

```
c[a-n]n
```

正则表达式也可能像下面的示例这样复杂：

```
[0-9][0-9][0-9]\.[0-9][0-9][0-9]\.[0-9][0-9][0-9][0-9]
```

这个表达式匹配美国电话号码的标准格式(例如，555.555.0000)。请注意，\ (反斜线)恢复了.(句号)原来的意义，因此这里的句号表示实际的句号，而不是句号元字符通常表示的意思(代表一个或多个字符)。

学习使用正则表达式和元字符是一个漫长而艰难的过程，但是了解这些基础知识有助于用户在 Unix 系统上完成自己需要做的事情。全面地讨论正则表达式已经超出了本书的范围，但是，如果要学习更多有关这个重要主题的知识，Andrew Watt 所著的 *Beginning Regular Expression* (Wiley, ISBN 0-7645-7489-2)是一本很好的参考书。

8.2 使用 SFTP 和 FTP

SFTP(安全文件传输协议, Secure File Transfer Protocol)和 FTP(文件传输协议, File Transfer Protocol)是用于在 Unix 系统之间传输文件的两个主要的协议。sftp 和 ftp 命令的语法与 telnet 和 ssh 的类似,即用户执行命令,然后指定希望登录的主机。例如,要使用 sftp 命令登录到名为 darwin 的 Mac OS X 机器上(远程机器的 IP 地址为 192.168.1.58),可以输入:

```
sftp 192.168.1.58
```

或者

```
sftp darwin
```

其输出类似于图 8-1。

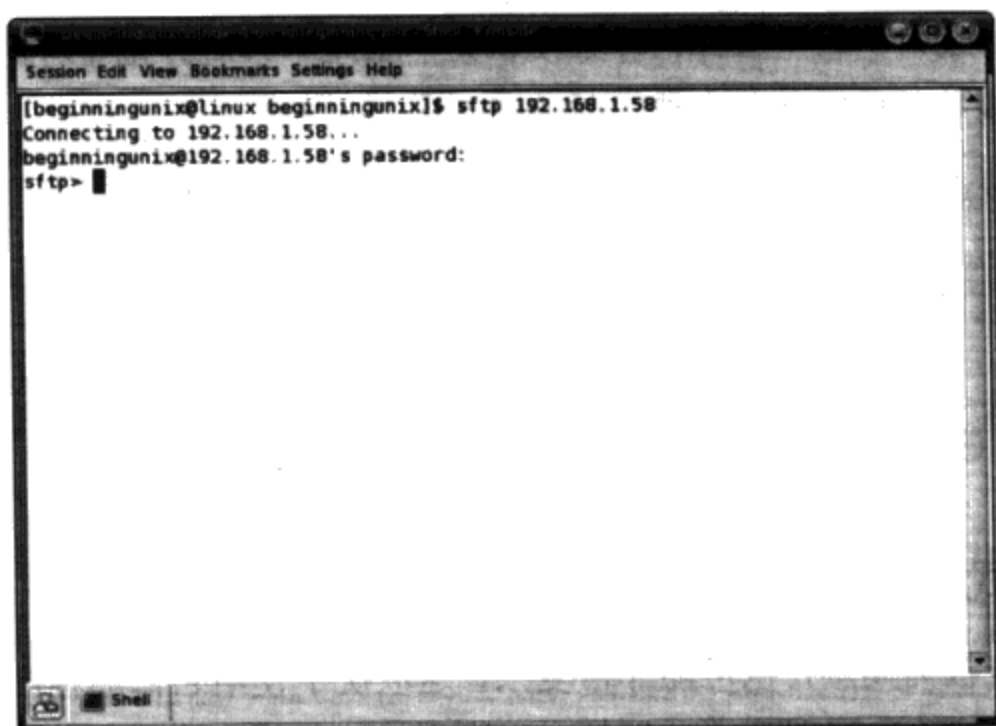


图 8-1

对于 ftp 命令,用户可以输入:

```
ftp 192.168.1.58
```

或者

```
ftp darwin
```

如果用户输入后者,那么输出将类似于:

```
$ ftp darwin
connected to darwin
Name (darwin:beginningunix): beginningunix
331 Password required for beginningunix
Password:
230-
```

```
Welcome to Darwin!  
230 User beginningunix logged in.  
Remote system type is Unix  
Using binary mode to transfer files.  
ftp>
```

图 8-2 显示了 `ftp 192.168.1.58` 命令的输出。可以看出，两者几乎一样，除了使用的是机器的 IP 地址而不是它的名字(darwin)。

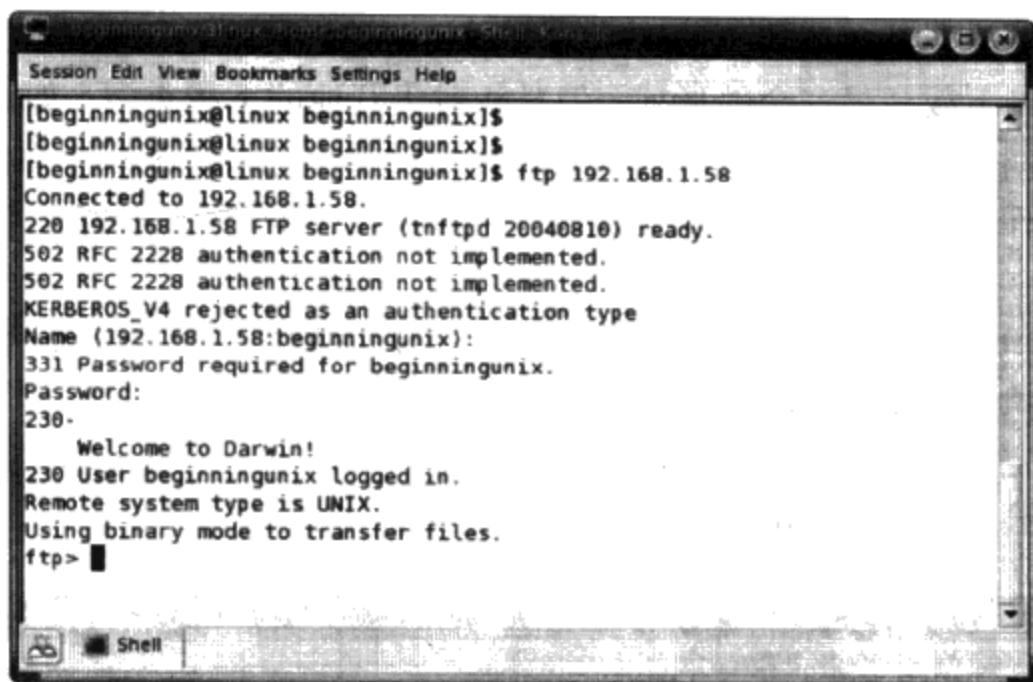


图 8-2

对于 `ftp darwin` 示例，下面给出它所引发的一系列事件，从最初的命令开始：

```
$ ftp darwin
```

如您所知，用户不一定非要使用机器名进行连接。用户可以使用机器的 IP 地址，如图 8-2 所示。

系统通过下面的消息声明对远程机器的连接成功：

```
connected to darwin
```

用户已经连接到系统，但是还没有通过验证，因此系统要求用户输入用户名。在冒号后输入用户名，然后按 **Enter** 或 **Return**(没有输入用户名而直接按 **Enter** 或 **Return** 将把默认用户名——登录本地系统的用户名——提供给远程系统。可以在 Name 后面的圆括号内看到默认的名称。通常情况下，默认的名称与用户的远程机器用户名是一样的)。然后系统提示用户输入口令。用户必须为远程系统输入正确的系统口令(这个口令与登录本地系统的口令可能一样，也可能不一样)。

```
Name (darwin:beginningunix): beginningunix  
331 Password required for beginningunix  
Password:
```

用户的身份通过验证之后，远程系统将提供相关的介绍信息：

```
230-
    Welcome to Darwin!
230 User beginningunix logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
```

第 4 行说明这是一个 Unix 系统，第 5 行说明二进制模式是默认的下载类型。二进制模式意味着以计算机能够提供的最低级别的表示方式下载文件，也就是使用 1 和 0，它们是核心的计算机语言。二进制下载是最安全的形式，但是相对于其他的下载类型，如 ASCII 类型，有时候会花费更长的时间，用这种类型下载的数据通常由键盘上的字母和符号组成。ASCII 类型的下载适用于文本文件，但是对于二进制文件、可执行程序 and 压缩文件而言，二进制模式是最好的选择。

在使用 sftp 或 ftp 命令登录到 Unix 系统之后，用户会收到不同的提示符，分别是 sftp>或 ftp>。当用户处于 sftp 或 ftp 连接状态时，不是使用交互式登录时所用的典型的 shell 环境，因此只能使用某个特定的命令集合。用户移动文件时也只能使用一个特定的命令集合。输入 help 并按 Enter 键或者输入?并按 Enter 键以查看命令列表，如图 8-3 所示。

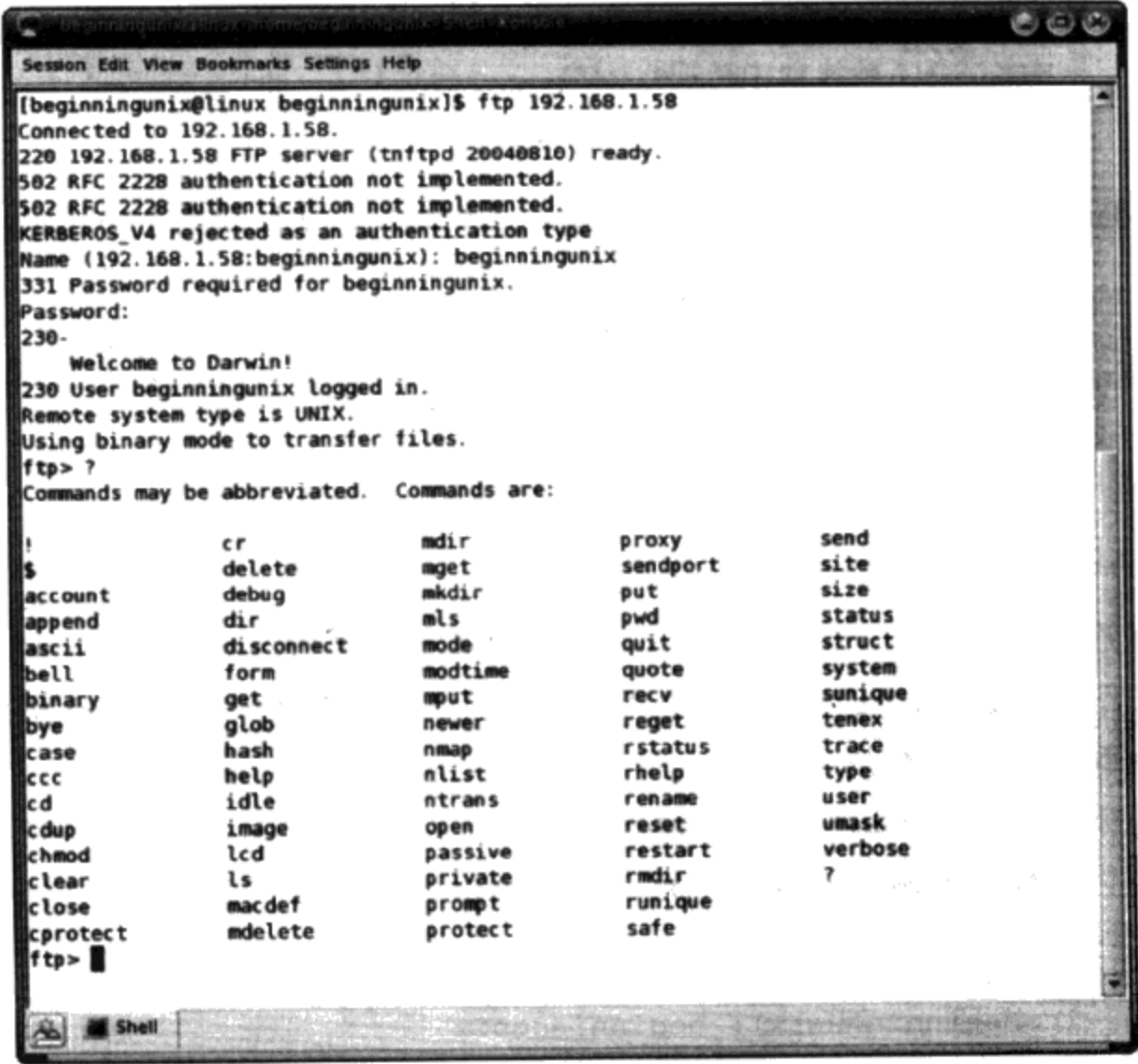


图 8-3

要查看更多与某个命令相关的信息，可以输入 help command。表 8-1 列出了一些在 sftp/ftp 会话可用的命令(在第 4 章中已经学习了其中一些命令——rmdir、mkdir、pwd、cd 和 ls)。

表 8-1

命 令	说 明	在 SFTP 中是否可用	在 FTP 中是否可用
ascii	将传输类型改为 ASCII(适用于文本文件)	否	是
binary	将传输类型改为二进制(适用于二进制文件)	否	是
bye	结束 ftp 会话	是	是
cd	切换目录; 用于在文件系统中浏览	是	是
dir 或 ls	显示目录的内容	ls, 是; dir, 否	是
get	从远程服务器中检索文件	是	是
hash	开启散列符号(#)以指示下载进度(默认时, 每个#符号等于 1024 字节)	否	是
help	显示命令列表	是	是
?	显示命令列表(与 help 一样)	是	是
lcd	将当前的本地目录改成指定的目录	是	是
pwd	显示当前的工作目录	是	是
mdelete	在远程服务器上删除文件	否	是
mget	从远程服务器中检索文件	否	是
mkdir	在远程服务器上创建目录	是	是
more	显示文件的内容	否	是
mput	将文件从本地机器复制到远程机器	否	是
put	将文件从本地机器复制到远程机器	是	是
rmdir	在远程服务器上删除目录	是	是
size	显示远程系统上文件的大小	否	是
system	显示远程机器的操作系统	否	是

登录到远程 FTP 服务器之后, 可以使用 cd(改变目录)命令在文件系统中进行切换:

```
ftp> cd /home/myfiles
```

要从远程 FTP 或 SFTP 服务器中检索文件, 可以使用 get 命令。下面的示例说明了从远程 ftp 服务器(darwin)中检索文件/home/myfiles/myfile.txt 的方法:

```
ftp> get myfile.txt
200 PORT command successful
150 Opening Binary mode data connection for 'myfile.txt' (722 bytes).
226 Transfer complete.
722 bytes received in 0.036 seconds (19 Kbytes/s)
221 Thank you for using the FTP service on Darwin
ftp>
```

第 1 行包含 get 命令, 后跟将要检索并放入本地系统上的当前目录中的远程文件(myfile.txt)。然后远程系统报告命令执行成功(200 PORT command successful)。第 3 行显示下载已经开始,

这个文件含有 722 个字节。接下来系统报告传输已经完成，然后给出详细信息：下载该文件花费的时间以及下载速度。第 6 行是一个简单的感谢用户使用本系统的信息。

如果用户在使用 `get` 命令之前输入了命令 `hash`，那么将会看到一连串的散列符号(#)，它们指示下载的进度，将散列符号用于长文件有助于判断 FTP 会话是否结束。

用户返回到命令提示符(`ftp>`)，从而可以执行其他的 FTP 命令。本例中，`quit` 命令结束 FTP 会话，中止对远程机器的连接并返回到本地系统，这一点通过 `$` 提示符表现出来：

```
ftp> quit
$
```

`sftp` 命令的操作方式与 `ftp` 的几乎完全一样，其输出也类似，但是它还提供了安全的、无须口令的登录以及其他的功能。FTP 和 SFTP 之间的主要区别在于，FTP 传送的任何数据都是不加密的，或者说是以普通文本表示的，而 SFTP 使用加密算法，隐藏了在机器之间传输的信息和数据。用户只需要用 `sftp` 代替 `ftp` 来建立连接。用户的会话将使用 `sftp>` 提示符而不是 `ftp>`。相对于 FTP，SFTP 具有更多优势；用户可以在 www.openssh.org 上找到更多相关内容。

8.3 更高级的命令

用户会经常用到本节所介绍的命令，因为有了它们，用户就可以在 Unix 上完成更多的操作。

8.3.1 grep

`grep` 是 Unix 中最有用的命令之一。它在文件中搜索用户所指定的序列，然后将结果打印出来。这看起来可能并不重要，但是用户可能每天都要用到这个命令来搜索文件或者在文件中进行查找。

`grep` 代表 `global regular expression print`(全局正则表达式打印)，它来自于 `ed` 编辑器(一个 `vi` 之前的编辑器)的一组命令，这些命令以 `g/re/p` 命令序列作为开始。该命令序列的使用非常频繁，以至于人们决定创建一个单独的具有搜索和打印功能的命令，也就是 `grep` 命令。

`grep` 的命令结构是：

```
grep string_to_search_for file_to_search
```

下面是一个简单的 `grep` 命令，用于在 `/etc` 目录及其子目录中搜索单词 `root`：

```
grep root /etc/*
```

也许您已经猜到，该命令将产生大量的输出，因为许多文件都含有单词 `root`。`grep` 支持前面所介绍的大部分元字符，这取决于该命令的版本。

`grep` 还具有参数 `-v`，它使得用户能够搜索不含有指定字符串的所有内容。要搜索 `/etc/passwd` 中不含有 `root` 字符串的所有账户，可以输入：

```
grep -v root /etc/passwd
```

输出将显示 `/etc/passwd` 文件中没有包含字符串 `root` 的内容。

还可以将 `grep` 和其他命令结合在一起，如下列所示：

```
cat /etc/passwd | grep root
```

该命令与上述不带 `-v` 选项的命令产生相同的输出，不过它示范了使用 `grep` 从其他命令的输出中搜索特定字符的方法。`grep` 命令几乎能够与任何其他命令结合在一起以产生有用的输出。

8.3.2 find

`find` 命令用于在目录结构中查找文件。它的语法是：

```
find path_to_look_in options
```

要找到 `/etc` 目录下的 `passwd` 文件，可以使用 `-name` 选项：

```
find /etc -name passwd
```

输出结果显示了在指定路径的目录和子目录下找到的所有相匹配的文件。

`find` 命令具有许多选项。表 8-2 介绍了一些最常用的选项。

表 8-2

选 项	说 明
<code>--help</code>	为 <code>find</code> 命令显示帮助信息
<code>--maxdepth n</code>	向下搜索到第 <code>n</code> 层目录。例如，若只搜索列出的目录，而不搜索子目录，使用 <code>--maxdepth 0</code>
<code>--mindepth n</code>	至少向下搜索 <code>n</code> 层子目录
<code>--mount</code>	防止搜索其他文件系统(例如远程安装的文件系统)。跨文件系统的搜索会消耗很多网络资源

在搜索文件时，还有其他可用的选项；请参考联机帮助页以获得更多信息。表 8-3 列出了 `find` 命令可用的一些测试(细化输出的参数)。如果没有 `find` 命令和相应的选项，这些测试不会起作用。

表 8-3

测 试	说 明
<code>+n</code>	搜索大于数字 <code>n</code> 的所有内容
<code>-n</code>	搜索小于数字 <code>n</code> 的所有内容
<code>n</code>	搜索准确匹配数字 <code>n</code> 的所有内容
<code>-amin n</code>	搜索过去 <code>n</code> 分钟之内访问过的文件
<code>-atime n</code>	搜索最近 <code>n</code> 天内没有访问过的文件(例如， <code>atime 1</code> 搜索 1 天以前访问过的文件)
<code>-fstype type</code>	搜索指定的文件系统 <code>type</code> 。选项包括 <code>nfs</code> 和 <code>ufs</code>
<code>-gid n</code>	搜索 <code>gid</code> (用数字表示的组 ID)等于 <code>n</code> 的文件
<code>-group group_name</code>	搜索组名等于 <code>group_name</code> 的文件

(续表)

测 试	说 明
-name filename	搜索名为 filename 的文件。用户可以使用元字符简化搜索
-perm mode	搜索权限设置为 mode(绝对的或者符号的)的文件
-size n	搜索大小等于 n 的文件。如果用户希望输出不是默认的 512 字节代码块的形式，可以用 c 表示字节，用 k 表示千字节。要找到一个 2 千字节的文件，使用-size 2k
-type file_type	搜索 file_type 类型的文件。选项 b 表示代码块设备，c 表示字符设备，d 表示目录，p 表示命名管道，f 表示正规文件，l 表示符号链接，s 表示套接字
-uid n	搜索 uid(用户 ID)为 n 的文件
-user username	搜索用户名为 username 的文件

有许多其他的可用选项，包括一些用于指定输出特殊格式的选项和一些使用正则表达式的选项。例如，要找到用户 beginningunix 在/home 目录(以及它所有的子目录)下拥有的所有文件，可以使用类似于下面的命令：

```
find /home -user beginningunix
```

假定系统管理员要求用户在自己的 home 目录下腾出空间。用户在 home 目录下拥有很多文件，不能确定哪个文件可能导致这个问题。要搜索大于 2 000 000 千字节(近似于 2 千兆字节)的文件，可以使用如下命令来显示所有满足条件的文件：

```
find /home/ beginningunix -size +2000000k -print
```

与大多数 Unix 命令一样，find 命令具有许多选项；请参考联机帮助页以确定是否有符合自己实际需要的选项。

8.3.3 sort

sort 命令是一个强大而短小的实用程序，它使得用户能够以特定的次序对命令或文件的输出进行排序。表 8-4 列出了 sort 的选项。

表 8-4

选 项	说 明
-d	以字典的顺序进行排序，忽略非字符和数字的内容或者空格
-f	排序时忽略大小写
-g	按数值排序
-M	按月份排序(例如，1 月在 12 月之前)
-r	给出倒序排列的结果
-m	合并经过排序的文件
-u	排序，只考虑惟一的值

sort 命令相当有用；如果用户尝试使用该命令及其选项，就会明白其中的原因。

实战

对文件进行排序

(1) 用下面的文本创建文件/tmp/outoforder:

```
Zebra
Quebec
hosts
Alpha
Romeo
juliet
unix
XRay
Xray
Sierra
Charlie
horse
horse
horse
Bravo
1
11
2
23
```

(2) 以字典的顺序排列文件:

```
sort -d /tmp/outoforder
```

结果是:

```
1
11
2
23
Alpha
Bravo
Charlie
Quebec
Rome
Sierra
XRay
Zebra
horse
horse
horse
hosts
juliet
unix
xtra
```

请注意, 以大写字母开头的字符串位于所有小写单词的前面。

(3) 单词 `horse` 在文件中出现了 3 次。要删除排序中额外的实例，可以使用如下命令：

```
sort -du /tmp/outoforder
```

工作原理

如果用户按照该例的做法，将文件作为输入进行排序，那么排序不会覆盖或修改输入文件的内容。在 `/tmp/outoforder` 文件上运行了这两个命令之后，该文件的次序与初始状态是一样的。这是因为 `sort` 命令将输出发送到标准输出(通常是屏幕)。要使命令能够写入文件，需要使用 `>` 或 `>>`，它们分别是写入文件或在文件末尾追加文本的运算符。

有些新用户使用 `sort` 命令，然后将输出重定向到输入文件。这将删除文件的内容或者可能导致其他不想要的结果，因此不要轻易尝试。

当 `sort` 命令与其他命令，如 `grep` 结合起来，以提供用户能够进行分析的、结构化的、有序的输出时，该命令就会变得更加强大。

8.3.4 tee

`tee` 命令使得用户能够将命令的输出发送到多个位置。例如，如果用户需要在屏幕上查看命令的输出，同时还需要将输出写入文件以备后来的用户查看，就可以使用 `tee`。要运行该命令，只需要定义一个命令，然后指定输出文件的位置。下面是一个示例：

```
ps -ef | tee /tmp/troubleshooting_file
```

该命令将 `ps` 命令的输出显示在屏幕上，同时还将它写入 `/tmp/troubleshooting_file`，这样以后就可以查看该文件。要将输出追加到文件而不是覆盖原有的内容，可以使用 `-a` 选项：

```
ps -ef | tee -a /tmp/troubleshooting_file
```

用户可以在 `tee` 后指定任意数目的文件，输出会写入每个文件中。

8.3.5 script

`script` 命令使得用户能够记录全部的交互式登录会话。从用户启动该命令开始，它就一直捕捉用户的每一个按键(以及它的输出)并保存到一个文件中，直到用户中止该命令时为止。对于排查问题或者将会话的内容保存起来以便稍后查看，`script` 特别有用。要使用 `script`，只需要输入该命令以及 `-a` 选项和文件名：

```
script -a /tmp/script_session
```

如果没有 `-a` 选项，并且指定的文件已经存在，那么该文件将被覆盖。

在 `script` 命令后输入的所有内容都记录在指定的文件中，本例中是 `/tmp/script`。如果用户没有指定文件名，那么这个命令将在启动该命令的目录下创建一个名为 `typescript` 的文件。当用户完成脚本会话后，输入 `exit` 以中止该命令。请注意，不要让脚本会话一直运行，因为所创建的文件将占用大量的空间，甚至会填满用户的文件系统。

8.3.6 wc

`wc` 命令使得用户能够打印文件中换行符、字符或单词的总数。表 8-5 列出了它的选项。

表 8-5

选 项	说 明
-c	显示文件中字符的总数
-l	显示文件中换行符的总数
-w	显示文件中单词的总数

例如，要显示文件/tmp/countfile 中的单词总数，可以使用下面的命令：

```
wc -w /tmp/countfile
```

产生的输出将显示指定文件中的单词总数。其他选项的操作方式类似。

只使用 wc 而没有任何选项将产生所有的结果：字符、换行符和单词。

8.4 小结

本章介绍了使用元字符和正则表达式的基础知识以及 sftp 和 ftp 命令。还研究了一些更高级的命令，包括 grep、find 和 sort。

在继续使用 Unix 的过程中，用户所掌握的工具将会相应的增多。在 Unix 系统上，用户需要完成的大多数操作都具有与任务相关联的命令，强烈建议用户使用联机帮助页及其搜索选项，以找到进一步学习所需要的工具。

8.5 习题

- (1) 演示如何使用 grep 命令在/etc/inetd.conf 文件中找到服务 telnet.d。
- (2) 已知/tmp 目录下有一个大型文件(超过 5 百万千字节)，上一次访问该文件是在 4 天以前。现在用户已记不得文件名了，应该怎样搜索该文件？

第 9 章 高级 Unix 命令：Sed 和 AWK

用户在学习 Unix 技能时，简单的命令可能不能提供所需要的全部功能。大多数程序员都依靠高级 Unix 命令来操作脚本和程序，无论他们使用何种语言。本章将主要介绍两个强大的命令，它们能够提供更高级别的灵活性，简化编程和一般的 Unix 工作：

- sed: 处理纯文本流的文本编辑器。
- AWK: 一种输出格式化语言。

命令 sed 和 awk 与普通的 Unix 命令稍有不同。这两个命令主要在已有文本上执行操作，而不是提供定位目录或者创建或删除文件的机制。所操作的文本可能是管理文件的内容，如 /etc/passwd；可能是另一个命令，如 ls 的输出；或者是用文本编辑器创建的实际文本文件的内容——可能是程序或冗长的文本文件，例如一本书的某个章节。

sed 和 awk 命令起源于老式的行编辑器 ed。现在几乎没有人使用 ed 编辑器，但是在大多数 Unix 系统上仍然能够找到它(或许在您的机器上就有！最起码，安装 ed 的代码可能位于用户的安装盘上或者包含在用户下载的安装包中)。ed 编辑器是一个用于编辑文本文件的命令行程序，是在计算机的早期编写的。那时候，终端屏幕不能显示多行输出，因此每次只能操作给定文件的一行文本，无论该文件有多长或者多复杂。ed 编辑器就是在这种限制下设计出来的，它提供了大量的命令，可以用于导航文件、执行编辑操作以及定位文件的特定行，然后在这些行上执行指定的操作。

如果用户对 ed 编辑器感兴趣，则可以试着在命令提示符后输入 ed，以查看系统上是否安装了该编辑器(输入 q 以退出 ed)。由于它是非常基本的程序，因此 ed 联机帮助页是学习这个编辑器的最好方法：在提示符后输入 man ed。如果没有安装联机帮助页，则可以在 <http://unixhelp.ed.ac.uk/CGI/man-cgi?ed> 上直接从网页阅读。

9.1 sed

sed 命令的功能与 ed 基本相同。两者之间的主要差别在于 sed 以非交互式的方式执行操作。sed 是流编辑器(因此而得名)，它根据用户预先设置的规则来操作指定的文本流。该文本流通常是前面某个操作的输出，这个操作可能是由用户执行也可能是一组自动运行的命令的一部分。例如，ls 命令的输出产生一个文本流——目录列表——可以通过管道将这个文本流传给 sed 并对它进行编辑。另外，sed 能够操作文件。如果用户具有一组内容相似的文件，并且需要对所有这些文件的内容执行特定的编辑，那么 sed 命令能够让用户轻易地实现这一点。例如，按照下面的“实战”部分进行操作，将两个文件的内容合并在一起，同时对两个文件中的名字“Paul”执行替换。

实战 使用 sed 进行操作

可以在命令行输入 sed 的编辑命令：

(1) 用 vi 创建两个文件，每个文件含有一组名字：

```
% vi names1.txt
Paul
Craig
Debra
Joe
Jeremy

% vi names2.txt
Paul
Katie
Mike
Tom
Pat
```

(2) 在命令行输入并运行下面的命令：

```
% sed -e s/Paul/Pablo/g names1.txt names2.txt > names3.txt
```

(3) 显示第三个文件的输出，以观察得到的名字列表：

```
% cat names3.txt
Pablo
Craig
Debra
Joe
Jeremy
Pablo
Katie
Mike
Tom
Pat
%
```

工作原理

sed 实用程序读取指定的文件和/或标准输入，并按照一组命令的指示来修改输入。然后将输入写到标准输出中，如果需要的话，也可以进行重定向。

本例中，sed 命令在命令行参数所提供的两个文件中搜索名字 Paul 的所有实例，并将它们替换为 Pablo。在完成搜索和替换之后，将输出从标准输出重定向到一个名为 names3.txt 的新文件中。请注意命令 s/Paul/Pablo/g 末尾的 g：

```
% sed s/Paul/Pablo/g names1.txt names2.txt > names3.txt
```

该参数指定 sed 进行全局查找。如果没有末尾的 g，并且名字 Paul 在同一行上出现两次，那么只替换第一个。

请注意，虽然每个文件只替换了一行，但是 sed 的输出将按照处理的先后次序显示两个文

件的所有行。原始文件并没有改变；只有输出结果，或者就本例而言，由输出创建的文件中包含了 Pablo 对 Paul 的替换。

9.1.1 使用-e 选项

通过使用-e 选项可以指定多个命令：

```
% sed -e 's/Paul/Pablo/; s/Pat/Patricia/' names1.txt names2.txt
Pablo
Craig
Debra
Joe
Jeremy
Pablo
Katie
Mike
Tom
Patricia
%
```

将多个编辑命令作为命令行参数时就需要用到-e 选项。请注意，虽然并没有要求将指令放入单引号内(第一个 sed 示例就没有使用单引号)，但是最好这样做。将指令放入单引号有助于用户区分与编辑有关的参数以及与其他信息，例如需要编辑的文件的有关参数。此外，单引号将防止 shell 解释在编辑指令中找到的特殊字符或空格。

有 3 种方法可以为 sed 提供一连串的编辑指令，以便在命令行进行处理。一种方法是使用分号将编辑指令隔开，如上例所示。

另一种方法是在每个编辑参数前加一个-e，如下所示：

```
% sed -e 's/Paul/Pablo/g' -e 's/Pat/Patricia/g' names1.txt names2.txt
```

第三种选择是，如果 shell 的多行记录项功能可用，则使用该功能。下面是在 Bash shell 环境而不是 C shell 中的用法：

```
% sed '
> s/Paul/Pablo/
> s/Pat/Patricia/ names1.txt names2.txt'
Pablo
Craig
Debra
Joe
Jeremy
Pablo
Katie
Mike
Tom
Patricia
```

9.1.2 sed 文件

当然，无论使用上述的哪种方法，在命令行上为 sed 输入一长串的编辑命令都是不实用的。

sed 能够读取纯命令文件以获取一连串的命令，该文件含有类似于命令行参数的编辑指令。利用 **-f** 选项可以实现这一点。

-f 参数指示的文件简单地指定一个文本文件，该文件含有一连串顺次执行的操作。这些操作大部分都可以在 **vi** 中手动完成：替换文本、删除行、插入新文本等。这种做法的优势在于所有的编辑指令都放在一个地方，可以一次执行所有指令。在下面的“实战”部分中，将把一组命令放在一起，这些命令编辑两个文件并把结果放入第三个文件中以便于保管。

实战 使用带有多个命令的 sed

实战 使用带有多个命令的 sed

(1) 定位先前示例中创建的两个含有一组名字 of 文本文件。或者简单地创建一个新的名字列表：

```
pdw% vi names1.txt
Paul
Craig
Debra
Joe
Jeremy

% vi names2.txt
Paul
Katie
Mike
Tom
Pat
```

(2) 利用 **vi** 命令创建一个名为 **edits.sedscr** 的新文件，并为 **sed** 列出一组编辑指令：

```
% vi edits.sedscr
s/Pat/Patricia/
s/Tom/Thomas/
s/Joe/Joseph/
ld
```

(3) 调用 **sed**：

```
% sed -f edits.sedscr names1.txt names2.txt > name3.txt
```

其结果是 **names3.txt** 文件中含有如下输出：

```
Paul
Craig
Debra
Joseph
Jeremy
Paul
Katie
Mike
Thomas
Patricia
```

工作原理

在文件中使用多个命令时，每行输入一个命令。sed 将按照文件中列出命令的先后次序，依次执行每个命令。同样，原始文件没有改变；只有输出中包含 sed 执行脚本文件时所产生的结果。

与前面的示例一样，本例将输出重定向到文件中。在大多数情况下，除非将 sed 的输出重定向到另一个程序，否则这是处理 sed 输出的首选方法——将其存放到文件中。通过指定一个 shell 的 I/O 重定向符号，后跟文件名，就可以实现该功能。

```
% sed -f edits.sedscr names1.txt > names3.txt
```

请注意，输出并没有重定向到原始文件，而是放入一个新文件中，在 sed 产生响应之后创建该文件。

9.1.3 sed 命令

sed 编辑器含有 25 个命令。表 9-1 列出了一些最有用的命令。

表 9-1

命 令	名 称	说 明
xa\ text	追加	将命令后的文本追加到匹配特定行地址(x)的每一行中。用希望追加 text 的行号(地址)来替换 x
xd	删除	删除指定行号的一行或多行。也就是说，这些行不会发送到标准输出中
xq	退出	遇到指定的地址就退出。首先将指定的行连同先前的追加或读取命令对它追加的所有文本一起写入输出中，然后 sed 退出对文件的处理
xr file	读取文件	读取文件的内容并在指定的行地址之后追加文本
xs/pattern/ replacement text/flags	置换	将每个指定行上的 pattern 替换成 replacement。如果存在有关置换的执行方式的额外信息，例如是否在所有的匹配模式上执行全局替换，或者只作用于一部分匹配的实例，那么将提供一个标记以修改置换命令的行为
[reg ex pattern] w file	写文件	将模式空间的内容追加到文件中。如果指定的文件不存在，那么该命令将创建这个文件；如果指定的文件已经存在，那么每次执行脚本都将覆盖该文件的内容。将输出定向到同一个文件的多个写入命令将内容追加到文件的末尾

虽然这些典型的命令在编辑文件时将行地址作为参数，但是行地址是可选的。要取代行地址，可以使用一些模式，这些模式包括两条斜线之间的正则表达式、行号或行取址符号。例如，前面示例中使用的替换命令列出了模式而不是行地址进行匹配：

```
s/Joe/Joseph/
```


如果使用行地址，大多数 `sed` 命令能够同时处理一行或多行。要处理多行，需要用逗号将指示行范围的行地址隔开。下面是一些示例：

```
# Delete the first and second lines
1,2d
# Delete lines 2 to 5
2,5d
# append the text to the first and second lines
1,2a\ Hello World!
# read the given file and append to after the second line
2r append_file.txt
```

但是，有些命令只接受单行地址，例如 `quit`。这是有实际意义的，因为不能将该 `sed` 脚本应用于一个范围内的行——`sed` 只能退出一次。

```
# Quit at the 100th line
100q
```

可以将相同的地址上执行的多个命令组合在一起并用花括号括起来：

```
# Delete the last line and quit
${d
  q
}
```

第一个命令可以与开始的括号放在同一行上，但是结束的括号必须单独放在一行上。另外，请注意括号内命令的缩进；在回顾一组命令以查看每步的操作时，行首的空格和 `tab` 使得用户更容易理解。如果不知道文件中最后一行的地址，可以使用美元符号(\$)来指定，如本例所示。

为了代码的可读性，不提倡将多个命令放在同一行上，因为即使每个命令自成一成一行，阅读 `sed` 脚本也是相当困难的。

9.2 AWK

`sed` 的作用类似于在任何类型的文本编辑器中手动编辑命令，因此，如果在非交互式的、批处理的环境下，在文件中编辑文本或者用其他命令编辑文本，它都是一个很好的选择。但是，`sed` 存在一些缺点，例如同时操作多行的能力有限，几乎没有能够用于构建更复杂脚本的基本编程结构。因此，在进行复杂的文本处理时，就需要用到其他的解决方法，`AWK` 就是其中之一，它提供了更一般的计算模型以处理文件。

`AWK` 程序的一个典型示例是将数据转换成格式化的报告。这些数据可能是由 Unix 程序，如 `traceroute` 产生的日志文件，而报告可能以一种对系统管理员有用的格式来归纳数据。该数据还可能从特定格式的文本文件中提取出来的，如下面的示例所示。换句话说，`AWK` 是一个模式匹配程序，与 `sed` 类似。

在命令行输入如下的 `awk` 命令：

```
%awk '{ print $0 }' /etc/passwd
```

其结果将类似于下面的内容，取决于 `/etc/passwd` 文件中的记录项：

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
webalizer:x:67:67:Webalizer:/var/www/usage:/sbin/nologin
ldap:x:55:55:LDAP User:/var/lib/ldap:bin/flase
mysql:x:27:27:MySQL Server:/var/lip/mysql:/bin/bash
pdw:x:500:500:Paul Weinstein:/home/pdw:/bin/bash
```

工作原理

AWK 有两个输入：一个命令、一组命令或一个命令文件，以及数据或数据文件。和 `sed` 一样，AWK 将命令或包含模式匹配指令的命令文件作为处理数据或数据文件的指导原则。

本例中，AWK 没有处理任何数据，只是简单地读取 `/etc/passwd` 文件的内容并将未经过滤的数据发送到标准输出中，类似于 `cat` 命令。调用 AWK 时，需要为它提供两条必要的信息：编辑命令以及所要编辑的数据。本例指定 `/etc/passwd` 作为提供数据的输入文件，并且编辑命令简单地指示 AWK 依次打印文件中的每一行。所有输出都发送到标准输出(可以定向到其他地方)、文件或者其他命令中。

9.2.1 用 AWK 提取数据

AWK 的真正强大之处在于从一个比较大的格式化文件中提取部分数据。

再次利用 `/etc/passwd` 文件，下面的命令将从 `/etc/passwd` 文件的每个记录项中提取两个字段，产生一个更为友好的输出：

```
# awk -F":" '{ print "username: "$1 "\t\t\t user id: "$3 }' /etc/passwd
```

其结果与下面的内容类似：

username: root	user id:0
username: bin	user id:1
username: sync	user id:5
username: shutdown	user id:6
username: halt	user id:7
username: mail	user id:8
username: nobody	user id:99
username: sshd	user id:74

```
username: apache      user id:48
username: webalizer    user id:67
username: ldap         user id:55
username: mysql        user id:27
username: pdw          user id:500
%
```

默认情况下, AWK 将空格作为输入数据的分隔符。要修改该设置, 可以使用 -F 开关来指示不同的字段分隔符——例如, /etc/passwd 文件中的分隔符是冒号。由此可以看出, 紧跟在 -F 开关之后, 放在冒号两边的引号指示了所用的分隔符。

本例还使用 `print` 命令提供输出的结构:

```
print "username: " $1 "\t\t\t user id:" $3
```

所有需要打印的文本都放在双引号内。`$1` 和 `$3` 是变量, 包含 AWK 正在过滤的数据。AWK 每次处理文件时都会初始化一组变量。变量 `$1` 包含从记录项的开头一直到第一个分隔符为止的文本; `$2` 是第 2 个字段, 依此类推, 因此该命令提取第 1 个(`$1`)和第 3 个(`$3`)字段的内容。变量 `$0` 指代整行。

AWK 编辑命令由两部分组成: 模式和命令。将数据文件中的记录行与提供的模式相匹配。如果没有提供模式, 那么 AWK 将匹配所有行, 并且总是执行命令。上面的示例就没有提供模式, 由于在确定是否执行命令之前不需要进行匹配, 因而 AWK 对每一行都执行命令。

9.2.2 使用模式

AWK 中的模式与 `sed` 中的模式类似; 它们由文本串或者一个或多个包含在斜线内的正则表达式组成。下面是一些模式示例:

```
# String example
/text pattern/
# Reg Ex example match any lowercase chars
/[a-z]/
```

遵循模式规则的命令提供了一些指令, 这些指令告诉 AWK 当模式匹配为真时应该执行的操作。一个命令可以包含多个指令, 指令之间用分号(;)隔开。常用的 AWK 指令包括 `=`、`print`、`printf`、`if`、`while` 和 `for`。

这些指令的行为与其他编程语言中的类似指令相似, 提供了在特定条件下(`if`、`while` 和 `for`)为变量赋值(`=`)、打印输出(`print` 和 `printf`)或执行代码段的功能。条件语句细化了模式匹配。

上例中, 在表示输出的具体含义的文本之间打印第 1 个和第 3 个字段, 供用户查阅。语句 `printf` 表示输出具有自己的格式, 允许使用转义序列如 `\t`, 该转义序列表示输出的间隔, 专门针对输出应该使用制表符隔开的情况, 本例使用了两个制表符。

表 9-2 列出了其他一些比较常用的转义序列。

无论是使用准确的字符串还是使用正则表达式, 模式匹配都是 `sed` 和 AWK 的核心部分。因此, 理解它的工作原理是有实际意义的。正则表达式使它能够最大限度地发挥各种工具的作用, 如 `sed`、AWK 以及 Perl(Perl 在第 17 章中讨论)。

表 9-2

转义序列	功 能
\f	换页，新页
\n	换行(\012 或者\015)
\r	回车，添印
\v	垂直制表符
\'	单引号
\"	双引号
\\	反斜线
\0	空(字符值 000)
\a	警报，警钟
\b	退格键
\040	空格键
\ddd	八进制符号
\xddd	十六进制符号

9.3 利用 AWK 编程

与 sed 不同，AWK 实际上是用于模式匹配的一种成熟的、结构化的、解释性的编程语言。这听起来似乎令人畏惧，但是一旦理解了基础知识，就会知道事实并非如此。

它的真正意思是，在使用正则表达式开发用于模式匹配的规则时，AWK 会更加健壮。只要把规则看作是一连串用于分开数据的步骤或命令。与 sed 一样，为了更好地进行全局管理，可以将多个命令放在一个文件中。在下面的“实战”中，将前面“实战”示例的命令放在一个文件中，然后 AWK 使用该文件来处理数据。

实战

使用 AWK 文件

(1) 使用 vi 命令输入如下内容，并将文件保存为 print.awk:

```
BEGIN {
    FS=":"
}
{ printf "username: " $1 "\t\t\t user id: $3 }
```

(2) 以如下方式执行 awk 命令:

```
% awk -f print.awk /etc/passwd
```

得到的输出与前例的相同:

```
username: root                user id:0
```

username: bin	user id: 1
username: sync	user id: 5
username: shutdown	user id: 6
username: halt	user id: 7
username: mail	user id 8
username: nobody	user id 99
username: sshd	user id 74
username: apache	user id 48
username: webalizer	user id 67
username: ldap	user id 55
username: mysql	user id 27
username: pdw	user id 500

工作原理

这里所执行的脚本实现与前面示例相同的功能；两者的差别在于，这里的命令位于文件中，其格式稍有不同。

由于 AWK 是结构化的编程语言，因此文件的布局具有一般的格式：

(1) 开始命令，只在文件的开头执行一次，放在以单词 BEGIN 开始的块中。这个代码块包含在花括号中，如下例所示：

```
BEGIN {
    FS=":"
}
```

(2) 模式匹配命令是由命令组成的块，对数据文件中的每行都执行一次。下面是一个示例：

```
{ printf "username: " $1 "\t\t\t user id: $3 }
```

(3) 结束命令，以单词 END 开始的命令块，只在到达文件结尾时执行一次。虽然该“实战”示例不包含 END 块，但是本例也可以包含一个类似于下面的 END 块：

```
END {
    Printf "All done processing /etc/passwd"
}
```

本例的 BEGIN 块中惟一的代码用于定义分隔符：冒号。这类似于前例中在命令行使用的 -F 开关。本例中，将分隔符赋值给变量 FS，在执行代码的主模式匹配块时，AWK 将检查该变量。

FS(field separator，字段分隔符)是 AWK 使用的几个标准变量之一。其他的变量包括：

- NF——用于提供特定行中的单词总数的变量。
- NR——用于表示已经处理了的记录数的变量。也就是说，NR 的值是 awk 正在处理的文件当前行。
- FILENAME——用于提供输入文件的名字的变量。
- RS——用于表示文件中每一行的分隔符的变量。

AWK 变量是无类型的。也就是说，AWK 允许输入任何值而不需要预先提供数据类型，这与其他编程语言不同，这些编程语言必须预定义变量的内容——例如，变量的内容可以用数学规则操作的数值，或者是用不同规则操作的包括文字和数字的字符串。换句话说，可以将数字或字符串赋值给变量，AWK 根据变量的内容来确定变量的意义。

程序的主要部分，即作用于每个数据行的模式匹配命令，是从第一行开始依次向下执行的。同样，数据文件中的记录行也是从文件的顶端开始逐行向下读取和计算的。

在 AWK 程序读取和计算数据文件时，命令每次只查看数据文件的当前行以及所有的 AWK 程序变量。程序对所有的行都执行模式匹配，并且自动将它们加载到特殊变量如 \$0、\$1 中。

与 BEGIN 块一样，END 块提供了只执行一次的一组命令，在到达输入数据的结尾时执行。刚才的示例没有包括 END 块。BEGIN 和 END 块不是对所有 AWK 程序都是必需的。但是，它们在安装和整理工作环境时是很有用的，大量的编程都是在工作环境中完成的。

有些系统上的 AWK 版本不支持 BEGIN 和 END 命令，但是在开始执行程序时，它会将所有变量设置为零或空。用户在实际使用 AWK 之前，一定要参考系统上的相关文档。

无论如何，BEGIN 和 END 块的要点在于 AWK 是一种无状态的编程环境。也就是说，AWK 用相似的方式对待每个新输入的行。开始和结束块以及变量和条件指令使得程序员能够创建一组状态，这组状态允许以不同的方式对待不同的数据行。这一点很重要，因为大多数实际的任务都需要一组状态以有用的方式来过滤数据。

9.4 小结

编辑器 sed 和编程语言 AWK 使得用户能够在命令行执行强大的文本操作和编辑任务。用户可以单独使用它们，也可以将它们结合起来使用，以管理多种 Unix 文件类型的内容。sed 和 AWK 提供了多种功能，其中包括如下功能：

- 将程序的输出用作文本操作函数的输入
- 排序并从冗长的文件中提取信息，只显示想要的信息
- 使用复杂的模式匹配功能
- 利用单个命令全局替换多个模式串

9.5 练习

创建下面的文件以便在练习中使用。将它命名为 addresses.txt。

```
Roger Longtwig:35 Midvale Ave.:Austin, TX:35432
Brad Brookstone:1044 E. 32nd St.:NY, New York:10001
Richard Smack:845 Pleasant Ter.:Dauberville, CT:06239
Django Steinbart:10 E. Point Way:Atlanta, GA:30374
Cliff Claymore:111 S. Main St.:Clevenger, IA:55472
```

- (1) 创建一个命令，该命令根据邮政编号对文件的内容进行排序，并将排序结果输入到文件 addresses-sorted.txt 中。
- (2) 创建一个 sed 脚本，该脚本将使用全称来替换所有的街道缩写(例如 St.或者 Ave.)或方位缩写(N.、S.、E.、W.)。
- (3) 创建一个命令，该命令将第 2 行中的“NY, New York”转换成“New York, NY”。

第 10 章 作业控制和进程管理

本章将介绍启动和停止进程、向正在运行的程序发送信号、查看正在运行的进程上的信息、shell 作业控制等内容。这些基本的功能使得 Unix 用户能够从命令提示符管理多个进程，以及了解多用户操作系统的功能。用户将学习利用基本的 Unix 工具来识别并控制系统和用户进程的方法。

10.1 进程

简单说来，进程(process)是一个正在运行的程序的实例。

正如第 2 章介绍的那样，操作系统在启动时创建 init 进程，它是所有进程的父进程。一直以来，它的进程 ID 都是 1。在启动其他程序时，系统为每个程序分配一个惟一的进程标识符，称为 PID。

在后台，使用 fork 库调用和 execve 系统调用来启动新程序。这些调用将复制当前运行的程序以得到子进程，从而产生分叉(fork)，子进程是正在运行的程序的精确副本。分叉得到的程序具有一个新的 PID 和不同的父进程 ID(显然)，并且系统全部重置子进程的资源。例如，在默认情况下，分叉得到的子进程和它的父进程共享文件描述符，并且能够共享打开的文件。

相对于产生子进程，用户还可以用一个新进程来代替当前正在运行的进程。Unix shell 含有内置的 exec 命令，该命令用一个新程序来代替正在运行的 shell(在后台，这个命令使用 execve 系统调用)。例如，输入 exec date 将运行 date 程序，并关闭原来的 shell。

通常情况下，进程 ID 是按从小到大的顺序依次分配的。当进程停止时，可以重复使用以前用过的 PID。通常，PID 的取值范围从 1 到 32768。有些系统具有 64 位的 PID 或者更大。在一个运行了 50 天的 NetBSD 工作站上，用户可能会看到 117 个进程，其 PID 范围从 0 到 27152。

有些系统分配伪随机的进程 ID，以试图阻止恶意的程序猜出 PID，从而引发临时文件竞争条件(当不同的程序试图在其他程序之前执行某些操作时就有可能产生竞争条件，例如，一个恶意的程序会试图用一个猜测的文件名创建对一个重要文件的符号链接，而此后另一个程序又试图使用这个文件名)。但是，如果存在安全性问题，随机性可能并没有太大的作用，因为总有办法猜出很多进程 ID。

要查看分配给 shell 的 PID，可以使用 shell 变量\$。例如：

```
%echo $$  
23527
```

输出的 23527 是正在运行的命令行 shell 的进程 ID。

启动一个进程与在 Unix shell 提示符后输入命令或者从菜单启动程序一样简单。软件中含有适用于用户平台的可执行代码。文件系统的属性指示了文件是否可执行以及哪些人有权限执

行该文件(文件的所有者、拥有该文件的组成员或者每个人)。用户可以使用 `ls` 的长列表命令(`ls -l`)来查看这些文件模式。

`file` 命令也可以告诉用户文件是否可执行, 该命令在大多数 Unix 系统上都可以找到。例如, 在 NetBSD 1.6.x 系统上, 用户可能执行如下命令:

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for NetBSD,
statically linked, stripped
```

在 Max OS X 系统上, `file` 显示:

```
$ file /bin/ls
in/ls: Mach-O executable ppc
```

在 Linux 系统上, 用户可能会看到

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for GNU/Linux
2.0.0, dynamically linked (uses shared libs), stripped
```

用户可以将所有类型的 Unix 文件的文件名作为 `file` 命令的参数, 以尝试使用该命令并查看它的结果。用户将会看到对文件类型的简短描述。

10.2 shell 脚本

用户还可以拥有可执行文件, 它们在编写时就可以执行, 而不必用编程代码编写。在加载这种程序时, 文件的第一个字符说明了该文件是哪类型的可执行格式。字符 `#!` (通常读作 *sh-bang*) 告诉内核运行在 `#!` 之后列出的程序, 包括根据需要设置命令行参数。然后该文件就变成正在运行的解释程序所使用的输入。

实战

创建一个简单的 shell 脚本

使用众所周知的 shell 脚本示例 “Hello World”, 来解释创建一个简单的 shell 脚本的概念。

- (1) 打开一个文本编辑器。
- (2) 向空文件中输入下面两行内容:

```
#!/bin/cat
Hello World
```

- (3) 将文件保存为 `cat.script` 并退出编辑器。
- (4) 利用下面的命令使文件可执行:

```
chmod a+x cat.script
```

- (5) 运行该文件:

```
./cat.script
```

用户也可以使用完整路径来运行该文件。其输出就是文件本身, 因为 `cat` 命令显示它的内容。

工作原理

从这个简单的示例中可以看出，sh-bang 程序甚至不需要是真正的编程语言解释程序，但是在大多数情况下，它用于/bin/sh 和 Perl 脚本。本书介绍了许多 shell 脚本编程的示例。要获得有关 shell 脚本编程的更多内容，请参阅第 13 章。

10.3 正在运行的进程

通过运行 ps(process status)命令可以方便地查看用户所拥有的进程。不带任何参数单独运行 ps 命令将显示用户所拥有的进程，这些进程与一个终端相关联。对于快速查看同一个系统上的不同用户正在运行的进程，ps 工具是非常有用的。还可以使用该命令查看是哪个进程耗尽了内存或过多占用了 CPU。

例如，在 Linux 2.6.x 系统上，用户可能得到如下输出：

```
$ ps
  PID TTY          TIME CMD
 18358 tttyp3      00:00:00 sh
 18361 tttyp3      00:01:31 abiword
 18789 tttyp3      00:00:00 ps
```

在 NetBSD 1.6.x 系统上将得到：

```
$ ps
  PID  TT  STAT      TIME COMMAND
 2205  p1  IWS+   0:00.00  bash
 17404 p2  Ss+    1:59.69  ssh -l reed montecristo
 26297 p3  Ss      0:00.04  bash
 26316 p3  R+      0:00.00  ps
```

来自 Mac OS X 10.3 系统的示例是：

```
$ ps
  PID  TT  STAT      TIME COMMAND
 29578 std Ss    0:00.03  -bash
 29585 std S      0:00.00  sleep 1000
```

请注意，在这些默认的示例中，ps 显示了进程 ID、进程所从属的终端(像 tttyp3 或 p2)、进程消耗的系统时间、以及启动该进程的命令(和一些参数)。

NetBSD 示例还显示了这些进程的运行状态(STAT 列)。进程的运行状态由第一个字母表示。表 10-1 列出一些常用的状态。

表 10-1

状 态	功 能
I	空闲，睡眠 20 秒以上
D	等待磁盘或者其他不可中断的等待
R	可运行，激活状态

(续表)

状 态	功 能
S	睡眠不超过 20 秒
T	被追踪的进程或已停止的进程
Z	僵死(zombie)的进程; 已死亡的进程

根据 Unix 系统和所安装的 ps 命令的类型, 用户可能会得到不同的或者额外的状态标识符。一定要阅读 ps (1) 联机帮助页以获得详细信息。

在前面的 ps 输出示例中, W 状态表示进程已被置换出内存。小写 s 状态表示该进程是会话期管理者。加号(+)表示该进程会用到终端。

10.3.1 ps 语法

ps 命令是几个常用的 Unix 工具之一, 它在不同的 Unix 风格上具有不同的语法和输出格式。有两种常见的 ps 工具格式, 一个是 BSD(或 Berkeley)ps 实现, 在 *BSD 和 Mac OS X 系统上可用, 另一个是 Unix System V 实现, 与 Solaris 系统上的类似。大多数 Linux 系统提供的 ps 工具接受两种 ps 工具格式。事实上, procps 套件为 Linux 系统提供的 ps 工具符合 Single Unix Specification 版本 2, 并模仿适合于 IBM S/390、AIX、Digital Unix、HP-UX、IRIX、SCO、SunOS 和 Unix98 的 ps。

ps 用法中的主要差别在于是否使用 - 作为选项的前缀。标准的 BSD ps 选项不使用破折号。例如, 要使用 BSD 风格的 ps 输出当前 shell 的进程状态, 执行命令:

```
ps $$
```

要用 System V 风格的 ps 实现同样的功能, 使用:

```
ps -p $$
```

请试用两种方法以确定自己的 ps 所用的语法(或许两种语法都支持)。

10.3.2 进程状态

在前面显示的 Linux ps 示例中, 输出没有显示有关进程状态的信息。要使 ps 输出这项额外的信息, 如果使用 BSD ps 语法, 可以使用 u 参数:

```
$ ps u
USER      PID %CPU %MEM    VSZ   RSS TTY STAT START TIME COMMAND
reed    18358   0.0   0.7   2460    668 ttty3 S      Sep13 0:00 -sh
reed    18361   0.0  11.6  32936  10756 ttty3 S      Sep13 6:12 abiword
reed    19736   0.0   0.8   2508    784 ttty3 R      18:12 0:00 ps u
```

可以看出, 其输出显示了正在运行进程的用户、进程 ID、进程占用 CPU 的百分比、实际内存的百分比、以千字节为单位的虚拟内存的大小、所用的物理内存的大小、所连接的终端、状态、进程开始时间、进程使用 CPU 的总时间(自进程开始以来)和命令名。

使用 System V ps 风格及其 -l 开关可以显示类似的信息。下面是 Solaris 系统上的示例:

```
$ ps -l
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
8 S 511 366 360 0 50 20 ? 332 ? pts/2 0:01 bash
8 S 511 360 358 0 40 20 ? 136 ? pts/2 0:00 sh
8 0 511 11820 366 0 50 20 ? 138 pts/2 0:00 ps
```

现在已不再使用第一个字段(F)，这里将它包含进来只是为了说明曾经使用过该字段。第二个字段(S)是状态。

请注意命令名是可以修改的，这一点很重要。有些进程修改它们的命令名，以显示有关正在运行的进程的状态信息。例如，POP3 服务器可能用命令名字段来显示当前正在执行的操作。初始的命令名可能显示 POP3 守护进程(daemon)软件的真实文件名，而修改后的名字可能指出当前正在检索电子邮件的用户。

10.4 系统进程

默认情况下，系统应该具有几个正在运行的进程(或者处于其他的运行状态)。例如，一个普通的工作站可能具有 76 个进程，其中 53 个正在运行。通过执行命令 `ps ax | wc -l`，然后去掉含有 ps 报头的行，就可以很容易地查看这些内容。

ps 的 a 参数使得它报告与所有用户的进程有关的信息。x 参数告诉 ps 显示没有控制终端的进程的相关信息。

系统进程是在后台运行的程序，它们处理许多基本的系统维护问题。通常情况下，系统进程没有 TTY(电传打字机，teletype)。其中的许多进程都称为守护进程，它们执行例行的操作。下面是在 Linux 系统上运行的系统进程示例：

```
$ ps ax
PID TTY STAT TIME COMMAND
1 ? S 0:00 init [3]
2 ? SW 0:00 [migration/0]
3 ? SWN 0:00 [ksoftirgd/0]
4 ? SW< 0:00 [events/0]
5 ? SW< 0:00 [khelper]
6 ? SW< 0:00 [kacpid]
20 ? SW< 0:00 [kblockd/0]
21 ? SW 0:00 [khubd]
31 ? SW 0:00 [pdflush]
32 ? SW 0:00 [pdflush]
33 ? SW 0:13 [kswapd0]
34 ? SW< 0:00 [aio/0]
618 ? SW 0:00 [kseriod]
646 ? SW< 0:00 [ata/0]
647 ? SW 0:00 [khpsbpkt]
670 ? SW 0:00 [kjournald]
788 ? SW 0:00 [kjournald]
793 ? S 0:00 /usr/sbin/syslogd
797 ? S 0:00 /usr/sbin/klogd
```

```

816 ?    S    0:00 /usr/sbin/sshd
829 tty1 S    0:00 /sbin/agetty 38400 tty1
830 tty2 S    0:00 /sbin/agetty 38400 tty2
831 tty3 S    0:00 /sbin/agetty 38400 tty3

```

下面是在 Solaris 系统上运行的一组系统进程。它使用 -e 开关输出每个进程的信息：

```

$ ps -e
PID TTY          TIME CMD
  0 ?             0:02 sched
  1 ?             0:18 init
  2 ?             0:00 pageout
  3 ?             8:06 fsflush
314 ?             0:00 sac
224 ?             0:00 utmpd
315 console      0:00 ttymon
 47 ?             0:00 sysevent
 54 ?             0:00 picid
130 ?             0:00 rpcbind
191 ?             0:00 syslogd
179 ?             0:00 automoun
153 ?             0:00 inetd
205 ?             0:00 nscd
213 ?             0:00 powerd
166 ?             0:00 statd
203 ?             0:00 cron
169 ?             0:00 lockd
318 ?             0:00 Xsun
317 ?             0:00 ttymon
307 ?             0:02 void
257 ?             0:00 sendmail
245 ?             0:00 afbdaemo
237 ?             0:00 smcboot
238 ?             0:00 smcboot
239 ?             0:00 smcboot
258 ?             0:00 sendmail
319 ?             0:01 mibiisa
290 ?             0:00 snmpdx
295 ?             0:00 dtlogin
299 ?             0:00 dmispd
322 ?             0:00 dtlogin
302 ?             0:00 snmpXdmi
320 ?             0:00 sshd
323 ??            0:00 fbconsol
335 ?             0:00 dtgreet

```

最后一个示例显示 Mac OS X 系统上的系统进程：

```

$ ps ax
PID TT STAT    TIME    COMMAND
  1 ?? Ss    0:00.02 /sbin/init
  2 ?? Ss    1:48.08 /sbin/mach_init
 78 ?? Ss    0:01.30 /usr/sbin/syslogd -s -m 0

```



```

84 ?? Ss 0:02.28 kexthd
106 ?? Ss 0:00.58 /usr/sbin/configd
107 ?? Ss 0:00.14 /usr/sbin/diskarbitrationd
112 ?? Ss 0:02.74 /usr/sbin/notifyd
128 ?? Ss 0:00.01 portmap
142 ?? Ss 0:04.01 netinfod -s local
144 ?? Ss 2:56.96 update
147 ?? Ss 0:00.00 dynamic_pager -F /private/var/vm/swapfile
171 ?? Ss 0:00.00 /usr/sbin/KernelEventAgent
172 ?? Ss 0:00.40 /usr/sbin/mDNSResponder
176 ?? Ss 0:00.68 /System/Library/CoreServices/coreservicesd
177 ?? Ss 0:00.14 /usr/sbin/distnoted
187 ?? Ss 0:00.88 cron
192 ?? Ss 0:01.21 /System/Library/CoreServices/SecurityServer -X
199 ?? S 0:00.00 /usr/libexec/ioupsd
200 ?? Ss 1:37.85 /System/Library/Frameworks/ApplicationServices.framework
203 ?? Ss 0:00.63 /System/Library/Frameworks/ApplicationServices.framework
210 ?? Ss 0:04.15 /usr/sbin/DirectoryService
228 ?? Ss 0:17.88 /usr/sbin/lookupd
254 ?? Ss 0:00.02 /usr/libexec/crashreporterd
279 ?? Ss 0:00.01 nfsiod -n 4
287 ?? Ss 0:00.02 rpc.statd
291 ?? Ss 0:00.00 rpc.lockd
306 ?? Ss 0:12.81 /usr/sbin/cupsd
310 ?? S 0:00.00 rpc.lockd
319 ?? Ss 0:00.05 xinetd -inetd_compat -pidfile /var/run/xinetd.pid
328 ?? Ss 0:01.30 sipd -f /etc/slpsa.conf
334 ?? Ss 0:00.03 mountd
337 ?? Ss 0:00.00 nfsd-master
338 ?? S 0:11.83 nfsd-server
345 ?? Ss 0:22.02 /usr/sbin/automount -f -m /Network -nsl
348 ?? Ss 0:00.01 /usr/sbin/automount -f -m /automount/Servers -fstab -
355 ?? Ss 0:00.00 /usr/sbin/postfix-watch
4560 ?? Ss 0:00.65 /System/Library/CoreServices/loginwindow.app/Contents
4561 ?? S 0:00.53 /System/Library/Frameworks/ApplicationServices.framework
4565 ?? Ss 0:00.17 /System/Library/CoreServices/pbs
4570 ?? S 0:00.86 /System/Library/CoreServices/Dock.app/Contents/MacOS/
4571 ?? S 0:09.84 /System/Library/CoreServices/SystemUIServer.app/Conte
4572 ?? S 0:03.95 /System/Library/CoreServices/Finder.app/Contents/MacO
4581 ?? S 0:07.51 /Applications/Utilities/NetInfo Manager.app/Contents/
4612 ?? Ss 0:00.18 /System/Library/CoreServices/loginwindow.app/Contents
4614 ?? S 5:29.32 /System/Library/CoreServices/SecurityAgent.app/Conten
4615 ?? S 0:00.18 /System/Library/Frameworks/ApplicationServices.fraiiiew
15147 ?? Ss 0:00.27 /usr/sbin/sshd -I

```

在前面的示例中，大多数系统进程都没有与终端(TTY)相关联。这用 TTY 字段中的问号来表示。还要注意输出可能被截断以适应控制台的宽度。使用 BSD ps，用户可以添加两个 ww 选项以获得额外宽度的输出。

10.5 进程属性

每个进程都有一个具有多种属性的环境，这些属性包括命令行参数、用户环境变量、文件描述符、工作目录、文件创建掩码、控制终端(控制台)、资源利用等等。其中许多属性都可以与父进程共享。

内核还知道进程的许多其他属性，并能够显示它们。在 NetBSD 系统上，内核为每个进程识别大约 85 个不同的属性。有些示例显示父进程的进程 ID、真实的和有效的用户 ID 和组 ID、调用进程的时间以及资源利用情况，例如内存的使用情况和执行该程序所花费的总时间。真实的用户或组 ID 通常是最初启动程序的用户和组。有效的用户或组 ID 是在进程以不同的(例如增强的)权限运行时产生的(用户可以在本章后面的“SETUID 和 SETGID”一节中了解更多的相关内容)。

要查看各种进程属性，用户可以使用 `ps -o` 开关，它可以用于两种风格的 `ps`。它用于定义输出格式。例如，要输出正在运行的进程的进程 ID、父进程 ID 和该进程在虚拟内存中的大小，可以执行命令：

```
ps -o user,pid,ppid,vsz,comm
```

表 10-2 列出了常用的输出格式字段。

表 10-2

字 段	定 义
user	进程的有效用户 ID
pid	进程 ID
ppid	父进程的进程 ID
pcpu	使用 CPU 时间的百分比
rss	以千字节为单位的实际内存大小
pmem	rss 相对于物理内存的百分比
vsz	进程在虚拟内存中的千字节数
tty	控制终端的名称
state(或 s)	进程的状态
stime	开始的时间
time	累计的用户和系统 CPU 时间
command(或 comm)	命令名

请参考 `ps (1)` 联机帮助页以查看可用的输出格式。

10.6 停止进程

有几种不同的方法可以结束进程。通常情况下，从基于控制台的命令界面发送 **CTRL+C** 按键(默认的中断字符)将退出命令。但是有时候进程会捕获或忽略中断字符。

用于结束进程的标准工具是 kill。从技术角度来讲，kill 命令并没有杀死一个命令，只是向进程发送一个特殊的信号。信号用于进程间的简单通信。程序员经常编写软件以不同的方式处理信号，例如告诉软件重新加载相关配置、重新打开日志文件或者激活调试输出。有些信号可能用于告诉软件网络连接已经关闭、存在不合法的或者有问题的内存访问、存在硬盘错误或者其他一些已经发生的事件。系统中有 30 多个信号。

如果系统中安装了开发人员联机帮助页，用户可以通过阅读 signal(3)和 sigaction(2)联机帮助页，学习更多有关在 C 语言下进行信号编程的知识。

kill 命令的默认信号是 SIGTERM(表示 terminate)。用户试图停止的软件在收到 TERM 信号时可能会完全备份文件或者停止工作，但是这些操作也许并不会发生。

要使用 kill 命令，只需要将进程 ID 作为命令行参数。例如，要将默认的 SIGTERM 信号发送给进程 ID 5432，运行命令：

```
kill 5432
```

在使用 kill 命令时要非常小心。一定要使用正确的 PID，否则用户可能会意外地关闭或者发送信号给错误的程序，这可能导致系统不稳定或者不可用。

用户还可以通过名字来选择信号，将-作为信号名的前缀，例如：

```
kill -SIGTERM 5432
```

信号名的 SIG 部分是可选的。

要列出所有可能的信号名，可以运行带有-l 开关的 kill 命令：

```
$ kill -l
HUP      INT      QUIT     ILL      TRAP     ABRT     EMT      FPE      KILL     BUS
SEGV     SYS      PIPE     ALRM     TERM     USR1     USR2     CLD      PWR      WINCH
URG      POLL     STOP     TSTP     CONT     TTIN     TTOU     VTALRM   PROF     XCPU
XFSZ     WAITING  LWP      FREEZE   THAW     CANCEL   LOST     XRES     RTMIN    RTMIN+1
RTMIN+2  RTMIN+3  RTMAX-3  RTMAX-2  RTMAX-1  RTMAX
```

在有些 shell 中，kill 命令是内置的。要使用独立的 kill 工具而不是内置的 kill，可以通过完整路径名来运行，例如/bin/kill(或者/usr/bin/kill，这取决于用户的系统)。

信号还可以通过编号来识别。例如，bash shell 的内置 kill 命令列出了所有的信号及其编号，如下所示：

```
$ kill -l,
1) SIGHUP  2) SIGINT  3) SIGQUIT  4) SIGILL
5) SIGTRAP 6) SIGABRT 7) SIGEMT  8) SIGFPE
9) SIGKILL 10) SIGBUS 11) SIGSEGV 12) SIGSYS
13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGURG
17) SIGSTOP 18) SIGTSTP 19) SIGCONT 20) SIGCHLD
21) SIGTTIN 22) SIGTTOU 23) SIGIO  24) SIGXCPU
125) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH
129) SIGINFO 30) SIGUSR1 31) SIGUSR2 32) SIGPWR
```

如果用户试图终止一个进程，而该进程并没有结束，可以试着使用不可忽略的 SIGKILL

信号。该信号不能由单个的程序操作。如前所述，程序在退出时会执行一些清理工作，因此不要将 `kill -9` 作为首选，否则在关闭程序后，文件或任务可能不一致。

`kill` 命令不使用命令名作为参数。尽管这样可能看起来不方便，但是有实际的意义——如果几个正在运行的进程具有相同的命令名，那么用户肯定不希望结束错误的程序。要找到希望结束的命令的 PID，常用的方法是使用 `ps` 命令。`ps` 的输出可以通过管道传给 `grep`，从而只列出匹配的进程。例如，要找到 FireFox Web 浏览器的进程 ID，用户可能执行命令：

```
$ ps auxww | grep firefox
reed    29591  0.0 16.2 40904 21224 ??   SNs   20Nov04  314:57.03
/usr/pkg/lib/firefox-gtk2/firefox-bin /home/reed/lynx_bookmarks.html
reed    14929  0.0  0.4   100   480 pj   DN+   3:23PM   0:00.01 grep firefox
```

`grep` 输出也显示了 `grep` 行本身。用户可以利用命令 `kill 29591` 来结束 FireFox 进程。

如果使用 System V `ps` 工具，可以使用 `ps -ef` 或者 `-el` 开关。

许多 Unix 系统提供了称为 `pgrep`、`pkill` 和 `killall` 的工具，这些工具使用命令名而不是 PID。在 Solaris 和 System V 系统上，`killall` 表示“kill all(结束所有进程)。”`shutdown` 命令使用它来终止所有活动的进程。在其他系统上，`killall` 用于通过名字向进程发送信号。在使用 `killall` 时一定要小心。在 Linux 系统上，这个危险的工具称为 `killall5`。

一个比较安全的工具是 `pkill`。`pkill` 命令的用法与 `kill` 命令类似，只是将命令名作为参数而不是使用 PID。例如，要关闭 FireFox 浏览器，可以运行：

```
kill firefox
```

`pkill` 的参数是用于匹配的简单正则表达式。另外，`pgrep` 工具用于简单地列出进程 ID 以供 `pkill` 查看，而没有发送任何信号(`pkill` 工具默认发送 `SIGTERM` 信号)。

`pkill` 命令是危险的，因为它可能匹配用户意料之外的进程。应该首先使用 `pgrep` 工具进行测试。一定要阅读系统的 `pgrep`(和 `pkill`)联机帮助页以查看它的特殊选项。

有些系统具有 `pidof` 命令，它与 `pgrep` 类似，但是不进行子串匹配。

10.6.1 进程树

进程树将子进程和它的父进程放在一起，显示了不同进程之间的衍生关系。请注意，每个子进程只存在一个父进程，但是每个父进程可以有多个子进程。例如，下面是一棵简单的 Linux 进程树：

```
$ pstree
init--3*[agetty]
    |-events/0--aio/0
    |
    |   -ata/0
    |
    |   -kacpid
    |
    |   -kblockd/0
    |
    |   -khelper
    |
    |   '-2*[pdflush]
    |-gconfd-2
    |-kdeinit--artsd
```

```
| -firefox-bin---firefox-bin---3*[firefox-bin]
| -3*[kdeinit]
| -kdeinit---sh---ssh
|-8*[kdeinit]
|-khpsbpkt
|-khubd
|-2*[kjournald]
|-klogd
|-kseriod
|-ksoftirqd/0
|-kswapd0
|-migration/0
|-sshd---sshd---sshd---sh---pstree
|-syslogd
'-xdm+-x
'-xdm---startkde---kwrapper
```

这个示例清楚地说明了 `init` 是所有进程的父进程。输出中另一个值得注意的进程是 `ps tree` 子进程，因为它显示了父进程和祖父进程。

可以使用 AWK 和 shell 脚本或者使用 Perl 脚本来编写进程树程序。简单的 pstree 命令可以从 www.serice.net/pstree/ 获得。pstree 实现可以从 <http://psmisc.sourceforge.net/> 上的 psmisc 套件中获得。另外，procps 套件中的 ps 命令提供了一个 --forest 开关，它输出进程树的 ASCII 副本。

10.6.2 僵死进程

通常情况下，结束一个子进程时，会通过 SIGCHLD 信号告诉它的父进程。然后父进程可以根据需要执行一些其他的任务或者重启一个新的子进程。但是有时候会终止父进程。在这种情况下，“所有进程的父进程”，init，将变成新的 PPID(父进程 ID)。用户经常能够遇到一些进程的 PPID 是 1，可能就是这种情况。

在结束一个进程之后,ps 列表可能仍然显示该进程,标识为 z 状态。这是一个僵死的(zombie)或者已死的进程。该进程已经死亡,没有在使用。在大多数情况下,进程会立刻结束,ps 输出并不会显示它。在极少数情况下,例如比较老的 Unix 内核,这些内核包含带有 bug 的软件,用户可能会发现有些进程不能终止,即使向它们发送-SIGKILL 信号。

如果一个进程处于挂起状态，可以首先试着向它发送两个-SIGTERM 信号。如果该进程仍然没有退出，可以尝试发送-SIGKILL 信号。如果用户发现该进程依然拒绝结束，可能需要重启系统。

10.7 top 命令

对于快速显示按各种标准排序的进程，top 命令非常有用。它是一个交互式的诊断工具，会经常进行更新，显示有关物理和虚拟内存、CPU 的使用情况、负载均衡和忙进程的信息。top 命令通常可以在 *BSD、Linux 和 Mac OS X 系统上找到。也可以在 www.groupsys.com/topinfo/ 上下载适用于其他各种 Unix 系统的版本。

下面是在 Solaris 系统上运行 top 命令的示例:

load averages: 0.19, 0.24, 0.12; up 3+19:01:58 06:14:02
37 processes: 36 sleeping, 1 on cpu
CPU states: 99.7% idle, 0.0% user, 0.2% kernel, 0.0% iowait, 0.0% swap
Memory: 512M real, 381M free, 26M swap in use, 859M swap free

PID	USERNAME	LWP	PRI	NICE	SIZE	RES	STATE	TIME	CPU	COMMAND
1	root	1	59	0	1232K	368K	sleep	0:17	0.00%	init
358	root	1	59	0	4520K	2696K	sleep	0:13	0.00%	sshd
307	root	2	59	0	2648K	2000K	sleep	0:01	0.00%	void
366	jreed	1	59	0	2672K	1984K	sleep	0:01	0.00%	bash
16492	jreed	1	59	0	1760K	1184K	cpu	0:00	0.00%	top
318	root	1	59	0	122M	11M	sleep	0:00	0.00%	Xsun
335	root	1	59	0	7912K	4864K	sleep	0:00	0.00%	dtgreet
322	root	1	59	0	6488K	2832K	sleep	0:00	0.00%	dtlogin
205	root	18	59	0	3264K	2528K	sleep	0:00	0.00%	nscd
302	root	2	59	0	3560K	2368K	sleep	0:00	0.00%	snmpXdmid
295	root	1	59	0	5008K	2048K	sleep	0:00	0.00%	dtlogin
179	root	3	59	0	3728K	1992K	sleep	0:00	0.00%	automountd
319	root	7	59	0	2400K	1960K	sleep	0:00	0.00%	mibiisa
299	root	2	59	0	3144K	1936K	sleep	0:00	0.00%	dmispd
257	root	1	59	0	4424K	1864K	sleep	0:00	0.00%	sendroail

一些比较老的 Solaris 和 SunOS 版本没有提供 top 命令。用户可以从第三方下载，如 www.sunfreeware.com，或者从源代码进行安装。

下面的输出显示了在 Mac OS X 系统上运行的 top:

Processes: 60 total, 2 running, 58 sleeping... 126 threads 06:14:04
Load Avg: 0.00, 0.00, 0.00 CPU usage: 0.0% user, 4.2% sys, 95.8% idle
SharedLibs: num = 115, resident = 34.0M code, 3.52M data, 9.45M LinkEdit
MemRegions: num = 3065, resident = 23.8M + 7.87M private, 52.0M shared
PhysMem: 106M wired, 93.8M active, 106M inactive, 306M used, 1.20G free
VM: 3.52G + 80.7M 256370(0) pageins, 0(0) pageouts

PID	COMMAND	%CPU	TIME	#TH	#PRTS	#MREGS	RPRVT	RSHRD	RSIZE	VSIZE
17705	gconfd-2	0.0%	0:00.91	1	13	47	792K	1.95M	2.75M	28.3M
17700	bash	0.0%	0:00.61	1	13	18	228K	964K	664K	18.2M
17699	sshd	0.0%	0:01.90	1	9	42	256K	1.69M	636K	30.2M
17697	sshd	0.0%	0:00.35	1	15	39	100K	1.69M	1.41M	30.0M
15361	bash	0.0%	0:00.33	1	12	17	192K	948K	844K	18.2M
15358	sshd	0.0%	0:01.02	1	9	39	112K	1.69M	468K	30.0M
15147	sshd	0.0%	0:00.27	1	15	39	100K	1.69M	1.41M	30.0M
9960	top	7.3%	0:00.36	1	17	26	268K	492K	648K	27.1M
4615	ATSServer	0.0%	0:00.18	2	44	50	392K	4.03M	1.81M	60.3M
4614	SecurityAg	0.0%	5:29.32	1	62	88	1.29M	13.8M	10.7M	144M
4612	loginwindo	0.0%	0:00.18	3	110	90	1008K	2.97M	3.05M	127M
4581	NetInfo Ma	0.0%	0:07.51	2	78	103	1.86M	8.79M	15.8M	154M
4572	Finder	0.0%	0:03.95	1	99	123	3.46M	16.9M	21.8M	169M
4571	SystemUISe	0.0%	0:10.05	1	169	113	1.80M	8.38M	17.0M	154M
4570	Dock	0.0%	0:00.86	2	76	100	612K	8.95M	10.6M	143M
4565	pbs	0.0%	0:00.17	2	47	39	516K	1.36M	4.28M	43.8M

top 的不同实现提供了不同的信息，并且具有用于交互式行为的不同按键。通常情况下，

top 命令在显示进程时，会将占用大部分 CPU 的进程显示在列表的顶端。这是可以定制的。

可以查阅 top 命令的联机帮助页，或者，如果 top 支持的话，在运行 top 时按问号(?)以获得帮助。按 Q 键退出 top。

前面的 top 输出示例显示了过去 1、5、15 分钟内的负载均衡。用户还可以运行 uptime 或 w 命令来显示负载均衡(load average):

```
netbsd3$ uptime
4:21AM up 139 days , 5 users, load averager: 0.22, 0.22, 0.24
```

负载均衡说明在过去的 1 分钟、5 分钟和 15 分钟内运行队列中作业的平均数。它是活动进程的指示器。该信息可以用于快速确定系统是否繁忙。

负载均衡信息与特定的操作系统版本和硬件有关。用户不能通过比较不同系统的负载均衡来评估其性能。例如，运行在 Alpha 硬件结构上的负载均衡为 10 的 NetBSD 系统感觉上可能比标准的 x86 结构上的负载均衡为 3 的 Linux 系统运行得更快。

即使在适度使用的系统上，也经常看到负载均衡为 0.00 或者接近于零。

10.8 /proc 文件系统

/proc 文件系统是一个动态产生的文件系统，它能够用于检索正在系统上运行的进程的相关信息。/proc 文件系统还可以用于检索和设置其他内核级的配置，这取决于 Unix 系统。Linux 和 Solaris 系统提供了 /proc 文件系统。其他 Unix 系统也可以使用 /proc 文件系统，但是通常没有默认安装。

/proc 文件系统包含了以 PID 为名称的活动进程的目录项。这些目录中包含了提供各种进程属性的文件。例如，下面的内容是一个 /proc 文件系统的目录列表，它属于 Solaris 系统上的个人 shell 进程。

```
$ ls -l /proc/$$
total 5402
-rw----- 1 jreed  build 2736128 Dec 16 12 06 as
-r----- 1 jreed  build    152 Dec 16 12 06 auxv
-r----- 1 jreed  build     32 Dec 16 12 06 cred
--w----- 1 jreed  build      0 Dec 16 12 06 ctl
lr-x----- 1 jreed  build      0 Dec 16 12 06 cwd ->
dr-x----- 2 jreed  build  8208 Dec 16 12 06 fd
-r--r--r-- 1 jreed  build   120 Dec 16 12 06 Ipsinfo
-r----- 1 jreed  build   912 Dec 16 12 06 Istatus
-r--r--r-- 1 jreed  build   536 Dec 16 12 06 lusage
dr-xr-xr-x 3 jreed  build    48 Dec 16 12 06 Iwp
-r----- 1 jreed  build  2112 Dec 16 12 06 map
dr-x----- 2 jreed  build   544 Dec 16 12 06 object
-r----- 1 jreed  build  2568 Dec 16 12 06 pagedata
-r--r--r-- 1 jreed  build   336 Dec 16 12 06 psinfo
-r----- 1 jreed  build  2112 Dec 16 12 06 rmap
```

```
lr-x----- 1 jreed build 0 Dec 16 12 06 root ->
-r----- 1 jreed build 1472 Dec 16 12 06 sigact
-r----- 1 jreed build 1232 Dec 16 12 06 status
-r--r--r-- 1 jreed build 256 Dec 16 12 06 usage
-r----- 1 jreed build 0 Dec 16 12 06 watch
-r----- 1 jreed build 3344 Dec 16 12 06 xmap
```

下面是 Linux 系统上 init 进程的 /proc 条目：

```
$ sudo ls -l /proc/1
total 0
-r----- 1 root root 0 Dec 20 04 41 auxv
-r--r--r-- 1 root root 0 Dec 20 02 11 crndline
lrwxrwxrwx 1 root root 0 Dec 20 04 41 cwd -> /
-r----- 1 root root 0 Dec 20 04 41 environ
lrwxrwxrwx 1 root root 0 Dec 20 04 00 exe -> /sbin/init
dr-x----- 2 root root 0 Dec 20 04 41 fd
-r--r--r-- 1 root root 0 Dec 20 04 41 maps
-rw----- 1 root root 0 Dec 20 04 41 mem
-r--r--r-- 1 root root 0 Dec 20 04 41 mounts
lrwxrwxrwx 1 root root 0 Dec 20 04 41 root -> /
-r--r--r-- 1 root root 0 Dec 20 02 11 stat
-r--r--r-- 1 root root 0 Dec 20 04 41 statm
-r--r--r-- 1 root root 0 Dec 20 02 11 status
dr-xr-xr-x 3 root root 0 Dec 20 04 41 task
-r--r--r-- 1 root root 0 Dec 20 04 41 wchan
```

使用 /proc 文件系统能够查看进程所使用的文件描述符、命令名、实际正在运行的可执行文件、定义的环境变量等等。

10.9 SETUID 和 SETGID

当文件具有执行时设置用户 ID(set-user-ID-on-execution)或执行时设置组 ID(set-group-ID-on-execution)的模式集时，程序能以增强的功能运行。如果一个程序为根用户所有，并且设置了设置用户 ID(set-user-ID)模式位，那么当普通用户运行该程序时，它将具有增强的(根)权限。例如，Mac OS X 系统上的 `setuid` 和 `setgid` 工具：

```
-r-xr-sr-x 1 root operator 23336 23 Sep 2003 /bin/df
-r-sr-xr-x 3 root wheel 41480 25 Sep 2003 /usr/bin/chsh
-r-sr-xr-x 1 root wheel 37704 23 Sep 2003 /usr/bin/crontab
-r-xr-sr-x 1 root kmem 24320 23 Sep 2003 /usr/bin/fstat
-r-sr-xr-x 1 root wheel 39992 25 Sep 2003 /usr/bin/passwd
-rwxr-sr-x 1 root postdrop 140556 25 Sep 2003 /usr/sbin/postqueue
```

其输出是标准的 `ls` 长列表格式(例如 `ls -l`)。用户和组的执行位用小写字母 `s` 表示，说明它们分别是 `setuid` 和 `setgid`。

输出说明存在许多有用的管理工具，它们放在 `SETUID` 和 `SETGID` 之后作为一个整体使用：

- `setgid df` 命令使得用户拥有操作人员组一样的权限，因此它能够查看那些只对操作人员组中的成员可读的磁盘设备。

- `setuid chsh` 和 `passed` 工具使得用户拥有根用户一样的权限，因此普通用户也能够更新用户数据库。用户使用该软件只能修改它们自己的记录项。
- `setuid crontab` 命令以根权限运行，因此它能够从 `crontabs` 目录中创建、编辑或删除用户自己的 `crontab`，该目录通常位于 `/var/cron/tabs/`。
- `setgid fstat` 工具使得用户拥有 `kmem` 组一样的权限，因此它能够从 `/dev/kmem` 设备中读取数据，该设备只对 `kmem` 组成员是可读的。`/dev/kmem` 设备用于检索各种对故障检修、调试和性能分析有用的系统数据结构。
- `setgid postqueue` 工具以 `postdrop` 组权限运行，因此它能够查看和管理 Postfix 电子邮件队列文件(Postfix 是一个邮件传输代理，它在所有的 Unix 系统上都没有默认安装。请登录 www.postfix.org 以获得更多信息)。

日常工作中会用到许多其他的 SETUID 和 SETGID 软件。用户可以使用 `find` 工具来列出系统上的这些软件。尝试使用下面的命令：

```
sudo find / -perm -4000 -print
```

该命令列出具有执行时设置用户 ID 位集的所有文件。

不难想象，SETUID 和 SETGID 软件可能导致潜在的安全问题。例如，可能将不能正确检查其输入的软件随意用于运行其他命令或访问其他文件，这些命令和文件拥有增强的权限。应该仔细编写和审计 SETUID 或 SETGID 软件。因为存在一些会利用 shell 脚本及其环境的技术，所以大多数 Unix 系统都不支持 `setuid`(或 `setgid`)。有些系统提供了 `suidperl`，可以使用它创建 `setuid Perl` 脚本(要了解 Perl 的基础知识，请参阅第 17 章)。

安全地利用 `setuid` 和 `setgid` 功能的主要方法是使用细粒度的组，这些组拥有用户需要对其进行读取和写入的目录和/或文件(包括设备)。例如，可以将用户数据库分成很多单独的文件，其中一些允许用户对其进行修改。

10.10 shell 作业控制

Unix 命令行 shell 可以用于在后台运行程序，因此用户能够同时运行多个程序。它还可以用于挂起命令以及重新启动挂起的命令。要让 shell 在后台运行特定的命令，只需要简单地 toward 命令行的末尾追加一个 `&`。例如，要在后台运行 `sleep`，执行命令：

```
$ sleep 60 &
[1] 16556
$
```

该命令使得 `sleep` 命令运行 60 秒。`sleep` 工具等待一定的时间然后成功退出。通过使用 `&`，将显示下一个命令行 shell 提示符，该提示符可以立刻用于运行其他命令。

这个输出示例来自 BASH shell。括号内的 1 是作业编号，16556 是进程 ID。这些信息是由 shell 而不是 `sleep` 命令显示的。

大多数 shell 都具有内置的 `jobs` 命令，可以用它来显示正在运行的 shell 作业：

```
$ jobs
```

```
[1]+ Running          sleep 60 &
```

在后台运行的命令不需要等待控制台的输入，但是可以向控制台输出文本。在前台运行的命令可以拥有完整的交互。

用户可以通过运行内置的 `fg` 命令将一个作业从后台转到前台。如果用户拥有多个作业，可以将作业编号作为 `fg` 的参数，例如：

```
$ fg 1
sleep 60
```

一旦用户将一个命令转到前台，系统将不会显示 `shell` 提示符，直到该进程结束时为止，并且直到此时用户才能运行另一个命令。

`shell` 还使得用户能够通过按下默认的挂起按键组合 **Ctrl+Z** 来挂起一个当前正在运行的前台进程。使用 `sleep` 命令是练习使用作业控制的一种简单的方法。下面的示例是在运行 `sleep` 之后按下 **Ctrl+Z**：

```
$ sleep 60
^Z
[1]+ Stopped          sleep 60
$ jobs
[1]+ Stopped          sleep 60
$ bg %1
[1]+ sleep 60 &
$
```

^Z(Ctrl+Z) 按键停止编号为 1 的作业。可以通过使用 `fg` 命令或者内置的 `bg` 命令解除挂起。本例中，`bg` 命令用于在后台运行这项作业。

还可以终止 `shell` 作业(或者向 `shell` 作业发送信号)。大多数 `shell` 都具有内置的 `kill` 命令，该命令使用百分号引用作业编号而不是 PID。例如，`kill %1` 将向编号为 1 的作业发送终止信号。如果用户正在使用该 `shell` 作业，那么一定要使用百分号，否则用户可能将结束信号发送给错误的进程。

用户还可以通过向进程发送信号 18(SIGSTP)来挂起一个进程，然后通过发送信号 19(SIGCONT)解除挂起。

```
$ sleep 120 &
[2] 15695
kill -18 %2
[2]+ Stopped          sleep 120
kill -19 %2
$ jobs
[2]+ Running          sleep 120 &
netbsd
$
```

当用户不能按 **Ctrl+Z** 执行挂起时，使用 `kill` 命令来挂起进程和解除对进程的挂起是很有用的。

10.11 小结

本章介绍了几种用于进程管理的简单却很基本的工具。进程是加载的程序，具有惟一的进程 ID。通常认为进程包括父进程和子进程——所创建的进程(新启动的程序)称为子进程。

- 可以使用 `ps` 和 `top` 工具列出进程信息，也可以通过查看 `/proc` 文件系统(在提供 `/proc` 的操作系统上)实现该功能。
- 可以使用 `ps` 和 `top` 工具报告性能和检修故障。
- 信号是进程间通信的方法。
- `kill` 命令是用于向进程发送信号的基本工具；它的默认值是发送终止信号。
- Unix shell 提供了在后台同时运行多个程序的简单方法，即在命令行的末尾添加 `&` 符号。

第 11 章 在指定时间运行程序

第一台数字计算机是为了处理重复任务而设计的，这些任务对大多数人来说是困难或无趣的。计算机的优势也正在与此。这种计算方法称为批处理计算，目前仍在广泛使用。如果计算机系统单纯地依赖于交互模式、固定的操作和用户的反应，那么数字计算机的价值就会大打折扣，无论它使用什么操作系统。本章将介绍 Unix 系统上的一些用于在特定时间运行程序的基本工具——这是充分利用数字计算机的第一步。

当然，如果没有时钟，用户就不能在指定的时间运行任何程序，因此本章首先介绍设置系统时钟的方法、将系统时钟和嵌在硬件中的时钟联系起来的方法、以及和网络中的其他系统时钟同步的方法。

本章还将介绍一些常用的工具，如用于调度和运行 Unix 系统上的程序和脚本的 `crontab`，并提供一些作业示例，用以说明系统的自动控制如何将用户从不必要的重复且令人厌倦的工作中解放出来。

11.1 系统时钟

大多数计算机系统实际上拥有两个 24-小时的时钟：硬件时钟和系统时钟。由电池供电的硬件时钟嵌在系统硬件中。它用于在系统关闭时记录时间。系统时钟是与硬件时钟相对应的软件版本，在系统初始化时创建。它根据硬件时钟进行设置。

在大多数 Unix 系统上，硬件时钟用于保存世界标准时间(UTC, Universal Time)，也称为格林威治时间(GMT, Greenwich Mean Time)，而不是系统所在时区的时间。可以配置系统记录 UTC 时间，并调整 UTC 和本地时间之间的偏差，包括夏令时。

例如，在启动一个基于 Unix 的系统(如 Linux)时，其中一个初始化脚本运行 `hwclock` 程序，该程序将当前的硬件时钟复制给系统时钟。这是一个重要的起始步骤，因为 Unix 正是通过系统时钟(也称为内核时钟或软件时钟)来获取当前的时间和日期的。

当夏令时影响不同的时区时(并且不是所有地区都使用夏令时)，UTC 并不会变化。

无论硬件时钟是否设置为 UTC，大多数系统都按照自 UTC 时间 1970 年 1 月 1 日午夜以来的秒数来存储和计算时间。为了让时钟每次增加 1 秒，Unix 只是简单地计算自 1970 年元旦以来的秒数。对时间表示法所做的任何修改都由与系统相链接的库函数或转换 UTC 和本地时间的应用程序来完成。

以这种方式存储时间的一个优点是基于 Unix 的系统上不存内在的 Y2K 问题。其缺点在于 Y2K 问题只是被简单地延后了，因为在 32 位的系统上，是用一个无符号的 32 位整数来存储自 UTC 时间 1970 年 1 月 1 日以来的秒数的，这个起始时间称为 Unix 新纪元，但是在 2038 年的某一天，自 Unix 新纪元开始以来的秒数将会大于 32 位的整数。

另一个优点是，系统工具(例如 `date`)或应用程序(例如 `ntpd`)能够即时设置本地时间，因此可以根据系统、应用程序或用户级别来调整用于显示本地时间的格式。考虑用户从不同的时区访问系统的情况。如果他愿意，可以设置环境变量 `TZ` 来调整所有的日期和时间，按照他所在的特定时区而不是系统所在的时区来显示它们。

系统时钟是系统的一个重要组成部分。如果没有它，很多程序和功能，例如自动控制将不能生效。设置系统时钟或硬件时钟需要超级用户(根)权限。在下面的示例中，使用 `sudo` 命令给需要权限的用户授予临时的超级用户身份，以便修改时钟设置。

11.1.1 使用 `date` 检查和设置系统时钟

要检查或设置系统时钟，使用命令行工具 `date`。如果没有参数，该命令将返回当前的日期和时间：

```
% date
Sun Oct 31 32:59:00 CST 2004
```

该命令还输出相关的时区。本例中，输出说明这个特定的系统时钟与中部标准时间同步。

假定要将当前的时间和日期设置为 2006 年 1 月 1 日，00:01:30，即新年过后的一分半钟，`date` 命令需要以 `CCYYMMDDhhmm.ss` 格式表示正确的参数，其中的 `CCYY` 是 4 位数字的年(2006)；`MMDD` 前两位数字表示月，后两位数字表示天(0101)；`hhmm.ss` 是两位数字的小时(24 小时表示法)，两位数字的分和两位数字的秒(0001.25)：

```
% sudo date 200601010001.30
Sun Jan 1 00:01:30 CST 2006
```

11.1.2 在 Linux 上利用 `hwclock` 同步时钟

有很多种方法可以设置给定系统的硬件时钟。一个常用的方法是在硬件的内置软件中设定其他初始化设置时配置时钟。然而，由于软件接口随硬件平台的变化而变化，因此另一种方法是首先设置系统时钟，然后同步硬件时钟和系统时钟设置。

`hwclock` 命令用于查询和设置硬件时钟。如果没有参数，该命令将显示当前设置：

```
$ /sbin/hwclock
Tue 30 Nov 2004 02:50:43 PM CST -0.600758 seconds
```

`hwclock` 命令没有位于用户的可执行程序路径中，因为许多系统，包括各种 Linux 发布版本，都将只能由根用户使用的命令(例如 `hwclock`)和其他程序放在称为 `/sbin` 的目录中。如前所述，设置硬件时钟是该命令的主要功能之一，它只能由超级用户账户，例如根用户使用。因此，要查询所设置的硬件时钟的日期，一个典型的 Linux 发布版本的用户需要包括 `hwclock` 命令的路径以及该命令本身。

要同步硬件时钟和系统时钟，`hwclock` 命令需要 `--systohc` 开关：

```
$ sudo /sbin/hwclock --systohc
```

如果硬件时钟设置为 UTC，那么完成该命令还需要一个额外的开关 `--utc`。

11.1.3 利用 NTP 同步系统时钟

另一个用于维持时钟秩序的常用工具是使用网络时间协议(NTP, Network Time Protocol), 利用该工具可以同步系统和外部系统。顾名思义, 这需要将系统连接到网络上。它在另一个运行 NTP 服务器的值得信赖的系统上寻找当前时间。而且, 如果系统总是在启动时就具有网络连接, 那么系统可能会跳过与硬件时钟的同步, 使用 `ntpdate`(通常包含在 Linux 系统上的 `ntp` 软件包内)从一个基于网络的时间服务器来初始化系统时钟。它并不考虑时间服务器是远程的还是位于本地网络上。

下面是使用该工具的一个示例:

```
$ sudo ntpdate
10 Dec 19:35:39 ntpdate[3736]:step time server 17.254.0.27 offset -11.081510 se
```

`ntpd` 程序是通过网络同步时钟的标准方法, 它提供一组公共的时间服务器, 系统可以使用 NTP 对它们进行连接。NTP 不仅提供了用于传输当前时间的方法, 还考虑到传输中的延迟, 提供了额外的精确级别。另外, NTP 使用信任级别(trust level)来确定具有权威时间的服务器。数字越小, 可信度越高, 因此级别为 1 的时钟是最好的, 因为它是由精确且值得信赖的时间源设置的。大多数系统——Apple、Linux、Windows——都提供了一个或两个公共 NTP 服务器, 如果没有其他的本地或网络服务器可用, 系统可以与它们同步。这些服务器的级别通常是 2。

要使用 NTP 同步系统时钟, 首先需要找到一个或多个可用的 NTP 服务器。可以咨询网络管理员或者服务器供应商, 他们可能已经安装了本地 NTP 服务器。有很多公开访问的 NTP 服务器的联机列表, 例如 <http://ntp.isc.org/bin/view/Servers/WebHome> 上的列表。最坏情况下, 可以检查 Unix 系统供应商是否提供了一个可以连接的公共 NTP 服务器。用户一定要了解任何服务器的使用政策, 如果需要的话, 可以申请相应的权限。

在许多 Unix 类型的系统上, 在 `/etc/ntp.conf` 文件中配置 NTP:

```
server ntp.local.com prefer
server time.apple.com

driftfile /var/db/ntp.drift
```

`server` 选项指定将要使用的服务器, 每个服务器自成一列。如果为服务器指定 `prefer` 选项, 像 `ntp.local.com` 那样, 那么首选该服务器。系统使用首选服务器的响应, 除非该响应与所列的其他服务器的响应之间有很大的差别。

如果需要的话, 可以在 `ntp.conf` 文件中提供额外的信息。例如, `driftfile` 选项指定一个文件, 用以存储系统时钟的频率偏移信息。如果切断 `ntpd` 与所有外部时间源的联系, 并持续一段时间, 那么 `ntpd` 使用该信息来自动弥补时钟的自然偏移。

任何 Unix 系统都可以作为使用 `ntpd` 的其他系统的一个 NTP 服务器。

11.2 安排将来运行的命令

为了将系统时钟用于自动操作任务, 所有 Unix 系统都提供了两个关键的工具——`cron` 和 `at`——用于在指定的时间运行命令、应用程序和脚本。

11.2.1 利用 cron 执行程序

cron 程序使得 Unix 用户能够在指定的时间和/或日期执行命令、脚本和应用程序。cron 使用一个 Unix 守护进程(称为 crond)，该进程只需要启动一次，通常在系统启动时运行，并且一直处于潜伏状态，直到需要用到它时为止。什么时候需要 cron 守护进程呢？在从某个配置文件或 crontab 运行指定的命令、脚本或应用程序时需要用到 crond 守护进程。

1. 启动并使用 cron

要从命令行启动 cron，简单地输入下面的命令：

```
$ sudo crond
$
```

该进程自动转到后台运行，与大多数应用程序不同，不需要对 cron 使用 & 参数以迫使它转到后台执行。

然而，在大多数 Unix 发布版本上，crond 是自动安装的，将在系统启动时运行。使用 ps(process status) 命令验证该守护进程正在运行：

```
$ ps auwx | grep cron
root      2068  0.0  0.0  2168  656 ?        S   Nov02   0:00 crond
pdw       24913 0.0  0.0  4596  568 pts/3    R   02:08   0:00 grep cron
```

一旦 cron 守护进程启动并开始运行，它将具有广泛的应用。

实战

安排重要文档的备份时

Unix 系统上的每个用户都拥有一个主目录，用于存储个人的文档、应用程序和系统选项。这个目录中的信息非常重要，不能对其进行替换，许多用户都竭尽全力地维护它的安全。在保证重要数据的安全性方面，一个重要的步骤是保存备份。当然，即使没有任何干扰的情况下，要记住按时备份所有的内容也是富有挑战性的，更不要说在工作或家庭中存在着大量令人分心的事情了。将 cron 用在这种地方是最合适的，不是吗？

按照下面的步骤为某个目录安排备份：

(1) 观察主目录，确定重要的文档及其位置。取决于个人的习惯，该目录可能看起来类似于：

```
$ ls
bin Desktop docs downloads Mail tmp
$
```

(2) 从含有信息的目录中选取一个，用以进行这个练习。docs 是一个很好的选择——它的内容通常都需要备份——本例中使用该目录。

(3) 在主目录中创建一个新的子目录，称为 backups。

```
$ mkdir backups
$
```

(4) 在文本编辑器中打开 cron 配置文件/etc/crontab(要打开并编辑该文件，用户需要超级用户权限)。

```
$ sudo vi /etc/crontab
```

该文件的内容类似于：

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# Adjust the time zone if the CMOS clock keeps local time, as opposed to
# UTC time. See adjkerntz(8) for details.
1,31 0-5 * * * root adjkerntz -a

# run reports on network usage
*/5 * * * * root /usr/local/bin/mrtg /usr/local/etc/mrtg/mrtg.conf

# routine maintenance for Weinstein.org
0,30 * * * * cd /home/pdw/public_html/weinstein.org; make tabernacle 2>&1
>/dev/null

# seti@home client for pdw
0 1,5,9,13,17,21 * * * cd /usr/local/bin/setiathome; ./setiathoe -nice 19 >
/dev/null 2> /dev/null
```

(5) 在这个文件的末尾添加如下内容：

```
# Copy docs directory to a local backup directory every hour
# Entered by (your name) on (today's date). Version 1.0
0 * * * * userid cp -r ~/username/docs ~/username/backups
```

在下一个小时开始时转入 **backups** 目录并查看详细的目录列表：

```
$ ls -l
total 1
drwxr-xr-x 0 userid userid 4096 Feb 8 14:00 docs
$
```

工作原理

在文本编辑器中再次打开 **/etc/crontab** 以了解 **cron** 的作用。前几行相当简单，它们为 **cron** 设置环境，以便 **cron** 进行工作。

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
```

下面是对每个记录项的解释：

- **SHELL**——**cron** 将在其下运行的 shell。
- **PATH**——包含在 **cron** 搜索路径中的目录。系统上安装的大多数程序都位于这些目录中，这些程序可能在指定时间调用 **cron**。

- MAILTO——如果 cron 正在运行的命令有输出，那么将每个命令的输出以电子邮件的形式传送给指定用户。如果没有指定用户，那么将输出以电子邮件的形式传送给产生该输出的进程的所有者。
- HOME——为 cron 应用程序指定的主目录。

crontab 文件的第二部分定义了时间和内容的状况，其细节在一系列由空格分隔的字段中设置。一个记录项中通常有 7 个字段：

```
0 * * * * userid cp -r ~/username/Mail ~/username/backups; cp -r ~/username/docs
~/username /backups
```

每个记录项的前面通常都有注释行(以#开头)，用于描述或说明该记录项。这些注释行将在接下来的“适当地注释 crontab 文件”一节中讨论。

下面是各个字段的概要：

(1) 分钟：命令将在每小时的第几分钟运行。该字段中的取值范围是 0~59。本例在整点运行，因此 minute 的值为 0。

(2) 小时：命令将在每日的第几小时运行。使用 24 小时表示法指定该记录项。也就是说，其值必须在 0~23 之间，其中 0 表示午夜。星号(*)代表每个小时，如本例所示。

(3) 日期：命令在每月的第几天运行(1~31)。例如，如果在每月的第 19 天执行命令，那么该字段的值为 19。星号(*)代表每一天，如本例所示。

(4) 月份：指定的命令将在每年的第几月运行。该字段的值可以使用数字 1~12 来指定，也可以使用月份的名称，Jan~Dec 来指定，这里只使用前 3 个字符(不区分大小写，sep 与 Sep 是一样的)。星号(*)代表每个月，如本例所示。

(5) 星期：命令在每周的第几天运行。其值可以是数字 0~7，也可以是每天的名称，Sun~Sat，这里只使用前 3 个字符(不区分大小写，thu 与 Thu 是一样的)。如果用数字表示，那么 0 和 7 都代表星期日。

(6) 用户：运行该命令的用户。

(7) 命令名称：所要调用的命令。该字段可以包含多个单词或命令——只要是能够完成作业的指令就行。

2. 适当地注释 crontab 文件

在任何配置文件中，注释都是一个有用的特性，应该尽可能地充分利用它。注释使得用户能够解释在文件中添加或修改的内容。注释以符号#开始——#右边位于同一行上的所有内容都是属于注释。因此，尽管可以在一行的任意位置开始一个注释，但是最好将注释本身放在单独的行上，如前一节的示例所示：

```
# Copy docs directories to a local backup directory every hour
# Entered by (your name) on (today's date). Version 1.0
```

注释应该传达一些重要的信息，包括：

- 添加/修改记录项的用户的名字。
- 如果出现问题，应该如何联系该用户。
- 添加或修改记录项的日期。

- 记录项(或修改)的目的, 指定它所作用的文件。

请记住, 只有最新的注释才是有效的。最新的注释有助于构建其他安排好的程序、理解普通的管理例程、帮助灾难恢复以及完成其他事情。运行计算机系统可能是一个复杂的作业, 但是当所有的作业都具有适当的注释时, 这项工作就会变得容易一些。

3. 构建复杂的调度

如果复制命令需要在多个小时执行, 但又不是每个小时都执行, 应该怎么办呢? 不用担心, 可以在一个字段中安排多个事件, 用逗号将所有的实例隔开, 之间没有空格。下面的示例将在每隔一小时的第一分钟运行脚本, 从午夜开始:

```
1 0,2,4,6,8,10,12,14,16,18,20,22 * * * userid cp -r ~/username/docs ~/username/backups
```

如果指定了日期和星期字段, 那么任何一个事件发生时都将执行该命令。本例将在每个星期一和每月 16 号的正午运行该命令:

```
* 12 16 * 1 userid cp -r ~/username/docs ~/username/backups
```

它产生的结果与下面两个记录项结合在一起产生的结果相同:

```
* 12 16 * * userid cp -r ~/username/docs ~/username/backups
* 12 * * * 16 userid cp -r ~/username/docs ~/username/backups
```

如果希望只在周一到周五调用备份, 可以输入:

```
* * * * 1-5 userid cp -r ~/username/docs ~/username/backups
```

除了为时间值提供取值的列表(如第一个复杂的示例)或范围(如最后一个示例)之外, crontab 文件语法还提供了一种输入步值的方法。可以将步值看作走石块(stepping stone)。在一条路上, 人们可以每块石头都走, 也可以隔一块走一块。例如, 如果一个备份重要数据的 shell 脚本在一天中每隔 5 小时运行一次, 那么可以输入步值 5。本例中, 相应的语法是 “*/5”, 其中 “*” 代表每个可能的小时, “/5” 代表每次间隔的小时:

```
* */5 * * * userid cp -r ~/username/docs ~/username/backups
```

而且, 还可以将步值和列表结合起来。因此, 如果在每月的中间几天每隔一天调用一次备份脚本, 可以使用类似于下面的命令, 该命令提供了每月中间几天的日期范围(10~16)和步值 (/2), 步值表示再次执行该命令之前要前进两天:

```
* 12 10-16/2 * * * userid cp -r ~/username/docs ~/username/backups
```

该命令与下面的列表示例作用相同:

```
* 12 10,12,14,16 * * * userid cp -r ~/username/docs ~/username/backups
```

4. Linux 上的报告和日志更新

cron 的另一个常见的用法是管理系统日志并产生 productive 使用报告。例如, 在大多数 Linux 发布版本的 4 个 /etc 子目录下有许多生成报告的脚本, 这 4 个子目录分别是: cron.hourly、cron.daily、cron.weekly 和 cron.monthly。cron 运行这些目录下的脚本, 周期分别是每小时、每

天、每周或每月。

下面是 Red Hat 企业 Linux 系统上/etc/ cron.daily 的一个目录列表示例：

```
$ ls
00-logwatch      inn-cron-expire  rhncheck        sysstat
00webalizer      logrotate       rpm             tmpwatch
0anacron         makewhatis.cron slocate.cron    tripwire-check
$
```

00webalizer 是一个 shell 脚本示例，用于处理 Web 服务器日志，它可能类似于：

```
#!/bin/bash
# update access statistics for the Web site

if [ -s /var/log/httpd/keplersol.com-access_log ] ; then
    /usr/bin/webalizer -cQ /etc/webalizer.keplersol.conf
fi

if [ -s /var/log/httpd/weinstein.org-access_log ] ; then
    /usr/bin/webalizer -cQ /etc/webalizer.weinstein.conf
fi

exit 0
```

第 13 章将介绍 shell 脚本的工作原理以及阅读和创建脚本的方法，而在这里，只需要知道该脚本验证了日志文件的存在，然后利用为 webalizer 程序创建的配置文件处理这些日志，产生可以在特定位置查看的报告，配置文件中预定义了这些位置。

下面的 crontab 记录项说明这些程序报告需要使用根用户权限来执行。也就是说，运行这些脚本就好像根用户调用 run-parts 命令一样，该命令的参数指定将要执行的脚本所在的目录。run-parts 命令将运行指定目录中的所有脚本。

下面是一个含有 4 个日志子目录的 crontab 文件的示例：

```
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

可以看出，每月都处理 cron.hourly、cron.daily、cron.weekly 和 cron.monthly 目录(月份字段中是*)。只在每月的第一天，上午 4:42 处理 monthly 目录。在星期日执行 weekly 目录脚本(星期字段为 0)。daily 目录在每天上午 4:42 运行(日期字段为*)。/etc/cron.hourly 中的脚本在每月的每星期的每天的每小时(所有这些字段都是*)的第一分钟运行。

5. 管理 cron 的输出

如前所述，cron 的输出以电子邮件的形式传送给进程的所有者，或者传送给在 MAILTO 变量中指定的用户。然而，如果需要将输出以电子邮件的形式发送给其他人，可以使用 Unix 管道，将输出通过管道传给 mail 命令。请注意，调用 mail 命令时要使用-s 开关，这个开关将所要发送的电子邮件的主题设置为在开关后提供的字符串：

```
0 * * * * userid cp -r ~username/docs ~username/backups | mail -s "Backup Script Output"
```

username

输出还可以重定向到日志文件：

```
0 * * * * userid cp -r ~username/docs ~username/backups >> log.file
```

本例将复制命令的输出重定向到一个日志文件，这在有些实例中是有用的。

6. 控制对 cron 的访问

管理员能够指定有权和无权使用 cron 的用户。通过使用/etc/cron.allow 和/etc/cron.deny 文件来实现这一点。这些文件的工作方式与其他守护进程的 allow 和 deny 文件相同。要阻止一个用户使用 cron，只需要将他的用户名放在 cron.deny 文件中；要让一个用户能够使用 cron，将他的用户名放在 cron.allow 文件中。要阻止所有用户使用 cron，在 cron.deny 文件中添加 ALL 行。

如果既没有 cron.allow 文件也没有 cron.deny 文件，那么 cron 的使用将不受限制。

crontab

由于 Unix 是一个多用户操作系统，许多应用程序，如 cron，必须能够支持多个用户。出于安全的考虑，授予所有用户访问/etc/crontab 的权力是不明智的，因为他们可以轻易看到正在运行的关键程序以及它们的运行时间。让每个用户拥有自己的 crontab 文件要好得多，可以使用 crontab 命令创建、编辑和删除这些文件。该命令创建个人的 crontab 文件，它通常存储在 /var/spool/cron/crontabs/user，但是也可能在其他地方(/var/spool/cron /user 或者/var/cron/tabs/user)找到该命令，这取决于所用的 Unix 风格。

表 11-1 列出了 crontab 的命令参数。

表 11-1

参 数	说 明
-e	编辑当前的 crontab 文件或者创建一个新的 crontab 文件
-l	列出 crontab 文件的内容
-r	删除 crontab 文件

命令 crontab -e 将用户的 crontab 文件加载到 EDITOR 或 VISUAL 环境变量所指定的编辑器中。

用户的个人 crontab 文件完全遵循主要的/etc/crontab 文件的格式，只是不需要指定 MAILTO 变量，因为默认的记录项是进程的所有者，通常指用户本身。

11.2.2 使用 at 命令进行一次性执行

要安排一个命令、一系列命令或者脚本在指定时间运行，而且只执行一次，那么可以用 at 命令代替 cron。at 命令是安排事件的一次性执行的方法，通常用于在确定的环境下安排特定的作业。换句话说，如果需要在每个周六更新日志，那么 at 是不适合的——这是 cron 的工作——但是，如果在周四下午为周五上午安排了一个会议，并且希望确保所需要的关键文档不会发生——损伤，那么用户可能希望在一天的繁忙工作结束之后能产生一个备份副本，这时用 at 来完成这个作业是非常合适的。要了解 at 的工作原理，可以选择一个文档并试用该命令。

(1) 定位文档并检查当前的日期，如下所示：

```
$ ls
important_report.swx
$ date
Thu Dec 9 14:53:06 CST 2004
```

(2) 输入 **at** 命令，后跟用户所希望的执行作业的时间：

```
$ at 17:00
```

(3) 输入复制命令(**cp**)、希望复制的文件名以及备份副本在文件系统中的位置、然后按 **Enter**：

```
at>cp important_report.swx ~username/backups
```

(4) 按 **Ctrl+D** 以退出 **at shell**；用户将看到类似于下面的内容：

```
job 1 at 2004-12-10 17:00
$
```

使用 **atq** 命令再次确认该作业已经安排好。

```
$ atq
1      2004-12-10 17:00 a username
$
```

要在执行某个命令之前将其删除，使用 **atrm** 命令：

```
$ atrm 1
```

工作原理

第一步是定位正在讨论的文档。这是因为从调用时开始，**at** 保留了工作目录、**shell** 环境——除了变量 **TERM** 和 **DISPLAY** 以及 **umask** 以外。

at 从标准输入或指定的文件中读取一个命令或者一系列命令。在 **at** 指定的将来某个时间执行 **at** 所收集的命令。使用用户的环境变量 **SHELL** 或 **/bin/sh** 所设置的任一个 **shell** 来执行这些命令。

at 允许指定相当复杂的时间。在最基本的情况下，**at** 接受以 **HH:MM** 格式表示的时间。如果今天的这个时间已经过了，那么假定是第二天，因此不需要指定日期。小时可以是 0~24，也可以是 0~12a.m./p.m.(数字和 **am/pm** 之间没有空格)，用户还可以指定午夜、正午，或者对(大不列颠)联合王国而言，指定下午茶时间(4 p.m.)。下面是各种 **at** 时间的一些示例：

```
$ at 17:45
$ at 5pm
$ at 5:15pm
$ at noon
$ at teatime
```

用户可以通过以 **MMDD** 或者 **MMDDYY** 格式给定一个日期，从而指定作业运行的日期。日期说明必须遵循下面的格式：

```
$ at teatime 010105
```

所指定的时间还可以与一个特定时间有关。也就是说，用户可以使用倒计时类型的参数，以分、小时、天或日期为单位，说明距离执行作业的特定时间还有多久。下面的示例将在调用 `at` 命令 5 分钟后执行某个命令：

```
$ at now + 5 minutes
```

或者从现在开始 2 小时后运行某个作业，可以使用：

```
$ at now + 2 hours
```

要在从现在开始 3 天后的 4p.m. 运行一个作业，可以使用：

```
$ at 4pm + 3 days
```

或者从现在开始 1 周后运行某个作业，可以使用：

```
$ at 1 week
```

无论是哪种情况，创建一个事件的下一步都是为 `at` 提供一个或一系列在特定时间执行的命令。回忆第一个示例，其中用于复制重要文档的命令是从命令行——也称为标准输入——输入的：

```
at> cp important_report.swx ~username/BACKUPS
```

命令也可以来自于文件。要做到这一点，需要使用 `-f` 开关，后跟存放命令的文件：

```
$ at 17:00 -f ~jdoe/binbackup
job 3 at 2004-12-10 17:00
$
```

控制对 `at` 的访问

与 `cron` 一样，Unix 管理员能够控制哪些用户有权使用 `at`。访问控制的一种方法是简单地创建一个 `/etc/at.allow` 文件。使用这种方法，`at` 守护进程将权限限制在 `at.allow` 文件中列出的那些用户，而其他任何人都没有权限。

访问控制的另一种方法是创建 `/etc/at.deny` 文件，大多数 Unix 系统默认使用该方法。`/etc/at.deny` 文件的工作方式与 `/etc/at.allow` 相反。它控制哪些用户不能使用 `at`——该文件中列出的用户无权使用 `at`，而其他任何人都可以使用该命令。

使用哪种访问控制方法，以及使用 `/etc/at.allow` 或 `/etc/at.deny` 中的哪个文件取决于用户的管理风格。作为管理员，如果希望每个人都可以使用这个命令，直到他们滥用该权限时才对他们进行限制，那么使用默认的 `/etc/at.deny` 文件。如果用户认为除非直接授予该权限，否则没有人需要使用 `at` 命令，那么创建并使用 `/etc/at.allow` 文件。

如果两个文件都不存在，那么只有超级用户有权使用 `at`。要允许所有人使用 `at`，管理员需要创建一个空的 `/etc/at.deny` 文件(管理员请注意：这通常是默认的配置)。

11.3 小结

截至目前，用户已经学习了系统自动化的两个基本部分：拥有一个有效的系统时钟设置和利用配置信息运行 `cron` 守护进程。请记住，设置命令、脚本或应用程序在特定时间运行时，有很多事情需要切记：

- 系统时钟是操作系统的时钟。
- 硬件时钟对于在启动时设置系统时钟很重要。
- 有很多方法可以保持系统时钟同步，包括与网络上的其他计算机同步。
- `cron` 是一个 Unix 应用程序，它记录何时运行所安排的任务。
- `crontab` 是 `cron` 守护进程的配置文件。
- `crontab` 还是一个命令工具的名字，该工具为单独的用户编辑 `crontab` 文件。
- `at` 是一个 Unix 应用程序，它记录何时一次性执行某个特定的任务。

11.4 练习

为每个用户的 `crontab` 文件创建一条记录项，它将列出该用户的目录并将输出以电子邮件的形式传给用户，该操作在一周中每隔一天执行一次。

第 12 章 安 全 性

面对越来越多对计算机系统的威胁，信息安全已经成为计算机领域的一个重要课题。这些危险包括外部的恶意攻击(从间谍到黑客和解密高手)，并且，如果系统上存在多个用户，还包括内部的安全隐患(从不满意的雇员到不道德的访问者)。硬件、软件、网络——其中都存在着大量的漏洞。Unix 的安全性与它的技术一样出色。本章将介绍主要的安全性概念和一些命令，用户可以使用这些命令让自己的系统更安全。

12.1 安全性的基础知识

计算机安全领域中有一个古老的谚语，是说绝对安全的计算机是断开任何网络的、关闭并嵌入混凝土块中的计算机。当然，这是一种夸张的说法，但是它反应了一个事实，即没有系统是百分之百安全的。安全性不是通过选中系统中的一个复选框就可以实现的，也不是简单地安装一个软件程序就可以高枕无忧了。安全性是一个不断发展的过程，而且可能是一个令人着迷的过程。

计算机的安全与现实世界中的安全非常类似。您可能在住宅的前门安装了一个或多个锁，以阻止陌生人进入自己的房子或使用自己的所有物。尽管您已经在努力地保护自己的家，但是不难想象一种会令安全措施失效的情况——例如，发生火灾时，消防人员需要进入您的房子，他们或许会通过暴力来达到目的。您可能将自己的积蓄存在银行里。还可能将自己的贵重物品存放在银行的贵重物品保险箱里，这比家里要安全得多。所有这些都类似于计算机的安全性，因为您决定了哪些需要最多的保护，哪些需要比较少的保护。

保护信息安全系统有 3 个基本原则：

- 机密性——必须阻止那些不需要知道的人了解信息(保持私有信息的私密性)。
- 完整性——信息必须避免未经授权的修改或污染。
- 可用性——对那些需要访问信息的人，信息必须是可用的。

12.1.1 资产价值保护

在开始保护系统之前，用户需要了解所要保护的内容。尽管每个人都拥有不同的特殊数据类型，具有不同的需求，但是创建一个非常通用的资产列表是可能的，该列表适用于公司或组织中的所有计算机和用户。请花费一定的时间来识别并定义这些资产。

保护系统免受攻击所花费的时间、努力和金钱应该基于与硬件、软件和信息资源的损失相关联的代价。例如，如果受到攻击将导致价值\$100 的损失，那么用户不会花费\$10 000 美元来保护系统。一旦用户了解了所保护资产的价值，就可以确定适当的资源级别，从而以一种有意义的、明智的方式来保护这些资产。

请花费一定的时间来识别并定义这些资产：

- **硬件**——计算机系统的物理组件是有价值的。如果系统位于一个容易进入的地方，没有任何物理保护措施，例如上锁的门，那么系统可能被偷走。如果有人能够接触到计算机系统，那么他可以访问系统所保存的所有信息，因为系统可以从其他的文件系统(CD-ROM、USB 等)启动，而忽略主要的操作系统。攻击者可能插入网络监控软件以捕捉在网络上传输的所有信息，例如 keylogger(捕捉用户的每个按键的秘密软件)或者许多其他的攻击方式。攻击者还可能简单地接管计算机，删除硬盘驱动器以获得对系统保存信息的访问权。
- **网络连接**——网络连接是一种非常宝贵的资源。攻击者通过对其他系统的各种攻击以占用带宽(和网络容量)。他们还能够利用其他用户的连接，有效地使用其他用户在 Internet 上的身份(至少隐藏自己的身份)，使得在外面的所有人看来，攻击都是来自于该用户的位置。
- **数据**——用户使用计算机所做的一切事情都存储在系统中：用户的电子邮件、地址列表、私人文件以及用户亲自创建或修改的其他内容。许多机密的文档都存放在用户的计算机中。可以想象，如果用户不得不重新创建硬盘上的所有内容是一件多么困难的事情。还有另外一种危险，就是有人可能使用这些信息来假冒用户的身份，访问用户的财政计划程序和电子邮件。
- **服务**——无论是在工作的部门中还是在家中，如果用户使用自己的计算机与任何其他用户共享信息，很重要的一点是系统必须提供连续的、不间断的服务。例如，如果用户经营家庭商务，依赖于 Web 服务器将产品销售给客户，就需要一直运行该 Web 服务器，否则将损失业务。

12.1.2 潜在的问题

从病毒到数据窃取再到拒绝服务攻击，在计算机系统中有许多情况可能造成破坏。为了尽量避免并最小化其他事件的影响，了解潜在的危险以及如何处理它们是有帮助的。

下面是可能发生的故障类型，以及那些希望避免并最小化其影响的故障类型：

- **用户过失**——到目前为止，数据丢失的最常见的原因是由于用户过失，特别是在 Unix 系统上，一个误放的空格或字符就可能导致严重的系统修改，而且没有一些其他操作系统默认提供的确认信息(Unix 最初是由程序员设计的，并且是为程序员设计的，倾向于在 shell 层使用简洁的命令语法，shell 通常假定用户熟悉运算符)。
- **硬件故障**——计算机是机械设备，尽管人们通常关注于软件和操作系统的安全，但是物理组件对系统操作而言也是必需的。电源和硬盘是两个可以破坏的组件。如果其中任何一个停止工作，那么结果可能是系统部分或全部不可用。
- **硬件失窃**——如前所述，保护物理计算机是非常重要的。如果机器没有放在安全的地点，那么它可能被偷走。如果有人可以轻易地从系统中取走硬盘并将它安装在自己的系统上，那么世界上所有的软件安全措施都不能保护您的数据。
- **数据操作**——如果获得对系统未经授权的访问，那么该系统上的数据可能被删除、修改或复制。如果用户通过 Web 提供信息，那么 Web 页上的数据可能被修改或删除。私有文件可能被复制并与用户的竞争者共享，或者其中的数据可能被简单地删除。
- **服务失窃**——如果用户的系统已被攻破，那么其他人将可以访问用户的网络带宽。例如，黑客可能在系统上创建一个文件夹；在其中存放各种带有版权的资料，如计算机

软件、音乐和电影；在用户的系统上安装一个 FTP 服务器；然后广告这些软件。在用户毫不知情的情况下，用户的机器将为不知详情的访问者提供服务。

- **窃听**——在使用计算机与其他系统通信时，如通过电子邮件或 Internet 进行通信，第三方或许能够窃听。这在许多方面都是有害的，特别是发布机密信息时，例如财政信息和鉴定信息，包括在其他系统上的口令和用户名。
- **病毒**——还存在其他的危险，例如病毒、间谍件、垃圾邮件以及其他在 Internet 上传播的恶意软件。在大多数情况下，这些软件都不会影响 Unix 系统。但是，在处理包含附件的邮件，特别是来自于未知实体的邮件时，一定要知道潜在的危险，用自己的常识进行判断。但是，由于 Unix 的安全特性以及它多变的实现，在 Unix 中遇到病毒的风险要比在其他操作系统中小得多。

12.2 保护 Unix 系统

本章其余的内容将帮助用户尽可能安全地运行机器，无论用户是机器上惟一的用户，还是维护一个多用户共享的系统。将安全性看作一个目标，而不是一个目的——它决不是一劳永逸的事情。如果在使用系统时总是考虑到安全性，那么用户将取得这场竞争的主动权。

12.2.1 口令的安全性

Unix 主机系统使用的基本验证方法是用户名和口令组合。实际上，由于通常在电子邮件的地址中使用用户名，因此大家都知道它。用户名通常是由系统提供的，系统通过一些程序，例如 who 将它传给本地用户，通过一些命令，例如 finger 和 rwho 将它传给远程用户。这就要求用户选择合适的口令，保证口令的隐秘并定期修改它们。口令是在操作系统级别上防止入侵者进入系统的第一道防线。有一些常用的专门试图猜测口令的程序。

怎样的口令才算是一个好口令呢？一个好口令是各种元素的结合。举例来说，一个好口令应该是极难猜测的，或者说是很难用口令破译工具破解的。这意味着不要使用任何个人的可识别性信息，例如生日或电话号码、宠物名、兄弟姐妹的名字、父母的名字——事实上，不要使用任何名字，包括运动队、商标名等等。用户还应该避免使用字典中出现的单词，以及特殊行业内的行话——特别是与用户有关的行业。不要使用常用的缩略词。一个好的口令应该由大写字母和小写字母以及数字和标点(非字符和数字)组成。

一个普遍适用的建议是在一个熟悉的惯用语中提取每个单词的首字母——例如，“Now is the time for all good men to come to the aid of their party”——并将这些首字母用大写字母表示，构成口令的基础：NITTFAGMTCTTAOTP。然后取出该序列的一部分——至少使用八个字符——并将其中的一些字母替换成数字和非字符和数字的符号。本例使用序列开头的字母，NITTFAGM，并将 I 替换成 1、F 替换成 4，得到 NITT4AGM。改变字母的大小写，如 nltT4aGm，并添加标点符号以得到最终的口令 nltT4-aGm&(当然，既然已经公布了这个口令，那么就不应该再使用它了，但是它是一个很好的示例，说明了创建安全口令的过程)。

无论用户创建了多少安全的口令，都一定要记住它。不要将口令写下来。再重申一遍：千万不要在任何地方写下您的口令。将口令写下来并把它放在计算机附近，与一开始就使用不安全的口令一样，将极大地降低系统的安全性。如果必须写下口令，那么一定要把它放在安全的地方，最好随身携带，尽管如此，还是强烈建议用户不要将口令写下来。

12.2.2 口令破译程序

通过使用多个字典和常用序列试图猜出口令的软件已经存在一段时间了。该软件一个接一个地尝试许多口令，直到它发现匹配的口令为止。这种技术称为暴力攻击。随着计算机的处理速度变得越来越快，该软件在破译口令时也变得越来越复杂(和快速)。过去破译一个类似于 theSafe9a55w0Rd 的口令需要几年时间或者根本不能破解，而现在只需要几天时间就可以实现。这也是频繁修改口令以获得最佳安全性的原因。通过频繁地修改口令，用户很有可能在攻击者已经破译口令但是还没有机会使用它之前将口令改掉。

两个最流行的口令破译程序是 Crack 和 John the Ripper。John the Ripper 是一个更加现代化的实现，它能够在几乎所有的现代化操作系统上进行编译。两个程序的操作方式类似，都是破解系统口令文件(通常是/etc/passwd 或/etc/shadow)，尝试各种口令直到找到一个匹配的为止。系统管理员和安全专家通常利用这些工具来测试用户账户的口令安全性。如果用户决定使用这些工具来审计系统口令，那么必须首先获得系统所有者的书面授权，否则用户可能会面临严重的指控或者民事处罚。

对于本章介绍的所有工具，如果没有完全了解使用它们的含义，不明白未经系统所有者的适当授权而擅自在系统上使用这些工具的潜在法律后果，就不要使用它们。这些软件程序的价值在于它们使得用户能够在攻击者之前测试潜在的系统弱点。用完之后要将这些工具删除，从而未经授权的用户就不能利用它们来攻击系统。

如果从系统所有者处获得了运行 John the Ripper 的适当权限，那么用户可以从 <http://openwall.com/john> 下载该程序。从该 Web 站点上可以获得有关安装和使用这个程序的完整说明。

12.3 限制管理访问

正如第 3 章所述，根用户——超级用户——拥有对系统的绝对控制权，包括随意修改其他用户的账户或系统。这个账户的至高权利决定了必须为它提供比其他账户更为高级的保护。由于根用户或超级用户的不受限制的权利，因此在恶意实体攻击系统时，他们总是试图获得对该账户的访问权。

12.3.1 UID 0

根用户的 `userid(UID)` 是 0。应该为根用户账户提供最高的安全性，包括最安全的口令。如果这个账户被攻破，那么至少对于数据存储，该系统是不可靠的，在最坏的情况下，对于遭到攻击的实体的全部领域都是不可靠的。为了保护系统，请用户只在绝对必要的时候，如用于系统管理时才使用根账户。从最初的系统启动开始，用户最好不要直接登录到根用户账户。取而代之，用户可以作为普通(无特权的)用户登录，然后在需要根账户时使用 `su` 命令切换到根账户。例如，要作为根用户查看/etc/shadow 中加密的口令字符串，可以使用 `su` 命令切换到根用户，运行需要的命令，然后退出根用户账户，如下所示：

```
jdoe@ttypl[~]$ su - root
root@ttypl[~]# cat /etc/shadow | grep root
```

```
exit
logout
jdoe@ttypl[~]$
```

通过不将根账户用于日常操作，如 Web 浏览或文本编辑器，用户可以避免作为根用户运行从 Internet 下载的程序，这可能导致系统被攻破。用户还要避免会将根用户账户暴露给恶意实体的各种情况。

另外，在普通用户的基础上查看/etc/passwd 可以确保只有真正的根用户才拥有 UID 0(字段 3)。例如，假定用户在/etc/passwd 上运行 cat 并显示下面的内容：

```
root:x:0:0:root:/root:/bin/bash
badguy:x:0:100:/home/badguy:/bin/bash
```

根据 root 和 badguy 的 UID(0)可知这两个账户都拥有完整的根用户权限。如果发现根以外的任何用户拥有 UID 0，请立刻调查该实体的合法性，因为他可能是已经获得对系统的访问权的潜在恶意黑客。

需要考虑的另一个安全措施是不允许用户从远程控制台直接作为根用户登录到系统。每个 Unix 系统处理这一点的方式不同，请用户参考系统文档以获得有关禁止远程根用户登录的信息。

12.3.2 根用户管理选项

传统上，使用 su(switch user)命令完成管理任务，该命令使得用户能够通过输入根账户口令变成根用户。但是由于多种原因，并不提倡这种做法。最重要的是，当系统拥有多个用户，并且所有用户都熟悉根用户口令时，根账户通常就没有责任可言了。如果其中一个用户使用 su 变成根用户，然后执行某个命令，那么系统日志将简单地把这个操作记录为根用户执行的命令，通常根用户的名字都比较长：System Administrator。在出现差错并试图查出登录到系统的用户以及执行特定命令的用户时，这是很不利的。

sudo 或(SuperUser Do)命令可以帮助用户解决这个问题，该命令使得用户能够追踪特权命令的使用，并且为了维护系统的安全，它提供了一些保护措施，防止将根用户口令透露给其他人。它允许灵活地管理任务授权，使得管理员能够授权个别用户以 UID 0 的身份执行某些具体的操作。用户通过自己的用户名使用 sudo。只需要简单地将 sudo 放在希望执行的命令前，例如 sudo shutdown。然后系统提示用户输入他们自己的口令，并执行相应的操作。系统日志记录用户名(如果已经配置了日志)，包括执行命令的用户 ID。无论访问是否成功，日志都将其记录在案。

sudo 远远优于 su，其原因是多方面的，特别是：

- 增强的日志——使用可配置的 syslogd 详细地记录了每个操作(使用命令参数)，包括用户名。
- 灵活的配置——sudo 使得管理员能够将管理访问权授予个别用户和组。
- 细粒度访问控制——sudo 使得管理员能够授权个别用户和组只访问所选择的具体的管理命令。

12.3.3 设置 sudo

大多数 Unix 版本都包含 `sudo`，但是如果用户没有相应的程序包或源代码，可以从 <http://courtesan.com/sudo/> 下载最新的版本。在安装了 `sudo` 之后，用户通常会拥有 3 个文件，它们的位置取决于安装 `sudo` 的方式：

- `sudo`——用户使用 `sudo` 命令时调用的二进制程序。该程序在二进制形式下是不可修改的。
- `visudo`——二进制程序，该程序使得用户能够安全地编辑 `sudo` 配置文件 `sudoers`。
- `sudoers`——普通的文本文件，用户编辑该文件以修改或添加权限。
- 在调用 `sudo` 时，该程序读取 `sudoers` 文件，这个文件通常位于 `/etc` 下。由于 `visudo` 会锁定进行编辑的文件，并且在提交修改之前对文件执行基本的语法检查，因此用它来编辑 `/etc/sudoers`。

下面是一个基本的 `/etc/sudoers` 文件：

```
# sudoers file.
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the sudoers man page for the details on how to write a sudoers file.
#

# Host alias specification
Host_Alias      LINUX_SEJWERS=linux1,linux2, 192.168.1.3
# User alias specification
User_Alias      ADMIN=jdoe
# Cmnd alias specification
Cmnd_Alias      BEADLOG=/usr/bin/less /var/log/lastlog, \
                  /usr/bin/less /var/log/system.log, \
                  /usr/bin/less /var/log/mail.log
# User privilege specification
ADMIN           LINUX_SERVERS=READLOG
```

这个配置文件包含 5 个部分。前 3 个部分允许用户创建别名，在文件的第 4 部分，用户权限(user privilege)中可以使用这些别名。有 3 种别名：主机别名、用户别名和命令别名。通过定义别名，可以简化特殊用户和命令的管理。在这个示例文件中，用户别名 `ADMIN` 包含用户 `jdoe`。如果用户需要一个新的管理员，只需简单地将这个管理员的名字添加到别名中，这样该管理员将拥有与其他管理员相同的访问权限——不需要配置单独的程序。相同的逻辑也适用于命令别名。第 5 部分是“Defaults specification”，用户可以在其中设置 `sudo` 的默认行为(要获得更多信息，请参阅 `sudo` 的联机帮助页)。

在使用网络文件系统(Network File Services, NFS)时，主机别名尤其有用，其原因是，尽管这些机器上的命令和用户可能不同，但是主机别名允许在多台机器上使用相同的 `sudoers` 文件。这使得用户能够标准化多台机器的管理选项，这一直都是一个很好的想法。用户能够标准化的内容越多，系统在攻击中的损失就越少，用户的设置也就越安全。

有关在系统上设置 `sudo` 的更多信息，请参阅本地的联机帮助页(`man sudo`)或者 `sudo` 的主页 <http://courtesan.com/sudo/>。

John Smith 是刚刚加入到系统管理员组中的系统管理员，他需要作为根用户运行一些特殊命令。John 的用户名是 jsmith，他必须运行命令 `cat /etc/shadow` 以验证用户的信息是否正确地保存在系统中。他只在 linux5 系统上运行这个命令，该系统拥有 `sudo` 命令。`sudo` 管理员必须修改 `sudoers` 文件以授予 jsmith 相应的访问权限。

如果系统上不存在 `sudo` 程序，用户可以按照第 19 章的介绍下载并安装该程序(如果这个系统不属于用户，那么用户必须获得适当的权限才能执行这些操作)。

(1) 作为根用户登录并使用 `visudo` 命令编辑 `sudoers` 文件：

```
visudo
```

(2) 在 vi 会话中显示用户的 `sudo` 文件样本。下面是一个示例：

```
# sudoers file.
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the sudoers man page for the details on how to write
# a sudoers file.
#
# Host alias specification
# User alias specification
# Cmnd alias specification
# User privilege specification
```

(3) 要授予 John 所需的访问权，在适当的部分添加下面的记录项(如果没有放在正确的位置，命令仍然起作用，但是将更加难以管理)：

```
Host_Alias Linux_System=linux5
User_Alias Administrator=jsmith
Cmnd_Alias CHECK_SHADOW=/bin/cat /etc/shadow
Administrator Linux_System=CHECK_SHADOW
```

用户的 `sudoers` 文件看上去类似于：

```
# sudoers file.
#
# This file MUST be edited with the 'visudo' command as root.
#
# See the sudoers man page for the details on how to write
# a sudoers file.
#
# Host alias specification
Host_Alias Linux_System=linux5
# User alias specification
User_Alias Administrator=jsmith
# Cmnd alias specification
Cmnd_Alias CHECK_SHADOW=/bin/cat /etc/shadow
```



```
# User privilege specification
Administrator      Linux_System=CHECK_SHADOW
```

(4) 保存所做的修改。

现在用户 jsmith 只需要运行下面的命令就可以作为根用户查看/etc/shadow 文件的内容：

```
$sudo /bin/cat /etc/shadow
```

完成这个练习之后，请删除添加到 sudoers 中的所有记录项。这样可以防止拥有 jsmith 账户的用户运行刚才所添加的命令(不要在一个真正的账户上进行这个练习)。

工作原理

在 sudoers 的 Host_Alias 部分，将 Linux_System 系统别名指定为 linux5。在 User_Alias 部分，将 Administrator 别名设置为 jsmith，即新的管理员用户名。在 Cmnd_Alias 部分，使用绝对路径设置命令别名。最后一行将所有的别名放在一起：Administrator 是运行命令的用户(jsmith)；Linux_System 是用户在其上运行命令的系统(linux5)；别名 CHECK_SHADOW 指示所运行的命令，即/bin/cat/etc/shadow。

要知道，sudo 用户能够利用一些命令来增强自己的特权。因此必须谨慎地考虑哪些命令可以通过 sudo 进行访问。一般情况下，用户不应该授予对下列命令的 sudo 访问权：

- 允许 shell 切换的命令。例如，如果授予用户对 vi 的 sudo 访问，那么该用户能够作为根用户切换 shell，并拥有对系统的完整访问权限。需要注意的其他命令有 less 命令和所有定制的程序。
- zip、gzip 或其他压缩/解压缩命令。可以使用这些命令作为根用户修改系统文件。

在 sudo 中使用命令的绝对路径，否则可能有人编写恶意脚本，并令该脚本的名称与一个有效命令的名称相同。

通常不应该授予用户利用 sudo 直接访问 shell 的权利，因为这将使得他们能够作为根用户导航并修改系统。

在授予对某个命令的 sudo 访问之前，用户必须完全了解该命令的功能。

12.4 系统管理的预防性任务

许多系统管理任务都可以称为预防性任务。用户执行这些任务有助于防止 Unix 系统被攻破。系统管理任务包括系统维护和清理。本节将介绍其中一些任务。

12.4.1 删除不需要的账户

Unix 系统的发布版本通常包含一组账户以方便系统的使用。有时候这些账户是非常有益的，并设置了足够的安全性，但是比较老的 Unix 版本默认提供的账户是不需要的，并且如果没有正确地进行设置，这些账户将会被攻破。删除不需要的账户是一项很好的安全措施。

为系统账户设置非常安全的口令，知道这些口令的人越少越好。

查看/etc/passwd 文件(或者其他的账户文件)，找出不熟悉的账户。仔细研究这些不熟悉的

系统账户的用途——用户可能会发现某些特殊功能确实需要这些账户。在删除任何不必要的系统账户之前(第 3 章中已经介绍过), 请在一个非生产所用的系统上测试所做的修改。如果不能使用这种方法, 那么用户可以通过下面的步骤禁用账户:

(1) 为/etc/shadow 和/etc/passwd 文件创建备份。如果在本系统上创建备份, 必须确保文件的权限是 700, 并且文件为根用户所有。

(2) 编辑/etc/passwd 文件, 为希望禁用的系统账户删除当前的 shell(例如/bin/sh), 同时将/bin/false 作为 shell 记录项。

(3) 编辑/etc/shadow, 在加密的口令字段中输入一个*。

通过这些步骤, 用户将不再拥有有效的 shell 和口令, 因此可以防止登录到这些账户。

对于需要的账户, 通过使用 sudo 命令使得用户能够利用 sudo 命令将 su 命令用于账户, 而不是让用户使用 su 命令本身。这将授予用户最大的可见度, 明白谁正在使用账户。通过禁止直接登录, 用户能够追踪哪些人正登录到组账户, 以防出现任何问题。

12.4.2 修补、限制或删除程序

Unix 通过系统所包含的程序提供了许多功能。这给用户提供了必要的灵活性, 但是也存在一定的问题, 因为恶意用户/攻击者可以利用相同的软件实现自己的不良意图。软件中会经常发现新的漏洞, 因此提供商或社区提供了相应的软件补丁。为了保持系统软件是最新的, 用户需要经常登录软件提供商/开发者的 Web 站点查找补丁。请记住需要为 Unix 系统软件打补丁。

删除或限制系统中不需要的软件, 以防止恶意用户以未被察觉的方式使用这些软件, 并减少为该软件打补丁的管理负担。以 gcc 编译器为例, 它对于编译源代码是很有用的, 但是恶意用户也能够使用它来编译危险的程序, 如口令解密程序, 或者其他能够授予他们访问计算机系统特权的软件。用户可能需要使用 gcc 编译软件, 这时应该严格限制(使用八进制权限 700)对该软件的访问, 或者, 如果不需要使用 gcc, 应该将它完全删除。

在删除任何软件之前, 请在一个非生产所用的系统上测试所做的修改。如果用户没有非生产所用的系统, 可以重命名软件(例如 gcc_old)并等待一段时间, 以确认该软件不需要应用于特殊的应用程序或用途, 这些应用程序或用途可能不会立刻显现出来。如果没有问题发生, 就可以删除该软件。

软件还具有不明显的依赖性, 因此使用系统内置的软件或程序包管理软件(如 Linux 中的 rpm 或 Solaris 中的 pkgmgr)将提供一些保护措施, 防止用户意外删除至关重要的软件。如果用户试图不通过程序包管理软件而直接删除某个软件, 则可能发生依赖性问题(一个程序可能依赖于另一个先前安装的程序)。如果其他程序的正确运行需要用到某个软件, 那么在删除该软件之前, 内置的程序包管理软件将阻止用户, 或者至少给出警告。

12.4.3 禁用不需要的服务

服务(service)(将在第 16 章中详细介绍)是提供特殊功能的程序, 它能够为用户提供需要的功能, 例如提供远程登录功能的远程登录服务。攻击者也可以利用服务获得对系统的访问权或者增强自己的权限。在大多数 Unix 系统中, /etc/inetd.conf 文件(Linux 中是 xinetd 文件)包含一组可用的有效服务。要删除某个服务的功能, 可以通过在服务名的前面加上英镑符号(#)注释掉该行并重新启动 inetd 或 xinetd 服务。例如, 要禁用 Solaris 系统上的远程登录服务, 用户可以编辑/etc/inetd.conf 文件以注释掉下面的行:

```
telnet stream tcp6 nowait root /usr/sbin/in.telnetd in.telnetd
```

从而该行将变成:

```
# telnet stream tcp6 nowait root /usr/sbin/in.telnetd in.telnetd
```

然后保存该文件并从命令行重新启动 inetd 服务, 如下所示:

```
#ps -ef | grep inetd
root    220    1    0    Nov  20    ?    0:00    /usr/sbin/inetd -s
#kill -HUP 220
```

重新启动 inetd 服务将迫使它重新读取配置文件(/etc/inetd.conf), 从而使用户的修改生效。

用户应该删除 rsh、finger、daytime、rstatd、ftp(如果不需要的话)、smtp 以及系统中不会特别需要的其他软件。如果日后需要某项服务, 用户可以通过删除配置文件中的相关#, 并按照前面的介绍重新启动 inetd 进程以取消对这个服务的注释。

12.4.4 监控并限制对服务的访问

用户能够限制对具有 TCP 包装器(TCP Wrapper)的远程服务的访问。这个小程序使得用户能够拦截对特殊服务的调用, 然后记录对这些服务的访问并根据请求的来源限制访问。

从 <ftp://ftp.porcupine.org/pub/security/> 下载 tcpd(TCP 包装器守护进程, TCP Wrapper Daemon)。按照第 19 章的介绍编译该软件, 然后为 tcpd 支持的服务(通常是 telnet、ftp、rsh、rlogin、finger 和一些其他服务)编辑/etc/inetd.conf 文件, 以指向 tcpd 包装器守护进程。例如, 在一个 Solaris 10 系统上, 如果用户需要运行 telnet, 那么要将 tcpd 用于 telnet 服务, 需要在 /etc/inetd.conf 文件中找到下面的行:

```
telnet stream tcp6 nowait root /usr/sbin/in.telnetd in.telnetd
```

将/usr/sbin/in.telnetd 部分替换为/usr/sbin/tcpd(或者保存 tcpd 二进制文件的任何路径), 从而该行将变成:

```
telnet stream tcp6 nowait root /usr/sbin/tcpd in.telnetd
```

这样将允许 tcpd 控制对 in.telnetd 服务的连接, 包括试图访问这个服务的日志。安装了 tcpd 之后, 用户可以创建两个文件:

- /etc/hosts.allow——列出允许访问服务的系统。用户可以指定允许访问服务的系统。例如, 要允许来自 192.168.1.2 的所有用户访问所有的服务, 可以添加如下记录项:

```
ALL: 192.168.1.2
```

- /etc/hosts.deny——列出不允许访问服务的系统。典型的/etc/hosts.deny 文件包含下面的记录项, 该记录项拒绝所有没有在/etc/hosts.allow 文件中显式列出的系统:

```
ALL: ALL
```

这些文件有许多不同的选项, 在实现修改之前, 应该在一个非生产所用的系统上测试所有的修改。要获得更多信息, 请登录 <http://www.cert.org/security-improvement/implementations/i041.07.html>。

12.4.5 实现内置防火墙

大多数现代化的 Unix 系统都含有内置的防火墙，它根据系统管理员设置的标准限制对系统的访问。如果所用的 Unix 版本包含或者拥有一个可用的防火墙程序，那么可以研究它的用途和实现。例如，Linux 拥有非常流行的 IPTables，Solaris 拥有 SunScreen，Mac OS X 拥有 Personal Firewall。这些防火墙需要了解系统的配置，具备确定系统访问请求的能力，因此使用它们时一定要仔细考虑。这些程序超出了本书的范围，但是用户可以在下面的网址找到更多信息：

- Linux IPTables——<http://netfilter.org/>
- Solaris SunScreen——<http://docs.sun.com/app/docs/coll/557.4>
- Mac OS X Personal Firewall——http://download.info.apple.com/apple_Support_Area/Manuals/servers/MacosxserverAdmin10.2.3.PDF

12.4.6 其他的安全程序

Unix 操作系统有很多可用的安全程序。尽可能多地研究这些程序，明白每个程序是怎样维护系统安全的。下面是帮助用户入门的一些程序(在使用其中的任何程序之前，用户应该获得系统所有者的许可)：

- Tripwire——文件完整性检查程序，它确定文件是否已经过修改。经过适当地配置，这个程序能够识别系统上已修改过的文件，该操作可以指示怀有恶意的行为，例如/etc/passwd 文件中的修改。用户可以从 <http://tripwire.org> 下载开放源代码版本，或者从 <http://tripwire.com> 下载商业版本。
- Nessus——漏洞扫描程序，它检查系统上处于危险中的服务。该程序拥有易用的图形用户界面，并且能够提供非常详细的报告。可以从 <http://nessus.org> 获得更多信息。
- Saint——漏洞扫描程序，跟 Nessus 的功能一样。可以从 <http://saintcorporation.com> 获得更多信息。
- NMAP——端口扫描工具，帮助用户识别系统上可用的端口或服务。可以从 <http://insecure.org/nmap/> 获得更多信息。
- Snort——网络入侵检测工具，它还可以嗅探网络通信量或者识别网络上潜在恶意实体的发作时间。可以从 <http://snort.org> 获得更多信息。
- GNUPG——GNU 隐私卫士(Gnu Privacy Guard)，该工具使得用户能够加密系统上的文件，以防止没有专用密钥的其他用户访问这些文件。可以从 <http://gnupg.org> 获得更多信息。

12.5 小结

安全性是一个不断发展的过程，它需要系统的所有者尽职尽责地进行维护。本章粗略地回顾了 Unix 安全性的一些基本方面。Unix 的每个实现都具有不同的安全性需求和特殊的配置文件。要想真正地确保系统的安全，用户应该参考针对相应 Unix 版本的 Web 站点，并查阅专门的安全性方面的出版物。

12.6 练习

请说明在一个主机名为 `linux_backup` 的单一系统上，如何设置一个具有 `sudo` 权限的用户以便将用户切换到 `backupuser` 账户。

第 13 章 基本 shell 脚本编程

shell 除了作为与计算机交互的环境以外，还能用于编程。shell 程序通常称为脚本(script)，可以用于很多方面，特别是用于系统管理。shell 脚本是存储在单个文件中的一系列系统命令。每次调用该文件时，这些命令就自动地依次执行。

考虑可能发生的事情。机器上的任何程序都可以从 shell 中运行，其输入和输出将根据用户的需要自动重定向。一旦用户掌握了 shell 脚本编程的方法，就可以利用少数的简单脚本自动完成大量的工作。更好的是，shell 脚本擅长处理的工作正是那些手工操作会让人非常烦躁的内容，无论是因为它们令人厌倦还是因为它们只是一味地重复。

本章将介绍 shell 脚本编程的基本概念。这些脚本很简单，而且看起来可能并不那么重要，但是它们清楚地演示了用户在进入后面的章节学习更高级的方法之前需要掌握的根本机制。

13.1 注释脚本

好的注释对好的程序而言非常关键。硬盘上很容易聚集大量的脚本，特别是在拥有多个管理员的系统上。其中的大多数脚本可能都是相同概念的变体，并且可能没有人了解所有这些变体。即使是在单用户系统上，用户也可能为了执行某些任务而编写一些脚本，而在以后打开某个脚本查看其中的内容时却不记得它的作用或者当初编写它的原因。

和所有的编程类型一样，解决这个问题的方法是良好的注释。shell 脚本是独立的文档，因此注释它们的最简单也是最好的方法是使用脚本自己的注释。大多数编程语言都具有在程序中嵌入注释的能力，shell 脚本也一样。

在 Unix shell 中，注释行以散列符号(#)开头。在散列符号之后出现的所有内容都认为是注释，不解释为命令。占用多行的注释在每行的开头都有一个散列符号，如下所示：

```
# Here is a comment that  
# spans multiple lines.
```

另外，还可以在一行的中间插入散列符号，将该符号右边的所有内容都看作注释：

```
some_command # This is a comment explaining the command
```

用于向脚本中添加注释的特定方法由用户决定。然而，有一些经验可以帮助用户避免麻烦：

(1) 在脚本的顶部放置一些基本信息。包括脚本的名称、日期和脚本的总体作用。如果脚本需要接受命令行参数，那么添加一个简短的注释，说明参数的正确语法。还应该包括可以说明脚本的不同寻常之处的所有记录或提示。

(2) 保存修改日志。如果用户对脚本做了修改，那么请保存所有修改的日志。将每个修改添加到修改日志的顶部，从而读者可以按照相反的年月日顺序查看所有修改。用户在执行这些操作时，需要同时修改脚本顶部的日期，从而保持修改日志和实际脚本的一致性。

(3) **注释每个部分**：大多数脚本都包含多个部分。每个部分都应该有一条注释以说明它的作用。不需要详细说明它的工作原理(通过清楚明晰的编程，用户应该能够明白)，但是应该在注释中说明它的目的。

(4) **识别必须由用户添加的数据**。如果脚本的用户(无论是您还是其他用户)需要为脚本的某一部分提供信息，那么应该添加注释来解释需要添加的信息以及该信息需要的格式。

虽然假定用户已经有一些阅读脚本的经验，但是研究脚本中的注释还是有帮助的(特别是如果用户在日常工作中没有为自己的工作添加注释)。下面是从 Red Hat Linux 上的 `/etc/rc.d/rc.sysinit` 文件中截取的一部分内容。该文件是控制机器启动方式的主要脚本，但是本例中它的功能并不是关键问题。我们仅仅考察其中的注释。

```
# If a SCSI tape has been detected, load the st module unconditionally
# since many SCSI tapes don't deal well with st being loaded and unloaded
if [ -f /proc/scsi/scsi ] && grep -q 'Type: Sequential-Access' \
    /proc/scsi/scsi 2>/dev/null ; then
    if grep -qy ' 9 st' /proc/devices ; then
        if [ -n "$USEMODULES" ] ; then
            # Try to load the module. If it fails, ignore it...
            insmod -p st >/dev/null 2>&1 && modprobe \
                st >/dev/null 2>&1
        fi
    fi
fi

# Load usb storage here, to match most other things
if [ -n "$needusbstorage" ]; then
    modprobe usb-storage >/dev/null 2>&1
fi

# If they asked for ide-scsi, load it
if grep -q "ide-scsi" /proc/cmdline ; then
    modprobe ide-cd >/dev/null 2>&1
    modprobe ide-scsi >/dev/null 2>&1
fi

# Generate a header that defines the boot kernel.
/sbin/mkkernelldoth

# Adjust symlinks as necessary in /boot to keep system services from
# spewing messages about mismatched System maps and so on.
if [ -L /boot/System.map -a -r /boot/System.map-'uname -r' ] ; then
    ln -s -f System.map-'uname -r' /boot/System.map
fi
if [ ! -e /boot/System.map -a -r /boot/System.map-'uname -r' ] ; then
    ln -s -f System.map-'uname -r' /boot/System.map
fi

# Now that we have all of our basic modules loaded and the kernel going,
# let's dump the syslog ring somewhere so we can find it later
dmesg > /var/log/dmesg
sleep 1
kill -TERM '/sbin/pidof getkey' >/dev/null 2>&1
```

```

} &
if [ "$PROMPT" != "no" ]; then
    /sbin/getkey i && touch /var/run/confirm
fi
wait

```

请注意，每个代码段前的注释说明了这一部分的作用。即使用户从来没有运行过 Linux，或者从来没有看过初始化脚本，也应该能够通过阅读其中的注释了解脚本的功能。考虑用户自己的脚本或程序。如果没有实际运行该程序，另一个随机选择的程序员能够理解您的工作和该程序的目的吗？如果不能，那么用户需要添加更多的注释，以使内容更加清楚。

警告：

用户在很多时候都会不由自主地忽略注释。跳过它们看起来很简单。毕竟，用户会这样想，“我是惟一会使用这段代码的人，而且我知道它的功能是什么。”这种想法几乎总是错误的，用户最终会发现自己不得不逐行地查看脚本以找到某个特定的部分，或者总结特定代码段的具体功能。在处理脚本的时候花费几分钟的时间进行注释将为用户节约大量的时间并在以后的开发中避免混淆。

13.2 开始脚本编程

正如本章开头所介绍的，shell 程序实质上是一组按顺序执行的命令。例如，用户可以编写一个基本的脚本，它包含如下命令：

```

pwd
ls

```

用户运行该脚本时，将得到一个关于当前工作目录及其内容的报告。这样一个脚本可能不是特别有用，但是用户可以从中获得基本的思想。shell 脚本编程就是从这些简单的概念开始的。

shell 脚本需要几个结构，这些结构告诉 shell 环境应该做什么以及什么时候做。本节将介绍这些结构，并说明如何使用它们。

13.2.1 调用 shell

在向脚本中添加任何其他内容之前，用户需要告诉系统将要启动一个 shell 脚本。使用 shebang 结构完成这项操作。例如：

```
#!/bin/bash
```

告诉系统接下来的命令由 bash shell 执行。

该结构之所以称为 shebang 是因为符号#称为 hash，符号!称为 bang。Unix 编程中有很多类似于这样的特殊术语。

要创建一个包含下列命令的脚本，用户应该首先添加 shebang 行，然后再添加这些命令：

```
#!/bin/bash
pwd
```

```
ls
```

如此简单的脚本是不需要太多注释的,但是本例仍然给出了详细的注释,这是一个很好的习惯:

```
# /home/joe/dirinfo
# 12/7/04
# A really stupid script to give some basic info
# about the current directory.

#!/bin/bash
pwd
ls
```

简言之,这些注释表明了编写脚本的目的。剩下的只是用合适的文件名(这里是 /home/joe/dirinfo,但是任何有效的文件名都可以)保存该文件,并令该文件可执行。完成这些操作之后,用户只需要在命令提示符的后面输入其文件名就可以运行该脚本了。

当然,如果用户的主目录不是 PATH 环境变量值的一部分,那么用户必须将自己的主目录添加到环境变量中,或者使用完整路径名调用该程序。

当然,大多数脚本都比这个脚本复杂。毕竟,shell 是一种真正的编程语言,它包括变量、控制结构等。然而,无论脚本多么复杂,它仍然只是一组按照顺序执行的命令。如果用户遇到困难,请牢记这一点,以保持冷静。

将自己的脚本保存在一个单独的目录下以便于查找,这是一个很好的习惯。可以考虑在主目录下创建一个用于存放脚本的/bin子目录。

13.2.2 变量

Unix 中的变量与其他编程语言中的变量的操作方式相同。正如第 5 章所述,环境变量的赋值非常简单,只需要使用赋值运算符(=),如下所示:

```
EDITOR="vi"
```

本例中,将 vi 赋值给变量 EDITOR。通过在变量名前加一个美元符号,用户就能够访问该变量的值:

```
echo $EDITOR
```

如果在命令行输入该命令,那么它将输出 vi。

在 bash shell(本章使用这个 shell)中,默认情况下将变量视作文本字符串。对于刚刚接触 shell,并且希望在 shell 程序中执行数学运算的程序员来说,这可能会引起一些问题。

在 bash 中是可以进行数学运算的,但是需要一些额外的工作。本章后面将介绍如何实现。

在 bash 中,如果用户利用命令 VAR="1"创建一个变量,那么 VAR 的值是文本字符 1,而不是数值 1。换句话说,系统将该变量的值看作数字 1 的文本表示(textual representation),而不是数值(numerical value)。如果用户编写如下的代码块:

```
VAR=1
VAR=$VAR+1
echo $VAR
```

那么将输出文本字符串 1+1，而不是数字 2。

在有些 shell 中(例如 bash 2.0 以上的版本)，用户可以声明变量类型。由于比较简单的 shell，如普通的 Bourne Shell 和它的克隆版本 ash 没有这项功能，因此本章不介绍这个内容。如果用户希望了解更多有关声明变量类型的内容，请参阅 bash 2 手册，网址是 www.gnu.org/software/bash/manual/bash.html。

如果变量名的前面有美元符号，那么可以将该变量替换成一个表达式，并且在运行脚本时执行这个变量。例如：

```
PERSON="Fred"
echo "Hello, $PERSON"
```

将得到如下输出：

```
Hello, Fred
```

按照惯例，Unix 变量用大写字母表示。这不是必需的，但是这样做使得在脚本中查找变量更容易。可以使用字母、数字和某些符号随意命名变量。但是，有一些特殊变量(本章后面讨论)在名字中使用数字和非字符和数字的符号。通常情况下，变量名最好使用字符和数字，并且以数字结尾。例如，VARIABLE3 是比较好的变量名，而 3VARIABLE 可能给用户带来麻烦。

13.2.3 从键盘读取输入

除了在脚本中为变量赋值以外，用户还可以通过读取键盘的输入值为变量赋值。当用户希望自己的脚本根据用户的输入或另一个命令的输出而做出不同的响应时，这种做法尤其有用。

例如，用户可能编写简单的脚本：

```
echo "What is your name?"
read PERSON
echo "Hello, $PERSON"
```

本例中，read 命令从键盘获取输入，并将它赋值给变量 PERSON。

13.2.4 特殊变量

在前面，本章警告用户不要在变量名中使用某些非字符和数字的符号。这是因为已在特殊 Unix 变量的名字中使用了这些符号。保留这些变量用于特殊的功能。例如，\$ 字符表示当前 shell 的进程 ID 号，或 PID(系统利用 PID 号识别正在运行的进程)。

如果用户在 shell 提示符后输入

```
echo $?
```

那么将获得上一个命令的退出状态。表 13-1 列出了很多可以在 bash 脚本中使用的特殊变量。

表 13-1

变 量	功 能
?	前一个命令的输出状态
\$	当前 shell 进程的 PID
-	当前 shell 启动时调用的选项
!	在后台运行的上一个命令的 PID
0	当前脚本的文件名
1-9	调用当前脚本时给出的第 1 个到第 9 个命令行参数: \$1 是第 1 个命令行参数的值, \$2 是第 2 个命令行参数的值, 依此类推
-	传递给在这个命令之前最近调用的命令的最后一个参数

13.2.5 退出状态

变量?表示前一个命令的退出状态(exit status)。退出状态是每个命令在完成时返回的一个数值。通常情况下, 如果命令成功执行, 则大多数命令将返回退出状态 0, 如果不成功, 将返回 1。有些命令由于特殊原因会返回额外的退出状态。例如, 有些命令区分不同的错误类型, 并且根据特殊的错误类型返回各种退出值。但是, 对于大多数的实际应用, 用户可以将 0 和 1 分别解释为成功和失败(或者真和假)。

13.3 流程控制

当然, 变量和其他必需的编程结构是有意义的, 但是, 按照字面上的意思, 它们看起来可能不是特别有用。然而, 在 shell 脚本中使用变量的一个主要原因是它们允许流程控制(flow control)。流程控制很关键, 因为它允许程序计算条件的值, 并根据这些条件执行相应的操作。简而言之, 流程控制允许程序做出判断。

流程控制通常可以分成两类: 条件流程控制和迭代流程控制。两种类型都描述了脚本设置条件的方法, 以及给定的条件满足或不满足时, 脚本如何做出反应。一旦用户开始编写涉及到这些类型的代码, 就会更加清楚两种类型之间的区别, 因为每种类型使用惟一的命令执行它们的任务。

13.3.1 条件流程控制

顾名思义, 条件流程控制(conditional flow control)主要关心特定的条件是否满足。条件结构使得程序员能够根据特定的约束是否满足来决定是否执行某个代码段。换句话说, 只有满足指定的条件才能运行这个代码段。如果条件不满足, 将跳过这个代码段。

1. if-then 语句

条件流程控制的核心是 if-then 语句。一般来说, if-then 语句类似于:

```
if some_condition
then
    something happens
fi
```

“something happens”部分可以是任意的代码块，从单条语句到非常复杂的大量代码。

和许多其他的编程语言一样，为了便于阅读，条件结构中包含的代码采用了缩进格式。
if-then 块以单词 fi 结束，它是单词 if 反过来拼写的。

下面是一个简单的脚本，它示范 if-then 语句的用法：

```
#!/bin/bash

echo "Guess the secret color"

read COLOR
if [ $COLOR="purple" ]
then
echo "You are correct."
fi
```

如果用户猜测的单词是“purple”，那么该脚本将输出“You are correct”。
这个结构可以变得非常复杂。用户可以通过向 if-then 结构中添加 else 子句来指定多个条件：

```
#!/bin/bash

echo "Guess the secret color"

read COLOR
if [ $COLOR="purple" ]
then
    echo "You are correct."
else
    echo "Your guess was incorrect."
fi
```

本例中，如果 if 子句中的条件不满足，那么将执行 else 子句中包含的命令。
另外，用户可以使用 elif 子句指定第二个条件：

```
#!/bin/bash

echo "Guess the secret color"

read COLOR
if [ $COLOR="purple" ]
then
    echo "You are correct."
elif [ $COLOR="blue" ]
    echo "You're close."
else
    echo "Your guess was incorrect."

fi
```

用户可以添加任意数量的 elif 子句。它们对于让脚本产生明确的回应非常有用，或者当用户知道只有有限的输入值，并且每个值都需要一个特定的响应时，它们也非常有用。

2. test 命令

test 命令用于计算条件的值。用户将会注意到，上例中包含方括号，需要计算的条件放在这些方括号内。方括号在语法上与 test 命令等价。例如，上面的脚本可以改写成：

```
if ( test $COLOR="purple" )
```

这个概念非常重要，因为 test 命令包含很多选项，这些选项能够用于计算所有的条件类型，而不仅仅是简单的等于。例如，使用 test 命令查看特定的文件是否存在：

```
if ( test -e filename )
```

本例中，如果该文件存在，则 test 命令将返回 true，或者 0，如果该文件不存在，则返回 false，或者 1。

通过使用方括号可以得到相同的结果：

```
if [ -e filename ]
```

表 13-2 列出了可以与 test 或方括号一起使用的其他选项。

表 13-2

选 项	测 试 条 件
-d	指定的文件存在并且是一个目录
-e	指定的文件存在
-f	指定的文件存在并且是一个普通文件(不是目录或其他特殊的文件类型)
-G	文件所有者的组 ID 与文件的 ID 相匹配
-nt	指定的前一个文件比后一个文件新(语法是 <i>file1 -nt file2</i>)
-ot	指定的前一个文件比后一个文件老(语法是 <i>file1 -ot file2</i>)
-O	执行该命令的用户是文件的所有者
-r	执行该命令的用户拥有对文件的读取权限
-s	指定的文件存在并且不为空
-w	执行该命令的用户拥有对文件的写入权限
-x	执行该命令的用户拥有对文件的执行权限

利用这些选项，根据语句的真假，命令将返回数值 1 或 0。

3. 比较运算符

条件流程控制使得用户还能够进行其他类型的比较。例如，要指定一个不等式条件而不是等式条件，用户可以通过在等号前加一个惊叹号来表示不等式：

```
if [ $COLOR != "purple" ]
```

对于 purple 以外的所有字符串，这个条件都返回真。表 13-3 列出了比较运算符，它们的工作方式与它们在算术运算中的一样。

表 13-3

运 算 符	示 例	测 试 条 件
=	string a=string b	文本字符串 a 与文本字符串 b 相同
!=	string a!=string b	文本字符串 a 与文本字符串 b 不相同
>	string a>string b	文本字符串 a 大于文本字符串 b
<	string a<string b	文本字符串 a 小于文本字符串 b

由于在通常情况下，字母没有数值，因此用户可能会感到奇怪，一个文本字符串如何能够拥有比另一个文本字符串更大或更小的值。但是，在这种情况下，它们确实有值。文本比较严格地按照字母的顺序进行：a 大于 b，b 大于 c，依此类推。因此，以字母 c 开头的字符串，如“cat”大于字符串“dog”。

在 shell 中进行数学运算

如前所述，默认情况下，shell 中的变量被视作文本字符串。如果用户将变量赋值为 1，那么该变量将保存文本字符(textual character)1，而不是数值(numerical value)1。这是一个理解的难点，许多 shell 程序员新手都在这个问题上会遇到挫折。不难想象，这一点使得在 shell 中进行数学运算更加困难。例如，考虑下面的代码块：

```
MYVAR=1
MYVAR=$MYVAR+1
print $MYVAR
```

这个代码块将输出 1+1，而不是 2。

幸运的是，shell 中有相应的处理方法。expr 命令允许用户在 shell 中执行简单的算术运算。利用 expr，用户可以将上面的代码改写成：

```
MYVAR=1
MYVAR='expr $MYVAR+1'
print $MYVAR
```

expr 命令迫使 shell 将文本字符串 1 解释成数值 1，现在代码段将输出 2。用户可以使用下面的运算符执行算术运算：

- + 加
- - 减
- * 乘
- / 除

expr 命令还具有一些其他的功能。请参阅该命令的文档以获得详细信息。

4. 多个条件

在脚本执行特定的功能之前，条件流程控制允许用户要求满足多个条件。用户可以使用几个逻辑运算符来实现这些多条件要求。假定用户希望在继续执行之前满足两个条件，就可以依照下面的做法：

```

if [ condition1 ]
then
    if [ condition2 ]
    then
        some action
    fi
fi

```

或者通过使用逻辑运算符 **and (&&)** 简化上面的语句:

```

if [ condition1 && condition2 ]
then
    some action
fi

```

还有逻辑运算符 **or (||)**:

```

if [ condition1 || condition2 ]
then
    some action
fi

```

它等价于:

```

if [ condition1 ]
then
    some action
elif [ condition2 ]
then
    the same action
fi

```

用户可能已经注意到, 这些示例使用缩进格式显示特殊语句中的各种操作。缩进, 也称为嵌套, 是一种排版的惯例, 目的是方便人们阅读机器上的代码。虽然脚本在没有嵌套时也能够正确运行, 但是用户最好养成这个习惯。

5. case 语句

除了 if-then 语句之外, 条件流程控制还使用 **case** 语句。这类条件流程控制允许程序员创建一组可选的操作, 根据满足的条件执行相应的操作。它的作用和具有许多 **elif** 子句的 if-then 语句一样, 但是其结果更加简练。下面是一般格式:

```

case expression in
    pattern1)
        action1
    ;;
    pattern2)
        action2
    ;;
    pattern3)
        action3
    ;;
esac

```

与 if 一样, case 语句以 esac 结束, 它是反过来拼写的 case。还要注意每个选项部分都以两个分号(;;)结束。

用 case 语句处理命令行参数非常方便。考察下面的脚本片断:

```
# See how we were called.
case "$1" in
  start)
    # Start daemons.
    action "Starting MFS services: " /usr/sbin/exportfs -r
    echo -n "Starting NFS quotas: "
    daemon rpc.rquotad
    echo
    echo -n "Starting MFS mountd: "
    daemon rpc.mountd $RPCMOUNTDOPTS
    echo
    echo -n "Starting NFS daemon: "
    daemon rpc.nfsd $RPCMFSDCOUNT
    echo
    touch /var/lock/subsys/nfs
    ;;
  stop)
    # Stop daemons.
    echo -n "Shutting down NFS mountd: "
    killproc rpc.mountd
    echo
    echo -n "Shutting down NFS daemon: "
    killproc nfsd
    echo
    action "Shutting down NFS services: " /usr/sbin/exportfs -au
    echo -n "Shutting down NFS quotas: "
    killproc rpc.rquotad
    echo
    rm -f /var/lock/subsys/nfs
    ;;
  status)
    status rpc.mountd
    status nfsd
    status rpc.rquotad
    ;;
  restart)
    echo -n "Restarting NFS services: "
    echo -n "rpc.mountd "
    killproc rpc.mountd
    daemon rpc.mountd $RPCMOUNTDOPTS
    /usr/sbin/exportfs -r
    touch /var/lock/subsys/nfs
    echo
    ;;
  reload)
    /usr/sbin/exportfs -r
    touch /var/lock/subsys/nfs
```

```

;;
probe)
    if [ ! -f /var/lock/subsys/nfs ] ; then
        echo start; exit 0
    fi
    /sbin/pidof rpc.mountd >/dev/null 2>&1; MOUNTD="$?"
    /sbin/pidof nfsd >/dev/null 2>&1; MFSD="$?"
    if [ $MOUNTD = 1 -o $NFSD = 1 ] ; then
        echo restart; exit 0
    fi
    if [ /etc/exports -nt /var/lock/subsys/nfs ] ; then
        echo reload; exit 0
    fi
;;
*)
    echo $"Usage: $0 {start|stop|status|restart|reload}"
    exit 1
esac

```

这个样本是从 Red Hat Linux 上的 `/etc/rc.d/init.d/nfs` 脚本中摘选的。这个脚本控制 NFS(网络文件系统, Network File System)服务。

请记住, 特殊变量 \$1 表示第一个命令行参数。该脚本提供了 7 个可能的选项: `start`、`stop`、`status`、`restart`、`reload`、`probe` 和 `*`。星号选项将匹配任何内容。将该选项放在最后以捕捉不匹配前面选项的所有内容。例如, 如果用户通过参数 `flerbnert` 调用脚本, 那么该脚本将只打印一个消息, 告诉用户所允许的参数, 然后退出。

13.3.2 迭代流程控制

还有许多控制结构, 它们本质上是迭代的(*iterative*)。也就是说, 通过使用这些结构, 代码块可以重复或迭代(*iterate*), 直到满足某个条件为止。

细心的读者可能会争辩, 认为这也是一种条件流程控制。不过这些结构的重要特性是重复性。

1. while 语句

只要满足特定的条件, `while` 语句就重复执行代码块。例如, 考虑下面的代码:

```

#!/bin/bash

echo "Guess the secret color: red, blue, yellow, purple, or orange \n"
read COLOR

while [ $COLOR != "purple" ]
do

    echo "Incorrect. Guess again. \n"
    read $COLOR
done

echo "Correct."

```

本例中，只要 COLOR 的值不是 purple，就重复执行 do 和 done 之间的代码块。

2. until 语句

until 语句中的条件与 while 中的相反。例如，用户可以将上面的示例改写成：

```
#!/bin/bash

echo "Guess the secret color: red, blue, yellow, purple, or orange \n"
read COLOR

until [ $COLOR = "purple" ]
do

    echo "Incorrect. Guess again. \n"
    read $COLOR
done

echo "Correct."
```

13.4 选择脚本编程 shell

在为脚本编程选择 shell 时，用户将面临很多选择。在一定程度上，所选择的 shell 纯粹取决于用户的喜好。但是，有一些重要的内容需要用户考虑：

(1) 几乎所有的 Unix 系统都是以 Bourne 或 bash 编写它们的控制脚本。这本身就是学习这两种 shell 中的编程方法的一个很好的理由。如果用户需要花费一定的时间运行 Unix 系统，那么能够读取和修改或写入(如果需要的话)新的控制脚本是非常重要的。

(2) bash shell 综合了 Bourne、Korn 和 C shell 的大多数优点。

(3) 许多 Unix 管理员认为将 C shell 用于繁重的工作时不太理想。构造一个使用类似于 C 语法的 shell 是一个很好的想法。理论上，这应该能够简化 C 程序员学习 shell 编程的过程。问题出现在执行过程中。C shell 中存在许多故障和不好的设计。事实上，网络中甚至有一个专门的文档，详细说明了 C shell 不应该用于编程的原因，请访问 www.faqs.org/faqs/unix-faq/shell/csh-whynot/。

(4) 其他的 shell，如 zsh、Plan9(rc)等，或许能够更容易地或者更好地完成某项任务，但是它们没有普遍安装在许多系统上。这意味着，如果用户需要切换系统，或者希望与其他人共享自己的脚本，可能会遇到困难。

(5) 大多数 Unix 用户使用 Bourne 或 bash。与其他事情一样，这是一个惯例，如果用户希望使用 Unix 用户的通用语言，那么 Bourne 和 bash 是正确的选择。

13.5 小结

shell 脚本是自动控制 Unix shell 命令字符串的一种极好的方法。脚本可以从键盘(与用户交互)或者其他命令的输出中获取输入。每个脚本都需要拥有几个基本的组件：

- 特定 shell 环境的调用
- 变量及其相关联的值
- 输入源

shell 脚本使用外部输入改变自身操作的方式主要有两种：条件流程控制和迭代流程控制。可以使用几个基本的编程结构来管理脚本流程控制：

- if-then 语句
- test 命令
- case 语句
- while 语句
- until 语句

13.6 练习

(1) 利用 if 语句编写一个名为 filescript.sh 的脚本，该脚本将：

- a. 从命令行获得一个参数。这个参数应该是一个目录路径。如果没有提供参数，那么该程序将默认使用当前目录。
- b. 列出该目录中的所有文本文件(名字中含有后缀.txt 的文件)。
- c. 列出文件名的同时，给用户提供一些选项，他可以选择显示文件的大小、权限、所有者、组或者“all of the above”(以上所有)信息。以交互的方式实现这一步。

(2) 用 case 语句代替 if 语句编写相同的程序。

一定要适当地进行注释。

用户的脚本可能因为所提供的方法的不同而变化，但是只要脚本能够工作，它们就是正确的。

第 14 章 高级 shell 脚本编程

第 13 章介绍了 shell 脚本编程基本元素的基础知识。尽管此时您可能对 shell 脚本的功能并没有太深的印象，但本章的内容可能会让您改变这种看法！本章完整地讨论了更高级的脚本功能，这些功能足以使得 shell 脚本编程成为 Unix 程序员开发工具箱中一个非常灵活而且强大的工具。

迅速回顾一下，您已经学习了怎样选择最适合自己需要的 shell，怎样把变量传递给脚本，并初步涉及了流控制语句。本章将在这些知识的基础上，介绍了很多新的操作符、概念和函数，这些内容可以用于大部分环境中。本章探讨了怎样编写和维护安全的脚本，以及应由谁执行这些脚本、这些脚本可以做哪些事情。受限 shell(restricted shell)和永久删除数据(Secure Wipe)等内容也将在本章讨论。

shell 并不局限于简单地逐行执行命令。它的更为强大的功能之一就是能够创建函数和函数库。因此，本章将研究这些内容并探讨其他一些相关的内容，比如变量的域，这个内容在处理较复杂的脚本时非常重要。

和任何优秀的程序员一样，您也需要考虑如何调试自己的程序。脚本在这方面和其他的语言没有什么不同，在本章的最后将讨论 shell 脚本解释器可用的各种调试方法，包括怎样检查错误、做什么和在何处查找，以及怎样添加停止信号(stop)。

学完本章之后，您将具备深入学习所必需的知识，并成为脚本编程好手。然而，和任何事情一样，脚本编程需要实践和一些技巧以便找到完成工作的最佳途径，所以务必要花一定的时间实际编写各种脚本并把它们放到一个固定的库中，以便在您的工作中能有所帮助。

14.1 高级脚本编程的概念

在何处以及如何使用脚本完全由您决定，惟一的限制就是您的想象力。本节旨在向您介绍使用脚本时内在的灵活性和各种各样的可能性。当然，这里不能教您如何在使用脚本的时候搞破坏，但能提供足够的信息，让您顺利地完成自己想做的任何事情。

脚本可以用来从某个地方接收输入，处理信息，然后把结果发送到别的某个地方——包括其他的脚本。脚本可以处理诸如多维数组等复杂的数据类型，而且能够利用它们的环境执行任务。需要管理进程？这对 shell 脚本毫无问题。希望自定义命令行交互的方式或者自动化文件处理任务(automate file-handling)？请考虑 shell 脚本。

您应该慢慢意识到 shell 脚本是非常有用的工具。完全正确——几乎所有不愿意手动一再重复的工作都可以考虑由脚本来完成。话虽然这么说，仍然有很多内容需要学习，现在就行动起来吧。

14.1.1 输入和输出重定向

为什么需要重定向输入和输出？实际上有很多原因。举个例子，假如正在运行一个脚本程序并希望把它的结果保存到一个文件中，或者更理想的情况是，希望把一个程序的结果发送给另一个程序，这种情况该怎么办？这要以某种方式，在某个阶段对信息进行重定向，因此学习如何完成重定向是很有价值的。

1. STDIN, STDOUT 和 STDERR

每次打开一个 shell 的时候，Unix 都会打开 3 个文件供您的程序使用：

- STDIN(标准输入)——通常是指终端的键盘。
- STDOUT(标准输出)——通常是指终端的显示器。
- STDERR(标准错误输出)——通常也是指终端的显示器。

这里需要记住的很重要的一点就是，默认情况下，输入来自键盘并打印到显示器上。虽然这是大多数交互方式所使用的方法，但这决不是程序和文件之间进行交互的惟一方法。重定向通常涉及到把一个文件和标准输入或者标准输出关联起来并把需要的信息发送到指定的文件中(其他几种重定向 IO 的方法稍后讨论)。

重定向非常简单；Unix 提供了几个简单的操作符来处理前面所说的关联。

2. 重定向操作符

为了对信息完成所需要的重定向，可以使用 Unix 提供的如下操作符。如表 14-1 所示。

表 14-1

操 作 符	动 作
>	把 STDOUT 重定向到一个文件
<	把 STDIN 重定向到一个文件
>>	把 STDOUT 添加到一个文件的末尾
	从一个程序或者进程获取输出，然后发送给另一个程序或进程
<< delimiter	把当前的输入流和 STDIN 关联起来直到到达指定的分隔符(delimiter)

前两个操作符的使用非常简单。例如，如果想把通常打印到屏幕(STDOUT)上的输出重定向到一个文件，可以输入如下命令：

```
$ ls > fileList
```

ls 命令通常直接在屏幕上显示结果，但在这个示例中，输出被重定向到 fileList 文件。当然，您可能不希望每次把输出发送到 fileList 文件时都把它原来的内容覆盖掉。在这种情况下，可以使用>>操作符定位到文件末尾，并把输出结果添加到文件的末尾：

```
$ ls \some\other\directory >> fileList
```

有时候希望把输出直接发送到另一个进程。要这样做，可以用如下方式使用管道(|)操作符：

```
$ ls | wc
```

在这个示例中，ls 命令的 STDOUT 被直接发送到 wc 实用工具的 STDIN，然后 wc 把运行的结果如实地打印到屏幕上(因为它的输出没有被重定向到别的地方)。

最后要讨论的一种重定向方法是 << delimiter，它通常用于一种称为 HERE 文件的特殊文档中。基本上，这种方法使用重定向从一个给定的输入中读取信息直到到达指定的分隔符。一种简单的用法是在屏幕上输出多行数据而无需反复使用 echo：

```
Cat << END
The cat
Sat on the
Mat.
END
```

对 HERE 文件的处理类型完全由您确定，因此可以尝试使用各种不同的重定向操作符，从而掌握它的用法。

在继续学习之前，有一件有趣的事情，就是 > 操作符有几个修饰符标记，这些标记可以改变它的行为。例如，> 和 & 一起使用将对 STDOUT 和 STDERR 同时进行重定向，如下所示：

```
$ ls >& fileList
```

在这种情况下，任何错误消息都不会打印到屏幕上，而是被发送到 fileList 文件中。

使用>!将迫使文件以 append 模式创建，或者以 normal 模式覆盖一个已经存在的文件，而使用>@将以二进制模式而不是文本模式打开一个文件。

稍后您将会看到更多在一个示例中进行多个重定向操作的示例。

14.1.2 命令替换：反引号和圆括号扩展

通常很重要的一点是把某个命令的结果保存到一个变量中以便脚本能够使用它。反引号(back tick)对这种情况非常有用，因为它提供了一种内联的方法来执行一个命令，并在执行余下的脚本之前返回结果。

例如，想象一下，您也许希望获得一个文件中文本的行数以便脚本可以根据这个数值决定采取特定的行动。反引号使这项任务变得非常简单：

```
Lines = `wc -l textFile`
```

变量 Lines 现在包含 textFile 文件中文本的行数并可以根据需要在脚本中的其他地方使用这个值。shell 也扩展双引号字符串的内容，所以用双引号也可以进行命令替换。这里特别感兴趣的是圆括号(brace)扩展，其格式是：\$(command)，其中 command 可以是任何有效的 Unix 命令。

因此，可以使用下面的命令获得和前面一样的结果：

```
Lines = "$( wc -l textFile ) "
```

记住在命令中使用引号来帮助您表明意图。如果不希望有任何类型的扩展发生，请使用反引号标记——换句话说，用它们包含一个文字串。否则，如果希望进行变量替换并执行命令，请使用双引号。

需要注意的是\$(command)格式支持嵌套，而无需跳出\$(和)字符，所以如果有需要，可以在一条命令中执行多个操作。如下所示：

```
$ echo "Next year will be 20$(expr $(date +%y)+1). "
```

很多人认为嵌套可能会使命令变得过于复杂，容易让人混淆不清。和其他事情一样，明智地使用嵌套就不会出现问题。让我们继续学习。

14.1.3 使用环境变量和 shell 变量

到目前为止，所看到的变量都是局部变量，将它们传递到哪个 shell 就只能在哪个 shell 中使用它们。自然，您不会希望自己的 shell 脚本孤立于系统的其他部分运行，因此通常 shell 脚本都需要和它所运行的环境交互。要这样做，shell 脚本必须能够存取环境变量和 shell 变量，这些变量可能是用户自行设置的，也可能是 shell 在启动的时候就自动设置好的。

环境变量不仅可以由 shell 本身访问，而且任何由 shell 产生的进程都可以对其进行访问。例如，如果一个 shell 创建了一个子 shell(subshell)，那么父 shell 中的所有环境变量在子 shell 中也都是可以访问的。如果希望 PATH 环境变量包含/bin 目录，则可以使用下面这条命令：

```
export PATH =/bin
```

export 命令用于修改 shell 环境。如果不带任何选项，它会显示很多环境变量。查看 mount 命令的联机帮助文档，可以找到当前 shell 的联机帮助文档。

对于 shell 后续的生命期，以及所有由 shell 启动的子 shell 和进程，PATH 变量将包含/bin 目录。可以通过如下方式向环境中输出任何类型的变量：

```
export name = value
```

为了查看环境中已设置的 shell 变量，使用 set 命令：

```
$ set
```

根据系统的设置和所使用的 shell，将得到不同的结果。

shell 变量不是由用户设置的，而是在 shell 初始化的时候设置的。表 14-2 显示了几个由 shell 设置的不同类型的变量。

表 14-2

变 量	包含的内容
CDPATH	用 cd 命令切换目录的快捷方式(类似 PATH)
COLUMNS	显示器上文本的最大列数
EDITOR	编辑器的路径
HISTSIZE	在命令历史记录中存放的命令条数(默认值是 500)

为 shell 设置的变量几乎没有什么数量限制，可以通过 set 命令查看当前已经设置好的 shell 变量清单。在对 shell 和它的变量变得更熟悉之后，可以修改它们来满足自己的需要。

14.2 shell 函数

什么是 shell 函数，为什么需要它们？简单的回答是，即使是最简单的脚本，使用函数也可以在很大程度上使代码保持良好的结构。函数使您可以把一个脚本程序的整体功能分割成比较小的、逻辑的子功能，在需要的时候可以调用它们执行各自单独的任务。

使用函数执行重复性的任务是实现代码重用的一种很好的方式。代码重用是现代面向对象编程原理中很重要的一个部分，而且很有必要——它意味着不必重复编写完成同样功能的代码，这将使得 shell 程序员的工作轻松很多。它还使脚本更加容易维护，因为所有的功能都封装于一个个准确命名的(希望是这样)函数中。

可以把函数看作是嵌入在主 shell 脚本中的一段脚本。毕竟，两者之间的区别非常小——函数只是执行它自己包含的脚本而不会创建一个新进程。当然，一个新的 shell 脚本总是启动它自己的进程。

要声明一个函数，只需要使用下面的语句：

```
name () { commands; }
```

函数的名称是 `name`，在脚本的其他地方调用函数的时候就会用到这个名称。函数名的后面必须是圆括号，再后面是包含在花括号中的一系列命令。

因为函数可以用于执行几乎所有能想到的动作，所以通常需要传递信息给它们处理。下面的“实战”中将介绍带有多个参数的函数。

实战 传递参数给函数

(1) 输入如下代码并将其保存为 `~/bin/func.sh`(如果在主目录中没有 `bin` 目录，请用第 4 章中讨论的 `mkdir` 命令创建一个)：

```
#!/bin/bash

# func
# A simple function

repeat () {
    echo -n "I don't know $1 $2 "
}

repeat Your Name
```

(2) 使脚本变成可执行文件(详细内容请参考第 4 章关于 `chmod` 的小节)。

```
chmod 755 ~/bin/func.sh
```

(3) 在命令行运行脚本：

```
$ ~/bin//func.sh
```

在屏幕上将打印出如下内容：

```
I don't know Your Name
```


工作原理

函数 `repeat` 的声明在其命令列表中只有一条 `echo` 命令。一旦定义之后，就可以在脚本的其他任何地方使用这个函数。在这里，它在声明的后面被直接调用，并给它传递了两个参数：`Your` 和 `Name`：

```
repeat Your Name
```

正如在第 13 章中看到的那样，函数可以通过特殊的变量访问参数。这只是一个非常简单的函数，相信您可以在这里完成更多的事情。不过，这里的关键内容是，在脚本中必须在函数声明之后的地方调用函数。

14.2.1 返回值

通常需要函数返回一个值，这个值可以在脚本的后续部分中使用。正如您已经知道的那样，脚本使用 `exit` 返回一个值，但是对于函数，则要使用 `return` 命令。由于函数可以用在条件语句中，所以完全可以根据函数的返回值来决定采取什么行动。另一种用法是，可以使用返回值告诉您函数中是否发生了错误，或者是不是把错误的信息传递给了函数。

要显式地设置函数的退出状态(默认情况下是最后一条语句的状态)，使用下面的语句：

```
return code
```

`code` 可以是所选择的任何内容，但显然，应该选择某个在脚本整体环境中有意义的或者是有用的内容。例如，如果使用函数来决定是否执行一个 `if` 语句块，就应该返回 0 或者 1，以使程序逻辑保持清晰。

14.2.2 嵌套函数和递归

关于函数的一个非常有趣的特点是它们可以调用自己，就像调用其他函数一样。我们将这种调用自己的函数称为递归函数，它可以用于很多反复执行同一个动作的情形。一个很好的示例是计算传递给函数的数字的总和或阶乘。

对于更复杂的情形，通常需要在函数中调用包含在另一个函数中的功能，我们把这种情况称为函数嵌套。下面的“实战”中，使用一个简单的示例演示了这种情况。

实战

使用嵌套函数

(1) 输入如下脚本，并把它保存为 `~/bin/nested.sh`：

```
#!/bin/bash

# nested
# Calling one function from another

number_one () {
    echo "This is the first function speaking..."
    number_two
}

number_two () {
```

```
    echo "This is now the second function speaking..."
}

number_one
```

(2) 使脚本变成可执行文件。

```
chmod 755 nested.sh
```

(3) 在命令行使用如下命令运行脚本：

```
$ ~/bin/nested.sh
```

以下两行消息将回显到屏幕上：

```
This is the first function speaking...
This is new the second function speaking...
```

工作原理

浏览执行的顺序，看看所发生的事情：

- (1) 调用 `number_one` 函数。
- (2) `number_one` 把它的消息回显到屏幕上。
- (3) `number_one` 调用函数 `number_two`。
- (4) `number_two` 把它的消息回显到屏幕上然后退出。

嵌套函数(有时候也称为链(chaining))是 shell 脚本解释器的工具库中非常强大的一个工具。它使得您能把很庞杂的问题分解成比较小的、容易理解的模块，然后使用这些模块化的代码产生一个整洁的解决方案。

这种脚本编程方式有什么不足的地方吗？是的，递归非常耗费资源而且执行起来可能会很慢，所以不要这样编写脚本。不过，在编写函数的时候有一个概念需要特别注意，这就是作用域(scope)。

14.2.3 作用域

作用域(scope)的概念实际上并不难理解，但是如果您和我一样，那么无论在什么地方，只要涉及到作用域，都要提高警惕。忘记了作用域的作用可能会导致错误的、无法预料的结果，这些结果并不是由于解释器的不精确性造成的。

有两种作用域——全局作用域和局部作用域。如果一个变量拥有全局作用域，这意味着在脚本的其他任何地方都可以访问该变量。而拥有局部作用域的变量则不是这样，只能在声明变量的作用域内访问它们。

为了更好地理解作用域的原理，请实践一下下面“实战”中的小示例。

实战 处理作用域

(1) 输入如下脚本，并把它保存为 `~/bin/scope.sh`：

```
#!/bin/bash

# scope
```

```
# dealing with local and global variables
```

```
scope ()
{
    local lclVariable=1
    gblVariable=2
    echo "lclVariable in function = $lclVariable"
    echo "gblVariable in function = $gblVariable"
}
```

```
scope
```

```
# We now test the two variables outside the function block to see what happens
```

```
echo "lclVariable outside function = $lclVariable"
echo "gblVariable outside function = $gblVariable"
```

```
exit 0
```

(2) 使脚本变成可执行文件。

```
chmod 755 ~/bin/scope.sh
```

(3) 在命令行使用如下命令运行脚本：

```
$ ~/bin/scope.sh
```

应该获得如下输出：

```
lclVariable in function = 1
gblVariable in function = 2
lclVariable outside function =
gblVariable outside function = 2
```

工作原理

这个示例的主要内容位于在 `scope` 函数中。为了定义一个局部变量，需要使用 `local` 关键字：

```
local lclVariable = 1
```

这行脚本把局部变量的值设置为 1 并且只给它赋予局部作用域。然后紧接着函数声明直接调用这个函数，在函数内部用 `echo` 命令输出每个变量的值。

这个调用给出了结果中的前两行，和您预料的一样：

```
local lclVariable = 1
local lclVariable = 2
```

最后，试图在 `shell` 中用 `echo` 命令输出两个在 `scope` 函数内部定义的变量的值。如您所见，只有拥有全局作用域的变量对 `shell` 是可见的——变量 `lclVariable` 在 `shell` 的上下文中是不可见的，只在定义它的函数中可见。最后，在讨论函数的时候几乎总要涉及到函数库，而这正是下一节要讨论的内容。

14.2.4 函数库

您可能会发现有一些函数使用得非常频繁。要不了多久就会发现把它们放到编写的每一个脚本中是一件非常单调乏味的事情。这就是函数库发挥作用的地方。

最简单的事情之一是把所有需要用到的函数都放到一个文件中，然后在每个脚本的开头包含这个文件。这使得该文件中的每个函数都像编写了它们一样可供使用。要这样做，只需要使用点号(.)。比如把 `scope` 文件(不需要是可执行文件，文本文件即可)修改成：

```
scope ()
{
    local lclVariable=1
    gblVariable=2
    echo "lclVariable in function = $lclVariable"
    echo "gblVariable in function = $gblVariable"
}

another_scope_function()
{
    echo "This is another_scope_function..."
}

yet_another_function()
{
    echo "This is yet_another_function..."
}
```

可以通过如下方式访问所有这些函数：

```
#!/bin/bash

. ~/lib/scope    # Define the path you need to access the function library file

scope
another_scope_function
yet_another_function

exit 0
```

这将在屏幕上输出两行回显文本：

```
This is another_scope_function...
This is yet_another_function...
```

如您所见，用这种方法可以很容易包含一个函数库。我们建议把所有的包含文件都放到 `~/lib/` 目录中，以避免主目录中出现混乱。

1. getopt

另一个让人感兴趣的内容是 `getopts`，它是 `shell` 的一个内置(built-in)(这意味着它是在 `shell` 内部实现的)工具，用于检查传递给命令行的选项是否有效。可以把它看作是一个命令行解析器。`getopts` 的语法如下：

```
getopts opstring name
```

在这里，`opstring` 包含了需要确认为有效选项的字符串。如果某个字符串的后面有一个冒号，意味着该选项应该有一个参数，选项和参数之间用空格分隔。`name` 是一个 shell 变量，用于保存选项表中的下一个选项。

`getopts` 通过两个变量来跟踪所有事情：

- `OPTIND` —— 保存下一个需要处理的参数的索引。
- `OPTARG` —— 如果需要一个参数，`getopts` 就把它放在这个变量中。

大多数时候，`getopts` 用于在某些类型的循环中检查传递给脚本的选项和参数。为了观察实际的运行情况，下面的“实战”将带领您编写一个非常简单的示例。

实战

使用 `getopts`

(1) 创建下面的脚本并把它保存为 `~/bin/get.sh`：

```
#!/bin/bash

# get
# A script for demonstrating getopts

while getopts "xy:z:" name
do
    echo "$name" $OPTIND $OPTARG
done
```

(2) 使脚本变成可执行文件。

```
chmod 755 ~/bin/get.sh
```

(3) 在命令行调用脚本，传递一些参数，例如：

```
$ ~/bin/get.sh -xy "one" -z "two"
```

应该获得如下结果：

```
x 1
y 3 one
z 5 two
```

工作原理

在这个示例中重要的代码只有两行。第一行处理传递给 `getopts` 的参数：

```
while getopts "XY:Z: " name
```

从这一行中可以看到选项 `y` 和 `z` 都应该有参数，而且 `name` 就是用于保存下一个选项的变量。

第二行重要的代码简单地打印出与 `getopts` 有关的各个变量：

```
echo "$name"$OPTIND $OPTARG
```

最后，在命令行执行 `~/bin/get.sh`，传递一些需要处理的参数：

```
$ ./get -xy "one" -z "two"
```

正如所料，您接收到了一组变量名，对应的 OPTIND 和 OPTARG 变量的值一起打印出来了。在这个示例中，给 y 和 z 两个选项都传递了参数，在第二行和第三行中，当 name 包含了需要参数的选项的时候，这些参数将通过 OPTARG 变量打印出来。

14.2.5 信号和陷阱

某些进程级别的事件会抛出信号，这些信号对于根据进程发生的事情来决定应采取某个操作是很有用的。例如，用户可以向一个正在尝试写临时文件的进程发送 Ctrl+C 命令。必须恰当地处理这种情况以防在下一次重启的时候删除临时文件，从而丢失重要的信息。

按下 Ctrl+C 组合键会向 shell 进程发送一个信号，怎样回应这个信号由您决定。可以根据接收到的信号使用 trap 命令来采取某个行动。为了获得所有可以发送给进程的信号列表，输入如下命令：

```
$ trap -l
```

表 14-3 列出了一些经常遇到的并且在程序中会用到的信号。

表 14-3		
信号名称	信号值	说明
SIGHUP	1	当进程挂起时系统发送该信号——终端连接断开
SIGINT	2	当用户发送一个中断信号(Ctrl+C)时系统发送该信号
SIGQUIT	3	当用户发送一个退出信号(Ctrl+D)时系统发送该信号
SIGFPE	8	如果试图进行一个非法的数学操作，系统会发送该信号
SIGKILL	9	如果某个进程收到这个信号它将立刻退出并且不会执行任何清理工作(例如关闭文件或者删除临时文件)

捕获这些信号很简单，trap 命令的语法如下所示：

```
trap command signal
```

command 可以是任何有效的 Unix 命令，或者甚至是一个用户定义的函数，而 signal 是任何一组希望捕获的信号值。一般来说，陷阱有 3 种常用的用法：删除临时文件，忽略信号和在进行某些特殊操作的过程中忽略信号。稍后将讨论一个使用信号和陷阱的示例。不过，信号和陷阱在文件处理中也能派上用场，所以让我们先来看看这方面的内容。

14.2.6 文件处理

在处理脚本的时候必须能够处理文件。无论是创建一个文件以保存数据或者检索数据，或者是作为 CVS 应用程序的一部分，抑或是简单地使用临时文件来帮助处理，都必须清楚地知道怎样安全而又整洁地处理文件。

本节讨论怎样使用 shell 脚本高效地执行与文件相关的任务。首先，您将学习如何判断一个文件是否存在。显然这对一个健壮的、鲁棒的(robust)文件处理脚本是很重要的。更进一步，您将学习在某些无法预期的事情发生的时刻怎样清除文件。

1. 文件检测

也许您已经猜到，能判断一个文件是否存在是非常重要的。如果试图覆盖一个已经存在的文件，或者向一个不存在的文件写入数据，就可能会发生很可怕的错误。幸运的是，有一个很快捷的解决方案。还可以很容易地判断文件是否可读或者可写。实际上，还可以通过测试判断很多关于文件的信息。

例如，如果希望确定是否写一个文件，就需要检查这个文件是否存在并且是否可写。要这样做，可以使用下面的 if 语句：

```
if [ -w writeFile ]
then
    echo "writeFile exists and is writable!"
fi
```

这里的测试由通用的形式给出：

```
[ option file ]
```

option 和 file 的内容由您需要测试的内容决定。表 14-4 列出了最常用的几个选项。

表 14-4

表 达 式	意 思
-d file	true, 如果 file 存在并且是一个目录
-e file	true, 如果 file 存在
-r file	true, 如果 file 存在并且可读
-s file	true, 如果 file 存在并且大小大于 0
-w file	true, 如果 file 存在并且可写
-x file	true, 如果 file 存在并且可执行

可以使用&&(逻辑与)或者||(逻辑或)来组合测试条件以便把测试调整得更规则：

```
if [ -r writeFile && -x writeFile ]
then
    echo "writeFile exists, is readable, and executable!"
fi
```

以及

```
if [ -r writeFile || -w writeFile ]
then
    echo "writeFile exists and is readable or writable!"
fi
```

2. 清除文件

在脚本中创建很多临时文件然后让它们散布在系统中是一种很不好的编程方式。各种信息都可能丢失或者驻留在系统中。例如，觉得某个脚本执行的时间太长，而现在该回家了，该怎么办？

绝大多数情况下，人们将按下 Ctrl+C 组合键结束脚本的运行。如果在合适的地方没有恰当的文件处理，这可能会导致一些奇怪的问题。这就是信号和陷阱发挥作用的地方。通过捕获发送给进程的信号，可以调用函数，这些函数会适当地进行处理。下面的“实战”小节会捕获一个由用户发送的中断，并使用 trap 命令清除在脚本的执行期间所创建的临时文件。

实战

用 trap 命令清除临时文件

- (1) 输入如下代码并保存到一个脚本文件中。把该文件命名为~/bin/sigtrap.sh:

```
#!/bin/bash

# sigtrap
# A small script to demonstrate signal trapping

tmpFile=/tmp/sigtrap$$
cat > $tmpFile

function removeTemp() {
    if [ -f "$tmpFile" ]
    then
        echo "Sorting out the temp file..."
        rm -f "$tmpFile"
    fi
}

trap removeTemp 1 2

exit 0
```

- (2) 使脚本变成可执行文件。

```
chmod 755 ~/bin/sigtrap.sh
```

- (3) 使用如下命令在命令行运行该脚本:

```
$ ~/bin/sigtrap.sh
```

- (4) 在命令行输入一些文本然后按下 Ctrl+D 组合键。一旦按下这个组合键，就会立即检查在 tmp 目录中创建的文件(它的名字可能和 sigtrap(procid)差不多)。

- (5) 现在再次运行代码，但在按下 Ctrl+D 组合键之前，先按下 Ctrl+C 组合键。这一次会收到一条消息“Sorting out the temp file....”。如果此时去查看名称类似于 sigtrap(procid)的文件，将发现它并不存在。

工作原理

第一行使用 shell 的进程 ID 创建一个临时文件，文件名的前缀为 sigtrap:

```
tmpFile=/tmp/sigtrap$$
```

下一行使用没有参数的 cat 命令。这使得脚本等待用户输入。您需要让程序等待一小会儿，以便有时间发送合适的信号:

```
cat > $tmpFile
```

接下来创建一个在退出之前删除由脚本创建的临时文件的函数：

```
function removeTemp() {
    if [ -f "$tmpFile" ]
    then
        echo "Sorting out the temp file... "
        rm -f "$tmpFile"
    fi
}
```

这就是陷阱的主要用途。您可能会希望根据接收到的信号和脚本正在完成的工作来调用某个特定的函数。

最后，设置 `trap` 命令在接收到信号 1 或 2(挂起进程的信号，或者用户发送一个中断，Ctrl+C) 时来调用 `removeTemp()` 函数：

```
trap removeTemp 1 2
```

您可能会发现需要在接收到不同信号的时候调用不同的函数。如果是这种情况，只需要用对应的函数创建多条 `trap` 命令来处理执行情况即可。

14.2.7 数组

处理数组通常都是必须的，而且很有用。`shell` 脚本在这方面和其他脚本或编程语言没有什么不同。自然，可以通过 `shell` 的符号来定义和使用数组。本节主要介绍 `bash shell` 的数组符号，但是在这里学到的任何内容都可以用到其他任何喜欢使用的 `shell` 中去。

需要对数组进行的处理有：声明数组，在数组中存放信息，以及根据需要操作这些信息。可以在 `shell` 的内部完成所有这些工作，而且，也许您已经料到，可以使用多种方法。

1. 声明数组

声明数组的方法很多，采取什么方法完全取决于当时什么方法对您最有利。甚至可以使用存放在文本文件中的数据来初始化数组。第一种，也是最简单的一种声明数组的方法如下所示：

```
array1[index] = value
```

上面的语句声明了一个名为 `array1` 的数组，它有一个值，这个值通过方括号中的索引引用。另一种声明和填充数组的方法是：

```
array2=(value1 value2 value3 value4)
```

在 `array1` 中填充了 `value1`、`value2` 等值。索引从 0 开始，自动赋值。第三种初始化数组的方法或多或少是前两种方法的组合：

```
array3=([0]=value1 [13]=value2 [7]=value3)
```

注意在声明数组中的数据的时候，索引号既不需要按特定的顺序排列，也不需要按顺序排列——可以任意地放置它们。

怎样获得数组中的数据呢？需要执行一个称为解引用(`dereference`)的操作。

2. 解引用数组

要获得数组中某个特定索引位置的数据，可以按如下方式使用花括号符号：

```
${array[index]}
```

很简单吧！所以要获得数组 `array3` 中索引为 13 的元素的数据，可以使用如下命令：

```
value=${array3[13]}
Echo "$value"
```

也许您已经猜到，这将打印出：

```
value2
```

可以使用特殊的符号来判断一个数组中的所有数据。还可以获得数组包含的元素数目。下面的两行代码分别执行这两个操作：

```
arrayelements=${array2[@]}
arraylength=${#array2[@]}
```

使用上一节声明的 `array2`，`arrayelements` 变量将包含数据 `value1 value2 value3 value4`，同时 `arraylength` 变量包含值 4。

最后，也许不想返回所有的元素，而只需要其中某个范围内的数据。如果是这样，使用下面的命令：

```
${array[@]:3}
${array[@]:3:2}
```

第一行从第四个元素(因为索引号是从 0 开始的)开始返回所有余下元素中的数据，而第二行返回第四个元素之后的两个元素的值。

`bash` 的版本 3 中增加了一个新的索引操作符号，不过严格地说它并不属于解引用操作。为了获得数组中包含的索引值，使用下面的命令：

```
${!array[@]}
```

对于前面出现的 `array2`，这将返回：0 1 2 3。

3. 从数组中删除数据

您或许希望删除数组中的某些数据，或者甚至丢弃数组中的所有数据。如果是这样，可以按如下格式使用 `unset` 命令：

```
unset array[1]
unset array[@]
```

第一行删除索引位置为 1 的数据，第二行删除数组中的所有数据。下面是一个简短的练习，演示怎样用一个文本文件中的数据来填充一个数组。

实战

在数组中使用文本文件中的数据

(1) 输入如下文本并保存为文件 `~/tmp/sports.txt`：

```
rugby hockey swimming
polo cricket squash
basketball baseball football
```

(2) 创建如下脚本，并把它保存为~/bin/array.sh:

```
#!/bin/bash

# array
# A script for populating an array from a file

populated=('cat ~/tmp/sports.txt | tr '\n' ' ')
echo ${populated[@]}

exit 0
```

(3) 使脚本变成可执行文件:

```
chmod 755 ~/bin/array.sh
```

(4) 在命令行使用如下命令运行脚本:

```
$ ~/bin/array.sh
```

示例的输出应该就是没有换行的 sports 文件的内容:

```
rugby hockey swimming polo cricket squash basketball baseball football
```

工作原理

关键的语句是:

```
populated=('cat ~/tmp/sports.txt | tr '\n' ' ')
```

这行代码使用前面介绍的声明符号来声明和填充一个名为 `populated` 的数组。圆括号中的命令通过管道把 `cat ~/tmp/sports.txt` 命令(该命令只是简单地把 `sports.txt` 文件的内容写到标准输出)的结果发送给 `tr '\n' ' '` 命令。

`tr` 把第一个字符串中的某个字符转换成第二个字符串中的另一个字符。在这里，删除了回车符(`\n`)并将其替换为空格，这样就可以正确地填充数组了。如果在文件中只有一行文本，在括号中可以只用 `cat sports.txt` 命令就行。

显然，需要确保圆括号内命令的执行结果具有正确的格式。否则，将得到某些奇怪的结果。

在理解了这些高级脚本编程技术之后，您也许会希望不让脚本为恶意的黑客利用。如今，在很多信息技术部门，安全是一个热门话题，需要严肃对待。下一节帮助您保证脚本的安全性，进而保证系统的安全性。

14.3 shell 的安全性

讨论安全的脚本编程就像打开了一个装着各种蠕虫的罐子。脚本编程的问题是它们用于执行包含了其他程序、文件或者实用工具的命令。通常，这些其他的程序会造成安全上的漏洞。即使您的安全意识非常强，恶意的用户仍然有可能绕过您的意图。另外，对于自负的人来说，

用您自己的代码来对付您并不是什么快意的事情。

如果对安全非常关注，那么最好的办法也许是即使是最简单的任务也不要使用脚本。在设计 shell 的时候并没有考虑到安全性——它们只是用于让工作变得容易些。让工作变得容易的部分内容就是将很多事情移到幕后完成。大多数脚本中的这些隐藏部分正是危险所在。

14.3.1 攻击可能来自何处

关于怎样利用 shell 输入来改变脚本行为的一个示例是修改文件名。脚本也许会忠实地搜索一组文件名并通过管道把它们发送给另一个实用工具，直到它遇到一个特殊的字符，例如一个分号，而该分号的后面是一条 shell 命令——很可能是用户添加的(useradd)命令。脚本也许会认为希望您运行这条用户添加的命令，结果在系统上创建了一个不需要的账号。也可能这条 shell 命令需要更高级别的系统特权才能执行，或者除了使脚本崩溃以外别的什么事情都做不了，但这里确实存在着恶意的企图。

另一个关注点是通常会在脚本中用到的临时文件。它们通常保存在/tmp 目录中，系统上的任何用户都看得见并能使用它们。要得知某个用户在不知道脚本原意的情况下做出某些恶意的东西是很困难的，但一个恶意的用户很可能会删除临时文件或者以某种方式修改它们，从而造成令人讨厌的或者意料之外的结果。

应该密切注意您的脚本正在运行的(或者将要运行的)环境，不但要注意存放脚本的位置，还要注意它运行的环境，以确保如果有未经许可的用户运行脚本，他将不会得到运行脚本所需要的访问权限。

14.3.2 采取预防措施

可以采取很多步骤来控制攻击，比如前面提到的那些。只要恪守一些通用的与安全相关的原则，就可以让别人在破坏脚本时变得很困难。

可以在/tmp 目录中创建一个子目录供脚本使用，这样任何由脚本创建的临时文件都不能由其他用户修改。这并不是一个绝对的安全保证，但确实能对那些可能的黑客形成障碍。黑客们仍然可以用自己的目录把您的目录替换掉，但除此之外也做不了别的什么事情。

把脚本保存在一个别人不能修改的地方。这意味着需要把它们存放到正确设置了权限的安全的目录结构中——换句话说，不要象别的文件那样让每个人都拥有写权限。更进一步，需要确保设置好隐藏的环境变量或者它们正是所希望的样子。如果修改了环境变量，脚本的行为可能在启动之前就被修改了，所以应该在脚本的开头设置自己的 PATH 和 IFS 变量。

最后，您也不会希望脚本拥有过高的权限而胡乱运行。只要让脚本拥有执行任务所需要的权限就好，其他权限一概没有。

接下来，看看一个特殊的 shell，它能帮助您成为一个编写安全的脚本的程序员。

14.3.3 受限 shell

受限 shell 正如它的名字所指的那样——是受到限制的。它只有足够的功能来执行一个有限集的任务。显然，这使得它更适用于安全地编程，但不能认为它是绝对安全的。前面已经提到，安全问题有时候是来自 shell 与其交互的程序而不是直接来自 shell 本身。

表 14-5 显示了受限 shell 的一些特点以及为什么会有这些限制。

表 14-5

特 点	原 因
不能使用 <code>cd</code> 切换目录	防止用户在一个不安全的目录中运行代码
不能设置某些环境变量，例如 <code>SHELL</code> 、 <code>PATH</code> 和 <code>ENV</code>	修改环境变量可能会导致无法预料的结果
不能通过 <code>></code> 、 <code>>&</code> 或者 <code><></code> 等操作符使用 <code>IO</code> 重定向	防止用户创建未经许可的文件
不能使用内置的 <code>exec</code> 来取代进程	任何子进程或子 <code>shell</code> 都不需要和父 <code>shell</code> 一样的运行权限
不能使用包含斜线的命令	防止使用命令切换到未经授权的目录的可能
不能退出受限模式	这是很显然的——您不会希望一个黑客轻轻松松地就把所有的限制取消

要运行一个受限 `shell`，只需要通过 `-r` 选项来启动。或者，如果只希望以受限的方式运行某些特定的命令，可以使用 `set -r` 和 `set +r` 选项。下面的代码片断显示了这种操作：

```
#!/bin/bash -r
# This script won't work

cd /some/other/directory
set +r

exit 0
```

也可以这样做：

```
#!/bin/bash
# This script won't work

cd /some/other/directory
echo "'pwd'"

set -r
cd
echo "'pwd'"
set +r

exit 0
```

如果运行这段脚本，就会看到两条 `set` 命令之间的 `cd` 命令没有起作用，因为不能在受限模式下切换目录。

安全性的另一个很重要的方面是一定要删除临时文件——也称为擦除(wipe)。在介绍文件处理的小节中您看到过一个很基础的示例，这个示例在脚本停止执行之前捕获由中断产生的信号并利用它来删除临时文件。

本节讨论了脚本安全的基础知识。这对普通用户很重要，对系统管理员保持其脚本的安全性就更加重要了。很多系统管理脚本都是以根用户运行的，正如将在下一节中所介绍的那样。

14.4 系统管理

如果希望成为一名系统管理员，就必须了解脚本编程。管理员为了让自己管理范围内的系统和平台能够正常地运行，需要完成大量的工作。在众多工作中，有一些是不会发生变化的，包括从系统中获取信息和找到关于系统的信息、协调系统对资源的使用，以及架设和安装软件和硬件。

脚本编程在这些工作中扮演了一个重要的角色。正如在本章和前面的章节中学到的那样，shell 为您提供了执行几乎所有类型的系统管理任务所需的命令和实用工具。从备份文件到添加新用户，任何与平台在远距离上有关系的事情都是系统管理的范围，而用户的日常工作就是在这些平台上完成的。

脚本的好处之一是可以很好地执行重复的任务。例如，如果您是某所大学计算机实验室的管理员，那么您很快就会发现每年给新用户添加权限是件很让人头痛的事情。编写一段简短的脚本来替您完成这些工作肯定能使您的工作变得轻松很多，不是吗？事实上，由于绝大多数学生都拥有完全相同的权限，也许可以编写脚本从一个文本文件中读取所有的名字并把他们创建成新用户，所有新用户都拥有标准的权限(或者任何您所希望的权限)。

到目前为止，您所做的很多事情都属于管理任务。您看到的所有命令，各种不同类型的变量、实用工具等等，都是管理员用于实现他们的管理角色所使用的工具包的一部分。大多数情况下，需要在下面列出的内容上访问信息或者执行操作：

- 文件系统
- 日志
- 用户
- 进程
- 资源

通常，您会反复需要同一类型的信息。很多管理员使用 cron(参考第 11 章，“在特定的时刻运行程序”)每小时、每天或者每周运行程序以便打印出他们需要的所有系统信息，无论这些信息是什么。由于已经学习过 cron 的实例，这里就不再深入讨论了。只需要知道任何在正常情况下可以运行的脚本都可以发送给 cron 来运行。

请注意很多系统管理任务都需要超级用户权限。结果，必须非常小心地决定哪些任务可以通过脚本来自动完成。

14.4.1 收集信息

自然，您希望能够从系统中获得有关系统是怎样运行的和用户、进程以及文件的状况的信息。您还希望保存系统正在执行的操作的记录和日志。首先要做的第一件事情是查看一下在检索关于系统的信息时有哪些工具可以使用。

当然，有很多命令是针对系统管理的。表 14-6 列出了常用的一些命令以及它们的用途。

表 14-6

命 令	用 途
df	提供了指定文件所在的文件系统的信息。如果没有给 df 提供一个文件名，它将显示所有在系统上已安装的文件系统的信息
du	为每个作为参数传递的文件给出文件系统的块用法(block usage)。如果没有提供参数，将给出当前目录中所有文件的块用法
lastlog	访问日志信息以打印出上一次登录的详细内容。可以指定一个用户，只检索这个用户的登录信息
lsdev	检索在/proc 目录下安装的硬件的信息。通过它可以获得系统中正在运行的硬件的概况
lsdf	显示所有当前打开的文件的列表。默认情况下它将列出由所有进程打开的所有文件
ps	提供关于进程的信息，按所有者和进程 ID 排序
stat	提供关于文件和目录的更详细的信息
top	提供一组占用 CPU 最多的进程
vmstat	提供系统上关于虚拟内存的信息——进程、内存、分页、块 IO 设备、陷阱以及 CPU 的活动
w	显示所有用户的信息，包括进程信息，如果不指定用户的话

还可以使用其他实用工具来收集信息，但以上这些工具是进行工作的基础。在脚本中使用这些工具按一定的规律(使用 cron)、或者在系统启动的时候显示信息，可以及时地了解到所发生的事情，而不用反复地搜寻这些信息。

很多这些工具都有一些选项可以让您在使用它们收集信息的时候详细地指定所需信息的类型。反复练习这些选项并查看联机帮助文档以便获得更多信息。下一节将讨论一些很常用的系统管理工作。

14.4.2 执行任务

收集信息只是一个方面。还必须维护系统良好的工作顺序。Unix 附带的实用工具正好可以帮助您完成这项工作。表 14-7 列出了一些系统管理员很常用的命令。

表 14-7

命 令	用 途
chown	超级用户可以使用 chown 来改变文件的所有者。另外，该文件的所有者可以使用 chown 来改变文件所属的组
chgrp	改变文件所属的组
dump	以二进制格式备份文件。然后可以使用 restore 恢复由 dump 备份的文件
mke2fs	创建新的 ext2 文件系统
mkswap	创建一个交换分区，该分区必须使用 swapon 激活
mount,umount	在设备上安装或者反安装文件系统
ulimit	设置进程可以消耗的资源限制
useradd,userdel	从系统中添加和删除用户
wall	向所有已登录的终端写信息。管理员可以用它在整个系统范围内发送消息
watch	以特定的时间间隔反复执行命令

系统管理员的工作绝不止这些。反复练习这个表和上一个表中给出的命令，您应该可以完成很多非常有用的、与系统编程相关的任务。然而，还有一个内容没有讨论到，而这对跟踪和记录信息是至关重要的——它就是日志！系统日志将在后面的章节中进行讨论，这里所要说的是利用好系统日志对任何系统管理员的工作描述都是一个很关键的部分。

14.4.3 调试脚本

与其他任何编程语言和脚本语言一样，shell 脚本需要某种形式的调试。shell 脚本在很多时候都会出错，从一个简单的语法错误到比较复杂的逻辑上微妙的错误等。然而，和其他功能完善的语言不一样，shell 没有内置的调试器，所以必须使用一些技巧和标记以确保脚本按所期望的方式执行。

怎样检查错误完全取决于所要跟踪的错误的类型。例如，运行下面的脚本将产生一个错误：

```
#!/bin/bash

# echobug
# Faulty echo shell

ech "Guess the secret color" \n"
```

应该立即就能看出错误所在：一个简单的打印遗漏(在代码的最后一行是 ech 而不是 echo)使得脚本不能正确地运行，因为 ech 命令并不存在。我们将这种错误称为语法错误，搜寻和改正这种错误的一种方法是通过 -n 选项来使用 sh 命令解释器，如下所示：

```
sh -n scriptname
```

这不会执行脚本；它只是简单地读每一条命令。更好的是，可以把 -v 选项和 -n 选项组合起来使用，这样可以得到一个很详细的输出。通常，这将强迫 sh 在执行的时候回显出所有的命令。这使得您能够更快地定位错误。

如果错误不是语法错误，而只是不能按照希望的方式运行该怎么办呢？有几种方法可供尝试以找到脚本的问题所在。

首先，可以使用 sh 的 -x 选项。这个选项将把脚本中执行的每条命令的结果回显到屏幕上，以便确认得到了所需要的内容。当然，对于更大的脚本，也许不希望同时调试所有的命令。如果是这样，可以在希望开始调试的地方插入 set -x 命令，并在希望结束调试的地方插入 set +x 命令。为了调试只有一行的 echobug 脚本，可以使用如下命令：

```
#!/bin/bash

# echobug1
# Faulty echo shell

set -x
ech "Guess the secret color"
set +x
```

当通过命令 \$ ~/bin/echobug1.sh 命令执行这段脚本的时候，将得到带有前缀符号(+)的输出。所有由前缀符号开始的输出行都是在 shell 上执行的命令。其他内容则是 shell 处理的结果。

```
$ ~/bin/echobug1.sh
```

```
+ ech Guess the secret color
~/bin/echobug1.sh: line 6: ech: command not found
+ set +x
```

除此以外，还可以在代码中插入 `echo` 命令来打印变量的值从而确定它们是否包含了期望的值。这种技巧对于判断使用了正则表达式的命令返回的结果是否是您需要的非常有用。

更进一步，在流控制语句中，例如 `if` 和 `while` 语句，插入 `echo` 语句可以判断条件是否正确。例如，如果希望一个 `if` 语句的条件在脚本的整个生命周期中至少有一次计算为真，并在 `if` 循环体中插入如下语句：

```
echo "You have reached the body of the statement ok! "
```

这样就可以很容易地判断脚本是否运行到代码的这个部分。如果没有，就可以知道要么是条件有问题，要么是脚本在执行到 `if` 循环之前就已结束了。

14.5 小结

本章介绍了比较高级的脚本编程概念，包括输入和输出重定向。您学习了很多关于 `shell` 函数的内容，包括：

- 怎样传递参数给函数
- 怎样从函数返回一个值到脚本中
- 怎样使用嵌套函数解决复杂的情形
- 怎样使用函数库来节约自己的时间和减少工作量

另外，学习了作用域的概念，包括全局作用域和局部作用域，并学习了怎样使用 `trap` 命令来处理进程发出的信号和清除临时文件。学习了通过把脚本保存到安全的目录结构中并使用受限的 `shell` 来维护脚本安全的重要性，还研究了其他一些系统管理任务，例如信息收集、系统维护和脚本调试的实用工具。

14.6 练习

- (1) 创建一个函数，利用该函数求出传递给它的两个值的和，并使它对所有的 `shell` 都可用。
- (2) 编写一个脚本，该脚本从一个文件的末尾读取信息并添加到另一个文件的末尾。为了让这个脚本更有用，可以让它从一个日志文件中读取信息。
- (3) 编写一个试图被零除的脚本。在结束执行之前它应该发送一个错误消息到屏幕上来。
- (4) 为系统管理员编写一段脚本。该脚本应该告诉管理员日期、已登录的用户、以及哪些文件正在使用之中。在退出之前，脚本应该向所有用户发送一条消息，告诉大家管理员现在正在办公室里。

第 15 章 系 统 日 志

如您现在所知，系统管理是一个广阔的概念，几乎包括维护系统运行的所有方面。它涉及到安装软件、升级软件、管理磁盘空间、控制系统访问以及管理用户账号等责任。一个最重要的(通常也是最单调的)管理工作是管理日志文件。由于系统涉及到很多和用户没有交互而且也不可见的任务，日志就成为跟踪系统的眼睛和耳朵。为了了解计算机在某个时刻正在做什么，必须监视日志文件，每当出了问题，可以参考日志文件。

Unix 系统有一个非常灵活而且强大的日志记录系统，它几乎可以记录下任何想象得到的内容并能处理这些日志以便检索到需要的信息。本章详细地解释了 Unix 的日志记录是如何工作的、怎样使日志跟踪需要知道的事情，以及怎样自动进行这个处理以便不用花费时间来阅读日志文件。

15.1 日志文件

日志文件对系统管理至关重要，因为它们是系统的声音——这正是系统和管理员交流的机制。通过阅读日志文件，系统管理员能知道系统上发生的事情，并可以用这些文件产生一个系统在任意时刻发生的事情的快照。

下面几个小节详细地介绍了日志记录的过程。用户将发现 Unix 的日志记录是一个非常灵活的过程，它能在很大程度上让用户完全控制记录什么内容以及把日志保存到什么地方。Unix 使得用户能详细地指定把什么样的消息写到系统的哪个文件中。

每个供应商在选择存放日志文件的位置上都有一些差别。通常从 SYSV 派生出来的系统(例如 HP-UX, IRIX)把它们的日志文件放在/var/adm 目录下，而 BSD 和 Linux(包括 MacOS X)把日志文件放在/var/log 目录下。/etc/syslog.conf 文件记录了日志文件位于何处，下一节将讨论这个文件。

日志文件包含了很多信息。写到日志文件中的最重要的信息包括：

- 系统启动消息和与启动相关的失败记录
- 用户登录和用户的位置信息
- 安全信息，例如失败的登录
- 电子邮件活动(E-mail activity)(成功和失败)
- 工作排程(cron job)的状态

15.2 syslogd

系统日志是运行在系统上的程序与用户进行交流的方法。当一切正确地按希望进行的时候，日志相当于程序运行清单。当出现错误的时候，日志能够准确地告诉您系统上所发生的事

情，这是日志最具价值的地方。记住，Unix 的基本原则是包含大量的小程序和实用工具，每个程序和工具完成一个特定的任务。由于有很多专用的程序同时运行，因此能够在任何时候知道任意程序在做什么是非常关键的。这正是 syslog 的职责所在。

syslog 是一个专用于系统日志的应用程序。它是 Unix 操作系统最伟大的、也是人们知道得最少的一个部分。syslog 是系统日志记录器(system logger)的意思，它是一个实用工具，为整个系统提供了重要的、统一的日志记录功能。syslog 守护进程(syslogd)管理系统上的所有日志，而不是让每个应用程序以独特的方式分别管理它们的日志文件。syslogd 并不产生任何记录到系统日志文件中的消息。它相当于计算机上一个无所不在的卫兵，指示由各个程序产生的消息应该位于何处。系统日志记录器的操作相当简单。各个程序把它们的日志记录项发送给 syslogd，后者参考配置文件/etc/syslogd.conf，并在找到一个匹配项的时候把日志消息写到指定的日志文件中。

有 4 个基本的 syslog 术语需要理解：

- 设备(facility)——用于描述递交日志消息的应用程序或进程的标志符。例如 mail、kernel 和 ftp。
- 级别(level)——消息的重要级别指示器。级别在 syslog 中是作为准则定义的，范围从调试信息到关键事件等。它们相当于标签。怎样处理这些级别完全由用户决定。
- 选择域(selector)——由一个或多个设备和级别组合而成。传递到 syslog 的日志消息将与配置文件中的选择域进行比较。当一个传递进来的事件和某个选择域匹配时，将采取一个动作。
- 动作(action)——对传递进来的与一个选择域匹配的消息进行的处理。这些动作能把消息写到一个日志文件中，在控制台或者其他设备上回显消息，把消息写给一个已登录的用户，或者把消息单独传递给另一个 syslog 服务器。

15.2.1 syslog.conf

下面是系统日志记录器配置文件/etc/syslog.conf 的一个示例：

```
*.err;kern.*;auth.notice;authpriv,remoteauth,install,none;mail.crit
                                         /dev/console
#.notice;*.info;authpriv,remoteauth,ftp,install.none;kern.debug;mail.crit
                                         /var/log/system.log

# Send messages normally sent to the console also to the serial port.
# To stop messages from being sent out the serial port, comment out this line.
#*.err;kern.*;auth.notice;authpriv,remoteauth.none;mail.crit
                                         /dev/tty.serial

# The authpriv log file should be restricted access; these
# messages shouldn't go to terminals or publically-readable
# files.
authpriv.*;remoteauth.crit                /var/log/secure.log

lpr.info                                   /var/log/lpr.log
mail.*                                    /var/log/mail.log
ftp.*                                     /var/log/ftp.log
netinfo.err                              /var/log/netinfo.log
install.*                                 /var/log/install.log
```

install.*

@192.168.1.50:32376

*.emerg

如您所见，`/ect/syslog.conf` 是一个纯文本文件，它的文本看起来呈两列。第一列——具体地说是每行的第一部分——是日志信息的来源，也就是提供信息的系统或程序。来源通常定义成两个部分，设备和由点号分隔的级别——`facility.level`——通常称为选择域。

第二列(每行的结尾)是信息的目的地，可能包含很多内容，包括文本文件、终端、串口号、甚至是运行在另一个系统上的系统日志记录器。在示例文件的末尾会注意到 `install.*`选择域出现了两次。把同一个结果发到多个地方是完全合法的。在这里，将属于设备安装的日志项添加到文本文件`/var/log/install.log` 的末尾，同时发送到 IP 地址为 `192.168.1.50`(端口号为 `32376`)的网络 `syslog` 服务器上。

表 15-1 中列出了对选择域可用的设置。

表 15-1

设 备	说 明
auth	与询问用户名和密码相关的活动(getty, su, login)
authpriv	和 auth 一样，但是把日志记录到只有特定用户才能读取的文件中
console	用于捕获通常直接发送到系统终端的消息
cron	来自 cron 系统调度程序的消息
daemon	系统守护程序 catch-all
ftp	与 ftp 守护程序相关的消息
kern	内核消息
lpr	来自行打印系统的消息
mail	与邮件系统相关的消息
mark	用于在日志文件中产生时间戳的伪事件
news	与网络新闻协议(network news protocol, nntp)相关的消息
ntp	与网络时间协议(network time protocol)相关的消息
local0-local7	每个站点定义的本地设备

表 15-2 解释了 `syslog.conf` 文件中可用的级别。

表 15-2

级 别	说 明
emerg	系统无法使用。消息发送到在所有终端上已登录的所有用户
alert	必须立即采取的行动
crit	关键条件
err	错误条件
warning	警告条件
notice	正常的但是很重要的条件

(续表)

级 别	说 明
info	信息性消息(informational message)
debug	调试-级别消息。平时使用时应处于关闭状态，因为这些本来就很冗长的消息将很快用完磁盘空间
none	用于指定不记录消息的伪级别

设备和级别的组合可以让您分辨出记录了什么以及信息去向何处。由于每个程序都如实地把它的消息发送给系统日志记录器，记录器将根据在选择域中定义的级别决定跟踪什么以及丢弃什么。指定一个级别之后，系统将跟踪所有该级别以及更高级别的消息。级别相当于一个最低水位标记(low-water marker)。例如，它表示如果希望在邮件系统中有警告消息的时候获得提示，那么当出现 `err`、`crit`、`alert` 和 `emerg` 消息中的任何一种的时候，也希望获得提示。

让我们看看/etc/syslog.conf 文件中的示例行：

```
lpr.info /var/log/lpr.log
mail.* /var/log/mail.log
ftp.* /var/log/ftp.log
```

来自打印系统(lpr)的消息，如果其日志级别等于或者大于 `info`，那么将记录到/var/log/lpr.log 文本文件中。后两行示例显示了通配符在选择域中的用法。通过使用通配符表示级别，系统将把所有来自该设备的消息记录到指定的地点。在这里，任何来自邮件设备的消息都将记录到 /var/log/mail.log 文件中，而来自 ftp 服务器的消息将记录到/var/log/ftp.log 文件中。

下面的“实战”小节创建一个条目把 ssh 的活动记录到一个新的日志文件 ssh.log 中。这个示例是在 Mac OS X 系统上完成的，所以如果在/etc 目录下没有 sshd_config 文件，就需要参考系统文档以找到该文件的位置。sshd_config 文件通常位于/etc/ssh 和/usr/local/etc/ssh 目录。另外，虽然 Mac OS X 把 SyslogFacility 设置为 AUTHPRIV，但是可能还会发现另一个默认的 SyslogFacility。无论使用哪一个，在这个练习中都要把它修改成 LOCAL7。

实战

配置日志文件

配置 syslog，把所有与 ssh 有关的活动都记录到一个名为 ssh.log 的日志文件中。

(1) 编辑/etc/sshd_config。把内容为 SyslogFacility AUTH 的行修改成：

```
SyslogFacility LOCAL7
```

(2) 编辑/etc/syslog.conf。在文件的末尾添加如下内容：

```
local7.* /var/log/sshd.log
```

(3) 输入命令：

```
sudo touch /var/log/ssh.log
```

(4) 通过输入恰当的系统命令(例如，`sudo kill -HUP 'cat /var/run/syslogd.pid'`)重启 syslog。

(5) 通过创建一个到本机的连接进行测试。

打开一个终端窗口，输入 `ssh localhost`，并以自己的账号登录系统。在收到提示时，输入

exit 退出系统。

然后输入命令：

```
sudo tail -f /var/log/ssh.log.
```

将看到一条关于连接的详细的记录。

如果没有看到记录下来的连接日志，那么需要执行如下命令重启 ssh 守护进程：

```
sudo /etc/init.d/sshd restart.
```

工作原理

上例中发生的事情是：

(1) 除了其他方式外，还可以通过编辑 sshd 配置文件来修改 ssh 守护进程记录它的活动方式。sshd 默认情况下使用 AUTH 设备记录它的活动。在该练习中，将其修改为一个用户设备，LOCAL7，该设备只能在本地使用，系统上存在的程序将不使用它。现在可以容易地把所有 sshd 消息都定向到一个特定的文本文件。

(2) 系统将指示 syslog 守护进程匹配 local7 事件并把它们添加到文本文件/var/log/sshd.log 的末尾。

(3) 在向该文本文件写入信息之前它必须位于系统中。syslog 并不会创建文件，所以可以使用 touch 命令创建文件/var/log/ssh.log。

(4) 对/etc/syslog.conf 所做的修改在重启 syslog 守护进程之前是不会起作用的。通过在系统上执行与发行商相关的(vendor-appropriate)命令来完成该操作(输入命令 man syslog 将告诉您如何在系统上执行该操作)。

15.2.2 消息

现在让我们看一看消息的内容，它们将由 syslog 管理。它输出的消息以非常特殊的格式结构化，正是这个特点使得解析系统日志文件对于 perl、sed 和其他处理文本、特别是处理正则表达式的实用工具来说是一件非常繁琐的事情。

并不是所有的消息都以相同的方式保存或者包含了同样的信息。出于这个原因，在编写脚本或正则表达式提取信息之前，应该熟悉希望解析的日志的特定格式。

下面是 ssh.log 文件中的一条消息：

```
Jan 7 15:31:37 pandora2 sshd[16367]: Accepted password for craigz from
192.168.1.222 port 5242 ssh2
```

该消息的格式如下：

```
Date time hostname process[pid]:action:version
```

因此该消息可以解释为：

1月7日下午3:31在主机 pandora2 上以 PID 16367 运行的 sshd 执行了如下操作：从主机 192.168.1.122 的 5242 端口接收用户 craigz 的一个连接；客户机的版本是 ssh2。

系统上所有 sshd 日志消息的格式都是一样的，无论该格式是什么。这意味着可以使用任何已经见过的实用工具(很可能是一个 shell 脚本)来提取需要的信息。只需要熟悉系统上的程序

和实用工具使用的格式即可。

15.2.3 日志记录器

大多数 Unix 都提供日志记录器(logger)系统实用工具, 可以使用它把消息发送给 syslogd。该程序可以把消息发送到用户指定的日志文件, 而且可以指定设备和级别。下面是 logger 命令的一个示例:

```
darwin$ logger -p local7.info -t test_logger_message "test message"
```

-p 标志表示优先级, 在这里是 local7.info, -t 标志表示-tag, 是和消息一起出现的一个标签。

在“配置日志文件”的练习中, 如果把 syslog 配置为把 local7 设备的消息记录到文件 /var/log/sshd.log 中, 该 logger 命令将把如下文本行添加到这个文本文件的末尾:

```
Nov 8 19:13:09 localhost test_logger_message: test message
```

在为系统管理创建的 shell 脚本中包含 logger 命令是有用的。通过使用 logger 命令, 可以访问标准的日志记录功能而不用从零开始编写代码。这使得跟踪脚本正在做的事情和确保与日志记录有关的代码没有引入错误变得更容易。

脚本编程已在第 13 章和第 14 章中讨论。

15.3 轮循日志

当 syslog 实时地评估事件并发送消息到各个在/etc/syslog.conf 中配置的日志文件的时候, 它将把消息添加到这些已经存在的文本文件的末尾。随着添加的消息越来越多, 日志文件变得非常大, 您很快就会意识到保留所有由系统发出的消息是不现实的。

作为系统管理员, 必须及时地复查最近的系统消息, 但还务必不能让日志文件占满所有的空闲磁盘空间。建立日志保留时间(log retention)策略是一个很好的解决方案。和日志记录的其他方面一样, 在管理日志文件的保留时间上也有很多选择。每个站点的需求都不一样, 需要设计一个适合自己以及自己的情况的策略。

下面的示例是一个基本的策略, 当然, 可以使用任何自己希望的策略和调整。

每个文件每周维护一次。也就是说, 每个日志文件包含了连续 7 天的记录。然后, 把一周以前的日志记录归档到一个单独的文件中。有 7 个归档文件, 按顺序每周使用一个, 每个文件包含一周的日志。这些保存在系统上的旧文件和当前的日志文件提供了两个月的系统日志数据。旧日志文件应该和当前的日志文件存放在相同的目录中而且应该对其进行压缩, 以便尽量少地占用磁盘空间。

大多数 Unix 系统都带有安装好的管理日志轮循的工具。可以对其进行配置以执行任何日志保留时间策略。这些程序通常都由 cron 工具、或者周期性的守护进程来运行, 它们将自动解析日志文件, 自动创建并归档以前的日志文件。如果成功实现了刚才描述的策略, 在目录中将产生类似如下清单中的文件:

```
system.log
system.log.0.gz
```

```

system.log.1.gz
system.log.2.gz
system.log.3.gz
system.log.4.gz
system.log.5.gz
system.log.6.gz
system.log.7.gz

```

在不同的 Unix 发布版本中日志记录策略执行的方式不尽相同，因而实现执行策略的程序也有所不同。例如，FreeBSD 提供了 newsyslog，而 Red Hat 提供了优秀且灵活的 logrotate。每个程序都能使工作变得轻松很多。

15.4 监视系统日志

系统日志记录配置的灵活性给系统管理员带来了极大的好处。知道保留了何种类型的消息和将这些消息存放在系统上的何处，将使得调查不正常的或者错误的行为变得容易很多。

然而，系统管理的中心任务是维持系统运行，所以虽然通过复查能够知道在某个系统失败的时候什么事情出了错是有用的而且也是必须的，但更重要的是精确地了解系统当前的运行状态。

假设有无穷多的时间和足够的耐心，可以为系统上的每个日志文件打开一个终端窗口，在每个窗口中执行 `tail -f` 命令，然后实时地阅读每个日志文件。阅读每条由 syslog 发送过来的消息的确能及时而准确地了解到系统上所发生的事情，因此可以采取任何必要的正确措施以确保系统能继续正确地运行。

然而，在实际操作中，这是完全不切实际的。即使不介意实时地阅读所有日志文件，也无法完成这项任务，因为必须花时间去休息，去做其他自己能够正常工作的必要的事情。但是，以同样高度的警惕性来阅读这些日志是非常重要的，就像亲自阅读它们一样。

自动化在这里变得非常关键。有几个优秀的能够自动监视日志的包可供使用。其中的两个工具是 logwatch 和 swatch，下一节将研究它们。这两个程序提供了优秀的操作。logwatch 提供了简洁的日志文件摘要，而 swatch(Simple WATCHer)根据预定义的触发器及时地监视日志文件并根据这些事件发送警报(alert)。

15.4.1 logwatch

可以从 <http://www2.logwatch.org:81/> 下载 logwatch。大多数基于 Red Hat 的发行版本都包含该工具，而且很可能在系统上已经存在该工具了。如果需要下载和安装该软件，RPM(Red-hat Package Manager)和基于源代码的下载包都可以得到(在写这本书的时候，logwatch 的最新版本是 5.2.2)。本节将介绍基于源代码的安装方法。

该软件是作为脚本运行的，所以不需要编译。只需要下载源代码并运行下面的命令：

```

$ tar -zxvf logwatch-5.2.2.tar.gz
$ cd logwatch-5.2.2
$ sudo mkdir /etc/log.d
$ sudo cp -R scripts conf lib /etc/log.d

```

如果想获得更多的安装信息，可以参阅随 logwatch 一起发布的 README 文件。

一起发布的还有联机帮助文档(man page)格式的文档。要安装这些联机帮助文档, 执行下面的命令:

```
$ sudo cp logwatch.8 /usr/share/man/man8
```

也许还希望把 logwatch 的 Perl 脚本复制到某个中心位置, 通常, /usr/sbin 是存放这些脚本的好地方。

```
$ sudo cp scripts/logwatch.pl /usr/sbin
```

logwatch 不需要任何额外的配置就可以处理绝大多数标准的日志文件, 但它非常灵活, 您能很容易地创建定制日志文件定义。所需做的事情是在/etc/log.d/conf/logfiles 目录下产生一个配置文件, 在/etc/log.d/conf/services 目录中定义一个服务过滤器, 然后在/etc/log.d/script/services 目录中创建该过滤器。该过滤器只是简单地读入日志记录项然后输出报告信息。更多关于配置自定义日志过滤器的信息请参考 logwatch 软件附带的文档, HOWTO-Make-Filter。

logwatch 的配置文件/etc/log.d/conf/logwatch.conf, 注释得很详细, 很容易阅读。参考该文件以定制 logwatch 的安装。

任务计划 (crontab)中的如下条目将在每晚 11:00 运行 logwatch:

```
0 23 * * * /usr/sbin/logwatch.pl
```

下面是一个运行中的服务器上的 logwatch 输出的示例(出于保密, 所有 IP 地址都用 192.168.x.x 代替)。这份报告正在跟踪来自于 ftp 服务器和 Apache 服务器的日志, 并报告 Mac OS X 系统上的空闲磁盘空间。这就是由 logwatch 产生的典型报告。该报告通过电子邮件发送到添加在 logwatch 配置文件中的 IP 地址处。

```
##### LogWatch 5.2.2 (06/23/04) #####
Processing Initiated: Sat Nov 13 23:00:05 2004
Date Range Processed: yesterday
Detail Level of Output: 5
Logfiles for Host: xxx.local
-#####>#####>#
----- ftpd-xferlog Begin -----
TOTAL KB OUT: 121287KB (121MB)
TOTAL KB IN: 718967KB (718MB)
Incoming Anonymous FTP Transfers:
192.168.1.1-> /Library/Webserver/Documents/file_depot/work/a_and/AE_1.mov b
----- ftpd-xferlog End -----
----- httpd Begin -----
1798.89 MB transfered in 550 responses (1xx 0, 2xx 373, 3xx 141, 4xx 35, 5xx 1)
74 Images (0.76 MB),
140 Movies files (1785.90 MB),
327 Content pages (0.17 MB),
2 Redirects (0.00 MB),
1 mod_proxy connection attempts (0.00 MB),
6 Other (12.06 MB)
```

Connection attempts using mod_proxy:

192.168.1.10 -> 192.168.1.12 ; 1 Time(s)

A total of 3 unidentified 'other' records logged

\x05\x01 with response code(s) 1 501 responses

GET /work/client_logo_hi_res.psd HTTP/1.1 with response code(s) 4 200 responses

GET /editors/ediitor HTTP/1.1 with response code(s) 1 401 responses

A total of 1 ROBOTS were logged

----- httpd End -----

----- Disk Space -----

Filesystem	1K-blocks	Used	Avail	Capacity	Mounted on
/dev/disk0s9	30005340	21754248	7995092	73%	/
devfs	92	92	0	100%	/dev
fdesc	1	1	0	100%	/dev
<volfs>	512	512	0	100%	/.vol
automount -nsl [291]	0	0	0	100%	/Network
automount -fstab [301]	0	0	0	100%	/automount/Servers
automount -static [301]	0	0	0	100%	/automount/static

----- Fortune -----

All science is either physics or stamp collecting.

-- Ernest Rutherford

LogWatch End

如您所见，输出很详细，组织得很好，非常清晰。这样的信息对于任何系统管理员来说都是很有用的。

15.4.2 swatch

每晚阅读日志文件摘要可以知道一天中所发生的事情。但是，对于某些特定类型的事件，需要在它们发生时或者发生后尽快地了解。这些事件包括磁盘或者文件集(volume)将要装满、根用户登录尝试失败、其他登录失败以及其他类似的关键事件。

swatch 是一个实用工具，它能在识别一个触发器事件的时候采取很多行动。例如，可以将写入到日志的一次失败的根用户登录尝试设置为一个触发器事件，而相应的操作可能是提醒系统管理员。

可以从 <http://swatch.sourceforge.net> 下载 swatch(编写本书时 swatch 的最新版本是 3.1.1)。要安装 swatch，请下载该归档文件并执行下列命令：

```
$ tar -zxvf swatch-3.1.1.tar.gz
$ cd swatch-3.1.1
$ perl Makefile.PL
$ make
$ make test
$ make install
$ make realclean
```

如果安装的过程中遇到类似如下错误，则需要安装在错误列表中列出来的 CPAN(Comprehensive Perl Archive Network)模块。

```
Warning: prerequisite Date::Calc 0 not found at (eval 1) line 219.
Warning: prerequisite Date::Parse 0 not found at (eval 1) line 219.
```

```
Warning: prerequisite File::Tail 0 not found at (eval 1) line 219.
Warning: prerequisite Time::HiRes 1.12 not found at (eval 1) line 219.
```

通过如下命令完成所需的安装:

```
$ sudo perl -MCPAN -e "install Date::Calc"
$ sudo perl -MCPAN -e "install Date::Parse"
$ sudo perl -MCPAN -e "install File::Tail"
$ sudo perl -MCPAN -e "install Time::HiRes"
```

swatch 希望在执行该程序的用户主目录下有一个名为 `.swatchrc` 的配置文件。另一种方法是在命令行通过 `-config-file` 指示符指定配置文件。

配置文件由制表符分隔的字段(field)组成(表 15-3 之后有一个配置文件条目的示例)。有两个字段是必须的, 其他字段可选。两个必须的字段是模式(pattern)和动作(action), `throttle` 是一个可选字段。模式是将要匹配的规则, 动作是当找到一个匹配的时候将要发生的事情。表 15-3 列出了可能的操作。

表 15-3

操 作	说 明
Bell	发出系统响铃
Echo	把匹配行发送到 STDOUT。格式选项, 例如 <code>red</code> 或者 <code>blink</code> 等此时可用
Mail	把匹配行发送到正在运行这个命令的用户。可以指定备选用户或对象
Pipe	把命令作为参数, 并把匹配行通过管道发送给该命令
Exec	和 <code>pipe</code> 类似, 把命令作为参数。使用特殊变量(<code>\$*</code> 和 <code>\$0</code>)可以把匹配行发送到该命令。匹配行的某些部分可以用 <code>\$1</code> 、 <code>\$2</code> 等指定
Write	把一个用户名或者一组由冒号分隔的用户名作为参数并使用系统写命令把匹配行发送给这个(些)用户
Throttle	设置在匹配行上采取操作之前必须等待的小时、分钟和秒数。例如, 控制 <code>2:00:00</code> 将限制对被监视的特定模式采取的动作只能每两小时发生一次。这对于防止反复触发相同的触发器从而产生大量的动作是有用的
Ignore	忽略匹配行, 不采取任何操作

下面是 `swatch` 配置文件的一个示例:

```
ignore /worthless_message/
watchfor /[dD]enied|/
echo=red
mail addresssadmin@my.organization.com, subject="Important Event"
throttle 1:00:00
```

该配置将丢弃任何包含 `"worthless_message"` 的文本行。但是, 对于找到包含 `"Denied"` (或者 `"denied"`) 的文本行, 将采取如下操作:

- 以红色文本把该行发送给正在运行该命令的终端
- 向 `admin@my.organizaton.com` 邮箱发送主题为 `"Important Event"` 的电子邮件

- 确保该邮件每小时只发送一次(throttle 命令)

swatch 可以从日志文件中非常准确地选择出文本,应该查阅该软件自带的 examples 文件夹中的示例,并阅读关于 swatch 的文档,以便理解怎样配置该软件才符合需求。

15.5 小结

本章介绍了日志是什么以及为什么需要它们。考查了系统日志守护进程(syslogd)并学习了 syslog 是如何工作的以及怎样使用它来定制自己的日志记录。

了解了怎样创建一个用于记录日志的文件和怎样使用 syslog.conf 配置文件设置 syslogd 把日志保存到该文件中,并研究了日志轮循。决定记录什么消息以及怎样保存它只是工作的一部分。了解了两个用于监视系统日志的实用程序: logwatch 和 swatch。

到现在为止,您应该认识到 Unix 强大的功能和灵活性,这些功能和灵活性能精确地维护系统的运行,并提供一个平台对到来的消息作出反应。创建一幅关于 Unix 中正在发生的事情或者是整个系统的场景,现在对您来说可能是一件很惬意的事情。

15.6 练习

- (1) 希望把内核警报打印到屏幕上。怎样实现?
- (2) 修改 syslog.conf 文件,以使邮件调试信息发送到另一台主机上(可以假设主机的名称是 Horatio 而且它上面有一个系统日志守护进程)。
- (3) 怎样通过 swatch 在发生登录密码错误的情况下向根用户发送一封电子邮件? 这个动作还应该包括发出系统响铃并提供一个有意义的电子邮件主题(假设登录失败时会在某个日志中出现 INVALID 字符串)。

第 16 章 Unix 网络互联

访问计算机网络对于计算机应用程序互通重要的数据仿佛一夜之间从昂贵的奢侈变成不可或缺的要求。虽然计算机网络的实现有很多种,但是网络进化的中心,Internet,只有一个协议,TCP/IP。这个协议在世界范围内广泛使用,几乎所有的计算机系统,从 Mac OS X 到 Novell Linux 桌上电脑、到 Sun Solaris 和 IBM 的 AIX、甚至 Microsoft 公司的非 Unix 操作系统 Windows,使用的都是 TCP/IP 协议。它是在当初为基于 Unix 的 BSD 操作系统实现的版本上开发出来的。

随着人们对 Internet 和其他网络的使用以一种令人难以置信的速度增长,有必要提供网络互联工具和服务以利用今天高速的计算带宽和计算能力。本章讨论这些重要的工具并研究底层的 TCP/IP 协议是怎样工作的,同时提出前面的章节中关于系统自动化和脚本编程方面的经验,从而显示怎样管理或支配 Unix 系统上的网络资源。

Mac OS X 系统使用 TCP/IP 进行网络互联,虽然 TCP/IP 协议传统上是通过 GUI 界面进行配置的,但是本章提到的所有命令行实用工具在 Mac OS X 以及其他 Unix 操作系统上都工作得很好。

16.1 TCP/IP

TCP/IP 实际上是指协议族的很多不同方面,该协议族提供了多种网间通信的方法,这些方法通过两种特定的协议在特定的网络层驱动或者内联:传输控制协议(Transmission Control Protocol, TCP)和 Internet 协议(Internet Protocol, IP)。

16.1.1 TCP

TCP 协议是 TCP/IP 协议族中可靠的、面向连接的传输层协议。它提供了一种方法,保证数据能够按顺序到达目的地。这里所说的按顺序很重要,因为该协议会把数据分割成数据包,通常称为分段(segment)。之所以有这样的保证,是因为 TCP 包含了一个重传方案(retransmission scheme),如果接收主机没有向发送方回送一个分段的确认,那么发送方将重发该分段。

TCP 数据包的头部由以下几个部分组成:

- 目标端口和源端口(destination and source port)——TCP 上层协议连接的起点和终点。
- 序号(sequence number)——特定数据包在所有数据包中的序号。
- 确认序号(acknowledgement number)——在发送一个确认包的时候,所有数据包的高位包序号都将用作确认序号。
- 窗口(window)——根据网络中一个慢连接处理数据包的时间长短来决定发送数据包的数量。
- 校验和(checksum)——确保发送的数据是正确的。

在 Internet 上使用的大多数应用层协议都使用 TCP 作为它们的传输层协议,包括 HTTP、FTP、SMTP、TELNET 和 POP。本章稍后将讨论这些协议。

16.1.2 IP

IP 是把数据从 Internet 上的一台计算机发送到另一台计算机实际所使用的方法或者协议。每一台计算机，或者称为主机，都拥有至少一个地址，这个地址能够将该主机和其他主机惟一地区别开。当 TCP 将数据分成多个数据包以便传输时，IP 将为每个数据包提供发送者和接收者的 Internet 地址，然后将数据包发送到网关。网关可确定 IP 在组成 Internet 的互联网络中所指向的位置。该网关读取目的地址并把数据包转发给一个邻近的网关，这个网关也是读取目的地址并进行转发，直到某个网关识别出该数据包属于它的一个直接邻居或者域(domain)。然后这个网关把数据包直接转发给特定地址的计算机。

由于一条消息分成了多个数据包，如果有必要，每个数据包可以通过 Internet 上的不同路由传送。数据包可能以与发送时不同的顺序到达，就像它们的分割是由 TCP 完成的一样，把它们组合成原来的消息也是由 TCP 来完成的。与 TCP 不同的是，IP 是一个无连接的协议，这意味着在进行通信的两个端点之间没有连续的连接。在 Internet 上传送的每个数据包都被当作一个独立的数据包，和其他任何数据单元没有任何联系。

16.1.3 与 TCP/IP 一起使用的其他协议

前面已经提到，TCP/IP 包含很多不同的协议，这些协议相互合作从而使 TCP/IP 网络能够工作。协议族中通常称为 TCP/IP 的其他协议部分如下所述，它们工作在整个连接的不同层上以保证正确的数据传送。

1. ICMP(Internet Control Message Protocol, Internet 控制消息协议)

Internet 控制消息协议(ICMP)用于在通过 IP 协议互联的主机和网关之间发送流量控制信息。创建和传送 ICMP 数据报的目的是为了响应某些 IP 数据包，这些数据包需要以某种形式的状态回应最初的发送方，通常表示主机之间的一次连接失败。因此，Unix 中的有些工具，例如 traceroute 可用于判断网络的状态信息，本章稍后将讨论这个工具。尽管 ICMP 不是一个面向连接的协议，而且很多网关都把 ICMP 放到一个比较低的优先级上，以至于在发生网络拥塞的时候 ICMP 数据很可能会丢失，但 Unix 中很多常用的工具，例如 traceroute，都使用 ICMP 协议。

2. UDP(User Datagram Protocol, 用户数据报协议)

UDP，也就是用户数据报协议，是与 TCP 和 ICMP 相似的传输层协议。然而，和 TCP 不同的是，从确保数据包到达其目的地这个意义上来说它是不可靠的。这反过来可以使那些希望自己进行流量控制的应用程序具有更大的灵活性，而且可以用于面向连接的方法不适合的地方，例如对于多播数据。也就是说，UDP 为其他在任意时刻向一台或多台计算机广播数据的协议层提供了一种方法，使得这些协议——例如 NFS、DNS 和 TRP 等协议——能够更多地控制数据流量或者向多个客户端同时进行广播。

3. ARP(Address Resolution Protocol, 地址解析协议)

ARP 是一个网络层(network-layer)协议，用于把 IP 地址转换成网络接口硬件的物理地址，例如把以太网地址转换成以太网网卡地址。希望获得物理地址的主机发送一个基于 TCP/IP 的

请求就需要用到 ARP 协议。网络上具有请求中包含的 IP 地址的主机于是以它的物理地址作为应答。

4. RARP(Reverse Address Resolution Protocol, 反向地址解析协议)

如果说 ARP 是把 IP 地址映射到网卡, 那么 RARP 则是把一个未知的网络接口卡的硬件地址映射到一个 IP 地址。这发生在开机的时候, 内嵌在硬件中的软件使用 RARP 从一个服务器或者路由器请求它的 IP 地址。大多数计算机使用自动主机配置协议(Dynamic Host Configuration Protocol), 本章后面将对其进行讨论, 而不是 RARP。

5. IGMP(Internet Group Management Protocol, Internet 组管理协议)

Internet 组管理协议(Internet Group Management Protocol, IGMP)是 Internet 中的 IP 多播标准, 用于在单个网络的特定多播组中建立主机成员关系。该协议的机制允许一台主机向本地路由器表明它希望接收某个特定多播组发出的消息。多播(multicast)是一种向网络中特定的一组计算机发送数据的方法, 而不是向所有的机器发送, 如果这样就成为广播(broadcast)了, 也不是向单个特定的机器发送。IGMP 是在 TCP/IP 网络中实现多播的协议。

6. HTTP(Hypertext Transfer Protocol, 超文本传输协议)

超文本传输协议(Hypertext Transfer Protocol, HTTP)是万维网(World Wide Web)使用的底层协议。HTTP 定义了怎样格式化和传送消息, 以及 Web 服务器和浏览器对各种命令应该采取什么样的操作作为回应。

人们将 HTTP 称为无状态协议, 因为它的每个命令都是独立执行的, 对前面执行过的命令没有任何了解。这就是实现智能地响应用户输入很困难的主要原因。

和 UDP 和 TCP 等协议不同, HTTP 以及其他协议, 例如文件传输协议(File Transfer Protocol, FTP)、简单邮件传输协议(Simple Mail Transfer Protocol, SMTP)和简单网络管理协议(Simple Network Management Protocol, SNMP), 都是应用层协议, 它们位于 TCP 和 IP 为网络连接创建的协议栈的顶端。

7. FTP(File Transfer Protocol, 文件传输协议)

FTP 也是一个应用层协议, 用于在 Internet 上交换文件。它的工作方式和 HTTP 把网页从服务器传送给用户的浏览器以及 SMTP 在 Internet 上传送电子邮件的方式一样, 都是使用 Internet 的 TCP/IP 协议族来实现数据传送的。

8. SMTP(Simple Mail Transfer Protocol, 简单邮件传输协议)

前面提到, SMTP 是在服务器之间发送电子邮件的协议。然而, 和 HTTP 或 FTP 不同的是, 其他协议, 例如邮局协议(Post Office Protocol, POP)或者 Internet 消息访问协议(Internet Message Access Protocol), 通常用于从上一个节点(point)检索消息、在邮件服务器中将消息排队、以及客户端请求在邮件队列中等待的消息集合。另外, SMTP 通常用于从客户端发送消息到邮件服务器。

16.1.4 网络地址、子网、子网掩码和 TCP/IP 路由选择

前面描述的包含在 IP 协议中的 IP 地址解析方案对于引导数据包在计算机网络或互联网中传输是整体的。每个 IP 地址都有特定的组成部分而且有一个基本格式，可以将这个格式细分并在子网内为特定的机器创建地址。

TCP/IP 网络上的每台主机都拥有一个惟一的 32 位逻辑地址，该地址被分为两个主要部分：网络号和主机号。网络号标识一个网络，必须由服务提供商分配。主机号标识一个网络上的一台主机，由本地网络管理员分配。32 位的 IP 地址分成 4 个 8 位组，组与组之间由点号分隔，并以十进制格式表示。每个 8 位组的值在 0 到 255 之间。

然而，IP 地址的什么部分表示网络和什么部分与主机有关取决于 IP 地址属于哪一类地址——也就是说，地址的类型决定了地址中的哪个部分属于网络地址和哪个部分属于主机地址。一个网络上的所有主机共享相同的网络前缀但是必须具有惟一的主机号。有 5 种网络地址类型，但最常用的只有如下 3 种：

- **A 类地址**——在一个 A 类网络中，4 个分组的第一个部分只能在 1 到 126 之间，标识一个网络，而余下的 3 个组指示网络内的主机。因此一个 A 类网络中可以有 16 000 000 台惟一的主机，而这样的网络总共有 126 个。在一个如 16.42.226.10 的 A 类地址中，16 标识网络，42.226.10 标识该网络上的一台主机。
- **B 类地址**——在一个 B 类网络中，表示 IP 地址的 4 个分组中的第一分组在 128 到 191 之间。在这种安排中，前两个分组标识网络，而余下的两个分组标识网络中惟一的一台主机。因此一个 B 类网络有 65 000 台主机，而这样的网络总共有 16 000 个。在一个如 128.204.113.42 的 B 类 IP 地址中，128.204 标识网络，而 113.42 标识这个网络上的一台主机。
- **C 类地址**——在一个 C 类网络中，4 个分组中的第一个分组只能在 192 到 223 之间，前 3 个分组标识网络而剩下的分组标识网络内部的主机。因此，一个 C 类网络上可以有 254 台主机，这样的网络总共有 2 000 000 个。在一个如 192.168.42.1 的 C 类地址中，192.168.42 标识网络，而 1 标识这个网络上的一台主机。

无论如何，一个子网中无论使用什么类型的网络，有几个 IP 地址都将保留下来，用于特殊的用途。例如，任何子网中都有一个特定的地址代表一个路由器或网关，网络内部的任何计算机都可以通过这个路由器或网关到达其他网络上的计算机。其他保留的 IP 地址可能包括广播地址——一个以广播 IP 作为地址的数据包将发送到子网上的所有计算机——或者是一个代表整个子网的地址。此外，任何系统上都应将地址 127.0.0.1 指定为本地回送接口(loopback interface)。也就是说，除了某个惟一给定的地址以外，127.0.0.1 表示代表对 Unix 系统与机器的软件连接。另外，地址 0.0.0.0 对于任何 Unix 系统都表示整个网络。换句话说，在一个 C 类网络上，对某个特定的子网，实际上只有 253(而不是 256)个 IP 地址可以用作主机名。

1. 什么是子网？

可以将 IP 网络分成更小的网络，我们把它们称为子网(subnetwork 或者 subnet)，这给网络管理员带来了很多好处，包括新的灵活性、更有效地使用网络地址，以及包含广播流量的能力，因为广播不会越过路由器。

虽然 IP 地址的网络部分必须由服务提供商分配，但子网和主机地址都是本地管理的内容。

同样地,从网络地址来看,外界将把整个公司看作是一个单独的网络,而不知道网络的内部结构。

因此可以将一个特定的网络地址分为一个或多个子网。例如,192.168.1.0、192.168.42.0和192.168.72.0都是192.168.0.0网络内部的子网,由于把一个地址的主机部分全部设置为0表示整个网络,那么对于在同一个子网上的机器就应该把子网掩码(netmask)设置成IP地址的前3个数字。

如果一个网络是分为多个子网来实现的,那么必须把它和一个不包含子网的网络区别开。前面已经提到,一个IP地址同时定义了网络和主机。如果一个网络需要分割成子网,那么必须能通过某种形式在IP地址中把它表示出来。这种表示方法称为子网掩码。

子网IP地址通过为IP地址中的每一位(bit)赋予一个掩码位来表示。如果掩码为打开(on),那么就将IP地址中相应的位视作网络地址的一部分;如果关闭(off),则将相应的位视作主机地址的一部分。

如果应将IP地址的某一段设置为打开,那么子网掩码的值就是255。如果设置成关闭,则子网掩码的值是0。例如,如果子网掩码设置成255.255.255.0,那么对于相同子网上的计算机,IP地址的前3段应该是一样的。

因此如果我们的IP地址,192.168.1.0、192.168.42.0和192.168.72.0的子网掩码都是255.255.255.0,那么如果这些计算机中的任意两台希望互相通信,它们必须首先把数据发送到网关再经由子网进行通信。

对于IP地址192.168.0.42和16.42.226.10,如果把子网掩码设置成255.0.0.0、255.255.0.0或者255.255.255.0,情况则和上面一样。

然而,如果IP地址192.168.1.0、192.168.42.0和192.168.72.0的子网掩码是255.255.0.0,那么就没有必要通过网关传送数据,因为这3个地址表示的主机在同一个子网上。

因此,如果IP地址类似,它必定在相同的子网上。否则,网络数据包必须经过网关才能传送。

2. IP 路由选择

IP路由选择协议是自动的,它要求路由设备上的软件每隔一个固定的时间之后自动计算路由表项。这和静态路由不同,静态路由是由网络管理员建立的,只有管理员修改它的时候它才会改变。

可以在同一个Unix box上设置两种方法。静态路由通过系统启动时初始化脚本中的路由命令设置。最通用的设置是使用一个默认的网关,子网中所有没有确定目的主机的数据包都将被导向该网关。计算出怎样把数据包发送到它们的目的地是默认网关的职责。

Unix下的动态路由意味着routed守护进程(routed daemon,一种利用RIP的寻径服务)正在系统上运行。routed守护进程使用路由选择信息协议(Routing Information Protocol)完成动态路由的发现。

3. 路由选择信息协议

很多网络都使用TCP/IP在内部进行路由选择。在一个网络内使用TCP/IP进行路由选择是由所谓的路由选择信息协议,或者RIP完成的。RIP在本地网上使用广播机制来获取怎样才能最好地把流量发送到目的地的信息。

在一个基于TCP/IP的网络中,每台主机都配置有一个静态默认的路由或者网关系统以便

管理网络。每次主机发送一个 TCP/IP 包到子网外的目的地时，首先将数据包发送到网关，并在网关中找到一个动态路由供该数据包使用。也就是说，RIP 使用子网的 IP 广播地址在所连接的子网上产生查询。

与某个子网连接的路由器或者网关主机根据它自己的路由表来回复这些 RIP 广播，在回复中包含了怎样到达其他子网的信息。

使用这种方法，网络上所有的路由器很快就可以知道到达整个网络上任意的远程目的地可能的路由选择路径。RIP 为数据包选择的路径是基于从源地址到目的地址的最小网络跳(hop)数，或连接数。

IP 路由选择表由目的地址或者某个可能的跳对(hops pairings)的地址组成，以便能够进行动态路由选择。这意味着数据包的完整路由并不是在一开始的时候就已经知道，而是在当前网络路由器的路由选择表中通过匹配数据包内的目的地址计算得到的。

由于整个网络是不可知的，每个节点并不关心数据包是否已到达它的最终目的地，在路由选择发生异常时，IP 也没有提供向源端返回错误报告的机制。这个任务由另一个 Internet 协议 ICMP 完成，本章的其他地方对该协议进行了讨论。

4. 域名和主机名

在决定到某个网络资源的一条路径之前，必须知道主机的网络地点。在一个小的网络中这也许只是简单地意味着用户把 IP 地址输入到应用程序中，或者应用程序可能已经知道正在讨论的 IP 地址。这种方法在一个分为多个子网的网络上或者是在整个 Internet 上是不切实际的。

为了解决这个问题，提供了域名系统(Domain Name System, DNS)网络服务把包括文字与数字的域名转换成 IP 地址。域名是帮助记忆的，是实际的 IP 地址的占位符，由于域名包括文字与数字，因此可以用易于记忆的单词或数字组成。但是，网络仍然是基于 IP 地址的，所以每次用到一个域名的时候，位于网络上的一个 DNS 服务必须把域名转换成对应的 IP 地址以便开始路由选择处理。例如，域名 `www.wrox.com` 可能会转换成 `208.215.179.178`，而域名 `www.weinstein.org` 则可能转换成 IP 地址 `69.36.240.162`。更好的是，因为 DNS 条目可以更新而且关联可以修改，所以一台主机可以变化网络或者子网而不需要每个用户或应用程序知道新的 IP 地址。

DNS 系统实际上就是它本身的网络。如果一个 DNS 服务器不能把某个特定的域名转换成一个特定的 IP 地址，那么这个 DNS 服务器将询问其他的 DNS 服务器，依次类推，直到返回正确的 IP 地址。

5. DNS

根据系统和配置，当一个域名或者主机名进入到系统时，系统将检查本地文件以查看对这个主机名是否已经存在一个主机名到 IP 地址的映射。如果找到，这个本地映射将被看作是权威映射，路由选择过程由此开始。这个映射将保存在 `/etc/hosts` 文件中。如果没有名字服务器(nameserver)，系统上的任何网络程序都将参考这个文件以判断和某个主机名对应的 IP 地址。

否则，将产生一个 DNS 服务请求，而且大多数情况下网络请求都会被发送到一个名字服务器上以获取结果。名字服务器只是网络上的一台把名字从一种形式转换成另一种形式的服务器，从域名转换到 IP 地址，或者在某些情况下，从 IP 地址转换到域名。注意这意味着如果 IP 地址不是位于一个可疑的系统上，那么为了让一个系统正确地运行 DNS 服务(即名字服务器)

中的 IP 地址，这个系统对于 DNS 进程必须是可知的，而且所有其他与它相关的网络请求必须是成功的。因此，大多数情况下，为某个请求寻找答案的系统将拥有至少一个(如果没有更多的话)名字服务器的清单，该清单以 IP 排列。

在/etc/resolv.conf 文件中可以找到 DNS 服务器的 IP 地址清单：

```
nameserver 192.168.0.2
nameserver 216.231.41.2
```

如果查询的服务器有该主机名的权威数据，它将返回 IP 地址。如果查询的服务器没有包含确定的结果，而且已将它配置成把查询转发给其他名字服务器，它将参考根域名服务器(root nameserver)列表。

从右边开始，将逐层分解请求直到找到包含该域名数据的具体机器。例如，为了处理 www.wrox.com，包含最顶层域名(top-level domain, TLD).com 的根域名服务器将回应这个 DNS 请求。如果是关于 www.weinstein.org 的请求，将查询另一组根域名服务器，它们知道以.org 为 TLD 的域名。接下来计算域名或者主机名的下一个右边的主元素，然后分别查询包含 wrox 或 weinstein 权威数据的名字服务器，对特定主机或子域名的请求将转发到那个 DNS 系统。这个过程将一直持续下去直到找到包含正在查询的目的地的精确信息的系统。一旦这个过程完成，该系统的 IP 地址将回答正在查询的域名已经知道，于是这两个系统之间 TCP/IP 路由选择即可开始。

16.2 为 Unix 系统设置 TCP/IP 网络

您也许已经料到，可以找到很多软件工具，用于把一个基于 Unix 的系统添加到 TCP/IP 网络中以及管理 TCP/IP 网络、子网或主机。

在对 TCP 协议进行讨论时提到，TCP 数据包中包含了端口的信息。TCP 可以在源主机的一个端口和目的主机的一个端口之间建立网络连接，数据包在实际的系统中正是通过这些端口传送的。而且，TCP 把一定范围内的端口设置为特殊的用途，例如，在所有的 Unix 系统上所有小于 1024 的端口通常都只能通过以根权限执行的应用程序访问，因为系统将这些端口用于建立 HTTP 和 FTP 等协议。例如，端口 80，所有的守护进程都指定通过这个端口提供 Web 服务、回应 HTTP 请求，它将绑定自己以便监听网络上远程客户机的请求。由于远程客户机知道 80 端口是 HTTP 端口，它将把所有的 HTTP 请求发送到 80 端口，除非被导向其他端口。TCP 支持多重端口，意味着一台主机可以回应多个协议的请求。而且，这意味着还可以产生多重输出请求，因为两个系统、客户机和服务器，都将多次响应并回应这些请求从而完成一次网络会话。

16.2.1 TCP/IP 网络请求配置

有两种方法为 TCP/IP 网络配置 Unix 系统，可以使用一个永远不会改变的静态 IP 地址，或者是一个动态的 IP 地址，该地址来源于一组 IP 地址，系统暂时使用这个地址来访问网络。两种方法各有其优点和缺点。

在各种不同的 Unix 系统上，您将发现两个非常有用的命令行工具，它们用于检查网络接口的配置以及为网络接口卡设置静态 IP 地址，它们是 ifconfig 和 route。

实战

使用 ifconfig 和 route

(1) 从网络管理员或服务提供商那里得到一个静态 IP 地址并把它指定给一个 Unix 系统(例如 IP 地址是 192.168.0.42)。

(2) 使用如下命令:

```
% sudo ifconfig eth0 192.168.0.42 netmask 255.255.255.0
```

(3) 为了测试设置是否正确完成, 在命令行再次输入 ifconfig, 这一次不要使用任何参数:

```
% ifconfig
```

输出看起来可能和下面的内容类似:

```
eth0      Link encap:Ethernet  HWaddr 00:E0:18:90:1B:56
inet addr:192.168.0.42  Bcast:192.168.0.255  Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:12 9 5 errors:0 dropped:0 overruns:0 frame:0
TX packets:1163 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
Interrupt:11 Base address:0xa800

lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:3924  Metric:1
RX packets:139 errors:0 dropped:0 overruns:0 frame:0
TX packets:139 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
```

(4) 为了把默认网关设置为 208.164.186.12, 使用下面的命令:

```
$ sudo route add default gw 208.164.186.12
```

工作原理

实用工具 ifconfig 用于设置和配置网卡。无论网络信息以是怎样保存的, 在启动时以及在系统运行的任何时刻, 用来配置网卡的工具都是 ifconfig。如果需要保存配置信息, 就一定要保存网络信息, 因为系统重启之后这些设置就不存在了。

通常, 在命令行直接使用 ifconfig 都是出于测试的目的, 即为了查看当前的设置或者为了手工修改 TCP/IP 网络设置。为了使设置长期有效, 用户必须在与网络互联功能相关的文件中进行设置, 例如, 在一台 FreeBSD 机器上, ifconfig 所使用的信息保存在 rc.conf 文件中:

```
defaultrouter="192.168.0.1"
hostname="chaffee.weinstein.org"
ifconfig_xl0="inet 192.168.0.15 netmask 255.255.255.0"
```

对于 Linux 系统, 网络配置信息保存在 /etc/sysconfig/network 文件中。

无论如何, 在前面的示例中, 默认路由器都由 route 命令工具设置为 192.168.0.1。

16.2.2 动态设置

设置静态 IP 地址和路由选择信息的一个问题是(例如前面的示例中),在携带便携系统,例如膝上电脑、或者笔记本、计算机等旅行时需要修改网络和配置。让管理员负责记录某个网络中已经分配了哪些地址,而哪些地址可以用于一组桌面系统是非常单调乏味的。多年来人们已经提出的一种解决方案是让网络管理员指定一些 IP 地址,这些地址可以通过称为动态主机配置协议(Dynamic Host Configuration Protocol, DHCP)的服务临时分配给移动系统。DHCP 使得用户能以动态的方式接收所需的 IP 地址、网络掩码、路由选择信息和 DNS 服务器的地址。

实战

使用 DHCP

- (1) 从系统管理员或服务提供商那里确认在网络上是否有 DHCP 服务正在运行。
- (2) 配置系统,以使用 DHCP 客户端为网络接口设置动态信息。例如,在 FreeBSD 系统上,在文件/etc/rc.conf 中输入如下文本行:

```
ifconfig_ed0="DHCP"
```

对于 Linux 系统,编辑文件/etc/sysconfig/network-scripts/ifcfg-eth0,使其包含如下内容:

```
DEVICE=eth0
BOOTPROTO=dhcp
CNBOOT=yes
```

- (3) 重启系统。
- (4) 为了测试设置是否正确完成,在命令行再次输入 ifconfig,这一次不要使用任何参数,如下所示:

```
% ifconfig
```

输出看起来应该和下面的内容相似,只是 IP 地址不再是包含在系统某个文件中的静态地址,而是由 DHCP 服务器分配的 IP 地址:

```
eth0      Link encap:Ethernet  HWaddr 00:E0:18:90:1B:56
inet addr:192.168.0.72  Bcast:192.168.0.255  Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:1295 errors:0 dropped:0 overruns:0 frame:0
TX packets:1163 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
Interrupt:11 Base address:0xa800

lo        Link encap:Local Loopback
Inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:3924  Metric:1
RX packets:139 errors:0 dropped:0 overruns:0 frame:0
TX packets:139 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
```

工作原理

Unix 在启动时希望给网卡指定一个 IP 地址。如果这个 IP 地址是静态的,就说明这些信息

保存在本机上并由 `ifconfig` 指定给网卡，如前例所述。如果要动态地设置信息，系统将广播一条消息请求一个 DHCP 服务器。请求中包含了客户端的硬件地址，任何接收到这个广播的 DHCP 服务器都将向客户端发送它自己的广播消息，提供一个 IP 地址让客户端使用一段时间，这段时间称为租借期(lease period)。

客户端从多个选择中挑选一个。通常，它希望获得最长的租借期。一旦选择了一个地址，客户端将通知 DHCP 服务器它已选择了一个提供给它的、可租借的 IP 地址并标识出所选中的服务器。

被选中的 DHCP 服务器于是发出一个确认，该确认中包含 IP 地址、子网掩码、默认网关、DNS 服务器和租借期。然后客户端使用 `ifconfig` 把正确的信息赋给网卡。

16.2.3 发送 TCP/IP 网络请求

对于一个 Unix 系统会有若干个客户端，具体数目取决于所请求的服务。另外，在不同的 Unix 系统上有很多命令行工具可以用于验证一个网络的配置是否正确和工作是否正常。

为了测试一个网络连接或者验证到一台或多台主机的连接是否完善或者速度的快慢，要使用 `ping` 命令，该命令从潜水员的角度来看就像声纳脉冲发生器(sonar pulse)，会发送一个 ICMP 响应请求(ICMP Echo Request)消息，该消息将请求远程主机发送一个响应应答，在这个应答中包含了接收到的消息。如果一切正常，远程主机将发送一个应答，请求和应答之间所花费的时间即可计算出来。

实战

用 ping 进行测试

在命令行输入如下命令：

```
% ping www.wrox.com
```

结果看起来应该和下面的内容差不多：

```
% ping www.wrox.com
PING www.wrox.com (204.178.64.166) from 192.168.0.15 : 56 data bytes
64 bytes from 208.164.186.2: icmp_seq=0 ttl=128 time=1.0 ms
64 bytes from 208.164.186.2: icmp_seq=1 ttl=128 time=1.0 ms
64 bytes from 208.164.186.2: icmp_seq=2 ttl=128 time=1.0 ms
64 bytes from 208.164.186.2: icmp_seq=3 ttl=128 time=1.0 ms

--- 208.164.186.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 1.0/1.0/1.0 ms
```

工作原理

正如所述，`ping` 命令向远程主机发送一个 ICMP 应答请求消息，该远程主机发送一个应答，在应答中包含了收到的消息。当返回应答消息时，就可以计算出 `ping` 请求的往返时间，并判断出远程主机的网络状态。

当尝试连接远程主机因超时而失败时，如果是“host unreachable(主机不可达)”和“network unreachable(网络不可达)”错误，或者包丢失率很高，那么该连接可能在路径中的某个地方没有走通。一种原因可能是网络防火墙(一种设计出来用于阻止或者过滤私有网和外部网机器之

间的网络流量的网络设备)不允许 ICMP 流量通过。

另一种原因可能是网络连接不稳定。尝试等待一小段时间以查看网络连接是否会自行重建,然后确认 DNS 系统正确地响应而且计算机的网卡配置正确,并正在正常运行。

实战

nslookup、dig 和 host

为了检查 DNS 服务器的响应能力或者手动检查域名/主机名的 IP 地址,可以使用下面的 nslookup、dig 或者 host 命令:

```
$ nslookup www.weinstein.org
```

根据所使用的命令,结果看起来将和下面的内容相似:

```
% nslookup www.weinstein.org
```

```
*** Can't find server name for address 192.168.0.4: Timed out
Server: dns.chil.speakeasy.net
Address: 64.81.159.2
```

```
Mon-authoritative answer:
Name: www.weinstein.org
Address: 69.36.240.162
```

注意在这个示例中第一个 DNS 服务器(地址是 192.168.0.4)没有响应,系统不得不检查排列在 resolv.conf 文件中下一个 DNS 服务器,以便解析 nslookup 命令提供的域名。

工作原理

nslookup、dig 和 host 等工具查询系统中配置好的名字服务器,以判断特定域名的 IP 地址。

在前面的示例中,Unix 系统向两个已经配置好的用于检索信息的名字服务器发送请求。第一个服务器(地址是 192.168.0.4)没有响应请求,所以系统查询下一个名字服务器,系统配置地址为 64.81.159.2。服务器不仅提供关于自己的信息,而且提供关于被查询的域名可能的 IP 地址的信息。

实战

使用 Netstat

尝试连接远程主机失败的另一个可能的原因也许是网卡的配置,这可以通过使用 ifconfig 查询,如前所述,或者用另一个工具 netstat。

为了迅速查看网卡的状态,使用 netstat -i 命令,如下所示:

```
% netstat -i
```

netstat 的结果如下所示:

Kernel Interface table

Iface	MTU	Met	RX-OK	X-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth0	1500	0	4236	0	0	0	3700	0	0	0	BRU
lo	3924	0	13300	0	0	0	13300	0	0	0	LRU
ppp0	1500	0	14	1	0	0	16	0	0	0	PRU

工作原理

netstat 命令象征性地显示各种与网络相关的数据结构的内容。有多种输出格式，取决于为显示信息所使用的选项；例如，在上一节中，netstat 提供了关于网卡的信息，这些信息已自动配置好。

```
eth0 1500 0 4236 0 0 0 3700 0 0 0 BRU
```

也就是说，表格中的每一行都是与某一特定网卡有关的已传送的数据包、错误和冲突的累积统计信息。网卡的地址和最大传送单元(maximum transmission unit)也一起显示出来。

netstat 的另一个选项检验关于所有当前网络连接的信息：

```
% netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address          Foreign Address         State
tcp      0      0 www.weinstein.org:ssh  dsl.chi.speakeasy.net:60371 ESTABLISHED
tcp      0    256 www.weinstein.org:ssh  dslchi.speakeasy.net:1830 ESTABLISHED

Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type        State         I-Node Path
unix  10      [ ]        DGRAM              754    /dev/log
unix   2      [ ]        DGRAM             4939915
unix   2      [ ]        DGRAM             372659
unix   2      [ ]        DGRAM             271499
unix   2      [ ]        DGRAM             1784
unix   2      [ ]        DGRAM             1606
unix   2      [ ]        DGRAM             1535
unix   2      [ ]        DGRAM             884
unix   2      [ ]        DGRAM             766
```

在这个示例中，默认显示的关于活动网络套接字的信息显示了协议、接收和发送队列的大小(以字节为单位)、本地和远程地址，以及协议的内部状态。

注意在任何特定的系统上，在任何时刻都可能有几个或很多网络连接。同样，netstat 命令的输出可能会比这个示例给出的结果长或短。

正如在本章后面的脚本编程小节中将要讨论的那样，可以自动收集和处理这些工具返回的数据。一组脚本能够从一个或多个测试中收集数据，并把这些数据和以前已经运行并处理过的测试数据作比较。于是可以把这些信息汇集到一起以获得在任意时刻网络和网络连接的健康情况和状态，并可以利用这些信息来帮助检查故障。

16.2.4 回应 TCP/IP 网络请求

/etc/services 文件提供从端口号到服务名的映射。一个典型的 Unix 服务文件看起来将和下面的内容类似：

```
# /etc/services:
# $Id: services,v 1.22 2001/07/19 20:13:27 notting Exp $
#
# Network services, Internet style
```

```

#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1700, "Assigned Numbers" (October 1994). Not all ports
# are included, only the more common ones.
#
# The latest IANA port assignments can be gotten from
#   http://www.iana.org/assignments/port-numbers
# The Well Known Ports are those from 0 through 1023.
# The Registered Ports are those from 1024 through 49151
# The Dynamic and/or Private Ports are those from 49152 through 65535
#
# Each line describes one service, and is of the form:
#
# service-name port/protocol [aliases ...] [# comment].

topmux          1/tcp                # TCP port service multiplexer
topmux          1/udp                # TCP port service multiplexer
echo            7/tcp
echo            7/udp
ftp             21/tcp
ftp             21/udp
ssh             22/tcp                # SSH Remote Login Protocol
ssh             22/udp                # SSH Remote Login Protocol
telnet          23/tcp
telnet          23/udp
smtp            25/tcp                mail
smtp            25/udp                mail
nameserver      42/top                name      # IEN 116
nameserver      42/udp                name      # IEN 116
http            80/tcp                www www-http # WorldWideWeb HTTP
http            80/udp                www www-http # HyperText Transfer Protocol
kerberos        88/tcp                kerberos5 krb5 # Kerberos v5
kerberos        88/udp                kerberos5 krb5 # Kerberos v5
pop3            110/tcp                pop-3       # POP version 3
pop3            110/udp                pop-3
nntp            119/tcp                readnews untp # USENET News Transfer Protocol
nntp            119/udp                readnews untp # USENET News Transfer Protocol
ntp             123/tcp
ntp             123/udp                # Network Time Protocol

```

正如所见，`/etc/services` 文件中的 80 端口指向名为 `http` 的服务，这表示一个协议：位于端口 80 的守护进程需要处理。

很多时候，在 Unix 系统上，一个称为 `inetd` 的守护进程负责管理不同的应用程序，这些应用程序将处理对网络服务的内部请求。`inetd` 的配置文件 `/etc/inetd.conf` 精确地指出了对于在 `/etc/services` 文件中给出的某个特定的服务需要采取什么操作。

16.2.5 inetd

在系统启动时由 `init` 启动的众多进程之一是 `inetd`，该进程是一个长时间运行的守护进程，

它密切地监视着进来的网络连接。当它发现进入内部的网络流量时，就检查进入的端口，参考 `/etc/services` 文件给出服务的名称，然后查阅其配置文件 `/etc/inetd.conf`，以决定怎样开始处理一个进来的连接：

```
# $FreeBSD: src/etc/inetd.conf,v 1.69 2004/06/06 11:46:27 schweikh Exp $
#
# Internet server configuration database
#
# Define *both* IPv4 and IPv6 entries for dual-stack support.
# To disable a service, comment it out by prefixing the line with '#'.
# To enable a service, remove the '#' at the beginning of the line.
#
ftp      stream  tcp      nowait  root    /usr/libexec/ftpd      ftpd -l
#ftp     stream  tcp6     nowait  root    /usr/libexec/ftpd      ftpd -l
#ssh     stream  tcp      nowait  root    /usr/sbin/sshd         sshd -i -4
#ssh     stream  tcp6     nowait  root    /usr/sbin/sshd         sshd -i -6
telnet   stream  tcp      nowait  root    /usr/libexec/telnetd   telnetd
#telnet  stream  tcp6     nowait  root    /usr/libexec/telnetd   telnetd
#shell   stream  tcp      nowait  root    /usr/libexec/rshd      rshd
#shell   stream  tcp6     nowait  root    /usr/libexec/rshd      rshd
#login   stream  tcp      nowait  root    /usr/libexec/rlogind   riogind
#login   stream  tcp6     nowait  root    /usr/libexec/rlogind   riogind
finger   stream  tcp      nowait  3/10    nobody /usr/libexec/fingerd   fingerd -s
#finger  stream  tcp6     nowait  3/10    nobody /usr/libexec/fingerd   fingerd -s
#exec    stream  tcp      nowait  root    /usr/libexec/rexecd    rexecd
```

在这个新示例中，大家可以看到有 3 个进程是激活的，而其余的进程都被禁用了，在这些被禁用的进程前有一个前缀注释符号(#)。3 个激活的端口一个是在 `/usr/libexec/ftpd` 的应用程序，它将由根触发以应答任何在 `ftp` 端口上的 TCP 请求，前面的示例中，该端口在文件 `/etc/services` 中定义成端口 21。对于 `telnet` 和 `finger` 可以给出类似的描述。

由 `inetd` 管理这些端口和应用程序的好处是在同一时刻管理一个单独的网络连接只需要一个专用的程序。好处很明显：这些端口监控程序本身的代码更简单，而且通用的 Unix 工具箱原则也得到了很好的应用，这个原则是使用输入-输出管道把小程序捆绑在一起使用。

然而，注意和前面的示例不同的是，这个示例中没有条目让某个应用程序来处理端口 80 上的 HTTP 请求。这是因为，和 FTP 或 TELNET 有所不同，Apache Web 服务器——Unix 上用于处理 HTTP 请求的最常用的应用程序——在 `inetd` 的管理范围外以独立的模式运行，在这种模式中它在自己的端口上处理 HTTP 通信流。

HTTP 通信流在 `inetd` 之外进行处理，因为使用类似 `inetd` 这样的通用 Internet 端口监控程序需要有一定的权衡。在 Unix 上启动一个可执行程序通常是一个很耗费资源的操作，要使用可能同时正由其他应用程序使用的 CUP、内存和磁盘访问。很多时候，实际运行一个已触发的程序会更耗费资源。因此，考虑网络服务将要处理的通信流类型是很重要的。如果连接相对不是很频繁但是需要进行很多处理，那么基于 `inetd` 的实现是合适的，因为它把网络代码归入 `inetd` 并让应用程序考虑处理请求的正确过程和资源。然而，如果连接非常频繁，例如对于一个很流行的站点的访问频率，那么独立的实现会更好。

16.3 网络管理工具

也许我们的想法很好，但是计算机并不是完全独立的系统，它们没有足够的知识来照顾自己和处理有时可能溜进系统的问题。然而，计算机系统是复杂的系统，包含互相依赖的组件，它们以预料之中或者预料之外的方式交互。无论使用的是什么计算机系统，一台个人桌面系统或者处理复杂计算的一组服务器，确保计算机系统启动并正常运行工作都是由系统管理员负责。

另外，现在的很多程序和服务都需要某种形式的网络访问方式。很多情况下网络管理员负责保证网络是正常工作的，并保证所需的系统有正确的访问权限。有时候，这些责任可能由一个人承担，这个人可能有也可能没有接受过处理手头上的问题的正规培训。

无论管理员的背景或技术怎么样，基本的目标是保持系统尽可能平稳地运行，使依靠系统的用户尽可能少地感觉到不方便。幸运的是，有很多 Unix 管理工具使得网络管理对每个人都变得简单很多。

16.3.1 通过 Traceroute 跟踪网络的性能

有很多系统和网络管理员都需要使用的工具。本节将讨论一个名为 `traceroute` 的工具并显示管理员怎样使用该工具和其他工具，例如 `AWK`、脚本编程和 `cron` 等，一起维持基于 Unix 系统上的网络连接的日常执行。

实战

使用 Traceroute

在命令行输入如下命令：

```
% traceroute www.yahoo.com
```

结果应该和下面的内容类似：

```
traceroute to www.yahoo.akadns.net (66.94.230.38), 30 hops max, 40 byte packets
1  192.168.0.1 (192.168.0.1)  0.8 ms  0.517 ms  0.383 ms
2  erl.localisp.net (66.42.138.1)  13.613 ms  12.664 ms  12.954 ms
3  220.ge-0-1-0.cr2.localisp.net (69.17.83.153)  12.741 ms  10.753 ms  12.042 ms
4  exchange-cust1.chi.equinix.net (206.223.119.16)  35.405 ms  28.537 ms  27.729 ms
5  ae0-p803.pat1.pao.yahoo.com (216.115.98.13)  69.158 ms  68.504 ms  68.886 ms
6  ge-0-0-2.msrl.scd.yahoo.com (66.218.64.134)  68.871 ms ge-1-0-
2.msrl.scd.yahoo.com (66.218.82.193)  76.226 ms ge-0-0-2.msrl.scd.yahoo.com
(66.218.64.134)  69.488 ms
7  vl42.basl-m.scd.yahoo.com (66.218.82.226)  70.22 ms unknown-
66-218-82-230.yahoo.com (66.218.82.230)  69.481 ms vl42.basl-m.scd.yahoo.com
(66.218.82.226)  70.71 ms
8  p7.www.scd.yahoo.com (66.94.230.38)  69.576 ms  70.754 ms  70.517 ms
%
```

工作原理

Internet 和其他计算机网络把包含信息的数据包从源计算机路由到目的计算机。考虑从一个大型航运公司的跟踪系统中查询包裹的信息。根据航运的方法(夜间，白天等)，包裹将从一

个航运点，或者中转地，及时地运送到下一个中转地以便按时达到目的地。理论上，人们可以跟踪包裹从一个中转地移动到另一个中转地所花费的时间，以及总的航运时间。

`traceroute` 就是这样工作的，跟踪它获取信息所经过的大型且复杂的网关或集线器集合所花费的时间和经过的路由。惟一所需的参数是目的主机名或者 IP 地址。

当然，不能随意使用 `tracetrout` 和其他像 `ping` 和 `nmap` 这样的网络互联工具。管理员和用户应该限制在他们负责的网络上使用这些工具。针对由其他人维护的网络使用这些工具，特别是频繁地使用，会被认为是一种敌对的手段，因为这些工具即可以用恶意的方式来使用，也确实有它们的用途。

无论何时，`traceroute` 都试图跟踪一个网络数据包到达某个主机所通过的路由，然后监听这条路由上的各个集线器以期得到一个应答。

`traceroute` 返回的结果显示了在 3 次不同的尝试中，数据包选取了那条路由，以及到达每个点所花费的时间，该时间以毫秒为单位。输出的第一行是关于跟踪的信息：目标系统、系统的 IP 地址、允许的最大跳数和发送的数据包的大小。

```
traceroute to www.yahoo.akadns.net (66.94.230.38), 30 hops max, 40 byte packets
```

接下来的文本行指示在两个点之间的每个集线器、系统或路由器。每行都显示了系统的名称，该名称由 DNS 记录判断，系统的 IP 地址和以毫秒为单位的 3 次往返时间。

```
exchange-cust1.chi.equinox.net (206.223.119.16) 35.405 ms 28.537 ms 27.729 ms
```

往返时间表示一个数据包从起点开始到达特定的位置并返回所需的时间长短。这段时间称为两个系统之间的反应时间(latency)。默认情况下，在路由上对每个系统发送了 3 个数据包，所以我们得到 3 个时间。

有时候，输出中的某一行会有一个或多个时间丢失，并用星号来表示它本应该出现在何处：

```
exchange-cust1.chi.equinox.net (206.223.119.16) 35.405 ms * 27.729 ms
```

这种情况表明，目的主机已经启动并且有响应但是对其中的一个测试数据包没有给出应答。这并不表示一定存在着问题。实际上，这是很普通的，仅仅意味着系统出于某种原因丢弃了数据包。

然而，一次跟踪中的 3 个数据包可能都以超时结束，例如：

```
4 exchange-cust1.chi.equinox.net (206.223.119.16) 35.405 ms 28.537 ms 27.729 ms
5 ae0-p803.pat1.pao.yahoo.com (216.115.98.13) 69.158 ms 68.504 ms 68.886 ms
6 * * *
7 * * *
8 * * *
```

如果是这种情况，多考虑一下就是正确的，因为这意味着目标系统可能不可达。更准确地说，这意味着无法将数据包送达目的地并返回。它们也许实际上已经到达目标系统但在返回的途中遇到了问题。但是，取决于目标系统的设置情况，这也可以是完全正常的。很多管理员刻意阻止 `traceroute` 这样的工具更进一步的访问，这是一种安全措施。

16.3.2 防火墙

本章前面已经简单地提到，防火墙(firewall)是一种设计出来用于阻止或者过滤私有网和外

部网机器之间的网络流量的网络设备，它可能不允许 ICMP 通信流通过。简单地说，防火墙将截获所有通向或者来自防火墙后面的机器的数据包，它根据一组预定义的规则，决定是否允许通信流通过。

网络的规则通常由网络管理员决定并配置，可能会很宽泛，很具体，或者两者兼有。例如，一个网络管理员可以决定阻止所有外部机器对内部私有网中任何 FTP 服务器的 FTP 请求。FTP 可能是一个安全隐患，因为它对任何要读取的网络设备都以明文的方式传送用户名和密码。或者网络管理员可能会担心有人利用某个已知的 FTP 服务器软件的弱点危及某个内部系统。如果是这样，管理员不得不决定让所有外部系统无法确定内部 FTP 服务器的位置并能创建一条规则阻止所有进来的 FTP 通信流。

但是管理员可能希望允许有限的来自远程网络的机器能够访问私有网内的 FTP 服务器。如果管理员知道远程办公室的 IP 子网，他(她)可以设置防火墙只允许来自那个子网的连接可以通过。管理员还可以进一步做出限制，只允许一组特定的机器根据它们的 IP 地址进行访问。

防火墙能做的还不止这些。位于防火墙后面的系统可以在任何时刻发起流向外部的通信流。同样，这个通信流在为外部网络看到之前必须通过防火墙。因此还可以配置防火墙以允许或拒绝以一组 IP 地址为目标的通信流。

以这种方式，网络管理员可以控制一个私有网络提供哪些信息，控制哪些关键业务或者共享哪些个人信息，限制内部系统或用户在访问一台已通过业务、安全性或个人策略确定为禁用的远程主机时可能遇到的危险。

16.3.3 例行检查网络延迟

数据在网络上完成一次往返所需要的时间对很多网络管理员来说都是一个重要的数字。网络的延迟会影响重要的网络服务，例如 DNS，或者更高层的应用系统，例如一个公共的 Web 站点。了解网络的正常性能使得管理员可以更好地解决任何遇到的问题。而且，收集的数据能提供网络通信流模式的信息并能帮助提供改善性能的基准。

实战

用 AWK 处理 traceroute 的数

(1) 在 vi 中，用如下内容创建一个名为 `mean.awk` 的文件并保存该文件：

```
/^ *[0-9]/ {
  trotime = 0; attempts = 0;
  for (field = 5; field <= NF; ++field) {
    if ($field == "ms") {
      trotime = $(field - 1) + trotime
      attempts = attempts + 1
    }
  }
  if (attempts > 0) {
    average = trotime/attempts
    printf "Point: " $2 ", Average Latency: " average "\n"
  }
}
```

(2) 接下来，创建一个名为 `mean.sh` 的 shell 脚本并输入如下内容：

```
#!/bin/bash
# run traceroute and parse the data with mean.awk
traceroute www.yahoo.com > trace_results.data
awk -f mean.awk trace_results.data
```

(3) 最后，使用命令 `crontab -e`，把下面的条目放到一个本地 `crontab` 文件中，以便对 shell 脚本进行自动化：

```
# run our shell script for testing the network
**/12**userid cd ~/bin; ./mean.sh | mail -s "Network Data" userid
```

工作原理

首先，AWK 脚本将计算 `trace_results.data` 文件中包含任意数字的每一行。这在文件的第一行由正则表达式定义。

```
/^ *[0-9]/
```

接下来，代码将两个变量 `trottime` 和 `attempts` 设置为零，在此之后定义了一个使用 `for` 语句的控制循环。

```
for (field = 5; field <=NF; ++field) {
```

循环条件首先定义一个名为 `field` 的变量并设置为 5。为什么要设置成 5？请看一看 `traceroute` 的输出：

```
ae0-p803.pat1.pao.yahoo.com (216.115.98.13) 69.158 ms 68.504 ms 68.886 ms
```

第 5 个字段是脚本感兴趣的数据，从这儿开始显示时间结果。(哦，其实并不是这样，第 5 个字段实际上是过去的第一个时间结果字段，但是这也没什么！)记住，默认的分隔符是空格。

下一个条件是测试条件。`field` 的值在 `loop` 的当前循环中小于或者等于正在计算的当前行的所有字段数吗？最后一个条件是，在每次循环结束时，`field` 变量的值增加 1。

在循环内部，第一行是另一个条件语句：如果变量 `field` 的值等于“ms”，继续执行下面的一组命令(记住，在循环的第一次重复中这个条件将为真因为这行的第五个字段包含了值“ms”，它是第一个时间值的结尾)。注意要测试一个变量看它是否等于测试条件，将一起使用两个等号(==)。如果只使用一个等号，值将赋给变量并且这个 `if` 语句将永远为真。

如果测试条件为真，就回退一个字段，`$(field-1)`，然后把这个值加到当前的 `trottime` 值上并增加 `attempts` 变量的值。当收集了一行中的所有数据，循环的限制条件也到达了，脚本执行到下一个部分——计算平均值并打印数据。

同样，这里也使用了一个 `if` 条件，为什么要打印没有结果可打印的数据呢？如果有数据，那就计算平均值，并打印该行数据的和。

```
if (attempts > 0) {
  average = trottime/attempts
  printf "Point: " $2 ", Average Latency: " average "\n"
}
```

其余的内容，shell 脚本工作的方式、以及 `crontab` 条目在前面的章节中已讨论过。

使这个任务或其他任务自动化存在一个问题。运行一个像 `traceroute` 这样的工具会发送一

些数据包，这些数据包将影响到网络上其他数据包的时间。由于网络上的这种影响，在正常的操作中或在自动化的脚本中如果没有必要，那么使用 `traceroute` 是不明智的。即使必须使用，也应该保守一点，前面的示例每天触发脚本两次。

16.4 小结

所有这些操作的关键是数据具有某种结构。考虑一个由多个抽屉组成的档案柜，每个抽屉保存特定的一组内容：在第一个抽屉中是与项目相关的文档，另一个里面是雇用记录。有时候抽屉中有隔板，使得不同的文件都可以保存在一起。所有这些结构都是根据事务的发展决定的，当从存放文件的地方挑选文件，并判断在何处可以找到它们时就会用到这些结构。

- TCP 是 TCP/IP 协议的传输层部分。
- IP 是实际在两台主机之间传送数据的网络协议。
- TCP/IP 网络具有拓扑，这种拓扑允许把它们细分成子网并为在网络和子网内、网络和子网之间正确引导信息提供了合适的方法。
- 域名提供了一种包含文字与数字的助记符，这种助记符可以定位网络和主机。
- DNS 是一种把域名转换成 IP 地址的网络服务。
- `ifconfig` 是配置网卡的工具，可以给它提供静态或动态信息。
- `netstat`、`ping` 和 `nslookup` 等工具可以提供与网络性能有关的信息。
- 从系统工具(例如 `traceroute`——它提供与网络上两点之间的性能有关的信息)，返回的数据可以由 AWK 进行处理以提供信息并帮助管理系统或者网络。
- AWK 可以和其他工具(例如 shell 脚本)组合以完成和管理复杂的任务。

16.5 练习

创建一段 shell 脚本，该脚本每两小时执行一次，使用 `ps` 命令产生一份报告，详细列出哪些进程是激活的以及它们正在使用的 CPU 资源是多少。

第 17 章 Perl 编程实现 Unix 自动化

Perl 是一种高级的、面向对象的编程语言，它支持很多可以在 C 和 C++ 这样的编程语言中找到的概念。而且，对很多 Unix 系统管理员来说，Perl 是他们解决问题时使用得最多的工具。

本章介绍 Perl 语言，该语言有什么好处，它怎样成为 Unix 系统管理员的得力工具，以及怎样编写和检修 Perl 脚本，从而使用 Perl。

Perl 是 Practical Extraction and Report Language 的首字母简写，开发它的最初目的是专注于为文件和文本处理工具开发一种解释性的脚本语言。由于这个目的，Perl 的使用者很多，他们发现 Perl 特别适合于快速原型设计和系统管理有关的任务。另外 Perl 在很多情况下通过提供把很多不同类型的应用程序联系到一起的编程工具而成为连接两个系统的桥梁，使得它在程序员、系统管理员、数学家、新闻记者甚至是商业管理人员之间特别流行。

Perl 的开放分布策略(open distribution policy)和众多用户的支持也使得该语言的流行趋势呈上升状态，同时使得为解释 Perl 语法而设计的解释器可以在不同的平台上运行。但是 Perl 库模块和文档的具体标准与不同的系统，例如 FreeBSD、Linux 和 Solaris 等有关。

下面的练习是一小段 Perl 代码，我们将从这里开始学习。

实战 用 Perl 编写的“Hello World”

(1) 用文本编辑器创建一个名为 hello.pl 的文件，内容如下：

```
#!/usr/bin/perl -w
# Classic "Hello World" as done with Perl

print "Hello World!\n";
```

(2) 保存文件并退出编辑器之后，修改文件的权限并运行下面的代码：

```
% chmod 755 hello.pl
% ./hello.pl
hello World!
```

工作原理

和 shell 脚本一样，代码的第一行告诉 Unix 系统怎样运行这段脚本——使用 Perl 解释器。在大多数 Unix 系统中，例如在编写和测试本章的脚本的 Mac OS X 系统中，Perl 解释器都位于 /usr/bin/perl。然而，情况并不总是这样。假如在执行这段示例脚本的时候，屏幕上输出了“command not found”或者“bad interpreter”这样的错误，而不是“Hello World!”，很可能是因为 Perl 解释器位于系统上的其他地方。

为了迅速定位或者确认 Perl 解释器的位置，可以按如下方式使用 which 命令：

```
which perl
/usr/bin/perl
```



```
/usr/bin/perl:
```

无论什么时候,下面这行代码除了指定在何处可以找到 Perl 解释器以外,还可以指定怎样调用 Perl 的选项:

```
#!/usr/bin/perl -w
```

在这个“Hello World”的示例中, -w 开关告诉 Perl 解释器,在把脚本转换成机器代码的时候,它除了应该提供关键警告或致命错误的报告以外,还要提供另一个基本的警告消息。

另一种让 Perl 提供完整的警告消息的方法,也是在该语言的最新版本中推荐使用的办法,是使用如下命令,而不是 -w 开关:

```
#!/usr/bin/perl
use warnings;
```

和其他已经讨论过的脚本编程示例一样, Perl 也为输入注释以记录脚本做什么以及怎么做提供了方法。任何以井号(#)作为起始符号的文本行都表示该行是文件内的一行注释。每行注释都必须以#作为开始,而且只限于该行本身,就像代码中的第二行那样:

```
# Classic "Hello World" as done with perl
```

正如在第 11 章和第 13 章中讨论的那样,注释对于任何配置文件或脚本文件都是一个很有用的功能而且有必要充分利用它。注释至少要传达如下信息:谁创建或修改了文件中的某个条目,在添加或修改这个条目的时候,使用了什么命令,正确执行这些命令需要哪些文件,如果发生错误需要联系谁,以及为什么要和怎样添加或者修改这个条目。

另外记住,注释只有保持更新才有意义。本例的最后一行是这个脚本的主体,即 print 函数,它输出双引号之间的文本。

```
Print "Hello World!\n";
```

和其他脚本编程环境一样,这段脚本使用\n 转义字符表示在这个字符串的结尾会产生新的一行。

17.1 Perl 的优点

如前所述, Perl 是一种解释性的语言,特别适用于字符串操作、系统集成和复杂应用程序的原型制作。Perl 之所以有助于系统管理和系统集成是因为它有和很多 Unix 系统命令相同或者等同的内置功能。而且, Perl 的语法容易学习,因为它混合了 Bourne shell、csh、AWK、sed、grep 和 C 的语法元素。

很多人认为, Perl 是一个多种特性和语法的大杂块。和其他 Unix 工具一样, Perl 有一个按照自己的语法创建的正则表达式引擎。对于那些认为正则表达式很难理解和使用的人来说,这可能有一定的坏处。

Perl 的另一个优点(也有人说是缺点)是它的综合性,允许在相同的代码中使用各种不同的编程风格(过程化的、函数化的和面向对象的)。同样,是否认为 Perl 有所帮助的差别在于您认为该语言的有机发展、以及它的“用多种方法完成相同的事情”的方法是一种优点还是一种缺点。

最后一个优点(同样,也有人说是缺点)是 Perl 是一种解释性的语言。对 Perl 这意味着解释器将在执行之前逐行地、动态地把该语言的高级(high-level)语法转换成一种中间形式。相比之下,编译器在应用程序执行之前把高级指令直接转换成机器语言。

然而,解释器的好处正是在于它不需要执行产生机器指令的编译阶段。因此在实际使用 C、C++这样的编译语言来完成一项任务之前,Perl 是为该应用程序制作原型的一个很好的选择。这意味着在正式编写代码之前,可以通过原型很好地理解和调整应用程序的处理过程和逻辑。

但是,Perl 解释器的处理过程非常耗费资源,因为解释器是动态地处理高级代码的。也就是说,与 C 和 C++编译器不同,Perl 解释器在低级(low-level)代码执行之前没有时间对其进行优化,使得紧凑的脚本在执行效率上打了一个折扣。

无论怎样,和本书中讨论的其他解释性编程环境(例如 shell 脚本)相比,Perl 有其明确的优点。

17.2 一些有用的 Perl 命令

如果确实有人想掌握 Perl 的所有特性,可能需要数年的时间。因此仅仅一章的内容并不能使您成为一名 Perl 编程的好手。然而,学习很多基础知识可能是一个漫长的过程,所以下面的几个小节将详细讨论 Perl 的一些基础知识。

17.2.1 变量

在第 9 章中我们介绍了 AWK 编程语言,它是一种无类型的语言,您应该还记得。也就是说,用 AWK 编程的时候不需要声明变量将包含的内容的类型。Perl 也一样,在使用变量之前不需要声明其类型。可以任意地创建变量并在创建的时候定义。所有标量型变量(scalar variable)——也就是只包含一个元素的变量——都是以美元符号(\$)作为前缀。下面是一些有效的变量和声明:

```
$name = "Paul";
$age = 29;
$Where_to_find_him = 'http://www.weinstein.org';
```

Perl 也允许变量有多个元素,通过使用符号(@)来定义这种变量。

```
@authors = ("Paul", "Joe", "Jeremy", "Paul") ;
@list = (1, 2, 3, 4);
```

数组元素也可以通过使用\$和元素的序号来解引用。也就是说,可以用如下方式访问 @authors 数组中的名字 "Paul":

```
@authors[4];
```

Perl 还允许声明关联数组(associative array)或散列(hash)。散列类似于数组,但它不是使用数字而是使用包含字符和数字的字符串作为索引。通常将索引称为键(key),散列由百分号(%)特别指定,如下所示:

```
%person = (
  name => 'Paul',
  age => '29',
```

```
url=> 'http://www.weinstein.org',
);
```

在这里名为 `person` 的散列有 3 个包含字符和数字的字符串索引或键：`name`、`age` 和 `url`。这 3 个索引对应的值分别是 `'Paul'`、`'29'` 和 `'http://www.weinstein.org'`。

17.2.2 运算符

Perl 允许使用所有其他语言都有的同样的基本数学运算符：

```
# Addition
$result = 5+5;
$a = 5;
$b = 6;
$result = $a + $b;

# Subtraction
$result = 5-5;
$a = 5;
$b = 6;
$result = $a - $b;

# Multiplication
$result = 5*5;
$a = 5;
$b = 6;
$result = $a * $b;

# Division
$result = 5/5;
$a = 5;
$b = 6;
$result = $a / $b;
```

在所有这些示例中，每个运算的结果都被赋值给变量 `result`，然后通过 `print` 函数输出这个变量的结果或者在脚本的后续部分把它用作一个控制分支的测试条件。在加法运算的第一个示例中，`$result` 的值应该是 10，也即 5 加上 5 的结果。在第二个加法示例中，`$a` 的值和 `$b` 的值相加并把结果赋给变量 `$result`。由于 `$a` 被设置为 5 而 `$b` 被设置为 6，显然 `$result` 中的值是 11。

17.2.3 基本函数

函数是绝大多数编程语言的一个基本组成部分，通常将它用作运算符，它可以修改变量或者返回一个可以赋给变量的值。Perl 提供了很多有用的、用于处理文本的函数，虽然下面列出来的函数非常有限，但是确实包含了几乎所有在每个 Perl 脚本中都会用到的函数。

1. print

函数 `print` 向标准输出或其他流(例如管道或者文件)输出一个字符串。它具有和其他很多语言一样的控制字符，例如在第 9 章中讨论的 AWK 语言的 `print` 函数。

```
print "Hello Again\n";
```

正如前面的示例(取自简单的 Hello World 脚本)显示的那样,向标准输出打印文本是很简单的。为了向一个具有文件句柄的文件打印文本,假设该句柄名为 FILE,请参考稍后关于写文件或者在文件的后面附加内容的讨论。这个示例的 print 语句看起来类似于:

```
print FILE "Hello Again\n";
```

如果想在文本字符串中打印一个变量的值,可以使用点号(.)字符把变量的内容加入到字符串语句中,如下所示:

```
print "How are on this day , the " . $date . "?\n"
```

2. chomp、join 和 split

函数 chomp、join 和 split 是 Perl 中操作变量的值的 3 个关键函数。例如,对于一个变量,无论它是标量还是数组,chomp 都将删除变量末尾的(多个)换行符。chomp 还返回删除的换行符的个数。

```
# chomp in action for a scalar and an array
chomp $name;
chomp @authors;
```

函数 join 把两个单独的字符串连接成一个字符串,就像点号运算符那样。然而,与点号运算符不同,join 函数的一个参数可以包含字段分隔符的值,该值用于界定合并的字符串。

```
# joining a number of strings together with a colon delimiter
$fields = join ':', $data_field1, $data_field2, $data_field3;
```

因此,在 join 函数的这个示例中,\$data_field1、\$data_field2 和 \$data_field3 的值将连接到一起,只有每个字符串之间的冒号将分开每个字符串。例如,假设 3 个变量包含的值依次是 Paul、Joe 和 Craig,那么 \$fields 将包含字符串 Paul:Joe:Craig。

函数 join 的反函数是 split。split 扫描一个字符串并根据参数指定的分隔符分割字符串,把字符串分割成一组字符串,像下面的代码演示的那样:

```
# splitting a string into substrings
($field1, $field2) = split ':/', 'Hello:World', 2;

# splitting a scalar and creating an array
(@fields) = split ':/', $raw_data;
```

在第一个 split 示例中,该函数将在冒号处分割字符串 Hello:World。也就是说,冒号前面的文本,Hello,将进入第一个变量;而 World,也就是冒号后面的文本,将进入第二个变量。注意函数参数中最后面的 2。这是一个限制,指定 split 最多把字符串分割成两个部分。这个限制并不是必须的,因为事先不可能总是知道它的值。

split 函数的真正威力在于它可以使用正则表达式进行模式匹配:

```
# split raw_data at any point in which a number is located and put into fields
(@fields) = split '/[0-9]/', $raw_data;
```

把 split 函数的这个新示例和前面的两个示例比较一下。在前面的两个示例中,split 是在分析字符串。在第一个示例中字符串只是“Hello:World”,在第二个示例中字符串是变量 \$raw_data

的值。在这两个示例中，`split` 函数只是在探测到冒号的时候分割字符串。

然而，在第三个示例中 `split` 函数在任何发现 0 到 9 之间的数字的时候就分割字符串，并把每个分组添加到数组字段中。正则表达式在这里的作用是即使程序员不知道 `$raw_data` 的值的格式，例如将来自某个没有固定字段分隔符文件的输入读到变量中，仍可以把它分割成有意义的数据。

3. `open`、`close`、`opendir`、`readdir` 和 `closedir`

Perl 使用 `open`、`close`、`opendir`、`readdir` 和 `closedir` 这些函数来访问 Unix 系统上的底层文件系统。

例如，`open` 函数打开一个作为参数传递给它的文件。然后就可以使用所获得的文件句柄处理文件的内容，对文件进行读、追加或写操作。

```
# open the file and slurp its contents into an array and then close the file.
open(FILE, "/etc/passwd") ;
@filedata = <FILE>;
close(FILE);
```

在这个示例中，文件 `/etc/passwd` 的文件句柄是 `FILE`。在打开文件之后，文件句柄把文件的内容填入一个数组。然后使用 `close` 函数关闭文件。下面的“实战”显示了怎样使用上面所讨论的函数。

实战

在 Perl 中打开文件和目录

- (1) 在主目录中用文本编辑器创建一个名为 `file.pl` 的文件，内容如下：

```
#!/usr/bin/perl -w
# Create a file that will have a directory listing of user's home dir

print "About to read user's home directory\n";
# open the home directory and read it
opendir(HOMEDIR, ".");
@ls = readdir HOMEDIR;
closedir(HOMEDIR);

print "About to create file dirlist.txt with a directory listing of user's
home dir\n" ;
# open a file and write the directory listing to the file
open(FILE, ">dirlist.txt");
foreach $item (@ls) {
    print FILE $item ."\n";
}
close(FILE);
print "All done\n\n";
```

- (2) 保存文件并退出编辑器之后，修改文件的权限并执行下面的代码：

```
% chmod 755 file.pl
% ./file.pl
```

- (3) 一旦脚本打印出 “All Done”，请找到 `dirlist.txt` 文件并用 `cat` 命令显示它的内容。输出

看起来可能和下面的内容类似：

```
% cat dirlist.txt
.
..
.bash_history
.bash_profile
.bashrc
Applications
Desktop
Documents
Downloads
file.pl
Library
Movies
Music
Pictures
Public
```

工作原理

函数 `opendir` 打开在参数中指定的目录并为打开后的目录创建一个文件句柄，该句柄也是在函数中指定的：

```
opendir(HOMEDIR, ".");
```

在“实战”示例中，`opendir` 函数打开的是当前目录，用双引号(“)之间的点号(.)表示。一个点号在 Unix 文件系统中表示当前目录，两个点号(..)表示上一级目录。除了在这个示例中使用的相对路径以外，`opendir` 函数还可以使用绝对目录路径作为参数。但是使用相对路径时必须小心，因为这些路径与脚本文件在文件系统中的位置有关，而不是与调用该脚本的用户或者应用程序的位置有关。

```
@ls = readdir HOMEDIR;
closedir(HOMEDIR);
```

一旦打开目录并为其指定了一个文件句柄，就可以使用 `readdir` 函数。这个函数接收一个文件句柄作为参数，该文件句柄代表着一个将要读取的目录。在这个示例中，该函数同时把目录中的条目发送到`@ls`中。

在讨论 `open` 和 `close` 函数的时候已经提到，`closedir` 函数关闭文件句柄表示的目录，在这个示例中是 `HOMEDIR`，它最初是由 `opendir` 函数创建的。

换句话说，`opendir`、`readdir` 和 `closedir` 函数与针对文件的 `open` 和 `close` 函数类似：

```
# open a file and write the directory listing to the file
open(FILE, ">dirlist.txt");
foreach $item (@ls) {
    print FILE $item ."\n";
}
close(FILE);
```

差别在于对于文件，没有读、写或追加函数。为了写文件，使用一个右箭头(>)表示将创建

一个新文件并向其中写入数据。为了向一个已经存在的文件中追加数据，需要使用两个右箭头(>>)。如果只是为了阅读文件，就不需要使用右箭头。

由于这个示例接收了从目录中读取的多个条目并把它们赋值给一个数组，所以使用一个控制循环 `foreach`，来处理数组中的每个元素，`@ls` 把它赋给变量 `$item`，然后打印到文件中。代码中的 `foreach` 循环和其他控制分支将在“循环和条件”小节中讨论。

注意，和讨论 `print` 函数时提到的那样，需要提供文件句柄让它知道是打印到文件流而不是标准输出。

4. `my` 和 `local`

`my` 和 `local` 运算符声明一个作用域受限的变量，例如只在一个子程序或循环中使用的变量。`my` 声明一个变量，这个变量只在一个有限的范围内存在，例如一个循环中；`local` 声明一个或多个已经存在的全局变量，这些变量拥有只在一个代码块中有效的值，例如在一个循环中。

```
# Global variable $name is given a name
$name = Paul

# Enter our loop
foreach (@filedata) {
    # declare a new variable for just the loop
    my $current_file_file;
    # create a local version of name to temporarily assign values within the
    # loop to
    local $name
    ...
}
```

在这段示例代码中，有两个定义了3次的变量：`$name` 和 `$current_file_file`。首先在任何循环或子程序之外定义变量 `$name` 并给它赋值，这称为全局声明。既然这样，这个全局变量中的值就可以在后续代码块中的任意位置上使用，不管它是循环、子程序还是其他的表达式。但是，变量 `$current_file_file` 不是全局变量。`my` 运算符声明这个变量，也就是 `$current_file_file`，它只在声明它的代码块中有效，这里是指 `foreach` 循环，一旦处理完这个代码块，该变量和它的值都将变为无效。也就是说，一旦循环结束，变量 `$current_file_file` 将不再存在而且分配给它的内存将被释放以作他用。

`local` 运算符和 `my` 相比稍微有些不同。和 `my` 一样，`local` 把变量限制到一个特定的代码块中。在前面的示例中，这个代码块是 `foreach` 循环。然而，和 `my` 不同的是，`local` 运算符修改了一个已经存在的变量的作用域。这意味着在前面的示例中进入 `foreach` 循环时变量 `$name` 仍是全局的，其值为 `Paul`。但是，在使用 `local` 运算符修改其作用域之后，在循环内可以用这个变量包含其他的值。一旦这个代码块执行结束，该变量将恢复其全局特性并重新获得值 `Paul`。

使用全局的和作用域受限的变量有助于管理内存资源，在编写大量耗费系统硬件资源的脚本时，例如处理一个大型的文本文件时，强烈推荐这种用法。

5. 循环和条件

说到循环，Perl 要是没有一组强大的、用于创建根据一组或多组条件执行的代码块，那还能有什么作用呢？

while

在循环内的任何代码执行之前，应先定义决定循环怎样运行的条件，循环体由起始花括号和结束花括号定义。也就是说，在定义和进入任何循环之前，需要先定义返回 **true** 或 **false** 的条件，否则循环将无限期地运行下去而程序将永远不会结束。

```
while ( $counter < 10 ) {
    print $counter . "\n";
    $counter++;
}
```

例如，前面的代码使用了一种通过计数器控制循环的普通方法。只要变量 **\$counter** 的值小于 10，循环就会继续。注意循环代码块中的最后一条语句：

```
$counter++;
```

这行代码把 **\$counter** 变量的值增加 1，它是程序员常用的一种简洁的语法形式。也就是说，它和下面的语句具有相同的效果：

```
$counter = $counter + 1;
```

do...while

do...while 循环和 **while** 循环类似，它在一次循环结束时检查测试条件，而不是在循环开始时，因此，它至少执行循环体一次。

```
do {
    print $counter . "\n";
    $counter++;
} while ( $counter < 10);
```

因而，比较一下这个 **do...while** 循环的示例和前面使用 **while** 循环的示例。在 **while** 循环中测试 **\$counter** 变量的条件在循环的起始处，而在这个示例中条件在循环的结尾处才进行测试。这意味着，在 **while** 循环的示例中，如果 **\$counter** 变量的值在执行到循环之前大于 10，那么这个循环永远不会执行，因为在循环开始之前测试就失败了。然而，在 **do...while** 循环中，即使 **\$counter** 变量的值大于 10，循环体内的代码也将至少执行一次，因为只有在进入循环代码块之后才会进行条件测试。

foreach

遍历数组的最好方法就是使用 **foreach** 循环来处理它的元素。例如，下面的循环将打印出一个名为 **authors** 的数组中的元素，在每次循环打印的时候把每个元素传递到一个临时标量中，直到打印完所有的元素。

```
foreach $temp (@authors) {
    print $temp . "\n";
}
```

if...else

这个古老的 **if...else** 语句可以用于测试条件并有选择地执行命令。其基本语法如下所示：

```
if ( $name eq 'Paul' ) {
```

```

print "Hi Paul\n";
} elsif ($name eq 'Joe') {
print "Hi Joe\n";
} elsif ($name eq 'Jeremy') {
print "Hi Jeremy\n";
} else {
    print "Sorry, have we meet before? ";
}

```

这个示例演示了很多需要测试的测试条件。如果一个条件为 `true`，那么程序将执行接下来的代码块。因此如果变量 `$name` 的值为 `(eq)Paul`，那么程序将执行花括号内的单行 `print` 语句。如果不相等，将测试下一个条件——`$name` 的值是否为 `Joe`。如果测试失败，程序将执行最后面的 `else` 语句块。

17.3 更多 Perl 代码的示例

理解 Perl 命令是怎样工作的最佳方法是分解一段脚本。下面的“实战”示例将给您一个实践的机会。

实战

检查来自标准输入的输入

(1) 在文本编辑器中，输入如下脚本，把它保存为 `check.pl`，并修改权限使得该脚本可执行。

```

#!/usr/bin/perl -T
# check.pl
# A Perl script that checks the input to determine if
# it contains numeric information

# First set up your Perl environment and import some useful Perl packages
use warnings;
use diagnostics;
use strict;
my $result;

# Next check for any input and then call the proper subroutine
# based on the test
if (@ARGV) {
    # If the test condition for data from standard in is true run
    # the test subroutine on the input data
    $result = &test;
} else {
    # else the test condition is false and you should run an error routine.
    &error;
}

# Now print the results
print $ARGV[0]." has ".$result." elements\n\n";
exit (0);

sub test {
    # Get the first array element from the global array ARVG

```

```

    # assign it to a local scalar and then test the local variable
    # with a regular expression
    my $cmd_arg = $ARGV[0];
    if($cmd_arg =~ m/[0-9]/) {
        return ('numeric');
    } else {
        # There was a error in the input; generate an error message
        warn "Unknown input";
    }
}

sub error {
    # There was no input; generate an error message and quit
    die "Usage: check.pl, (<INPUT TO CHECK>)";
}

```

(2) 在命令行使用不同的参数或者不带任何参数运行该脚本，如下所示。注意脚本输出的不同文本行取决于提供了什么样的参数：

```

% ./check.pl
Usage: check.pl, (<INPUT TO CHECK>) at ./check.pl line 41.
%

% ./check.pl 42
42 has numeric elements

%

% ./check.pl h2g242
h2g242 has numeric elements

%

% ./check.pl hg
Unknown input at ./check.pl line 35.
hg has 1 elements

%

```

工作原理

为了调用 Perl 解释器的注释，这个示例使用了-T 开关，这使 Perl 以不洁模式(taint mode)运行：

```
#!/usr/bin/perl -T
```

如果某个用 Perl 编写的脚本将由多个用户使用，或者如果脚本是在一段外部代码的基础上开发的，不洁模式将确保不隐式信任任何由运行这段代码的用户提供的输入。也就是说，在不洁模式下，Perl 解释器假设所有用户输入都是不洁的，或者是潜在恶意的，并严格限制脚本将对输入执行的操作。

Perl 使用一组特殊的规则来限制可能在不洁数据上执行的操作。例如，不洁数据将不会在和系统底层交互的函数中使用。当解释器遇到一个以一种它认为不安全的方式使用不洁数据的

操作时，它将简单地停止执行并返回一个错误。然后由系统管理员检查代码(如果脚本来源未知)并决定为了使代码变安全需要进行的必要修改，或者仔细检查用户的什么输入导致了这样一个问题。

最重要的是，千万不能把不洁模式看作是 Perl 脚本万无一失的安全保障。虽然不洁模式可以帮助防止缓冲区溢出攻击，但它不能检测到所有可能的恶意输入，例如一条 SQL 命令，如果正在运行的脚本和一个后台数据库有交互，这条命令将删除整个数据库。

下面的代码段为 Perl 解释器执行示例脚本的时候设置了一些环境条件：

```
# First set up your Perl environment and import some useful Perl packages
use warnings;
use diagnostics;
use strict;
my $result;
```

首先，`use` 函数向当前的脚本中导入一些代码。这些代码的来源是一个已命名的、供 `use` 函数使用的 Perl 模块，例如，`use warnings` 导入定义在一个标准的名为 `warnings` 的 Perl 模块中的条件。该模块中的有些警告看起来可能很晦涩。而 `use diagnostics` 正好可以解决这个问题。在激活这个模块之后，由 `use warnings` 和 `-w` 开关产生的警告都将大幅地展开。当然，设计这两个函数都是用于程序开发周期内，检查完代码的问题之后也许应该将它们关闭，因为它们都是很大的模块，无论出于什么原因您都不会希望载入它们。

`use strict` 模块强迫所有的变量都必须用 `my` 声明，以跟踪它们的作用域并限制那些只需要使用某个(些)变量的代码占用的内存。`strict` 模块还要求在代码中绝大多数字符串都要包含在引号中而且不能出现没有冠上形态符号的字(bare word)(或未由引号包含的字符串)。最后，`strict` 还强制执行在脚本内不可以使用符号引用的规则。换句话说，`strict` 强制好的编码习惯。是的，即使在一种认为所有的编程风格都有效的语言中，确实存在好的编码习惯这样的事情，稍后将向您解释其优点。

最后，由于使用了 `strict` 模块，需要在脚本中定义一个全局变量 `$result`。

在这个 `if-else` 条件语句中以前没有讨论过的新内容是 `@ARGV` 数组和子程序调用：

```
# Next check for any input and then call the proper subroutine
# based on the test
if (@ARGV) {
    # If the test condition for data from standard in is true run
    # the test subroutine on the input data
    $result = &test;
} else {
    # else the test condition is false and you should run an error routine.
    &error;
}
```

`@ARGV` 数组包含调用 Perl 脚本时使用的命令行参数。这段脚本希望只有一个参数，所以这个数组中的第一个元素就是需要测试的内容。

根据测试的结果，程序将执行两个子程序 `&test` 和 `&error` 中的一个。我们希望子程序 `test` 返回一个值并把这个值赋给变量 `$result`。当然，在这个示例中，`$result` 是一个全局变量，所以可以在子程序 `test` 内部进行赋值，但是这种方法提供了一个从子程序返回结果的示例：

```
# Now print the results
print $ARGV[0]." has ".$result." elements\n\n";
exit (0);
```

示例脚本主体的最后一段代码打印结果然后使用 Perl 的 `exit` 函数结束执行。`exit` 函数能够返回一个值，这个值可以由 Unix 命令行计算。在这个示例中，它返回 0，表示 true 或执行成功。

```
sub test {
    # Get the first array element from the global array ARVG
    # assign it to a local scalar and then test the local variable
    # with a regular expression
    my $cmd_arg = $ARGV[0];
    if($cmd_arg =~ m/[0-9]/) {
        return ('numeric');
    } else {
        # There was a error in the input; generate an error message
        warn "Unknown input";
    }
}
```

在进入 `test` 子程序之后，所做的第一件事情就是声明一个名为 `$cmd_arg` 的局部变量并把 `$ARGV[0]` 的值赋给它。接下来要做的事情是测试在这个字符串中是否包含任何数字：

```
if ($cmd_arg =~ m/[0-9]/) {
```

程序将匹配 `(m/)` 表示 `[0-9]` 之间的数字的正则表达式然后绑定 `$cmd_arg` 中待搜索的字符串。如果匹配成功，程序将在调用该子程序的地方返回字符串 “numeric”。如果匹配失败，`warn` 函数将在标准错误上打印一个错误消息 (Unknown input)，该错误消息包含了给它提供的字符串和触发错误消息的程序代码的行数。对于检修问题和调试，这是非常有用的信息。本章稍后将讨论检修问题和调试。

`warn` 函数不会结束脚本的执行。下面是调用这段脚本的一个示例：

```
% ./check.pl hg
Unknown input at ./check.pl line 35.
hg has 1 elements

%
```

程序打印出错误消息，但脚本仍将退出这个子程序，然后执行脚本主体中最后的 `print` 语句。注意这个子程序已经结束了，它返回一个传递给 `$result` 的值，该值为 1，然后由程序打印出来。可以为 `print` 语句加一个 `if` 条件，例如如果 `$result` 的值为 1 (在 Perl 中表示 false，和大多数 Unix 系统的情况一样)，就不执行最后一条 `print` 语句：

```
print $ARGV[0]." has ".$result." elements\n\n" if $result;
```

也就是语句：

```
if ( $result) {
    print $ARGV[0]." has ".$result." elements\n\n";
}
```


要讨论的最后一段代码是 `error` 子程序：

```
sub error {  
    # There was no input; generate an error message and quit  
    die "Usage: check.pl, (<INPUT TO CHECK>)" ;  
}
```

`die` 函数向标准输出打印一条消息，就像 `warn` 做的那样。然而，`die` 将结束脚本的执行而不会继续。因此在查找任何输入失败时脚本的输出看起来可能和下面的内容类似：

```
% ./check.pl  
Usage: check.pl, (<INPUT TO CHECK>) at ./check.pl line 41.  
%
```

关于这段脚本最后需要注意的是：本章已经说过，Perl 是一种“只要我高兴，怎样都可以”的语言。这个示例脚本的功能实际上可以在一行代码中完成：

```
#!/usr/bin/perl -T  
print $ARGV[0]." Has numeric elements\n\n" if ($ARGV[0] =~ m/[0-9]/);
```

如果有数字出现，这行脚本将打印如下结果。如果没有匹配的数字或者没有提供输入，什么也不会打印：

```
% ./check.pl h2g2  
h2g2 has numeric elements  
%
```

而且，即使使用了 `strict` 和 `warnings` 模块，脚本的有效性足以使得它可以处理任何问题，但正如下一节讨论的那样，在检修错误或修改该代码时，这些快速开发的代码可能会对很多人造成损害，包括您在内。

17.4 检修 Perl 脚本

不讨论执行，在调试一段 Perl 脚本之前，需要验证脚本的语法无误。当然，这在每次试图执行脚本的时候会自动进行。例如，如果 Unix 系统在脚本的第一行不能找到 Perl 解释器，shell 将返回 `command not found` 错误。其他常见的语法错误包括在一行的最后面遗漏分号或者把分号放在本不该出现的地方。

还可以在不用执行脚本的情况下检查脚本的语法，例如检查 `hello.pl` 或者 `file.pl` 的语法，如下所示：

```
% perl -c hello.pl  
hello.pl syntax OK  
%
```

`-c` 开关告诉 Perl 解释器只检查作为参数提供的文件的语法并跳过执行脚本，这个开关是在命令行而不是在脚本的第一行调用的。

在命令行还可以使用 `-w` 开关打开警告。

然而，注意，如果试图在 `check.pl` 脚本上使用 `-w` 开关，将产生如下错误：

foo late for "-T" option at check.pl line 1.

这是因为在首先调用 Perl 解释器并设置脚本环境的时候需要提供-T 开关，这个开关告诉 Perl 解释器以不洁模式运行。由于 Perl 是在命令行而不是在前例中脚本的第一行调用的，因此解释器会打印错误，指示由于 Perl 已经在运行，要建立不洁模式已太迟了。为了避免出现这个消息，在测试和调试的时候把-T 开关注释掉，如下所示。只是不要忘记在找到所有的 bug 之后取消对-T 的注释。

```
#!/usr/bin/perl #-T
```

如果在脚本中包含了 use strict 模块，语法检查还会测试不适当的标量作用域，这和已经知道的普通的语法问题有点不同。

当然，语法检查成功并不意味着任何以这种方式表达出来的语句都会返回所期望的结果。

另一方面，Perl 调试器是用于跟踪脚本自身执行的工具。调试器是一个交互式的 Perl 运行环境，它在使用-d 开关调用 Perl 时初始化：

```
% perl -d check.pl
```

和前面检查语法的示例一样，需要提供一个文件名作为参数，该文件中包含了要调试的脚本。由于是交互式的，下一步就是使用调试器的命令控制脚本的分步执行以便发现正在查找的 bug。表 17-1 解释了这些命令。

表 17-1

命 令	说 明
s	仅执行脚本中的一步，或者说一行
n	仔细地按步执行以避免进入子程序中
c	继续处理直到遇到某种断点
r	继续运行直到从当前的子程序中执行返回命令
w	在窗口中显示当前行前后的代码
b	设置一个断点(使用行号或者子程序的名字)
x	显示变量的值(包括数组和关联阵列)
W	设置一个监视表达式(watch expression)
T	显示栈回溯跟踪(stack back trace)
L	列出所有的断点
D	删除所有的断点
R	重新启动脚本以便再次测试

表 17-1 中列出的命令用于在 Perl 的交互式调试环境中控制 Perl 执行什么内容，这样就可以检查脚本的行为以便定位任何逻辑错误。

无论如何，在运行调试器来调试 check.pl 的时候，输出的第一部分总是如下所示：

```
Default die handler restored.
```

Loading DB routines from perl5db.pl version 1.07

Editor support available.

Enter h or 'h h' for help, or 'man perldebug' for more help.

```
main::(check.pl:10):    my $result;
DB<1>
```

调试器跳过注释等内容并在第 10 行的可执行代码上停下来。使用 **w**——注意是小写的 **w**，大写的 **W** 是另一个命令——来确认调试器位于何处：

```
DB<1> w
7:      use warnings;
8:      use diagnostics;
9:      use strict;
10==>   my $result;
11
12      # Next check for any input and then call the proper subroutine
13      # based on the test
14:      if (@ARGV) {
15          # If the test condition for data from standard in is true run
16          # the test subroutine on the input data
```

w 命令显示了即将执行的那部分代码。也就是说，它显示了即将执行的代码行。箭头符号 (**=>**) 指出即将执行的代码，以及该行代码前后的代码，这样就提供了一幅远景图。要按步执行每条命令，输入：

```
DB<1> s
main::(check.pl:14):    if (@ARGV) {
DB<1> s
main::(check.pl:20):    &error;
DB<1> s
main::error(check.pl:42):    die "Usage: check.pl, (<INPUT TO CHECK>)";
DB<1>
```

调试器执行第一条 **if** 语句然后执行 **error** 子程序。可以检查数组的值以确保正确地执行了 **if** 语句：

```
DB<1> x @ARGV
empty array
DB<1>
```

显然，数组是空的，所以对脚本来说 **error** 子程序是正确的路径。为了测试这段脚本中的其他分支，需要在调用调试器的时候提供一个文本参数。记住脚本的作用：它检查输入的字符串的文本并根据输入字符串中是否包含数字打印一条消息。由于在调用这段脚本的时候没有为脚本提供参数(只有一个参数——**-d** 开关——该参数是为 Perl 本身提供的)，脚本选择了打印错误消息的分支。因此如果想测试脚本的其他分支，需要提供一个输入字符串以便测试这些分支。

可以在 <http://debugger.perl.org> 上在线获得更多关于 Perl 调试器的信息和一对一逐项对应的 (one-on-one) 帮助。

您现在应该明白 Perl 调试器的基本使用方法了。当然，在调试或者检修脚本的时候需要记

住的最重要的事情是，确保在整个调试过程中对程序的每个主要分支、每个模块的动作有一个清晰的认识。在调试脚本或程序的时候，您将发现很多逻辑错误之所以会发生是因为程序代码正以一种与您想象的不同方式执行。

17.5 小结

Perl 是一个强大的 Unix 工具，它的作用已远远超出最初为处理文本而设计的一种语言。学习 Perl 的关键是要理解它对很多人意味着很多内容。Perl 的用处使得有些人把它看作是瑞士军刀或是连接 Internet 的胶水和管道。

由于其本身的特性，Perl 是一种易于学习的语言，大多数任务都只需要 Perl 语言的一个很小的子集即可完成。而且，虽然可以在很多系统上找到 Perl，类 Unix 的或非类 Unix 的，但是理解基于 Unix 的系统是怎样运行的将使 Perl 更容易学习，因为 Perl 最初就是伴随着 Unix 一起开发的，它和 Unix 的联系最紧密。Perl 完全是基于 Unix 的系统的一部分，它是源代码公开的，能进行文件处理和正则表达式处理。除了这些内容以外，本章还讨论了以下主题：

- Perl 是一种功能强大的编程语言，它没有其他 Unix 工具(例如 shell 脚本和 AWK)的限制。
- 选择 Perl 解决问题的好处和坏处。
- 文本处理的基本命令和函数，例如 split 和 join。
- 理解怎样编写和执行 Perl 脚本。
- 检修 Perl 脚本的基本步骤。

17.6 练习

- (1) 编写并调试一段脚本，该脚本读取/etc/passwd 文件的内容并只输出用户名和 ID 字段。
- (2) 使用下面的命令来对本章的 check.pl 脚本调用 Perl 调试器。逐行执行每条代码。每个调用的区别是什么？

```
perl -d check.pl h2g2
perl -d check.pl h2
perl -d check.pl
perl -d check.pl 42
```

- (3) 修改 check.pl 脚本，使得在输入不包含数字的字符串时，脚本将向用户打印消息表明这一点。例如指定的测试字符串是“hg”，脚本现在的输出不应该是“Unknown input at ./check.pl line 35. hg has 1 elements”，而应该是“hg has no numeric elements”。

第 18 章 备份工具

备份(backup)——这个术语在您曾经阅读的很多书籍中都讨论过，而且都解释了这样做的理由。这个概念很简单：把不能替代的或者关键的文件和数据的一份副本(备份)保存到一个安全的地方以防主系统(primary system)出现什么问题。备份是到目前为止在系统发生崩溃、出现故障、配置错误或者有恶意行为的时候防止关键信息丢失的最佳方法。虽然如此，很多家庭用户和公司用户并没有对备份给予足够的重视，而是常常把它们放到次要的位置。然而，如果在系统上保存有重要的信息，除了备份以外别无选择。备份和其他任何生产事务一样很有必要。本章讨论备份的基本概念和方法，以及备份数据的恢复。

18.1 备份基础知识

有效的备份需要进行周详的计划，包括决定怎样、何时以及在何处存放备份。在第一次需要使用备份来恢复丢失的数据的时候，这些计划的价值就充分体现出来了。

每个人都会犯错。如果曾在某个地方使用过计算机，很可能偶然删除过非常重要的文件，或者突然掉电或系统在没有任何警告的情况下重新启动时丢失一个非常重要的文档。这种事情时有发生，备份可以把这种情况从一次灾难减小成一次小小的不便。大多数用户，特别是家庭用户，会把他们的数据保存到本地硬盘上，这使得检索和修改数据非常方便。如果硬盘出现故障怎么办？任何硬盘到最后总会出现故障——平均故障时间 (Mean Time Before Failure, MTBF)——如果没有有规律地把数据备份到独立的介质中，您将面临丢失重要信息的危险。

如果您是家庭用户，请考虑一下如果丢失了一年多以来收集的所有数据会让您在时间和金钱上蒙受多大的损失。即使只丢失了 30 天的工作数据会怎样？如果公司的每个员工都丢失了他们近 3 天的数据又会怎样？如果没有有规律地对数据进行备份而不得不重新创建所有丢失的数据，这些情况中的任何一种都会造成时间、金钱和生产上的重大损失。

备份数据不仅可以防止硬盘失效，而且还可以防止其他外部的和内部的事件，例如自然灾害、恶意损坏数据的实体、以及对公司不满的员工从系统中删除信息——或者其他任何可能损坏数据的事故。

18.1.1 决定备份什么数据

决定备份什么数据很耗费时间，但是完全值得，特别是在必须恢复数据的时候。您肯定不希望成为告诉管理员无法恢复一个关键的业务文件因为没有对它进行备份的人。即使没有管理一个公共的备份系统，也不会希望因为在备份文件之后却发现遗漏了一个关键的目录或者文件而陷入麻烦。

通常希望备份整个文件系统。如果这由于资源的限制(财力和/或时间)不可行，就不得不选出每个需要备份的文件系统或者文件。最少，需要备份主目录、用户定制的配置文件、已安装

的特定软件和它们的配置文件(包括许可文件), 以及任何其他如果丢失就会造成严重的财产损失或者使得个人或业务被迫中断的数据。

备份用户主目录非常重要, 因为普通用户都是在这个目录中完成他们的工作的, 这个备份通常是在系统中进行的最重要的工作。用户很可能会意外地删除或丢失他们的数据, 有时候这些数据是他们工作数周或数月的成果。通过备份用户的主目录, 可以确保在丢失某个文件的时候能使生产力的损失降到最小。

在修改配置文件的时候, 用户会希望保存这些修改以便在发生系统灾难的时候, 可以恢复这些修改而不用打开配置文件从头再做一次。这样将大大减少系统停机和系统备份之间的时间。这包括/etc 目录中的很多文件, 但在整个系统中还有很多其他可能需要备份的文件, 这将根据系统配置而有所不同。

其他需要备份的文件是已经安装了的特殊的、用户定制的或者内部开发(in-house-developed)的应用程序, 包括它们的配置文件和许可文件。例如, 用户可能希望备份 Oracle 数据库, 因为公司的大多数重要信息可能保存在里面。这些文件是公司信息程序的支柱, 所以需要仔细地研究它们以保证备份的有效性并有规律地对其进行备份。

您也许正在运行一个很小的商店或家庭系统, 并认为没有必要备份整个文件系统, 而只需要备份包含了重要文件的主目录。如果是这种情况, 仍然需要制订一个备份计划——即使只是每周向 CD-ROM 中备份一小部分数据——以便不会丢失保存在系统中的工作。

如果不能备份整个文件系统, 那就需要花点时间来决定应该备份哪些文件和目录。投入到研究哪些文件对您和您的系统非常关键的时间和与对系统非常熟悉的人交谈的时间在恢复一台机器的工作能力、或者恢复个人使用的文件时, 会为您节约很多时间。

18.1.2 备份介质类型

备份数据的介质类型是非常重要的。如果所使用的介质保存不下所有的关键数据, 或者不能在期望的时间段内有效地保存数据, 那么备份就失去了其作用。备份数据的介质类型很多, 包括:

- **磁带**——备份数据最古老的方式之一。它经受住了时间的考验并已证明是非常耐用的。最新版本的磁带非常快, 而且可以保存非常多的数据。使用磁带进行备份所需的介质和硬件可能会非常昂贵而且最初架设时非常困难。这种介质非常适用于长期保存(20 年甚至更长)。大多数公司都是使用磁带备份数据。
- **CD/DVD-ROM**——一种比较新(相对而言)的备份数据的方法, 由于能适用于多种平台的硬件, 所以其使用已变得很普遍。这项技术还在发展之中, 其速度比较快而且能存储相当数量的数据。使用这种方法进行备份所需的介质和硬件很容易获得。人们还不能确定这种存储类型的长期性, 但根据所选择的介质的质量, 一般认为在 10 到 20 年之间。这是现在家用计算机最常用的备份硬件。
- **大容量磁片(Zip Drives)**——一种比较过时的技术, 由于容量很小而且速度很慢已失去其市场份额。使用这种方法进行备份所需的介质和硬件仍然很容易获得, 但越来越少。这种类型的存储可以维持数据相当长的一段时间(10 年以上)。
- **硬盘/网络备份**——由于硬盘空间的价格比较低, 这种类型的备份越来越普遍。这不是一种优先选择的方法, 因为它对硬件和可以篡改或意外地删除数据的能力的依赖性很高。这种方法不适于长期存储而且在备份数据时也不推荐使用它。

还有其他类型的介质，包括 USB 驱动器，它的空间一般都有限，不足以存储相当数量的备份数据。软盘的价格也下降了，因为它的存储容量实在是太有限了。

在决定使用什么类型的介质进行备份的时候，需要考虑要备份的是什么、数据的重要性(它对您和您的业务的价值)、需要备份的数据量，以及需要备份开始运行的时间限制(判断是否需要更快的硬件等)，和为了能够恢复数据计划保存备份多长时间。所有这些问题在决定什么备份硬件和介质最适合自己的情况时都会有一定的影响。例如，如果需要保存数据 30 年，可能会选择磁带而不是在线硬盘存储，因为人们已经证实磁带备份只要存储得当就可以保存很长时间。

18.1.3 备份类型

备份数据是为了在需要的时候能恢复它。出于这个原因，必须决定备份数据的最佳计划。下面是 3 种主要的备份类型：

- **完全(full)备份：**在备份定义的范围内备份所有数据。例如，如果设置备份命令备份/etc 和/home 目录，那么不管这两个目录中的文件在上次备份之后是否发生过改变，系统都将对它们进行完全备份。
- **差分(differential)备份：**备份所有自上次备份之后发生过变化的数据。例如，如果在周五对/etc 和/home 两个目录进行了完全备份，然后在周一对这两个目录进行差分备份，那么系统将只对从上周五以后发生过变化的文件进行备份。为了进行完整的恢复，需要用到完全备份和最近的差分备份。
- **增量(incremental)备份：**备份上次备份完成之后所有发生变化的数据，而不考虑上次备份是什么类型。如果周五对/etc 和/home 两个目录进行了差分备份，然后周一又进行了增量备份，那么系统将对这两个目录中周五之后发生过变化的所有文件进行备份。在恢复时，需要用到恢复时段之前的一个完全备份和差分备份，然后是所有的增量备份。

每一种备份都有其优点和缺点，如表 18-1 所示。

表 18-1

类 型	优 点	缺 点
完全	恢复数据比差分备份或增量备份快	<ul style="list-style-type: none">• 备份所需的时间明显比其他类型长得多• 需要的介质空间最多
差分	<ul style="list-style-type: none">• 恢复数据比增量备份快• 备份速度比完全备份快• 与完全备份相比，需要较少的介质空间	<ul style="list-style-type: none">• 备份速度比增量备份的慢• 备份所需的介质空间比增量备份所需的多
增量	<ul style="list-style-type: none">• 备份速度比完毕备份或差分备份快得多• 备份所需的介质空间比完全备份或差分备份所需的明显小很多	恢复数据所需的时间比完全备份和差分备份所需的长

本章将讨论这些备份类型，使您能理解每种备份类型的优点和缺点。

18.1.4 备份时间

通常希望在系统上的活动最少的时候进行备份，以便捕获所有的变化。这通常意味着在午夜的时候进行备份，这可以用 cron(在第 11 章中讨论)轻松实现。另外，备份将耗费大量的系统

资源，如果还有其他耗费资源的进程正在运行，系统将变得很慢。第一次备份的时候，应该手动进行以确保备份正确，并确保系统在运行其他进程时不会变慢。

另一件需要考虑的事情是备份的频率。您将希望在如果数据丢失影响会非常大时备份数据，例如潜在的金融影响或生产力的损失。如果即使是损失一天的数据也是难以承受的，那么就不需要制定每周的数据备份时间表。通常，可以每周进行一次最小化的完全备份，然后在一周的时间里面再进行差分备份或增量备份以便捕获这一周中数据发生的所有变化。使用的频率和类型取决于所要求的备份有效时间段和恢复时间。

18.1.5 验证备份

在完成备份之后，可以通过恢复数据来验证备份。为了获得最大的保证，可以先恢复数据，然后再在其上工作一段时间以确保不会丢失关键数据。任何人都不希望发现自己的备份不完整，或者在需要它们的时候却无法使用。应该经常通过多种方法验证备份，从恢复整个备份到恢复单个文件以检查错误。正确的做法是，在一个测试系统上恢复数据并让用户判断是否丢失了数据。在恢复的时候，希望恢复系统然后试验该系统一段时间，以确保备份了需要备份的所有内容。最好是在测试时间表的时候就发现备份计划遗漏了某些数据或文件，而不是在发生灾难的时候才发现。根据需求和所涉及数据的价值，建议每个月或者每个季度测试一次备份。

18.1.6 保存备份

备份的保存是设计备份策略的时候经常忽略的一个关键部分。就地保存备份似乎是一个好主意，因为随时都可以使用它们，但是如果在很长一段时间内、甚至是永远无法进入放置服务器和备份的房间该怎么办？例如，如果就地保存备份，而房屋被烧毁，那么将丢失备份以及系统。

不能进入工作场所意味着需要在全新的硬件上恢复生产系统，所以必须能够使用备份介质。这就是为什么至少应该在其他地方、或者最起码在一个防火的地方保存备份的原因。如果确实要就地保存备份，则确保根据数据的敏感性妥善地保存并严格限制无关人员进入，因为这些备份可能包含了公司的所有商业机密。

如果选择在其他地方保存备份的方案，应该保证为您保存备份介质的公司提供 24 小时的服务；判断该公司的响应时间是否满足要求；检查保存设备的安全性；并查看一下该公司收到的客户投诉记录。为您保存备份的公司，其实保存了您的公司的商业机密，所以详细了解该公司保存备份的实际操作和过程是非常关键的。每隔一段时间，应该要求取回备份介质以检查该公司的响应时间和及时高效地提供所需介质的能力。

保存备份介质最常用的一种方法是把包含最近一周备份数据的备份介质就地保存在一个防火的地方。以便在发生任何意外的删除或数据丢失的时候可以迅速进行恢复。一周之后，可以将备份介质转移到别的地方一段时间(从一年到无限期)，以便在紧急情况下可以使用它。有些备份需要保存更长的时间以满足一些关于档案的需求，但在为公司做这件事情的时候应该咨询公司的法律部门，因为根据所从事的行业，相关的法律会规定必须保存数据多长时间。

18.2 备份命令

默认情况下，Unix 包含了很多备份数据的实用工具，这些工具可以向多种类型的介质备份，它们使得您可以选择最适合自己备份需要的工具，并提供了很大的灵活性，特别是在对工具进行脚本编程方面，这些内容在本书的其他地方已讨论过的。

很多备份命令都需要以一定权限的用户身份才能运行，例如根用户，因为备份可能需要访问根据许可限制只有特殊用户才能读取的文件，例如/etc/shadow 文件。还需要确保在备份的时候没有用户在使用文件或文件系统，因为这可能会让您遇到一些问题，例如备份发生错误但当时并没有看出来，直到试图恢复时才发现。

18.2.1 tar

可以使用 tar(磁带档案(tape archive))命令创建磁带归档文件，而且如果有必要，可以把这些文件直接发送到磁带设备。即使没有磁带驱动器这个命令也很有用，因为可以用它把很多文件合并到一个文件中，我们把这种文件称为 tarfile。例如，可以把一个包含 100 个文件和 20 个目录的目录合并成一个 tarfile。这和 WinZip、gzip 或 bzip 命令的压缩功能不同，但是可以联合 tar 命令和 gzip 或 bzip2 命令创建一个压缩的 tarfile。在压缩之前，tarfile 和它所包含的目录占用同样大小的磁盘空间。如果使用了 tar 命令但没有写入 tarfile 的磁带驱动器，可以简单地利用 CD-ROM 或稍后讨论的其他方式备份它。

tar 命令有很多功能，包括：

- t —— 显示包含 tarfile 内容的表格。
- x —— 提取或恢复 tarfile 的内容
- u —— 更新 tarfile 的内容

有些目录是不能归档的，例如/proc，因此使用/作为目录名会造成权限错误。还有很多目录是不需要归档的，例如/tmp。

创建 tarfile 的语法是：

```
tar -cvf tarfile_name_or_tape_device directory_name
```

参数 c 表示开始在 tarfile 的开头写入数据，v 表示 verbose，这个参数将提供很多输出，显示命令正在做的事情。f 选项表明用户将为 tarfile 提供一个名称，而不是使用在/etc/default/tar 中标识的默认名称。

例如，为了以整个/etc 目录中的文件创建一个 tarfile 并把它写到一个名为/dev/rmt0 的已配置好的磁带设备上，可以使用下面的命令：

```
tar -cvf /dev/rmt0 /etc
```

这将向指定的磁带设备写入/etc 目录的内容的一个副本。

如果没有磁带设备并希望把 tarfile 文件保存到一个名为/backups/etc-backup-122004.tar 的文件，可以运行下面的命令：

```
tar -cvf /backups/etc-backup-122004.tar /etc
```

在 tarfile 的名称后面加上.tar 扩展名是一种习惯，这样可以更容易地识别它们。如果使用压缩(本节稍后讨论)，可以添加.tar、.gz 或.tgz 后缀表示一个压缩的 tarfile 文件。这个命令创建指定的文件，该文件包含了/etc 目录中的所有文件和目录的副本(这可能会相当大)。

创建 tarfile 之后，可能希望查看它的内容。要这样做，可以对一个归档文件使用如下命令：

```
tar -tvf tarfile_name_or_tape_device directory_name
```

所以要查看在磁带设备上创建的/etc 备份的内容，可以运行如下命令：

```
tar -tvf /dev/rmt0
```

要查看创建在/backups/etc-backup-122004.tar 文件中的/etc 备份的内容, 可以使用:

```
tar -tvf /backups/etc-backup-122004.tar
```

如果只需要一个简单的列表, 可以省略 v 选项, 减少输出的内容, 其命令是:

```
tar -tf /dev/rmt0
```

或者

```
tar -tf /backups/etc-backup-122004.tar
```

这两个命令的输出可能类似如下内容:

```
#tar -tf /backups/etc-backup-122004.tar
etc/
etc/profile.d/
etc/profile.d/ssh-client.sh
etc/profile.d/alias.sh
etc/profile.d/xhost.csh
etc/profile.d/xhost.sh
(...The rest omitted for brevity)
```

为了恢复某个 tarfile 的内容, 可以使用 tar 命令的-xvf 选项。下面是提取 tarfile 的语法:

```
tar -xvf tarfile_name_or_tape_device directory_name
```

为了从前面发送到磁带设备的备份中获得/etc/passwd 的一个副本, 可以输入如下命令:

```
tar -xvf /dev/rmt0 etc/passwd
```

为了从前面创建的/backups/etc-backup-122004.tar 中提取相同的文件, 可以使用如下命令:

```
tar -xvf /backups/etc-backup-122004.tar etc/passwd
```

为了把/backups/etc-backup-122004.tar 的所有内容提取到当前的目录中, 可以使用:

```
tar -xvf /backups/etc-backup-122004.tar
```

这个命令在当前目录中创建一个名为 etc 的目录, 然后提取归档到/backups/etc-backup-122004.tar 中的所有文件的一个备份到这个 etc 目录中(它不会存放到/目录中, 除非/就是当前目录)。

提取文件时要小心, 因为它们将提取到当前目录中。因此如果在位于/目录的时候从/backups/etc-backup-122004.tar 中提取/etc/passwd 文件, 系统中当前的/etc/passwd 文件将被提取出来的那个文件覆盖(如果拥有恰当的权限)。为了避免意外的文件覆盖, 通常在一个临时目录中提取文件比较安全。

-u 选项可以更新 tarfile, 在一个 tarfile 文件的后面添加额外的文件。如果在 tarfile 中已经存在同名的文件, 将更新这个文件。这个过程可能会很慢, 所以请耐心等待。更新 tarfile 的语法是:

```
tar -uf tarfile_name_or_tape_device directory_name
```

为了用一个新的 passwd 副本更新/backups/etc-backup-122004.tar 中的/etc/passwd 文件, 可以执行如下命令:

```
tar -uf /backups/etc-backup-122004.tar /etc/passwd
```

这个命令对磁带归档文件不会起作用，因为并不是所有的磁性介质都可以在磁带上非连续的片断中写入数据。但是，如果文件自从上次加入到 `tarfile` 中之后发生了变化，那么在 `tarfile` 的末尾将添加该文件的一个新副本。

GNU `tar`(在一些 Unix 版本上可用，或者可以下载)还包含一些用于简单备份和恢复文件的脚本。

很多版本的 `tar` 都使您能够使用内置的脚本以及很多其他功能创建完整的文件备份，而在其他版本的 `tar` 中没有提供这些功能。为了回顾更流行的 `tar` 版本的完整功能集合，请通过 http://gnu.org/software/tar/manual/html_node/tar_toc.html 查看 GNU 的在线手册。

18.2.2 使用 `gzip` 和 `bzip2` 压缩

压缩数据是指得到一个正常大小的文件并减小它占用的磁盘空间。这需要一种称为压缩程序的特殊应用程序。把备份保存到介质上时，无论是哪种类型的介质，一般都希望尽量保留空间从而最大限度地发挥介质的作用。一般情况下，压缩文件不能被通常对其操作的标准应用程序读取，因为为了压缩对其进行了优化，这使它变得不可读。因此如果压缩了一个文本文件，就不能使用 `vi` 这样的文本编辑器来打开它。另外，取决于文件的类型，压缩之后这个文件通常会从原来的大小明显减小很多，表示它占用的磁盘空间比其未压缩的形式所占用的空间少。

Unix 有一些免费的压缩工具，这些工具把数据压缩成 `gzip` 和 `bzip2` 的形式。本节将对这些工具进行讨论。

1. `gzip`

GUN `zip(gzip)`是 Unix 操作系统提供的最流行的压缩程序，这是因为它可以在不同的系统架构之间移植而且它的功能很全面。

使用 `gzip` 对一个文件进行简单压缩的语法是：

```
gzip filename
```

为了通过压缩让一个名为 `mytextfile` 的标准文本文件占用更少的磁盘空间，可以执行命令：

```
gzip mytextfile
```

如果在执行上面这条命令之后用 `ls` 命令显示该目录的内容，输出将包含一个名为 `mytextfile.gz` 的新文件。扩展名 `.gz` 是 `gzip` 向任何使用该工具压缩的文件添加的标准扩展名。在该命令完成时，它将删除旧的未压缩的文件并只留下已压缩的版本。如果正在压缩一个很大的文件并非常关心速度，那么可以添加 `-1` 选项使 `gzip` 以更快的速度压缩这个文件：

```
gzip -1 mytextfile
```

如果更关心节约空间，可使用 `-9` 选项进行最大限度的压缩：

```
gzip -9 mytextfile
```

可以使用 1 到 9 之间的任意数字指出偏爱的速度或压缩程度。默认的压缩状态是 6，它是以速度为代价进行更好的压缩的边界值。

为了查看由 `gzip` 压缩的文件的信息，可以使用 `-l` 选项，它的输出和下面的内容类似：

```
#gzip -l mytextfile.gz
```



```
compressed uncompr. ratio uncompressed_name
614 1660 64.8 mytextfile
```

这个输出显示了已压缩文件的大小、未压缩文件的大小、用于该文件的压缩率，以及 `gzip` 归档文件中每个文件未压缩时的名称。

为了把文件恢复成原来的未压缩的状态，使用 `gunzip` 或者 `gzip -d`：

```
gunzip mytextfile.gz
```

或

```
gzip -d mytextfile.gz
```

这将创建一个名为 `mytextfile` 的新文件，该文件应该是刚才创建的那个普通文件的未压缩版本。如果正在解压的文件已经存在，`gzip` 命令就会提示覆盖这个文件。为了跳过这个动作并让 `gzip` 解压的内容覆盖已经存在的文件，需要在使用 `gzip -d` 或 `gunzip` 解压时添加 `-f` 选项。该命令结束之后，它将删除压缩文件并只留下未压缩的文件。在 Unix 中，压缩文件和未压缩文件都称为正式文件(`regular file`)。

在 `gzip` 的联机帮助文档中可以找到其他选项，这些选项用于更高级的压缩。

下面的练习提供了一个尝试使用 `gzip` 的机会。

实战 使用 `gzip`

- (1) 使用如下命令在 `/tmp` 目录中创建一个名为 `test-file` 的文件：

```
cd /tmp
touch test-file
```

- (2) 使用 `vi` 在这个文件中输入一些文本——10 行左右的随机语句——然后保存这个文件。
- (3) 使用 `ls -l` 命令显示文件的大小：

```
ls -l test-file
```

- (4) 为了证明可以查看这个文件，因为它是一个普通的文本文件，使用 `cat` 命令：

```
cat test-file
```

将看到前面创建的 10 行随机文本。

- (5) 使用 `gzip` 压缩这个刚才创建的文件：

```
gzip -l test-file
```

- (6) 现在在 `/tmp` 目录中产生了一个名为 `test-file.gz` 的文件。使用 `-l` 选项显示压缩文件的统计信息：

```
gzip -l test-file.gz
```

可以看到 `gzip` 所使用的压缩率。

- (7) 使用 `ls -l` 命令查看压缩文件的大小：

```
ls -l test-file.gz
```


这个文件应该比未压缩的版本占用较少的空间，因为 **gzip** 压缩了它的大小。现在这个文件是不可读的，如果用 **cat** 命令查看它将会看到一些不同寻常的字符(不要这样做，因为这有可能在终端窗口中产生问题)。

(8) 使用 **gzip** 或 **gunzip** 解压文件：

```
gzip -d test-file.gz
```

或

```
gunzip -d test-file.gz
```

(9) 现在/tmp 目录中有一个名为 **test-file** 的文件(**test-file.gz** 文件已不存在)。使用 **cat** 命令查看该文件的内容：

```
cat test-file
```

(10) 看到的输出应该和压缩该文件前使用 **cat** 命令看到的一样。再次使用 **ls -l** 命令，将看到这个文件和原来的文件大小相同。

工作原理

text-file 文本文件由 **gzip** 命令成功压缩。使用 **gunzip** 或 **gzip**，可以成功解压缩这个文件，使它看起来好像以前从来没有被修改或压缩过一样。

由 **gzip** 工具压缩的文件通常都以 **.gz**、**.z** 或者 **.Z** 扩展名结束，所以如果遇到带有其中任何一个扩展名的文件，这个文件很可能就是由 **gzip** 压缩的。

2. bzip2

bzip2 命令提供和 **gzip** 命令一样的功能，但在压缩能力上有所改善。**bzip2** 类型的压缩文件没有 **gzip** 类型的压缩文件使用得广泛，但将会时不时地遇到它，或者您可能更希望使用它。通常，由 **bzip2** 压缩的文件以 **.bz**、**.bz2**、**.bzip2**、**.tbz** 或 **.tbz2** 扩展名作为结尾。如果遇到带有其中任何一个扩展名的文件，那么这个文件很可能是使用 **bzip2** 压缩的。

下面是使用 **bzip2** 进行简单的文件压缩的语法：

```
bzip2 filename
```

为了通过压缩使一个名为 **mytextfile** 的标准文本文件占用更少的空间，可以执行命令：

```
bzip2 mytextfile
```

ls 命令将显示在目录中有一个名为 **mytextfile.bz2** 的新文件。

和 **gzip** 命令一样，可以使用 1 到 9 之间的任意数字指出偏爱的速度或压缩程度。如果正在压缩一个很大的文件并非常关心速度，那么可以添加 **-1** 选项使 **bzip2** 以更快的速度压缩这个文件：

```
bzip2 -1 mytextfile
```

如果更关心节约空间，可使用 **-9** 选项进行最大限度的压缩：

```
bzip2 -9 mytextfile
```

bzip2 的默认压缩状态是 9，这是为了获得最大限度的压缩结果。

为了把一个文件恢复成原来的未压缩状态，应该使用 `bunzip2` 或者 `bzip2 -d`：

```
bunzip2 mytextfile.bz2
```

或

```
bzip2 -d mytextfile.bz2
```

这将创建一个名为 `mytextfile` 的新文件，该文件应该是刚才创建的那个普通文件的未压缩版本。关于 `bzip2` 需要记住的一点是，默认情况下它不会覆盖已经存在的文件。如果希望 `bzip2` 在解压缩的时候覆盖文件，需要使用 `-f` 选项。

`bzip2` 命令有一个称为 `bzip2recover` 的相关命令，该命令将试图修复一个损坏的 `bzip2` 归档文件。关于该命令的更多信息和其他功能，请查阅 `bzip2` 的联机帮助文档。

3. 同时使用 tar 和 gzip 或 bzip2

`gzip` 或 `bzip2` 和 `tar` 命令可以很好地相互补充，而且，事实上这两个命令经常都通过命令整合一起工作。为了在同一个命令中创建一个 `tarfile` 并压缩它，使用下面的命令通过 `gzip` 筛选目标文件：

```
tar -cvzf tarfile_name_or_tape_device directory_name_to_tar_and_compress
```

如果希望使用 `bzip2` 进行压缩，可以使用如下命令：

```
tar -cvjf tarfile_name_or_tape_device directory-to-tar-and-compress
```

习惯上，给由 `tar` 命令和 `gzip` 命令创建的文件添加 `.tar.gz` 或 `.tgz` 扩展名，而给由 `tar` 命令和 `bzip2` 命令创建的文件添加 `.bz` 或 `.tbz` 扩展名，所以如果使用前面为 `tar` 命令示例创建的文件，可以使用下面的命令来归档和压缩这个文件：

```
tar -cvzf /backups/etc-backup-122004.tar.gz /etc
```

或

```
tar -cvf /backups/etc-backup-122004.tgz /etc
```

使用下面的命令来创建一个由 `tar` 归档、由 `bzip2` 压缩的归档文件：

```
tar -cvjf /backups/etc-backup-122004.tar.bz /etc
```

可以通过单独的命令解压缩和解归档(`untar`)文件，先执行解压命令。例如，为了解归档和解压缩 `/backups/etc-backup-122004.tgz` 文件，首先使用

```
gunzip /backups/etc-backup-122004.tgz
```

这将产生一个名为 `/backups/etc-backup-122004.tar` 的文件。然后解归档这个文件：

```
tar -xvf /backups/etc-backup-122004.tar
```

这样就可以解归档前面创建的 `/etc` 目录的归档文件。也可以用一条命令完成同样的工作：

```
tar -xvzf /backups/etc-backup -122004.tgz
```

-xvzf 选项将采取前面的两个步骤并把它们合并成一个命令。结果将把前面归档的/etc 目录的文档放到当前目录的新子目录(称为 etc)中。用 j 替换 z, 就可以使用相同的步骤来处理由 bzip2 压缩的文件。

18.2.3 cpio

cpio(copy in/out)命令用起来比 tar 命令困难得多, 但它提供了一些特殊的功能, 例如保存特殊的文件。cpio 命令从其他命令获取输入的文件(需要备份的文件), 这使得创建自动化脚本很容易。这个命令只存在于某些版本的 Unix 上, 而且学习它比学习其他方法困难得多, 所以在使用它进行备份之前一定要想好是否愿意阅读联机帮助文档。cpio 有 3 个主要选项, 如表 18-2 所示。

表 18-2

选 项	说 明
-i	从标准输入提取文件
-o	读取标准输入以获取需要备份的一组路径名称(运行重定向到标准输出)
-p	从标准输入读取文件路径名, 在后面指定的目录中保存副本

下面是创建当前目录简单备份的示例:

```
find . -print | cpio -ov >cpio_archive
```

可以通过如下命令(以及很多其他命令组合)获得同样的结果:

```
ls * | cpio -ov >cpio_archive
```

如果把 ls 作为 ls -l 命令的别名, 这个命令将运行失败。

这将收集当前目录中的所有文件(find .)并打印它们。find 命令的结果通过管道(|)重定向到 cpio 命令。cpio 于是读取 find 命令的输出(-o)并列出这些文件名(-v)。cpio 命令的输出然后被重定向到一个名为 cpio_archive. cpio_archive 的文件中, 对于磁带设备或者软盘设备, 可以分别使用设备名/dev/rmt0 或者/dev/fd0 来替换这个文件名(这些设备名只是示例, 您的设备名可能与此不同)。

例如, 假设有一个名为/tmp/cpio-directory 的目录, 该目录中包含了 cpio1-test-file、cpio2-test-file 和 cpio3-test-file, 并希望创建/tmp/cpio-directory 目录的一个备份到一个名为 cpio-archive-file 的文件中, 可以执行下面的命令, 先用 cd 切换到/tmp/cpio-directory 目录, 创建一个 cpio 归档文件, 然后用 ls 命令查看该目录的内容:

```
#cd /tmp/cpio-directory
#find . -print  cpio -ov >cpio-archive-file
cpio: .: truncating inode number
.
cpio: cpio1-test-file: truncating inode number
./cpio1-test-file
cpio: cpio2-test-file: truncating inode number
./cpio2-test-file
```

```

cpio: cpio3-test-file: truncating inode number
./cpio3-test-file
3 blocks
#ls
cpio1-test-file cpio2-test-file cpio3-test-file cpio-archive-file

```

这将创建一个名为 `cpio-archive-file` 的新文件，正如 `ls` 命令的输出显示的那样。可以使用设备名来替换文件名 `cpio-archive-file`，这样该文件将被直接写入设备。

为了查看 `cpio` 归档文件的内容，需要使用 `-itv` 选项并把这个归档文件重定向到 `cpio` 命令。例如：

```

#cpio -itv <cpio-archive-file
drwxr-xr-x  2 user user      0 Dec 20 22:44 .
-rw-r--r--  1 user user    37 Dec 20 22:38 cpio1-test-file
-rw-r--r--  1 user user    37 Dec 20 22:38 cpio2-test-file
-rw-r--r--  1 user user    37 Dec 20 22:38 cpio3-test-file
3 blocks
#

```

这个命令显示刚才创建的 `cpio-archive-file` 文件的内容，正如命令之后的输出显示的那样。如果这个文件在磁带设备上，可以使用命令：

```
cpio -itv </dev/rmt0
```

如果已经把文件保存到这个磁带设备上，将得到同样的结果。

为了从 `cpio` 归档文件中检索文件，这个示例在另一个目录 `/tmp/cpio-directory2`，中执行如下命令：

```

#cd /tmp/cpio-directory2
#cpio -i </tmp/cpio-directory/cpio-archive-file

3 blocks
#

```

如果此时执行 `ls` 命令，将看到这个目录包含了前面在 `cpio` 中归档的 3 个文件(`cpio1-test-file`、`cpio2-test-file` 和 `cpio3-test-file`)。

使用 `cpio` 进行文件备份的选项还有很多。关于这个多功能、高灵活的命令的更多信息，请参考联机帮助文档。

18.2.4 dump、backup 和 restore

`dump` 命令，从其是为以一种系统的、连续的方式备份整个文件系统而设计的意义上来说，是一个真正的备份工具。`dump` 命令有两个组成部分：根据用户的每条指令实际执行备份的 `backup` 命令，和恢复已被备份的文件的 `restore` 命令。在有些系统上这两个命令称为 `ufsdump` 和 `ufsrestore`，例如 Sun 公司的 Solaris 系统，但这两组命令在实现上的差别很小。使用 `dump` 而不是其他命令行备份工具的好处在于 `dump` 可以：

- 识别文件上一次备份的时间。
- 标记已经备份的文件。

- 在一个磁带设备上备份多种文件系统(或者在多个磁带上备份文件系统)。
- 很容易地备份远程系统。

另外，dump 命令的恢复过程相当直观和简单。

使用 dump 命令的第一步是要理解它的可用的命令行选项，如表 18-3 所示。

表 18-3

选 项	说 明
-0 到 -9	转储(dump)的级别。级别 0 表示将所有文件都复制到备份设备中。其他数字表示增量备份，这种备份将复制自从上一次使用更低的级别进行备份之后所有新的或者修改过的文件
-a	不考虑磁带长度的计算并一直写入数据直到收到 end-of-tape 消息(用于向磁带中添加数据)
-A	创建一个表格，该表格显示紧接在-A 之后的文件转储的内容。这个选项将在后面和 restore 一起使用
-f output_name	将备份写入 output_name 文件(包括磁带设备，例如/dev/rmt0，普通文件，或者甚至是远程系统，例如 linux2:/dev/rmt0)。例如，可以在 dump 命令中使用 -f /var/my_backup_050105 选项将备份写入一个名为 my_backup_050105 的文件。为了在本地屏幕上查看备份，可以在 f 选项后面添加 a-表示标准输出(检查问题时很有用)
-F script_name	在备份完成之后运行 script_name 脚本。这对数据库备份很有用，因为备份完成之后应该重新连接到数据库
-j option	使用紧接在选项 1-9 后面指出的压缩级别。这里使用 bzip2 进行压缩
-L lable_name	使用 lable_name 标记由 restore 读取的备份
-q	当需要操作者干预时立即退出
-w	显示需要转储的所有文件系统的列表
-W	显示所有转储文件中已被备份的文件系统的列表(从/etc/dumpdates 文件派生而来)
-z options	类似于-j 选项，除了在压缩时使用的是 gzip

正确的数据备份非常关键，所以 dump 在遇到错误时非常谨慎，通常在出现问题时都需要用户干预。

通常必须以根用户或者具有备份权限的用户身份来进行备份。

在第一次使用 dump 命令的时候，可以通过添加 0 开关来运行 dump 以确保一次备份了所有的内容，例如：

```
/sbin/dump -0u -f /dev/st0 /home
```

这个命令备份/home 目录中的所有内容并将备份发送到设备/dev/st0(通常是 Linux 中的磁带设备)。为了将/home 目录备份到一个名为/var/backup-122204 的文件中，可以运行：

```
/sbin/dump -0u -f /var/backup-122204 /home
```

在开始执行该命令之后，将看到与图 18-1 显示的内容类似的输出。

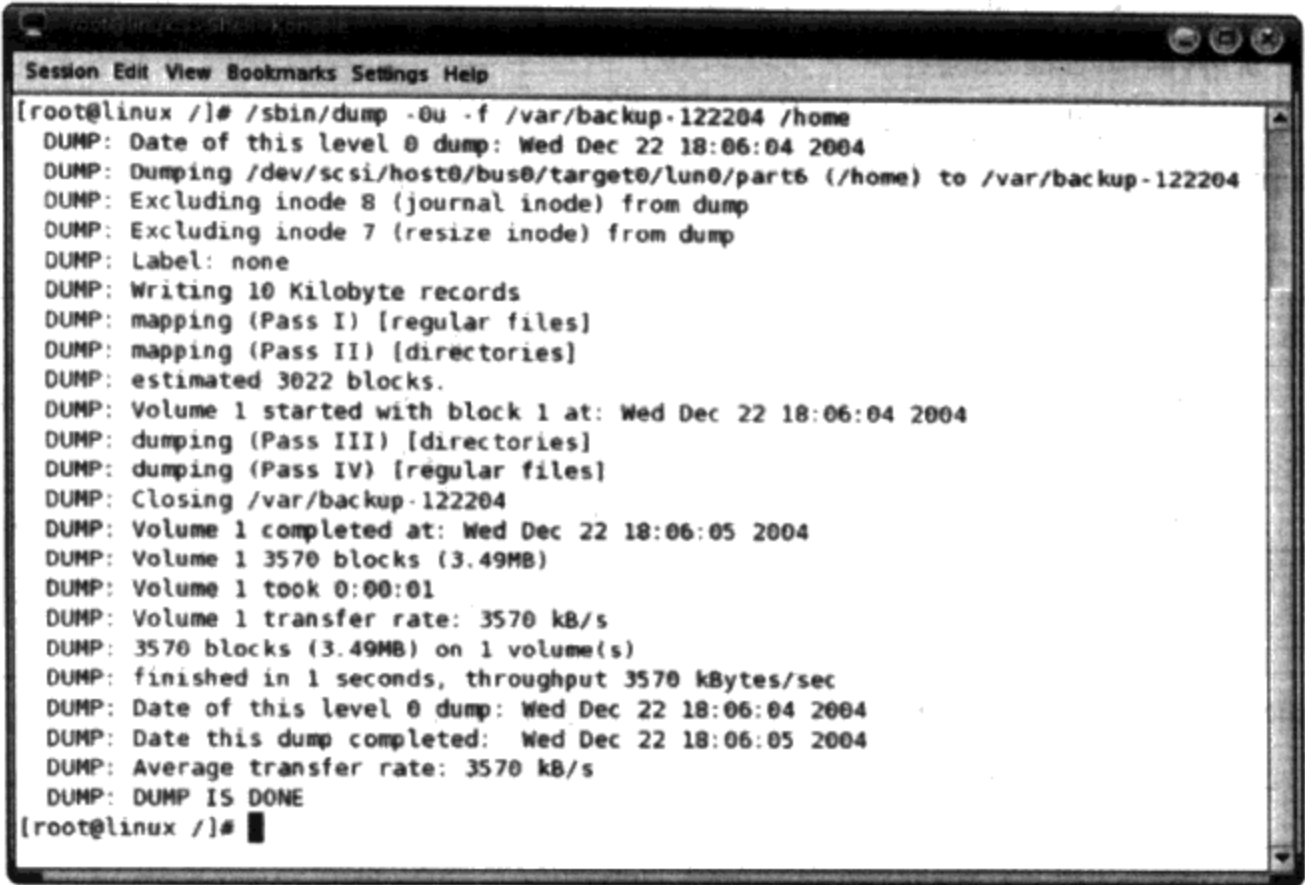


图 18-1

现在，可以运行 `dump -W` 命令查看文件系统备份的状况，如图 18-2 所示。

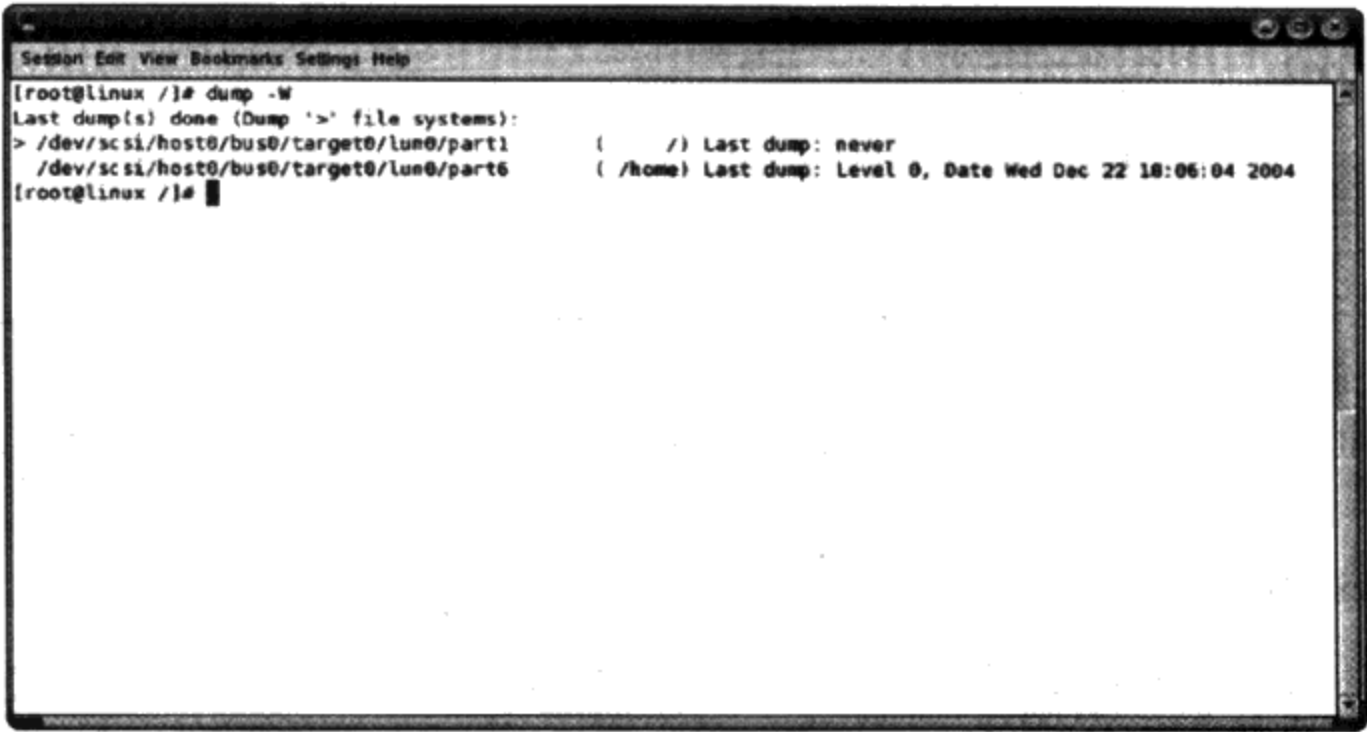


图 18-2

如果希望查看需要对哪些文件系统进行备份(如图 18-3 所示)，这些文件系统则由 `dump` 命令决定，执行 `dump -w`(小写的“w”)。

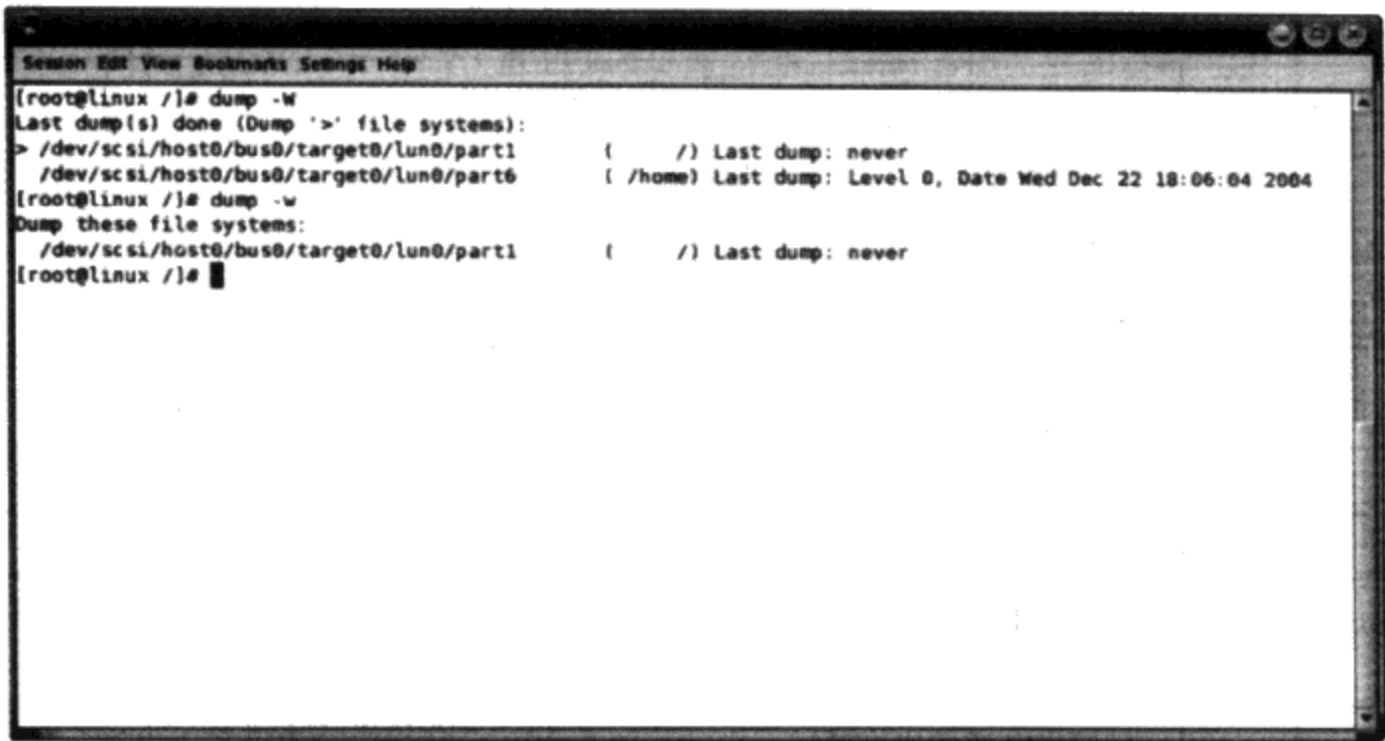


图 18-3

关于通过脚本编程进行备份的更多信息请参考联机帮助文档，以确保找到高效且正确地备份文件系统所需的选项。

在备份之后，需要恢复数据以验证备份，或者为一次意外的删除或系统故障恢复数据。人们将从转储文件中恢复数据的命令恰当地命名为 restore。表 18-4 显示了 restore 命令的一些命令行选项。

表 18-4

选 项	说 明
-f	指出从何处(文件或者设备)恢复数据
-F script	在备份开始的地方运行指定的脚本
-h	恢复实际的目录，而不是所指的文件
-i	以交互的模式开始恢复数据
-M	激活多卷备份(multivolume)功能(当为转储文件使用-M 时)。和-f 选项一起使用时，它将使用 filename001、filename002、filename003 等文件名，以此类推
-N	仅仅打印文件名，不恢复实际的文件
-r	恢复整个磁带或文件系统(必须小心使用)
-v	显示详细的输出
-V	读取多卷备份介质而不是磁带(例如 CR-ROM)
-x	恢复特别指定的文件或目录
-X filestring	从恢复的数据中提取 filestring 中列出来的文件
-y	如果有错误，不中断，继续恢复

restore 命令的交互模式(由-i 选项初始化)有它自己的提示符(通常是 RESTORE>)。在交互模式中，可以使用表 18-5 中描述的选项。

表 18-5

选 项	说 明
add directory_or_file	如果单独使用，它将恢复当前目录。如果添加了目录或文件名，将添加到恢复目录中
cd directory	切换到指定的目录
delete directory_or_file	如果没有参数，将从恢复列表中删除当前目录。如果指定了一个目录或文件名，将不会恢复指定的目录或文件
extract	提取所有选择需要提取的文件(恢复已启动)
help	显示所有可用的命令
ls directory	如果不指定文件或目录，将显示当前目录中的文件列表。如果指定了文件或目录，则显示这个文件或目录。需要恢复的文件旁边有一个*
pwd	显示当前的工作目录
quit	立即退出恢复交互对话
setmodes	设置文件的所有者、模式和时间
verbose	显示扩展信息

要恢复数据，使用下面的命令(使用刚才由 `dump` 命令创建的备份文件)：

```
restore -if /var/backup-122204
```

图 18-4 显示了 `restore` 在交互模式下的一些命令和这些命令产生的输出。

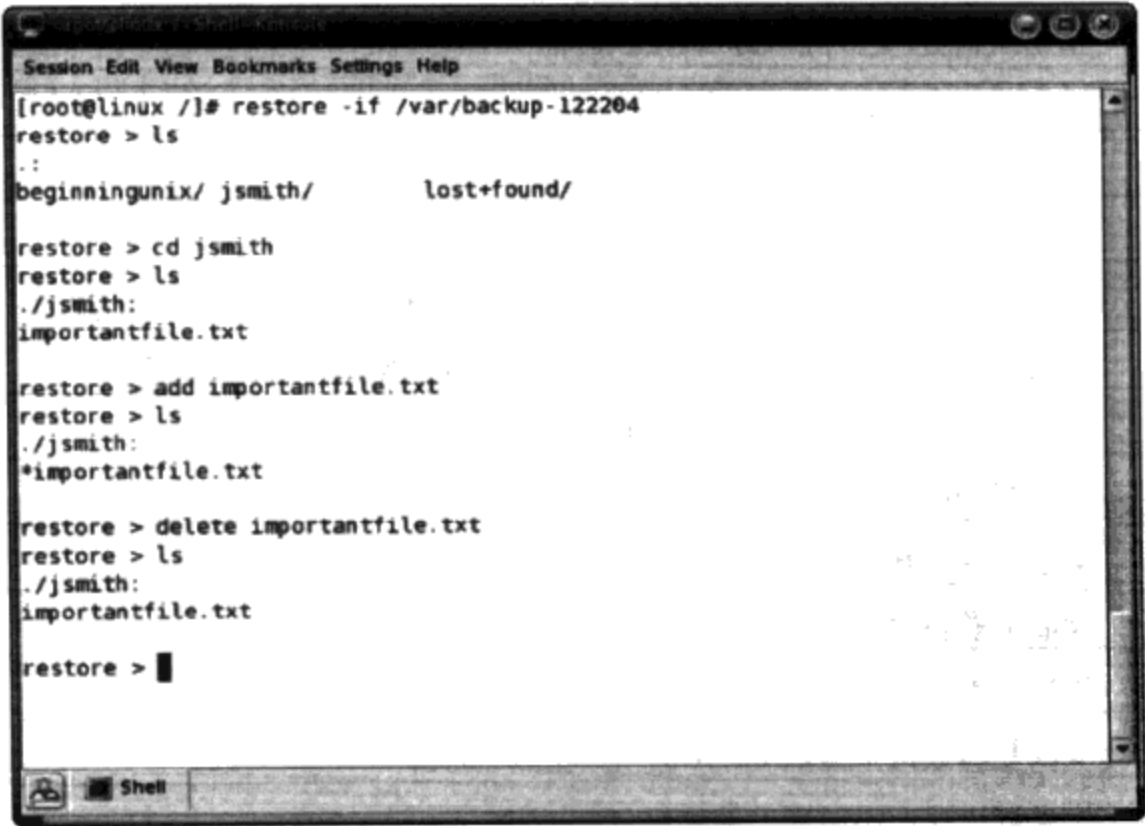


图 18-4

在交互模式的数据恢复中，可以使用 `cd` 命令在目录之间切换，使用 `add` 命令添加目录或文件，然后使用 `extract` 实际恢复文件。

恢复数据的命令相当简单和直观。dump 和 restore 也可以和 mt 命令一起用于磁带操作。关于这个特定主题的更多信息，请参考 mt 命令的联机帮助文档。

18.2.5 其他备份命令

Unix 传统上是可以使用多种方法完成同一件事情。如果前面讨论的命令不能使用，Unix 还有其他一些值得研究的备份命令。其中一些是：

- pax(portable archive exchange)与 tar 和 cpio 类似，但提供了一些其他功能。更多信息请访问 <http://directory.fsf.org/GNU/paxutils.html>。
- rsync——用于高速的远程文件同步。更多信息请访问 <http://samba.anu.edu.au/rsync/>。
- rdist——用于文件和备份的分布式副本。更多信息请访问 <http://www.magnicomp.com/rdist/>。

18.3 备份套件

对于生产或商业环境，可能决定需要更多的功能、易用性和对整个备份方案的技术支持。一些公司为 Unix 操作系统提供了很多非常优秀的软件应用程序，包括：

- ARCserver——www3.ca.com/Solutions/ProductFamily.asp?ID=115
- Arkia——www.arkeia.com
- IBM Tivoli Storage Manager——www-306.ibm.com/software/tivoli/products/storage_mgr
- LoneTar——www.lone-tar.com
- Veritas NetBackup——www.veritas.com/Products/www?c=product&refId=2

下面的免费备份套件与其他命令行界面备份套件相比提供了更好的易用性：

- Amanda——www.amanda.org
- Bacula——www.bacula.org
- Kbackup——<http://kbackup.sourceforge.net>

18.4 小结

本章中，学习了备份数据的基础知识，包括建立备份介质的存储的备份策略。还学习了备份数据的主要命令，以及可以用于备份的不同介质类型。

使用 backup 和 restore 命令可以进行简单的备份。gzip 和 bzip2 命令可以压缩文件以节约备份介质的空间。您还学习了怎样使用 cpio 和 tar 命令从大量的文件中创建简单的、便于携带的文件。既然已经具备了备份 Unix 系统的工具，就应该通过备份系统来使用这些技术。

18.5 练习

演示怎样使用 tar 和 gzip 命令压缩/etc(整个目录)目录中的配置文件并把这个文件命名为/tmp/etc_backup。

第 19 章 从源代码安装软件

虽然 Unix 系统预装了很多软件，或者可以从系统安装包中安装这些软件，但是 Unix 的一个好处是可以从源代码编译并安装软件。实际上，绝大多数 Unix 系统的软件都是以源代码的形式提供的，而且只有源代码，所以必须知道怎样处理它。

本章讨论查找和检索软件、使用通用的编译工具、编译和安装软件以及查找和解决问题。另外还讨论 make(GNU 编译系统)、GCC 和管理包的基础知识。为了检验对这些内容的掌握情况，您将从源代码安装几个程序。

为了从源代码编译软件，Mac OS X 系统的用户必须在他们的系统上安装 Xcode Tool。如果计算机在出厂前已安装了 Mac OS X 10.3 Panther，那么在/Applications/Installers/Developer Tools/目录可以找到一个安装程序。如果在系统上找不到这个安装程序，可以从 Apple 公司的网站上下载这个软件。与这个软件有关的信息，包括下载说明，可以在 <http://developer.apple.com.com/tools/downloads/> 获得。为了从 Apple 公司的网站上下载 Xcode Tool，您需要在 Apple Developer Connection 注册一个免费的账号。Xcode Tool 包括 make、GCC 和其他本章中提到的工具和实用程序。Xcode Tool 目前的版本是 1.5。

19.1 源代码

源代码是创建软件的原始数据。可以用 C、C++ 这样的编程语言来编写源代码，在使用这些代码之前需要进行编译，或者使用 Perl 或 Python 这样的脚本语言编写源代码，这些代码不需要编译即可运行。绝大多数情况下，源代码都提供编译说明和/或编译工具，它们有助于编译和安装最后的产品：系统立即可用的软件。已编译好的软件和机器可执行格式的软件不是源代码。

由于 Unix 平台的种类实在是太多，而且为每种平台提供立即可用的二进制程序(可执行程序)非常耗费时间，所以 Unix 的开发人员通常只以源代码下载包的形式提供他们的软件。一般情况下，他们的软件具有足够好的移植性，使您可以在自己选择的 Unix 系统上编译它。通过获得源代码，可以为自己的系统编译程序，而且可以定制它以满足自己的特殊需要，这取决于您的能力。

下载源代码之后，需要查看与您的特定 Unix 版本相关的说明或者 README 文件。可能需要下载其他的包或文档才能使软件在系统上正确地运行。

下载包通常提供由 C、C++、Perl、Python 或其他编程语言编写的源代码和文档，其中包括编译说明、帮助编译和安装软件的脚本和实用工具、数据文件、配置示例和各种其他文件。有时候下载包仅仅包含可立即安装(也可能是立即可执行的)的脚本而没有任何说明。

在 Unix 分销商或供应商提供更新的预编译发布之前，开发者通常以源代码的形式提供安

全漏洞修补(security fix)或其他必需的改进。想象在中病毒的紧急情况下,如果惟一可用的软件补丁是源代码而不是符合您的 Unix 版本的预编译二进制形式。为了安装补丁,必须知道怎样使用代码。即使绝大多数时候都可以使用预编译好的软件补丁,知道怎样编译代码也是很有用的,以便在紧急时刻更新系统。

当然,由开发人员(或软件供应商)直接提供的软件比操作系统的版本要新得多。从代码安装是试用最新软件的最新功能的好方法,不过这也意味着软件的测试非常少,可能有很多 bug——下载人必须小心!

19.2 开放源代码许可证

以源代码形式提供的软件通常称为开放源代码,但这并不是完整的定义。开放源代码软件是带有特定许可条款的软件。通常可以自由地查看、共享和使用它,(最重要的是)还可以自由地修改它。某些开放源代码软件或额外的附加品,例如 CD 或者服务,可能需要出钱购买。

开放源代码软件许可证声明对源代码拥有版权,并确定代码的使用和发布、以及派生代码的使用和发布的自由范围和限制。软件的作者通常会创建自己的许可条款,但是如果重用了其他开发人员的代码,则必须尊重已经存在的许可证。

理解开放源代码许可证的基本内容——特别是从商业的角度——是必须的,这样就可以判断是否可以使用或者发布某个软件。如果计划参加一个现存的开放源代码软件项目,或者创建自己的项目,该项目将以典型的开放源代码许可证发布,那么必须熟悉适用于正在使用的代码的许可条款。下面介绍两种主要的开放源代码许可证:BSD 和 GPL。

19.2.1 BSD 许可证

可以自由地修改持有 BSD 式许可证的源代码,而且不需要重新发布这些修改。有些持有 BSD 式许可证的源代码要求在对程序做广告时,如果这个程序包含了 BSD 代码,则必须在广告中表明这一点。持有 BSD 式许可证的源代码经常用于私有的或商业的软件中,例如 Microsoft 公司的 Windows 系统和 Apple 公司的 Mac OS X 系统。

TCP Wrapper 软件是由 Wietse Venema 开发的,这个软件非常流行,在该软件的源代码中有一个简单的 BSD 许可证的例子:

```
Copyright 1995 by Wietse Venema. All rights reserved. Some individual files may be
covered by other copyrights.
```

```
This material was originally written and compiled by Wietse Venema at Eindhoven
University of Technology, The Netherlands, in 1990, 1991, 1992, 1993, 1994 and 1995.
```

```
Redistribution and use in source and binary forms, with or without modification,
are permitted provided that this entire copyright notice is duplicated in all such
copies.
```

```
This software is provided "as is" and without any expressed or implied warranties,
including, without limitation, the implied warranties of merchantability and
fitness for any particular purpose.
```

X11 和 MIT 许可证与 BSD 许可证类似,很多 X Window 系统的开放源代码软件都使用这种许可证,同时也是流行的 Apache Web 服务器使用的最早的许可证。

19.2.2 GNU 公共许可证

GNU 通用公共许可证(GNU General Public License, GPL)也很常用。人们也把它称为 copyleft 许可证,因为它要求对已发布的软件所作的任何修改都必须以相同的许可证发布。GNU 工程(GNU Project)和自由软件基金会(Free Software Foundation)对 GPL 下发布的软件不使用术语 open source(开放源代码),而更偏向于使用术语 free software(自由软件),它们都是 GPL 的支持者。

如果修改具有 GPL 的代码,要注意在该代码上所做的工作可能将受到 GPL 条款的控制。也就是说,GPL 决定了可以怎样使用从持有 GPL 的代码派生出来的代码,而这通常都意味着所做的工作必须在相同的 GPL 条款下发布。对于那些认为最好的软件就是开放使用和开发的人来说,GPL 是一个非常棒的许可证。如果不愿意发布源代码让别人修改和利用,最好的办法是从零开始编写代码,或者至少避免使用任何带有 GNU 公共许可证的代码。

常用的开源许可证有 30 多种。更多关于各种许可证的信息和示例请通过网址 www.opensource.org 访问开放源代码促进会 (Open Source Initiative) 的网站和网址 www.gnu.org/licenses/license-list.html 访问自由软件基金的评论。

如果使用了某份代码的一部分,一定要在最终形成的新代码中明确地记录对这份代码的使用情况并包含这份代码的许可证副本。在某些情况下,使用具有许可证的其他代码可能会淹没您的代码,因为这个许可证将凌驾于您的所有代码之上(是的,的确是这样,在某些情况下,列出来的许可证可能比实际的代码还长)。

19.3 寻找和下载 Unix 软件

为 Unix 系统寻找新软件的方法有好几种。最简单的方法是使用特定 Unix 版本的内置功能。例如,很多 Linux 版本提供了一些工具,这些工具为已安装的软件自动搜索升级版本或补丁(最让人灰心的方法是使用最喜欢的搜索引擎在网络上搜索。虽然这偶尔也会有所发现,但即使是最好的搜索引擎通常也只能获得毫无意义的结果,例如找到不安全的或者旧的版本)。

最有可能找到的新的、让人激动的 Unix 软件是 Web 上的软件文档。这些站点的内容非常广泛,从包含 Windows 和 Mac 程序的一般文档到仅仅包含在特定软件许可证下发布的程序。下面是一些最流行和可靠的文档,它们包含了很多 Unix 程序:

- Freshmeat(www.freshmeat.net)—— 最新的(cutting-edge)发布。
- Linuxberg(www.linuxberg.com)—— Tucows 文档的一部分。
- Tucows(www.tucows.com)—— 多种 Unix 风格,还有源代码。
- iBiblio(www.ibiblio.org)—— 所有主题的数字简档,包括软件。
- 自由软件基金(www.fsf.org)—— 在 GPL 下发布的软件。
- Linux Software Map(www.boutell.com/lsm)—— 植入 Linux 的软件。

19.3.1 选择软件

开发人员通常会为相同的程序提供多个下载包:最新的正式版本、在多种平台上稳定运行的以前的版本、以及几个主要用于测试的开发版本。怎样选择呢?

测试版本通常称为 alpha、beta、版本候选(release candidate)或每日快照(daily snapshot)，使用它们时必须小心。根据定义，测试软件是潜在不稳定的。

通常，应该下载正式版本，一般以“LATEST”来标记它。在安装该版本并正确运行之后，可能会发现下载某个开发版本——例如多个 bug 补丁或新的功能——是非常好的选择。重要的是要意识到开发版本通常要求必要的软件开发版本。其他软件可能与新的测试代码不兼容，这些软件依赖于正在安装的软件。安装新软件通常会产生一系列的影响，可能需要进行全面的升级。

在某些情况下，虽然开放源代码项目的实际供应商或正式开发人员不提供修补(补丁)或升级版本，但还是可以获得该项目的安全补丁或重要的 bug 修补。实际上，正式的 Web 站点通常甚至都不会提到有安全问题存在。根据已往的经验，最好快速搜索一下安装的软件的安全问题。某些技术新闻网站，例如 Slashdot(www.slashdot.org)，可以给您指明出路。

19.3.2 下载文件

在选择好所需的包之后，就该下载源代码文件了。下载的内容通常由 tar、gzip 或 bzip 命令压缩过。形成的文件称为 tarball，其典型的文件扩展名是 .tgz、.tar.gz 或 .tar.bz2。有些源代码下载包只能作为 shell 文档、旧式压缩的 tar 文件(扩展名为 .Z)或 ZIP 文件。Unix 系统应该可以把 tarball 展开成这些格式中的任意一种，所以选择哪种格式并不重要。

1. 检查文件的大小

软件文档通常会说明下载文件的大小。开始下载文件之后也可以查看它的大小，如果文件对硬盘过大可以中断下载。不过，请记住，这是一个压缩文件，它可能包含了大量的文件夹和单独的文件。查看 README 文件、程序自己的 Web 站点或其他文档可以获知解压后和安装后的软件的实际大小。安装一个没有足够磁盘空间来容纳的程序将导致非常危险的崩溃。同时还需要足够的内存来处理所有的解压和安装过程。

遗憾的是，要知道多大的空间才算足够大几乎是不可能的。下载和安装一个很小的命令行工具与安装一个全新的 shell 或一个强大的程序，例如 Apache，所需的资源是完全不一样的。另外，所选择的功能也会影响到需要的磁盘空间。例如，提取 Apache 2(下载包 6MB)需要大于 22MB 的空间，编译它需要差不多 60MB 的空间，安装它另外需要 20MB 的空间。所有这些问题都归结为“看情况”和“调查一下”。幸运的是，磁盘空间现在很便宜。如果怀疑磁盘不够用，就升级！

2. 下载技巧

可以使用 Web 浏览器或 FTP 客户端，例如 nsftp 或 wget，来下载文件。一定要把下载客户端设置成二进制下载模式，以便下载完成后可以执行 tarball。如果以 ASCII 模式下载，文件将以文本格式下载而且不可用(大多数浏览器和 FTP 客户端会自动探测文件的类型并相应地设置下载模式)。记录下存放下载文件的位置以免事后还要查找它。

通常用于存放下载到的源代码 tarball 的位置有：/usr/src，存放系统源代码；/usr/local/src，存放第三方软件；~/src，存放只能由您使用的软件。其他可用的目录包括 \${HOME}/src 或 \${HOME}/downloads。无论选择在何处存放 tarball，尽量使用相同的目录以便控制这些文件并在安装了包之后删除它们。

下面将实际下载一个文件。在这个例子中, curl 程序用于从 MaraDNS 工程(www.madns.org)中检索一个特定的 tarball。在这个工程的网页上直接点击 Download 链接并选择 Current Stable Release 选项会更简单, 但是使用 FTP 程序能更好地说明这个过程。

在命令行调用 curl 程序并指定保存包的目录和需要下载的包:

```
curl -o maradns-1.0.23.tar.bz2 -v http://www.madns.org/download/madns-1.0.23.tar.bz2
* About to connect() to www.madns.org:80
* Connected to www.madns.org (66.33.48.187) port 80
> GET /download/madns-1.0.23.tar.bz2 HTTP/1.1
User-Agent: curl/7.10.4 (i386-netbsdelf) libcurl/7.10.4 OpenSSL/0.9.6g ipv6
zlib/1.1.4
Host: www.madns.org
Pragma: no-cache
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*

% Total    % Received % Xferd Average Speed          Time      Curr.
          Dload Upload Total   Current Left   Speed
100  423k  100  423k    0    0  70000  0          0:00:06 0:00:06 0:00:00  76326
* Connection #0 left intact
* Closing connection #0
```

工作原理

-o(或--output)选项指定 curl 保存下载包的文件名, 而-v(或--verbose)选项显示一些调试信息。也可以使用-O(或--remote-name)选项保存一些按键。该文件将以和源站点上相同的文件名保存。

```
curl -O -v http://www.madns.org/download/madns-1.0.23.tar.bz2
```

下面的命令下载同样的文件, 但是使用 wget:

```
wget http://www.madns.org/download/madns-1.0.23.tar.bz2
--16:07:10-- http://www.madns.org/download/madns-1.0.23.tar.bz2
=> 'madns-1.0.23.tar.bz2.1'
Resolving www.madns.org... done.
Connecting to www.madns.org[66.33.48.187]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 433,164 [application/x-bzip2]

100%[=====>] 433,164  77.43K/s  ETA 00:00
16:07:15 (77.43 KB/s) - 'madns-1.0-23.tar.bz2.1' saved [433164/433164]
```

有趣的是, wget 可以为 FTP 和 HTTP 续传已中断的下载包。它有一些非常有趣而且有用的功能, 可以在 www.gnu.org/software/wget/ 上找到。

19.3.3 验证源代码

代码验证在下载和安装的过程中是一个可选的步骤。通常, 开发人员提供签名、消息摘要

或校验和，可以利用这些数据来验证已下载的软件完整性。这个过程将验证已正确地下载了软件，而且这个软件确实是想下载的那个软件。一般情况下，只要服务器上提供了文件，下载工具将准确地完成下载文件的工作，并防止任何形式的数据损坏。验证软件包的主要原因是担心正在下载的内容可能是一个已被恶意修改的软件包，包内也许包含着特洛伊木马或其他类型的病毒。

会发生这样的事情吗？是的。一些有问题的包可以由验证系统自动识别，例如 Gentoo Linux 版本就带有这种验证系统。这样的例子包括先前记录下来的校验和(使用 MD5 算法)后来在编译源代码的系统中用于检验下载的包。2002 年 10 月，有一个用于 Sendmail 的下载服务器中了特洛伊木马病毒。2002 年 11 月，一个流行的下载镜像提供了多个包含特洛伊木马的 tcpdump 和 libpcap 副本。在过去的十年中，其他受损程序通过很多不知名的软件文档四处流窜。

NetBSD、OpenBSD 和 FreeBSD 也提供自动化的源代码编译系统，这种系统将根据先前产生的摘要检验下载的源代码的完整性。本章稍后将对此简要地讨论。

幸运的是，在安装软件并危及系统之前，可以使用验证工具查看是否下载了受损的软件。最简单的测试是使用校验和，它是一个数字标签，表示下载包中应该包含多少比特。当然，必须根据真正的、原始的下包计算最初的、正确的校验和，而且应该通过独立的源提供给下载方。例如，如果校验和与受损的下包由同一台服务器提供，那么它也有可能受损。关于独立源的例子是电子邮件声明，这种声明为下包列出 MD5 摘要或使用下载服务器以外的服务器。

获取 MD5 摘要只需要对文件使用 digest、openssl、md5 或 md5sum 工具(取决于 Unix 系统)。下面的例子演示了怎样在 NetBSD 下获取校验和：

```
$ md5 httpd-2.0.50.tar.gz
MD5 (httpd-2.0.50.tar.gz) = 8b251767212aebf41a13128bb70c0b41
```

可以将这个校验和与 www.apache.org 站点上提供的 MD5 摘要相比较。

多年以来，人们一直认为 MD5 可以为每个不同的文件提供惟一的摘要。但是在 2004 年，有人发现两个同样大小的不同文件拥有相同的 MD5 摘要，由此找出 MD5 算法的冲突。结果，很多项目都开始采用 SHA1 摘要作为替代。可以通过 digest、openssl、shal 或 shalsum 工具输出 SHA1 摘要：

```
$ openssl shal Mail-SpamAssassin-2.64.tar.bz2
SHA1 (Mail-SpamAssassin-2.64.tar.bz2) = ea4925c6967249a581c4966d1cefdla3162eb639
```

可以将这个校验和与 SpamAssassin.org 站点上提供的 SHA1 摘要相比较。

为了更好地验证下载的源代码的完整性，很多开发者使用数字签名，提供下载包的一个签名验证信息。如果后来修改过下载包，这个签名验证信息就不会匹配。该数字签名由开发人员拥有的密钥创建。公共密钥通常可以在公共密钥服务器上获得，有时候则直接在源代码 tarball 中提供。通常，签名验证信息可以通过 PGP 或 GNU 隐私守卫(Gnu Privacy Guard, GPG)来处理。

实战

获取数字签名

可以使用数字签名验证在前面的“实战”例子中下载的 MaraDNS 包。只有安装了 GPG 软件并设置好之后才能运行这个例子。也可以在下下载 MaraDNS 包的站点下载 GPG 签名。MaraDNS 下载包中包含了公钥(在这个例子中公钥是 `maradns-1.0.23/maradns.pgp.key`)。

- (1) 首先需要从前面下载的 tar 文件中提取 PGP 密钥:

```
bzcat maradns-1.0.23.tar.bz2 | tar xf - maradns-1.0.23/maradns.pgp.key
```

- (2) 为已下载的 bzip2 版本的源代码文件检索 GPG 签名:

```
wget http://www.mardns.org/download/maradns-1.0.23.tar.bz2.asc
```

- (3) 为了把这个密钥添加到自己的数字密钥环中, 使用命令:

```
gpg --import maradns-1.0.23/maradns.pgp.key
```

另外, 也可以使用:

```
gpg -recv-keys 1E61FCA6
```

其中 1E61FCA6 是使用 gpg 标识的密钥 ID。

- (4) 验证下载文件:

```
gpg -verify maradns-1.0.23.tar.bz2.asc maradns-1.0.23.tar.bz2
```

.asc 文件是从下载 MaraDNS 包的网页上下载的数字签名。

工作原理

gpg 工具根据公共加密密钥检验已下载的文件, 这个公共加密密钥包含在已下载的原始软件包 tarball 文件中的。如果它们是相同的, 这个软件将通过验证, 而且很可能没有遭到什么损坏。如果它们不相同, 那么软件可能已经遭到损坏, 更糟糕的是, 该软件可能是恶意的代码。与其后悔还不如采取安全的措施, 所以不要打开或安装这个软件, 立即联系项目开发人员。

更详细的信息请参考 GnuPG 文档或 PGP 文档。

19.4 编译和安装

编译和安装软件的通常步骤是:

- (1) 把源代码文件提取到目录中。
- (2) 切换到源代码目录。
- (3) 配置系统的编译环境。
- (4) 运行 make 编译软件。
- (5) 安装软件(可能需要根用户身份)。

假设已经下载了软件, 就可以开始第(1)步, 把源代码文件提取到目录中。

检索软件下载包和从源代码编译软件应该以普通的、非根用户的身份来进行。如果有需要, 安装软件可以以根用户的身份来进行。当然, 如果非常信任软件并以根用户的身份来安装和运行它(或者允许其他用户运行它), 那么基本上也可以信任编译步骤。然而, 最好只对那些真正需要提高权限的任务才使用这种权限。另外, 如果关心软件的编译和后续的使用, 可能需要考虑为这些步骤创建一个专用的用户。

常用的编译目录有：`/usr/src`，在这里编译系统软件；`/usr/local/src/`，在这里编译其他软件(有时指第三方软件或附件软件)；`${HOME}/src`，在这里为自己的账号软件。通常将所有的源代码保存在`~/src`目录，并在这个目录中提取 `tarball` 和编译软件。把所有的代码保存在一个集中的地方易于清除，在主目录中保存文件可以提醒我们以普通的、没有特权的用户身份来提取和编译软件。

如果系统有多个管理员，可能希望使用一个集中的地方，管理员之间可以相互配合，或者使用一个单独的地方以免和其他人发生冲突。有些管理员使用专用的用户账号编译软件，这有助于组织、维护和测试。

19.4.1 提取文件

不能直接使用 `tarball` 来安装软件。首先，必须展开压缩文件看看它实际包含的文件。提取 `tar` 源文件最常用的方法是使用如下命令：

```
tar xvzf source-download.tar.gz
```

`tar` 命令和其他 Unix 命令的工作方式稍微有些不同。特别是，`tar` 命令的选项不使用连字符(-)。在这个例子中有 4 个选项：

- `x` —— 标识一个需要提取的压缩文件。
- `v` —— 触发 `verbose` 模式，列出提取的每一个文件。
- `z` —— 解压 `gzip` 格式的文件。
- `f` —— 定义压缩文件是 `source-download.tar.gz`。

如果所使用的 Unix 版本没有 `z` 选项，可以使用下面的命令来解压由 `gzip` 压缩的文件：

```
gunzip -c source-download.tar.gz | tar xvf -
```

有些 `tar` 版本支持 `j` 或 `J` 选项，这两个选项表示使用 `bzip2` 压缩格式。另外，有些 `tar` 版本具有足够的智能，在使用 `z` 选项的时候也可以处理 `bzip2` 格式的压缩文件(相关细节请查看 `tar` 的联机帮助文档)。

大多数情况下，源 `tarball` 文件将在与软件的名称和版本同名的子目录中解压。例如，`tar` 文件 `Mail-SpamAssassin-2.64.tar.bz2` 的所有文件将解压到 `Mail-SpamAssassin-2.64` 子目录中。有些 `ZIP` 或 `tar` 文件可能会把所有的文件存放在归档文件的最顶层，在提取文件时可能把文件放到当前目录中(这很可能会覆盖同名的文件)。考虑到这种情况，在解压之前可以先查看一下文件列表，或者在解压之前把 `tarball` 文件转移到它自己的子目录中。解压之前可以使用 `tar` 命令的 `t`(内容表格)选项查看 `tarball` 文件包含的文件列表。

19.4.2 开始编译

提取源代码之后，切换到新创建的目录(第(2)步)。使用 `ls` 命令列出这个目录的内容。通常都会有 `README` 文件和 `INSTALL` 文件(这两个文件的文件名有时候会稍有不同，或者位于某个子目录中，该子目录很可能是 `docs` 或 `help`)。查阅这两个文件以便找到事先需要安装的程序或库，另外还可以找到关于准备编译、编译软件和安装编译结果的说明。另外在进行简单的配置和开始使用软件的时候这两个文件也有用。

通常 `INSTALL` 文件是一个预先做好的模板，并不是您要安装的软件才有的。如果是这种情况，可以使用和本章描述的标准编译(build)过程一样的过程。如果 `INSTALL` 文件描述了一

个不同的过程，就按照它来，因为一个标准的过程可能不能正确地安装这个软件。

参考软件的网站也是一个好主意。通常在网站上可以找到编译和安装说明，还有其他经常问到的关于编译的问题的答案。

我们的目标是配置编译过程以便它正确地检测编译环境并为操作系统正确地安装软件。有时候，需要手动修改一个 `makefile` 文件以定义怎样进行编译和安装。`makefile` 是一个文件，它指示怎样完成编译。多少有点迷惑的是，通常将 `makefile` 文件命名为 `Makefile`。本章后面将进一步学习 `makefile`。幸运的是，大多数现代软件都使用脚本来帮助完成编译，而不是强迫您手工配置 `makefile`。

1. 使用配置脚本

为了设置好 `makefile`，先在展开的 `tarball` 的顶层目录中找到名为 `configure` 的可执行文件。通过下面的命令运行这个 `configure` 脚本(编译和安装列表中的第(3)步)：

```
./configure --help
```

在命令的前面使用点号和斜线表示这个脚本的实际位置，因为当前的工作目录很可能不是 `PATH` 变量的值的一部分(关于 `PATH` 变量的更多信息请参考第 5 章)。--help 选项的意思一看就明白。该命令的输出将显示很多开关，在运行 `configure` 命令的时候可能会用到它们。这个命令定义怎样编译和安装代码。例如，下面是 `Blackbox` 窗口管理器代码的 `Beta` 版本的输出：

```
$ ./configure --help
'configure' configures blackbox 0.70.0beta2 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE. See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
-h, --help                display this help and exit
  --help=short            display options specific to this package
  --help=recursive       display the short help of all the included packages
-V, --version            display version information and exit
-q, --quiet, --silent    do not print 'checking...' messages
  --cache-file=FILE      cache test results in FILE [disabled]
-C, --config-cache        alias for '--cache-file=config.cache'
-n, --no-create          do not create output files
  --srcdir=DIR            find the sources in DIR [configure dir or '..']

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, 'make install' will install all the files in
'/usr/local/bin', '/usr/local/lib' etc. You can specify
an installation prefix other than '/usr/local' using '--prefix',
for instance '--prefix=$HOME'.

...
```


配置输出通常还列出其他设置，这些设置指定将安装的文件类型、可以定义的环境变量和可以启用或禁用的不同功能。

如果只需要使用默认值而不想找到合适的选项，只要运行 `./configure` 命令即可。配置脚本可能会有助于解决一些细节问题，这些问题在编译软件的过程中经常涉及到，包括：

- 寻找安装工具
- 探测 AWK 实现
- 查看 `make` 命令怎样工作(用于实际执行编译的命令)
- 查看 C 编译器和 C++ 编译器的类型
- 学习怎样运行 C 预处理器(cpp)
- 自动探测系统架构和操作系统
- 检查各种标准包含文件(头文件)
- 决定是否编译共享的或静态的库

在探测编译环境的时候 `configure` 脚本可能会有更多的用处。更多详细内容可以参考本章后面的“帮助创建 Makefile 的工具”小节。

通常都会设置的一个选项是 `--prefix`，这个选项定义在何处安装软件。一般情况下，软件默认安装到 `/usr/local` 目录，并在将软件编译到目录层级内的参考文件期间配置软件。文件安装在恰当的子目录中，例如在 `/usr/local/bin/` 目录中安装可执行文件，在 `/usr/local/man/` 目录中安装联机帮助文档，在 `/usr/local/share/` 目录中安装文本和数据文件，而在 `/usr/local/lib/` 目录中安装库。在 `/usr/local` 目录中安装软件可以防止新的程序覆盖系统本来的文件。

FreeBSD 在它的包系统中使用 `/usr/local` 安装第三方软件，所以在 FreeBSD 中可能要使用不同的 `--prefix`。关于文件系统的更多信息请参考第 4 章。

下面是怎样使用 `--prefix` 选项在主目录中安装软件的命令：

```
./configure --prefix=${HOME}
```

然后编译过程可以在主目录下创建 `bin`、`sbin`、`lib`、`man` 和/或其他子目录。

很多管理员都在根据软件命名的目录下安装软件，例如使用 `--prefix="/usr/local/apache"` 或 `--prefix="/opt/gnome-2.6.2"` 这样的选项。在第一个选项中，编译过程将创建很多子目录：`/usr/local/apache/bin/`、`/usr/local/apache/conf/`、`/usr/local/apache/modules` 等。第二个例子将把所有的文件都安装在 `/opt/gnome-2.6.2` 目录下。使用专用目录安装软件有一些好处，包括可以很容易地清除软件，快速查看安装了哪些软件，另外还可以使用不同的安装前缀多次安装同一个软件(只是简单的升级)，例如使用版本号。

记住如果把软件安装到某些目录，那么该软件在使用可执行文件路径搜索时不能执行(在第 5 章中已讨论)。在工作环境中，把新安装的 `/bin` 目录添加到 `PATH` 环境变量中，或者在运行这个软件的命令中使用绝对路径。本章稍后将讨论如何以及在何处安装软件。

通常配置脚本成功执行之后会显示它至少创建了一个 Makefile。也可能有其他的输出，例如显示为编译软件进行了怎样的配置。

运行一个配置脚本可能需要 20 秒到数分钟不等。

2. 使用 make

在 `configure` 脚本成功完成之后，应该获得一个名为 `Makefile` 的新文件。编译和安装软件都是通过一个叫做 `make` 的命令来完成的(本章的下一节将更详细地讨论 `make` 命令)。简单地说，我们使用 `make` 创建程序(以及文档、网页和几乎其他所有文件)，它能确保软件的每个组成部分都是最新的。它对开发人员非常有用，如果只是修改了一个小文件，有了 `make` 就不用重新编译所有的代码(本章稍后讨论 `make` 的其他用法)。

两种常见的实现是 GNU `make`(也称为 `gmake`)，大多数 Linux 系统上使用的都是这种 `make`，和 `pmake`。几个 `pmake` 的派生产品都由 BSD 操作系统工程组维护，有时候也把它们称为 `bmake`。这些 `make` 命令都使用标准的语法，但它们的不兼容性有些差别。在大多数情况下，系统自带的 `make` 命令都能很好地工作。

为了继续编译和安装，只要在命令提示符后面简单地输入 `make`(第(4)步)即可。屏幕上将会滚动很多输出，这些输出中的大多数都是表明 `make` 在每个文件上调用 GCC 编译器的信息。

系统的速度和要编译的代码量可能会使这一步耗费从几分钟到几个小时不等。一些大型项目，例如 GNU `Lic`、`Koffice` 和 `X.org`，在比较老的机器上编译要花一天多的时间。`OpenOffice.org` 可能需要一周的时间来编译。最好使用 Celeron 1000、AMD 兆赫的机器来编译项目。

当 `make` 这一步成功完成之后，它通常只是简单地退出，而不会显示任何明显的消息。如果有问题出现，`make` 通常会退出并显示消息 `Error 1`(本章后面将详细讨论 `make` 和 `makefile`)。

很多软件编译过程(例如 GNU `tar`、`PCRE`、`OpenLDAP`、`Ruby` 和其他很多软件)还包括在安装软件之前进行测试，通常可以运行 `make test` 或 `make check` 来进行测试。这种方法可以用于在安装和使用软件之前检验软件能否正常工作。

也许在安装之前可以在编译目录中直接运行新程序以对其进行测试。很多时候这样做可能行不通，因为这些程序可能会引用到其他还没有安装的库、配置文件或其他数据，通常 `make` 这一步并没有完成所有的编译。

一般情况下，最后一步(第(5)步)是执行 `make install` 命令。这个命令将完成编译(如果需要的话)，创建安装目录，复制可执行文件、配置文件、库、文档和各种数据等。取决于选择在什么目录安装软件，可能需要以超级用户的权限来执行这一步，例如：

```
su root -c "make install"
```

这个命令由两个部分组成。首先，第一部分 `su root` 假定可以获得超级用户权限，`-c` 选项通知 `shell` 运行包含在命令行中的后续命令。第二部分是要以超级用户身份运行的命令，`make install`。使用这种方法需要提供超级用户密码，但是可以不用单独登录和退出。

也许希望以日志的方式记录下安装了哪些软件，这些日志以软件名、版本、日期和目的排序。也许还希望产生一个列表，这个列表中包含了所有已安装的文件。

有些安装将会备份以前已安装的比较老的、同名的文件，但不要依赖这种功能。另外，有时候必须运行多个安装目标以便安装额外的组件。查看安装说明以获得确定的信息(本章后面的 `Lynx` 安装示例将显示如何额外安装一个文档)。

当所有的安装都完成之后，可能希望删除编译目录以节约磁盘空间。有时候临时保存它们有助于查找和解决问题，但没有必要长期保存 `tarball`。随时都可以重新下载它们。

既然已经知道了从源代码安装的基本步骤，现在就应该试一试。按照下面的步骤在 Debian Linux 系统的主目录中从源代码安装 OpenSSL。

OpenSSL 为 SSL 和数据加密提供工具和库。

(1) 在主目录下，创建一个目录(例如 src)，软件的编译将在这个目录中进行，然后切换到这个目录：

```
$ mkdir ~/src
$ cd ~/src
```

(2) 获取源代码。可以在 Google 或 FreshMeat 站点上搜索到 www.openssl.org 网页。如果没有更新的版本，则使用浏览器下载 openssl-0.9.7e.tar.gz 版本，大小为 2.7MB。

(3) 提取源代码并切换到新目录中：

```
$ tar xzf openssl-0.9.7e.tar.gz
$ cd openssl-0.9.7e
```

不需要 tar 命令的 v(verbose)选项，在这个例子中它将在提取的时候列出 1900 多个文件名。新创建的目录命名为 openssl-0.9.7e，大小为 17MB。

(4) 使用 ls 命令查看包含在安装目录中的文件(有多个 README*和 INSTALL*文件)。

(5) 使用 less 或 more 命令阅读这些找到的文件。主要的 README 文件中指出，为了在 Unix 的派生系统中安装 openssl 需要阅读 INSTALL 文件。这个文件给出了安装的快速指南以及配置的详细信息。相同的目录中还包含了一个 FAQ(Frequently Asked Questions)文件，这个文件对排查编译中的各种问题很有用。

您可能已经注意到在新目录中有 config 命令和 Configure 命令，但没有 configure 脚本(文件名全是小写)。此时 OpenSSL 的编译不使用标准的 autoconf 配置系统，但这里演示的是一种类似的方法。

(6) 根据 INSTALL 文件中的说明，运行带有--prefix 开关的 config 脚本：

```
$ ./config --prefix=${HOME}/openssl shared
```

这个命令将配置编译环境，使得安装程序存放在主目录下的 openssl 子目录中。shared 参数告诉这个脚本编译并安装共享库。config 脚本检测操作系统、硬件平台和所使用的编译工具；定义怎样编译 openssl 库；并确保这个本地编译结果可以立即执行。输出的速度可能会很快，无法阅读，但有些状态信息在这个步骤结束的时候才显示出来。

(7) 开始编译：

```
$ make
```

使用 Makefile 中的指令，make 跳转到不同的子目录中并按顺序对不同的 C 文件运行编译器。这里使用的是 AMD-K6 系统，内存为 256MB，使用 gcc 2.95.4，编译过程持续 6 分钟。

(8) 为了在安装之前测试已编译好的程序，运行带有 test 的 make 命令：

```
$ make test
```

这个命令测试 SSL 和 TLS 验证、证书、哈希(例如 MD5 和 SHA1)算法、私用/公用密钥的产生、加密和解密密钥和其他内容(这个过程可能需要几分钟才能完成)。

(9) 安装软件。记住已经选择在自己的主目录中安装它(在运行 `config` 命令的时候由 `--prefix=${HOME}/openssl` 选项指定), 所以可以用自己的账号开始安装了:

```
$ make install
```

工作原理

这个安装步骤将创建必要的子目录并安装 340 多个文件, 共 7.6MB(安装过程需要 2 分钟多一点)。下面是新安装的文件:

```
$ ls -l ~/openssl
total 16
drwxr-sr-x  2 reed  reed    4096 Sep  7 11:00 bin
drwxr-sr-x  3 reed  reed    4096 Sep  7 11:00 include
drwxr-sr-x  3 reed  reed    4096 Sep  7 11:00 lib
drwxr-sr-x  7 reed  reed    4096 Sep  7 11:00 ssl
$ ls -l ~/openssl/bin
total 1264
-rwxr-xr-x  1 reed  reed    3656 Sep  7 11:00 c_rehash
-rwxr-xr-x  1 reed  reed  1283686 Sep  7 11:00 openssl
```

本例中, 在没有 `configure` 脚本的情况下编译了软件。虽然没有通常使用的 `configure` 脚本, 但是 OpenSSL 提供了另一种配置方法, 这种方法可以帮助正确地安装软件。

实战

编译需要预装软件的代码

在这个例子中, 您将安装 Lynx, 它是为文本终端提供的一种流行的 Web 浏览器和 FTP 客户端。这个软件使用 `autoconf` 配置系统。一定要注意为使用 OpenSSL 是怎样配置的。OpenSSL 用于支持 HTTP, 在前面的例子中已安装了它。

(1) 切换到源代码编译目录(这个例子使用 `/src` 作为编译目录):

```
$ cd ~/src
```

(2) 找到 Lynx 的源代码。官方站点是 lynx.browser.org。这个例子使用最新的开发版本并通过 `wget` 下载:

```
$ wget http://lynx.isc.org/current/lynx2.8.5rel.1.tar.bz2
```

(3) 提取源代码并切换到新创建的目录中:

```
$ tar xvjf lyjf lynx2.8.5rel.1.tar.bz2
$ cd lynx2-8-5
```

注意 `tar` 目录的 `j` 选项表示文件是用 `bzip2` 压缩的。

(4) 查找 `README` 或 `INSTALL` 文件。在这里, Lynx 提供了一个名为 `INSTALLATION` 的文件, 这个文件包含了安装说明。编译这个版本的 Lynx 将会用到很多选项。

(5) 运行 `./configure --help` 查看编译选项列表。

(6) 必须执行一个特殊的步骤，这个步骤配置 Lynx 编译程序以便安装到主目录中，并使用前面已经安装的 SSL 库：

```
$ LDFLAGS=-Wl,-R${HOME}/openssl/lib
$ ./configure --prefix=${HOME} --with-ssl=${HOME}/openssl -- \
libdir= ${HOME}/share/lynx
```

默认情况下，Lynx 并不知道在何处找到共享库，因为主目录中安装共享库的位置或许并不在默认搜索路径中。即使使用 `--with-ssl` 命令定义共享库的路径，`make` 命令也将报告编译出错。

(7) 准备安装：

```
$ make
```

在这个编译系统上编译过程将持续 4 分半钟。

(8) 为了安装软件，执行命令：

```
$ make install
```

这个安装例程将备份已经存在的 Lynx 二进制文件，并把它命名为 `lynx.old`。

(9) 如果希望安装其他 Lynx 文档，则可以根据屏幕上给出的提示进行操作。使用下面的命令：

```
$ make install-help
$ make install-doc
```

`install -doc` 并不是必须的，它提供了一些测试 Lynx 的例子和网页，主要由 Lynx 开发人员使用。

工作原理

这是一个非常简单的安装，只需要按照安装顺序一步步往下做就行。惟一不同的是使用 `LDFLAGS` (与前一个例子的第(6)步相比)告诉 `make` 在所需库所在的位置进行编译。例如，`--libdir` 选项告诉 Lynx 配置文件和本地文档保存在何处。

使用 `ldd` 命令可以看到 Lynx 确实使用了 `openssl` 库：

```
$ ldd ~/bin/lynx
libncurses.so.5 => /lib/libncurses.so.5 (0x40017000)
libssl.so.0.9.7 => /home/reed/openssl/lib/libssl.so.0.9.7 (0x40055000)
libcrypto.so.0.9.7 => /home/reed/openssl/lib/libcrypto.so.0.9.7
(0x40085000)
libc.so.6 => /lib/libc.so.6 (0x40177000)
libdl.so.2 => /lib/libdl.so.2 (0x40294000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

`ldd` 工具用于列出运行程序所需的共享库。注意 Lynx 还会探测本地系统的 `ncurses` (用于控制终端的屏幕)并使用它的共享库。

通过输入下面的命令运行新安装的 Lynx：

```
$ ~/bin/lynx
```


19.5 make、Makefile 和 make 目标

使用 `make` 编译和安装软件并不需要知道它是如何工作的，但理解 `make` 的原理有助于排查问题和移植软件。

除了编译软件之外，`make` 还可用于管理 Web 站点和处理其他文档。它的主要作用是保证每个组成部分都是最新的。例如，`make` 通常用于编译很多独立的源代码文件，然后产生最终的程序。开发人员不希望在处理某个文件的时候反复编译项目中的每个文件。`make` 可以解决这个问题，它只重新编译更新过的文件。定义所需步骤的规则存放在一个 `makefile` 文件中(如前所述，通常将这个文件命名为 `Makefile`)。

本章前面已经提到，有几种不同类型的 `make`。它们有一些通用的语法和用法，但也提供了互不兼容的功能和配置。Linux 系统上的 `make` 是 GUN Make。BSD 系统上的 `make` 是 BSD Make，通常称为 `pmake`，但也称为 `bmake`。FreeBSD、NetBSD 和 OpenBSD 都有自己版本的 BSD Make，所以它们之间有一些不兼容性。GNU Make(也称为 `gmake`)也可以安装到 BSD 系统上，可以通过它的压缩包获得它。也可以从源代码编译和安装 GNU Make。从 ftp.gnu.org/pub/gnu/make/ 下载相关文件。在 Mac OS X 系统和 Darwin 系统上同时安装了 `bsdmake` 和 `gunmake`，默认情况下 `make` 是 `gnumake` 的一个符号链接。

更让人迷惑的是，XFree86 或 X.org 的 `imake` 工具和 QT 的 `qmake` 工具都不是 `make` 程序。它们用于产生 `makefile`，但执行的其他任务和标准的 `make` 都没有关系。

很多项目需要特定版本的 `make`，但是绝大多数 `makefile` 都是以一种可移植的、标准的格式编写的。如果在运行 `make` 的时候接收到类似于下面的错误：

```
Makefile:18: *** missing separator. Stop
```

这可能表示正在使用的是 GNU Make，应该用 BSD Make 取代它。然而，如果接收到的错误消息类似于：

```
make: don't know how to make something. Stop
```

则应该用 GNU Make 来取代 BSD Make。

根据经验，最好只使用 GNU Make，因为它是开放源代码项目中使用得最广泛的 `make`。

19.5.1 Makefile

`makefile` 文件的语法有时可能会非常琐碎，让人迷惑不解。通常，`makefile` 由不同的定义和目标组成。`makefile` 也可以有注释，以井号(#)开始。

第 5 章已经学习过，变量的定义是通过向变量标识符赋值完成的。一般情况下，变量名全是大写，但这并不是硬性要求。定义一个值有好几种方法。让我们看看常用的一些方法。例如，使用等号赋值意味着将覆盖原来的值：

```
SHELL = /bin/sh
```

可以通过加-等号(+=)附加一个值：


```
CFLAGS += -s
```

在新值前面还有一个空格。赋值的时候通常会忽略空格(但通常能使程序的可读性更好), 所以可以使用 `CFLAGS += -s` 完成赋值操作。

如果已经定义了一个变量, 使用问号-等号(`?=`)可以确保不会重新定义该变量。例如, 在下面的代码行中, 如果还没有定义了 `CRYPTO`, 就把它赋值为 `yes`(否则它将保持原来的值不变):

```
CRYPTO ?=yes
```

通过 `$(NAME)` 或 `${NAME}` 引用变量。`make` 使用一个称为扩展(expansion)的概念, 这个概念的意思是: 变量的值只有在引用到变量的时候才知道。

定义目标(也称为规则)时, 把一个关键字放在一行的开始, 后接一个冒号。接下来的代码列出处理这个规则时使用的 `shell` 命令。每条命令必须缩进一个制表符(不是空格)。例如:

```
showmetheday
→@date +%A
```

`at` 符号(`@`)表示显示输出但不打印命令的名字(第 11 章详细地讨论了 `date` 命令)。如果把把这个目标放到一个 `makefile` 文件中, 然后在某个周五的时候运行 `make`, 输出应该是:

```
$ gmake
Friday
```

通常, 把主要目标命名为 `all` 并定义为第一条规则。另一个常用的目标是 `install`。可以选择运行某个特定的目标并在命令行指定它, 例如:

```
$ make showmetheday
Friday
```

下面的例子通过 `BSD Make` 执行一个不存在的目标:

```
$ make whatever
make: don't know how to make whatever. Stop
make: stopped in /home/reed/tmp
```

由于指定的目标不存在, `make` 进程打印一个错误消息并退出。

可以在一个目标的冒号后面列出其他目标以便先处理这些目标之间的依赖关系。看看下面的 `makefile` 文件。这个例子中包含了简单的规则依赖, 并在同一个命令行上定义变量:

```
# This is an example Makefile
MYVARIABLE?= alphabet soup

all: rule2
→@echo This is the "all" target.

rule1:
→@echo Hello from rule 1.

rule2: rule1
→@echo $(MYVARIABLE)
```

记住目标体中使用的是制表符(由右箭头表示), 而不是空格。

现在运行这个进程。注意它是怎样使用最顶上的目标作为默认目标的:

```
$ gnumake
Hello from rule 1
alphabet soup
This is the all target.
```

如您所见, 它设置了变量, 因为代码中还没有定义这个变量。接下来 `make` 执行第一个目标, 该目标具有依赖关系。也可以在执行 `make` 命令的时候选择希望运行的目标:

```
$ gnumake rule1
Hello from rule 1
$ gnumake rule2
Hello from rule 1
alphabet soup
```

下一个示例显示定义 `make` 变量的一种方法, 把变量放在命令行作为 `make` 的参数:

```
$ gnumake MYVARIABLE="Unix is fun"
Hello from rule 1
Unix is fun
This is the all target.
```

也可以在 Unix shell 的环境中设置这个变量, 而不使用 `make` 参数(关于 shell 和环境变量的详细内容请参考第 5 章)。例如:

```
$ MYVARIABLE=whatever
$ bsdmake rule2
Hello from rule 1
whatever
```

下面是另一个运行不存在的目标的例子:

```
$ gnumake help
gunmake: *** No rule to make target 'help'. Stop.
```

虽然这些例子都很简单, 但 `make` 可以完成的工作很多。`make` 的配置语法通常是一种编程语言。例如, 它可以使用 `if/else` 条件、模式匹配和替换, 以及其他很多功能。一般情况下, 使用 `make` 来运行 GCC, 以编译和链接软件, 然后安装软件。

19.5.2 帮助创建 Makefile 的工具

有一套流行的开发工具可以帮助创建配置脚本和 `makefile`。通常将这些工具称为自动工具, 或 GNU 编译系统。应该记得, 我们使用 `configure` 脚本检测编译环境并自定义怎样编译和安装软件。`configure` 脚本按照模板产生一个或多个 `makefile`。它的作用在于不用手动编辑 `makefile`, 简化了处理并为您节约了大量的时间。

大多数时候, 作为软件的终端用户, 并不需要使用自动工具。主要是最初的开发人员使用它们。但有时候可能会获得没有 `configure` 脚本或 `makefile` 模板的源代码(例如预发布的软件)。要正确安装这样的代码, 自动工具的作用非常大。下面是一些关键的自动工具:

- automake 工具用于产生 makefile 模板,稍后 configure 将使用该模板创建最终的 makefile。最初的文件名为 Makefile.am。于是由 automake 产生的模板就被命名为 Makefile.in。
- autoconf 工具用于创建可移植的 configure shell 脚本。autoconf 的输入是一个名为 configure.ac 或 configure.in 的配置文件。开发人员可能还会提供一些插件特性,这些插件通常命名为 acsite.m4 和 aclocal.m4,它们用于定义 autoconf 宏。大多数时候,configure 使用 config.h.in 文件产生 config.h 头文件,该文件定义了编译期间所使用的特性或功能。

除了 makefile 文件以外, configure 脚本还输出 config.status(实际产生 makefile 和头文件的 shell 脚本)、config.cache(记录大量配置测试的结果)和 config.log(包含输出和调试信息)。

另外还与 autoconf 和 configure 脚本相关的是 autom4te 文件 config.guess(检测实际的硬件平台、操作系统类型和版本)和 config.sub。

- libtool 工具用于在不同的系统中、使用不同的工具、创建不同对象格式的共享(静态)库。可以使用它创建和安装可移植的、标准化的库,这些库可以由不相关的项目使用。

与 libtool 相关的文件包括: ltconfig, 用于产生系统特定的 libtool 工具; ltmain.sh, 该文本包含库编译子例程; ltcf-c.sh, 该文件由 ltconfig 用于为系统编译器在创建库的时候选择正确的选项和参数。以 .la 为扩展名的文件是 libtool 的库文件, 它们是普通的文本文件, 其中定义了很多相关库的属性。

另一种使用得越来越多的软件编译工具是 pkg-config, 它提供已安装的库的元信息(meta-information)。更多内容请访问 www.freedesktop.org/Software/pkgconfig。

有时候会发现软件中并没有包含 configure 脚本, 但提供了 autoconf 文件, 可以使用这个文件产生一个 configure 脚本。一定要查看编译说明。源代码中可能包含了一个 autogen.sh 脚本, 可以运行这个脚本产生 configure, 然后再运行 configure。在其他情况下, 可能需要手动运行多个 automake 和 autoconf 命令以便创建 configure 脚本, 和下面的命令顺序一样:

```
$ aclocal
$ autoheader
$ automake -a --foreign -i
$ autoconf
```

很重要的一点是, 要注意到 automake 和 autoconf 有多个不同的版本, 有些功能是不可用的, 或者即使可用也不能获得支持(例如, 可能会接收到错误, 这些错误显示无法找到指定的宏)。再次强调, 必须阅读所有的编译说明(或者询问开发人员)以确保所使用的 automake 和 autoconf 是正确的版本。

使用自动工具系统的软件有一个问题, 那就是它们包含了很多预先产生(pregenerated)的文件, 这些文件很大, 有时候修改起来非常困难。当新版本的 automake 和 autoconf 发布之后, 它们的功能得到改善, 并带有新的补丁。那些已经发布的、带有预先产生的老的 configure 脚本的软件将难以利用这些补丁带来的好处。为了发挥新的自动工具的优势, 可能需要下载软件的新版本。

19.5.3 GNU 编译工具

在从源代码安装软件的时候, 一个步骤就是编译代码以使它在系统上可执行。编译器非常多, 它们可以处理各种语言编写出来的代码。然而, 在基于 Unix 的系统上, 可以使用 GCC 和

与它相关的编译工具。

GCC 并不仅仅是一个 C 编译器：它是 GNU 编译器集(GNU Compiler Collection)。GCC 套件支持 C、C++、Objective C、Java、Fortran(F77)以及 Ada 语言。GCC 的移植性非常好，使用范围也很广——几乎所有用 C 或 C++编写的开放源代码软件都使用它来编译。可以使用 GCC 进行跨平台编译，例如为不同的操作系统和/或使用不同主机平台的硬件架构软件。

GCC 很可能在安装系统的时候就默认安装了。如果没有安装或者需要一个最新的版本，请访问 www.gnu.org/software/gcc/gcc.html。

GCC 工具是控制很多其他工具的前端，这些工具将在编译过程用到，例如 `cpp` 预处理器、编译器(类似 `cc1` 或 `cc1plus`)、汇编器和链接器。`gcc` 是很多工具的前端，它有 500 多个选项可以使用。

最常用的汇编器(可移植的 GNU 汇编器 `as`)和链接器(GNU 链接器 `ld`，可以使用它合并库归档文件)是和 GNU Binutils 套件一起提供的。GNU Binutils 还提供其他工具，这些工具用于修改库(`ar`)、列举符号(`nm`)、显示目标文件信息(`objdump`)、产生文档索引(`ranlib`)以及列出以扇区为单位的目标文件的大小(`size`)。如果没有 Binutils，可以从 www.gnu.org/directory/binutils.html 下载。

GNU Binutils 同时还提供了“瘦身(`strip`)”工具和字符工具。`strip` 命令用于提取符号，例如从目标文件中提取调试信息。它可以在很大程度上减小文件的尺寸。很多管理员使用 `strings` 命令从文件中输出可打印的字符。例如，如果有几个二进制文件或私用文档格式，可以使用 `strings` 命令过滤这些文件从而仅显示对人类可读的内容。

前一节已经提到，`automake`、`autoconf` 和 `libtool` 用于为运行 `gcc` 和 `linker` 建立编译环境。在不同的 Unix 版本上编译软件和库可能会有很大的差异，而且在大多数情况下，`configure` 脚本和 `makefile` 已经存在，所以不用关心它。

19.5.4 diff 和 patch

很多开发人员会推迟正式的版本，而只为最近的安全问题提供补丁和 bug 修补。另外，有些项目以补丁的形式提供升级以节约带宽和时间。其他开发人员为插件、新功能或项目的其他改进提供补丁，而这些代码并不由他们自己维护。知道怎样使用 `patch` 和 `diff` 命令对处理这种情况特别有用。`diff` 是用于比较两个文件的命令；它的输出称为补丁(patch)。`patch` 命令可以使用 `diff` 命令的这个输出来修改一个文件(或多个文件)。

`cmp` 命令也可以比较两个文件是否相同，但 `diff` 要有用得多，因为它能够告诉您两个文件的区别，并提供用于重新创建差异的信息。

例如，为了查看两个文件之间的差异，使用命令 `diff old.file new.file`，具体如下所示：

```
$ diff motd.orig motd
5,7c5,6
< Please note that the library will be closed on Saturday for
< construction. Any books can be put in the box outside. Videos
< due on Saturday will have their due dates extended for two days.
---
```

```
> Student Board Elections begin next week. Be sure
> to vote on Tuesday or Wednesday!
```

小于号(<)表示该行位于第一个文件(motd.orig)而不是第二个文件(motd)。大于号(>)表示添加到第二个文件中的新行。

通常,使用 `diff -u` 命令创建补丁以便获得统一的输出。很多用户认为阅读统一的输出会比较容易:

统一的输出是两个文件之间不同之处的另一种表现方式。也许 `diff` 和统一的输出之间的最大差别就是统一的输出提供上下文。一些开发人员偏向于使用统一的输出来发布补丁,因为 `diff` 可能会产生某些无规律的输出。请访问 www.kcl.ac.uk/humanities/cch/ma/courses/ucmtls/gnu/diff.html 以获得 `gnu diff` 实用工具的更多信息。

```
$ diff -u motd.orig motd
--- motd.orig   Mon Sep 27 10:21:14 2004
+++ motd        Mon Sep 27 10:21:07 2004
@@ -2,8 +2,7 @@
```

```
Welcome to the University Shell System!
```

```
-Please note that the library will be closed on Saturday for
-construction. Any books can be put in the box outside. Videos
-due on Saturday will have their due dates extended for two days.
+Student Board Elections begin next week. Be sure
+to vote on Tuesday or Wednesday!
```

```
To get help on using this system, type "help" and press Enter.
```

如您所见, `-u` 开关包含修改过的内容前面和后面的文本行,从而添加了更多的内容。以减号开始的行只是原来的数据,以加号开始的行是添加到新文件中的行。

如果需要查看多个文件之间的不同之处,有些文件包含在子目录中,可以使用 `diff -ruN directory1 directory2` 命令。`-r` 开关使得 `diff` 命令递归地比较子目录。`-u` 开关使输出以统一的格式打印出来。`-N` 选项运行 `diff` 比较不存在于其他目录中的文件(相对于其他目录把它看作是一个空文件)。

用于 `diff` 输出的选项很多。更多详细信息请阅读 `diff` 的联机帮助文档。

`diff` 的输出包括表明修改位于何处的行号。`patch` 命令可以将 `diff` 格式用于更新文件,甚至创建新文件。使用 `patch` 命令的常用方法是切换到包含了需要打补丁的文件的目录,然后把 `diff` 的输出(补丁)作为 `patch` 命令的标准输出来运行该命令。`patch` 工具还有其他有用的选项,例如它能在修补原来的文件之前保存它们,以防出现问题。

19.6 利于维护的安装技术

从源代码安装软件的主要问题包括清除旧文件、升级到新版本、运行时配置、记住编译步骤和编译配置以及跟踪依赖关系。当一个程序需要用到其他软件提供的功能时就产生了依赖关系。例如, `psmerge`(一个用于合并 Postscript 文件的工具,由 `psutils` 项目提供)是一个 Perl 脚本。

它有一个依赖关系(一个需要)在机器上安装 Perl 解释器。

如果依赖关系是共享库,那么情况会变得非常麻烦。例如,很多 GNOME 软件都使用 GConf 库,但是有些 GConf 版本不兼容。如果升级与 GNOME 相关的软件,新包就可能无法编译,因为所需库的版本号不对。另一种方法,如果升级 GConf,又可能使 GNOME 无法运行。您也许会认为只有一个依赖关系,修改它将非常容易。然而,取决于不同的 GNOME 软件,可能有 60 多种依赖关系!跟踪软件则会更困难。

一种技术是把某个特定项目中的所有软件都安装到一个目录中,这个目录以软件的名称和版本命名。例如,对于用到 `configure` 脚本的编译,可以使用命令 `./configure --prefix=/opt/name-version` 把软件安装到 `/opt/name-version` 目录中。然后可以升级不同的依赖关系而不用覆盖任何文件,因为相同的软件将会根据版本号安装到不同的目录。可以升级任何使用组件的程序,这些组件通过 `./configure` 开关(以及环境变量)告诉程序依赖关系位于何处。

这种技术对于处在不断开发阶段的软件特别有用。例如,可以在 `/usr/local/httpd-2.0.49/bin/httpd` 运行 Apache Web 服务器,然后配置一个新的编译以安装到 `/usr/local/httpd-2.0.51` 目录。修改 `/usr/local/httpd-2.0.51/conf/` 目录下的配置(根据旧的配置),然后就可以自由选择测试或实际使用该产品了。把软件包安装到它们自己的目录中也非常有用,当需要复制到其他系统时,无需再次编译它们。只要用 `tar` 命令将该目录打包,传送到其他机器,然后在相同的目录结构中解压。

也可以使用符号链接把某个 `bin` 目录(例如 `/opt/bin` 或 `/usr/local/bin`)和手册目录放到 `PATH` 中,以便您(和您的用户)可以容易地运行这些命令并使用这些帮助文档。

符号链接已在第 4 章中讨论过。

```
$ cd /opt/foo-1.2.3
$ for file in bin/* sbin/* man/*/* ; do ln -sf /opt/foo-1.2.3/$file /opt/$file;
done
```

在不同的项目中很少会有同名的文件,所以要使用正确的版本必须使用绝对路径。

如果希望同一软件的不同版本使用相同的配置目录,也许可以通过 `configure` 的 `--sysconfdir` 选项指向正确的目录。注意,在某些情况下,新的安装可能会覆盖配置信息。大多数时候,新安装的配置可能会因为新的功能和不兼容性而使目前正在运行的软件无法工作。

通过 `DESTDIR` 变量,可以把使用 `configure` 脚本和 GNU 编译系统的软件安装到与默认目的目录不同的目录中。例如:

```
# make install DESTDIR=/tmp/abc-3.4.5
# cd /tmp
# tar cvzf abc-3.4.5.tar.gz abc-3.4.5
```

(可以把 `abc-3.4.5` 替换成软件的名称和版本。)

在这个例子中,安装过程把所有的文件都放到一个单独的目录中,然后可以归档这个目录,以便在其他系统上重用,或者在同一个系统上重新安装。

在很多情况下,路径都是硬编码到可执行文件或文档和其他文件中的,所以这个软件最终应该安装在 `configure` 的 `--prefix` 选项指定的目录中。

19.7 排查编译问题

本书不大可能一一列出编译源代码和安装软件时可能遇到的错误或问题。例如磁盘空间错误、编译时内存溢出错误、用 gcc 编译库时使用了错误的开关、不兼容的库、头文件遗漏以及很多其他问题。下面的例子显示了一个错误的配置步骤：

```
checking for iconv_open... no
checking for iconv_open in -liconv... no
configure: error: Blackbox requires iconv(3) support.
```

类似的错误可能会意味着很多问题：一个确实不存在的库、configure 在错误的地方寻找库、configure 检测到库不正确，或者编译过程可能在进行了 iconv 探测时在某个地方出了错。接收到 configure 脚本的错误之后，查看新的 config.log 文件会有所帮助。

在发现错误消息之后，最好是通过偏爱的搜索引擎和 Google 群组(http://groups.google.com/advanced_group_search)进行搜索。多数情况下，其他人可能已经遇到过相同的或类似的问题——而且很可能已经有了解决方案。

另外，可以查看项目组的 FAQ，通常可以在项目的网站上找到，或者参考项目适当的邮件列表以寻求帮助。在使用邮件列表时，一定要表达清楚最终的目标是什么、正在使用的相关软件是哪个版本、采取了什么步骤、具体的错误消息或问题是什么、以及采取了其他什么步骤来排查问题。只有包含所有这些信息其他人才可能给您提供帮助。

19.8 预编译软件包

通常只安装预编译的软件包会更容易，而且所花的时间也比较少。这些软件包由操作系统供应商提供(或者是为操作系统提供的包)。预编译包是某个项目或程序立即可用的软件。包中通常含有可执行文件、配置示例或基本的配置、文档和辅助数据文件。包中还包含了元数据(metadata)，这些数据包含了一些信息，例如包的简要描述、谁或在何处编译了它、它提供了什么、以及它依赖于哪些包(如果有的话)。包系统通常提供一些工具，这些工具用于安装软件、自动安装必须预备的包、显示已安装的包、列出可用的包、显示文件列表、删除包、检索包、验证已安装的文件、提供包安全提示器，还有其他用途的很多工具。

可以使用的包系统和技术非常多。在开放源代码社区中最常用的两种是 RPM 和 Debian 公司的 DPKG 格式。RPM 原来表示 Red Hat Package Manager，但现在它是一个首字母递归缩写，RMP Package Manager。很多系统都可以使用 RPM 工具和 RPM 式的包，例如 Mandrakelinux、SUSE 和 Fedora Core。OpenPkg 也使用这个 RPM。所有的 Unix 系统都可以使用 RPM 工具。DPKG 格式主要是在 Debian GNU/Linux 和 Debian 派生系统中使用。RPM 一直以来的问题是依赖关系的自动更新能力非常弱。Debian 提供了一个工具，apt-get，这个工具对安装和明确地升级包非常有用。人们还扩展了这个工具以使它能和 RPM 一起工作(Red Hat 和其他使用 RPM 的系统也使用 Yum、Up2date 和其他升级系统)。

Mac OS X 系统上的 Fink 程序是 apt-get 和其他很多软件的前端，这些软件组织成扇区的形式，和下一段中提到的端口集非常类似。Fink 软件集包含很多补丁，这些补丁将简化 Mac OS X

系统上的安装过程，自动使用 diff 命令并收集不同的库路径，这个路径和其他 Unix 系统的库路径不一样。关于 Fink 的更多信息请访问该项目的主页 <http://fink.sourceforge.net/>。

BSD 操作系统提供从源代码编译(build-from-source)包的系统。FreeBSD 和 OpenBSD 称它们为端口(port)，同时 NetBSD 维护 pkgsrc(package source)。这些是众多软件套件说明和描述的目录(FreeBSD 提供了 1000 多种软件套件的说明，包含成千上万个文件)。pkgsrc 是一个可移植的包编译系统，它可用于 Linux、NetBSD、Darwin、Mac OS X、Irix、SunOS/Solaris、AIX、HPUX、BSD/OS、FreeBSD、Windows(带有 Unix 插件)和其他类似 Unix 的操作系统。通常这些编译系统会通过很多功能自动化整个从源代码编译系统：

- 下载源代码和补丁
- 根据事先记录的摘要校验和验证源代码
- 以递归的方式安装所有的编译或运行时依赖关系
- 根据需要为源代码打补丁
- 为特定的操作系统配置源代码
- 编译源代码
- 安装软件和相关文件
- 创建包(tarball 加上元数据)以便在其他地方使用或重新安装

一些操作系统使用类似的从源代码编译系统，例如 Gentoo Linux。与使用预编译的、立即可用的包相比，从源代码编译系统是非常慢的，但与编译纯代码相比仍然可以节约很多时间。例如，在没有安装任何必备的软件的情况下，为了从 FreeBSD 端口安装 KDE，很多系统可能都需要数天的时间。

从源代码和包来混合和比较软件编译结果以解决依赖关系是一种很糟糕的做法，尤其是在强制安装一个包(使用 RPM 的 --nodeps 选项，例如)时却没有安装所需的包的情况下，情况将变得更糟。这是因为一个包提供的必要条件或功能可能无法满足其他软件的需要。

让我们快速查看一下怎样使用 RPM 工具首先所需的软件，然后安装一个已经下载的包。

实战

使用 RPM

(1) 在命令行输入如下命令：

```
$ rpm -qa | grep mysql
mysql-server-3.23.58-9.i386
```

根据个人的安装情况，要么会获得一些输出，显示在机器上安装了哪个版本的 MySQL，要么什么输出都没有。

(2) 要安装新版的 MySQL，需要从 MySQL 的网站(<http://dev.mysql.com/downloads/mysql/5.0.html>)下载相关的 RPM 包并保存到所选择的文件夹中。

(3) 执行下面的命令

```
$ rpm -Uvh MySQL-server-5.0-2-0.i386.rpm
Preparing... ##### [100%]
1:mysql-server ##### [100%]
```

就这几个步骤！现在已经安装好 MySQL 了！

工作原理

RPM 能完成很多工作，更多信息请参考联机帮助文档。在这个练习中，选项-qa 使得 RPM 查询所有已安装的包文件，然后输出经由管道转发到 `grep` 命令以便该命令搜索任何包含字符串“mysql”的数据。这仅仅是演示了另一种使用 RPM 的方法。

接下来，使用 RPM 从 RPM 包安装新版本的 MySQL。请注意-Uvh 选项，该选择在使用 RPM 进行安装的时候经常用到。U 把 RPM 升级到最新的版本，v 使输出变得很详细，而 h 将在屏幕上打印一行散列值以便跟踪安装过程。

注意，在安装过程中如果有未满足的依赖关系，RPM 将失败。例如，如果新版本的 MySQL 依赖于另一个还未安装的包，那么安装将不会成功。保持对每个包或软件依赖关系的跟踪可能会非常单调乏味，建议寻找可以帮助完成自动更新以及其他功能的工具。例如 Yum，它是 Fedora Core(<http://fedora.redhat.com/>)的一个标准功能，这个工具不但能自动判断出需要升级的 RPM 包，还能判断出所有支撑包——并安装这些包。

19.9 小结

从源代码编译软件是 Unix 类型的系统常用的一种方法。对很多程序而言，这是惟一的方法，因为无法获得立即可用的、已编译好的软件。另外，很多及时安全升级或其他软件修补程序通常只能以源代码的形式获得。学习怎样从源代码编译软件最重要的内容包括：

- 找到并下载源代码文件
- 提取源代码文档
- 查找并阅读编译和安装说明
- 配置编译脚本和/或说明
- 编译软件
- 安装软件
- 清除已编译的文件
- 从源代码编译和安装软件所使用的技术可以用于很多 Unix 系统。

19.10 练习

- (1) 在基于 Unix 的系统上，需要一个程序获取基于 IMAP 的电子邮件。在使用偏爱的搜索引擎搜索之后，您决定安装 Fetchmail。
 - a. 下载 Fetchmail 的源代码。
 - b. 提取源代码文件。
 - c. (从源代码中)阅读安装指令。
 - d. 配置编译。
 - e. 从源代码编译 Fetchmail。
 - f. 在系统上安装已编译好的 Fetchmail。
 - g. 其他任务：配置 Fetchmail 以便接收邮件。

- (2) 您需要安装一个很少用到的、简单的 Web 服务器并决定使用基于 `inetd` 的 `micro_httpd`。
- a. 找到并下载 `micro_httpd` 的源代码。
 - b. 提取 `micro_httpd` 的源代码。
 - c. 查阅安装说明。
 - d. 使用 `make` 编译软件。
 - e. 手动复制联机帮助文档和新的可执行文件到目的目录。
 - f. 其他任务：配置 `inetd` 或 `xinetd` 以运行 `micro_httpd`。

第20章 转换：适用于 Mac OS 用户的 Unix

Apple 公司自从 1984 年引入 Macintosh 计算机时就为用户提供了一种基于图形用户界面的操作系统。从 1984 年到现在，图形用户界面已经发生了很大的变化，Macintosh 当前的操作系统版本是 Mac OS X，这个系统为用户提供了一个优雅而强大的 GUI 界面，同时在外观上保持了非常简单的用户友好性。然而，在这个友好的界面之下是一个完全不同的引擎。虽然 Macintosh 以前的操作系统都是由一个专有的引擎支持的，但是 Mac OS X 是由 Unix 支持的。

本章旨在与两类不同的用户讨论操作系统的变化：开始使用 Mac OS X/Unix 的 Macintosh 老用户以及对其他 Unix 系统感兴趣的 Mac OS X 用户。本章使用 Mac OS X 作为 Unix 的参考平台，因为很多仍在使用 Macintosh 经典版本操作系统的用户很可能会把他们的系统升级到 Mac OS X，要么在当前的计算机上，要么在购买新设备的时候。另外，本章的信息对那些感兴趣于 Mac OS X 的 Unix 老用户也很有用。

20.1 Mac OS X 简史

Macintosh 操作系统终止于 20 世纪 90 年代末的一次重大检修。Apple 公司曾试图创建一个新版的操作系统，并将该系统命名为 Copeland，它将在古老的 Mac OS 7 系统中加入一些非常有价值的现代功能。后来证实这项工程比一开始时预料的情况要复杂得多，最终 Apple 决定走出公司，从外面购买新的 OS。Apple 公司起初决定购买 BeOS，这个系统由 Be Inc 公司生产。该公司的两位创建人，Jean-Louis Gasse 和 Steve Sakoman，都曾是 Apple 公司的主管。该公司于 1991 年创建，现在已经倒闭了。这个计划因为与 NeXT 软件公司的风格相互冲突而被迫终止，这家公司由 Apple 以前的主管 Steve Jobs 经营。在 1996 年年底前后，Apple 公司以大约 \$400 000 000 的价格收购了 NeXT 软件公司。NeXT 公司在 20 世纪 80 年代末到 90 年代初期间生产计算机。在退出硬件业务之后，接下来的几年里，NeXT 继续销售它的操作系统 (NeXTStep，后来是 OPENSTEP) 以及 WebObjects 软件，这些软件可以运行在 x86、HP 和 Sun 硬件平台上。

当 1998 年第一台 NeXT 计算机面世的时候，它简直是一个技术奇迹。它的 CPU 是 33Mhz 的 Motorola 68030 芯片，这使它比市场上任何已经存在的 Macintosh 机器和基于 Intel 的机器都快。它自诩拥有一个整洁、高雅的 GUI，这个 GUI 以 Unix 为核心。这个系统主要面向教育市场而且是提供网络功能的第一台 PC 系统，具有一定的网络通信能力。该系统的销售起价为 \$6000，是那时正在销售的 Macintosh 机器或其他 PC 价格的两倍。价格成了妨碍该系统销售的主要因素。

Mac OS X 的公开 β 版是在 2000 年发布的, 10.0 版本是在 2001 年发布的。在 Rhapsody(公开 β 版本的代码名称)刚出现的时候, 很多人就已经意识到 Apple 公司拥有一个强大的竞争武器。这个操作系统保留了 NeXTStep 系统优雅的视觉效果, 同时也保留了 Macintosh 的外观。这个版本的 Mac OS 和其他版本之间有好几个明显的差别, 例如在桌面上有一个停靠栏, 而家用垃圾箱则从桌面上删除(转移到停靠栏上)。但是, 除了新的浅绿色视觉效果以外, 这个系统在外观上和以前的版本大部分相同。当然, 视觉上的相似性是这个版本的 Mac OS 和以前的版本惟一相同的方面。包含在 10.0 版本中的软件称为 Classic。Classic 是一个应用程序, 在启动之后, 它将执行一个基于软件的 Macintosh 模拟器。用户可以在一个硬件平台上同时安装 Mac OS 9 和 Mac OS X, 通过 Classic 可以启动 Mac OS 9 并运行该系统设计的应用程序。早期的用户通常都会用到这种功能, 在等待为 Mac OS X 发布的软件的同时运行旧版的软件。

现在, Mac OS X 已成为基于 Unix 的桌面系统的主流, 以它为系统生产的基于 Unix 的桌面系统比其他所有 Unix 供应商加起来的还要多。对这些系统的大多数用户而言, Unix 的加强作用是看不见的, 这些加强作用提高了系统的稳定性和执行效率, 但没有增加任何复杂性, 也没有影响到 Macintosh 系统“易于使用”和“就是好”的总体形象。硬件和软件以相同的即插即用的简洁性整合到一起, 这是 Macintosh 用户期待已久的。

20.2 Mac OS 9 与 Mac OS X 之间的差别

Mac OS X 和旧版本的 Macintosh 操作系统之间存在着很多差别。这些差别大多位于底层, 换句话说, 它们对于只限于使用系统 GUI 的用户来说是不可见的。然而, 除了只是减小死机或重启计算机的频率以外, 系统底层的一些基本变化却非常明显地改变了操作 Macintosh 系统的感受。

两类系统之间最明显的差别是, Mac OS X 是一个多用户操作系统, 而 Mac OS 9 是一个实实在在的单一用户系统。和其他所有 Unix 系统一样, 多用户的概念渗透了 Mac OS X 的每个角落。即使您是计算机的惟一用户, 而且计算机也没有连接到任何网络上, 系统上仍然有多个账户。这些账户是在安装系统时为不同的系统守护进程添加的, 它们使得系统的权限分离, 同时提供账户管理(accounting)。eppc 就是这样的一个用户, 它是远程 AppleEvents 服务, 系统在安装的时候创建了这个 eppc 用户。该用户即没有 shell 也没有主目录(本章稍后将讨论主目录)。然而, eppc 用于登录和其他目的。其他伪用户用于 postfix、cyrus、lp、mailman、MySQL、qtss、smmsp、sshd 和 catchall 守护进程以及未知的账户。

另外, 文件夹在硬盘上的组织方式也不同。虽然旧版本的 Macintosh 操作系统对硬盘上的文件夹名称和位置的处理非常灵活和随意, 但 Mac OS X 对文件夹的存放, 特别是在引导硬盘的根目录层级上, 有非常严格的规定。操作系统就安装在引导硬盘上。

Mac OS X 与旧版本的 Mac OS 之间其他的主要差别涉及到管理操作和权限。由于 Mac OS 9 和更早的版本都是单一用户系统, 所以在访问文件和文件夹或修改系统配置的时候没有任何限制。只要能够访问计算机, 就可以任意地进行修改。Mac OS X 的安全模型要严格得多, 只有管理用户才拥有管理权限, 他们可以控制文件和文件夹的访问权限并修改系统设置。另外, 由于系统上存在多个用户(或者系统上可能同时存在多个实际用户), 全局的设置和首选项没有以前那么多了。虽然 Mac OS 9 和更早的版本只有一个首选项文件夹, 在这个文件夹中可能保存了应用程序的设置, 但是您将看到 Mac OS X 对系统上的每个用户都安排了首选项设置。

20.3 文件夹也是目录

一直以来, Macintosh OS 都是以文件夹的方式为用户在嵌套的目录中显示成组的文件。自从 Macintosh OS 出现以来, 它就是这种操作系统可视化比喻的一部分。Mac OS X 中的 Finder 程序也使用文件夹并这样称呼它们。然而, 在 Unix 操作系统中, 文件夹通常也称为目录。这种说法起源于 Unix 的命令行, 在命令行下不需要图标引用。在使用 Mac OS X 的时候, 文件夹和目录两种说法都有用到。两种说法都是正确的, 而且它们表达的意思完全一样。可以使用任何习惯的术语。通常在使用 Finder 查看文件以及进行其他基于 GUI 的操作时会使用文件夹, 而在提到终端和其他命令行操作时会使用目录。

在文件夹和目录的导航方面, Mac OS X(以及 Unix)和 Mac OS 9 之间的主要差别是命令行界面。旧版 Mac OS 的很多用户曾经使用 AppleScript 或其他脚本编程语言来导航文件系统。对这些用户而言, 文件夹由冒号(:)分隔, 例如 Macintosh HD:System Folder:Preferences。Unix 和 Mac OS X 使用正斜线(/)作为分隔符, 例如/Users/craigz/Desktop。另外, 在使用典型版本的 Mac OS 的时候, 文件路径名将以卷标开始, 例如: Macintosh HD:Users:craigz:Desktop。然而, Unix 有其引用根目录的方式, 这种方式是一个正斜线(/)。因此, 在 Mac OS X 中同样的示例将表示成/Users/craigz/Desktop。在使用 Finder 导航文件系统的时候这些差别并不会造成任何问题, 但在使用终端导航文件系统或者使用其他 Unix 系统的时候, 这些差别就变得非常重要, 因为基于冒号的导航方式会产生错误消息。记住, 在 Unix 中, 所有的文件都位于根目录(/)下, 它是所有路径名的引导字符。在 Mac OS X 系统上, 不同的硬盘驱动器可以具有不同的标签, 例如 Macintosh HD 和 HardDisk。Finder 在桌面上显示额外的卷标, 然而在终端上, 额外的卷标位于隐藏目录/Volumes 中。

20.3.1 必需的文件夹

在典型的 Macintosh 操作系统中, 为了能正确引导系统, 只有一个文件夹是必需的: System 文件夹。实际上, 可以删除 System 文件夹中的大部分内容, 只需留下 Mac OS ROM、Finder 和 System, 一台典型的 Macintosh 机器就可以正常启动, 虽然启动后的状态不是很好。按照传统, Mac OS 9 对文件夹的名称和位置的处理非常灵活。在开发 Mac OS X 的时候, Apple 公司加强了结构的组织和文件夹的命名约定, 特别是在根目录层级。这基本上没有必要, 因为基于 Unix 的系统要求特定的内容必须按照特定的方式组织。这并不会在使用系统的方式上形成很多实际的差别; 但是, 应该知道这种差别的存在。

在 Mac OS X 中有 4 个安装时创建的永久文件夹。这些文件夹在安装 Mac OS X 的时候由系统创建, 不能移动、删除或修改这些文件夹, 因为它们对系统的运行非常关键。这些文件夹是 Applications、Library、System 和 Users, 如图 20-1 所示。

还有很多在安装时就创建的文件夹, 这些文件夹要么与特定的 Unix 有关, 要么与网络功能相关, 例如 Network、Volumes, 还有就是标准的 Unix 文件夹, 例如 ect、bin、sbin、tmp、usr 和 var。Finder 程序不会显示这些文件, 因为在日常工作中它们并不是必须的, 而且不显示它们可以保持界面清洁, 不易混淆。如果要显示引导硬盘上的所有文件夹, 但这些文件夹对 Finder 不可见, 请参考.hidden 文件, 它位于引导硬盘的根目录下。这个文件对 Finder 也是不可见的, 因为 Finder(类似于 Unix 的 ls 命令)不会显示以点号开头的文件或文件夹。

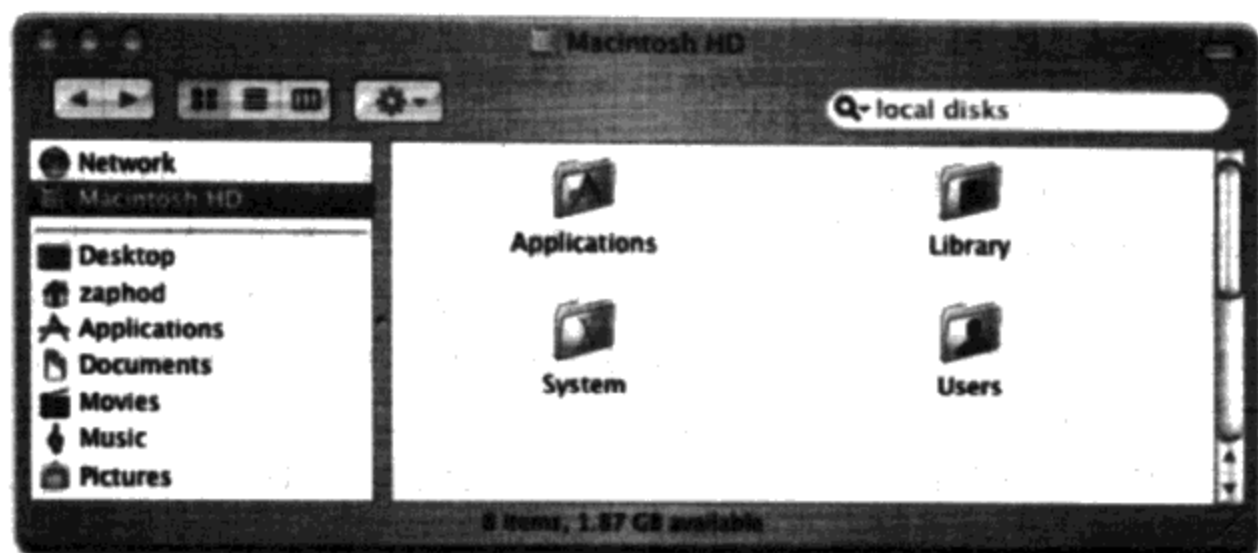


图 20-1

下一节描述这些文件夹并解释它们的作用。

1. Applications

Applications 文件夹用于保存应用程序。在安装 Mac OS X 的时候由 Apple 公司安装的所有应用程序都安装在这个目录中。这个目录中的所有程序对系统上的所有用户都是可用的。如果希望限制只有一个用户能使用某个特定的应用程序，那么需要在该用户的主目录中的 Applications 目录内安装这个应用程序。

在这个文件夹中有一个名为 Utilities 的文件夹，在安装 Mac OS X 的时候，系统将在这个文件夹中安装一些实用工具性质的应用程序。实用工具软件有 Disk Utility、Print Center、Installer、Console、Activity Monitor、Network Utility、NetInfo Manager 以及终端应用程序，还有其他用于配置、监视和维护系统各个方面的程序。

2. Library

Library 文件夹大致相当于包含在典型的 Macintosh 系统的 System 文件夹中的 Preferences 文件夹。Library 文件夹表现了 Mac OS X 的模块化性质。对这个文件夹包含的内容并没有什么正式的要求或定义。应用程序默认设置(application preference)、字体、打印机驱动程序、共享代码库以及表示信息(通常是与系统相关的信息)的内容都可以保存到 Library 文件夹中。实际上，Library 文件夹在 Mac OS X 系统上有 3 个不同的位置。位于引导硬盘根目录下的 Library 文件夹包含的文件对所有本地用户都是可访问的，但只有具有管理员权限的用户才能修改这些文件。另外，Library 文件夹位于 System 文件夹中，本节稍后讨论这个文件夹。另一个位于用户主目录中，下一节讨论这个目录。位于用户主目录的 Library 文件夹中的文件只能由这个用户使用。

3. System

这个文件夹大致相当于典型的 Macintosh 系统的 System 文件夹。这个文件夹中只包含了一个名为 Library 的文件夹。该文件夹包含了与主 Library 文件夹类似的文件夹，前面刚讨论过主 Library 文件夹。然而，这个位置是为 Apple 公司的软件保留的，不应该对其进行修改。如果计算机上有文件夹会大喊“请勿动手！”的话，这个文件夹就是 System。虽然管理员用户可以修

改该文件夹中的文件，但并不建议这样做，因为在软件升级的时候 Apple 很可能会修改这些文件，所以对这些文件所做的任何修改随时都可能被覆盖。

4. Users

从用户的角度看，Users 目录可能是操作系统最明显的变化。旧版的 Macintosh 操作系统曾试图引入多用户机制，但是从来没有真正变成包含在 Mac OS X 中的 Unix 标准的多用户系统。Mac OS X 是一个真正的多用户系统，本章一开始就讨论了这一点。

Unix 系统上的每个文件和文件夹都属于某个特定的用户和组。系统把创建文件的用户看作是文件的拥有者。作为拥有者，该用户可以决定其他用户是否能够阅读或编辑这个文件。而且，每个文件或目录都属于一个组，这与拥有者类似，只是拥有者只能是一个用户，而组可以包含多个成员。在安装了 Mac OS X 之后，系统默认创建两个组。这两个组是 **staff** 和 **admin**。在系统上创建的所有用户都是 **staff** 组的成员。拥有管理权限的用户，无论是在安装系统的时候创建的，还是后来才拥有权限的，都将添加到 **admin** 组中。通过设置某个用户或这两个组中的其他用户对文件或文件夹的访问权限，就可以控制这些类型的用户对文件的访问。为了更明确地控制文件和文件夹的访问权限，必须创建其他的组并把恰当的用户设置为这些组的成员。关于 Unix 中的用户和组的详细讨论，请参考第 3 章。Users 目录保存了系统上所有用户的主目录。

5. 其他文件夹

如果已经安装了 Developer Utilities，系统上还会出现 Developer 文件夹。另外，如果在系统上安装了 Mac OS 9，将会看到几个与这个操作系统有关的文件夹。这些文件夹是：

- **Application(Mac OS 9)**——这个文件夹是 Mac OS 9 安装应用程序的地方。如果已将系统升级到 Mac OS X，可以在这个文件夹中看到原来的应用程序。
- **Desktop(Mac OS 9)**——这个文件夹包含了 Mac OS 9 的 Desktop 文件夹。如果已将系统升级到 Mac OS X，可以在这个文件夹中看到原来的桌面内容。
- **System Folder** ——这个文件夹是 Mac OS 9 的 System 文件夹。Classic 应用程序将使用这个文件夹来启动模拟器。

20.3.2 主目录

主目录是计算机上存放个人数据的地方。这个目录中的所有数据都属于一个用户，而且只属于这个用户。在这个文件夹中，用户完全可以按照自己的偏爱自定义任何内容，而且不用担心所做的修改会影响到机器的正常运行、或者影响到系统上的其他用户使用机器的方式。桌面和包含在桌面上的所有文件和文件夹都位于主目录中一个名为 **Desktop** 的文件夹中。可以在自己的 Library 文件夹中安装字体，系统上的其他用户将无法看到这些字体。这个文件夹很像 Mac OS 9 和更早的 Macintosh OS 中的顶层文件夹 Macintosh HD。在这个文件夹中，用户可以完全根据自己的需要创建文件夹，将文件命名为任意喜欢的名称，而且决不会被任何身份验证对话框打断。

Apple 为了帮助用户在 Mac OS X 中组织数据做了很多努力。在创建账户的时候，系统会在该账户的主目录中默认创建几个文件夹。绝大多数计算机用户都创建和使用类型相似的文件，可以将这些文件宽泛地归类为多媒体。Apple 创建了几个目录，并推荐用户把音乐、电影和图片这样的通用类型文件保存到这几个目录中。另外，还有一个包罗万象的文件夹

Documents, 这个文件夹可以保存任何文件。用户主目录中的内容只对该用户可见, 但是 Public 和 Sites 两个目录除外, 稍后将讨论这两个目录。如果将本地系统作为 Web 服务器, 那么可以通过 Web 浏览器查看位于 Sites 中的任何文件。Public 目录的作用是和系统上的其他用户共享该目录中的文件, 而且如果打开共享功能, 还可以和网络用户共享这些文件。

这些文件夹都是默认安装的, 如图 20-2 所示, 下面的列表给出了这些文件夹的描述。

- **Desktop** —— 这个目录中的文件和文件夹就是在桌面上看到的内容。Macintosh 计算机上总是显示这个文件夹。然而, 以前 Finder 把它解释成不可见的。
- **Documents** —— 可以在这个目录中保存自己创建的文档和其他内容。用户可以根据自己的习惯决定是否使用这个文件夹。
- **Library** —— 这个文件夹包含了与其他 Library 目录中相似的内容。该文件夹中的文件是私有的, 而且只能由包含该目录的主目录用户使用。个人偏爱、字体、屏幕保护程序等都保存在这个目录中。如果希望系统上的其他用户能共享某些文件, 那么应该将这些文件保存在硬盘最顶层的 Library 目录中。
- **Movies/Music/Pictures** —— 这些目录是保存各种媒体文件的默认位置, 这些媒体文件由不同的 iLife 应用程序(iMovie、iTunes 和 iPhoto)产生。系统只是推荐用它们保存相似类型的文件。可以忽略这些文件夹或者在它们的内部创建文件夹, 总之, 按照对自己最有意义的方式来使用它们。
- **Public** —— 如果希望和系统上或网络上的其他用户共享文件, 可以把这些文件存放到主目录中的 Public 文件夹中。另外, 在这个文件夹中有一个名为 Drop Box 的文件夹。其他用户可以把您需要的文件保存到这个目录中, 但他们无法看到这个文件夹中的内容。这称为盲投(blind drop), 它通过阻止其他用户查看 Drop Box 文件夹中的内容来保护隐私。
- **Sites** —— Mac OS X 包含了 Apache Web 服务器软件。如果把本地计算机用作 Web 服务器, 那么这个文件夹中的任何文件都可由 Web 客户端访问。Sites 文件夹可以通过下面的 URL 访问: `http://your-ip-address/~username`, 其中 *your-ip-address* 是计算机的 IP 地址, 而 *~username* 是您的用户名(例如, `http://192.168.1.20/~craigz`)。

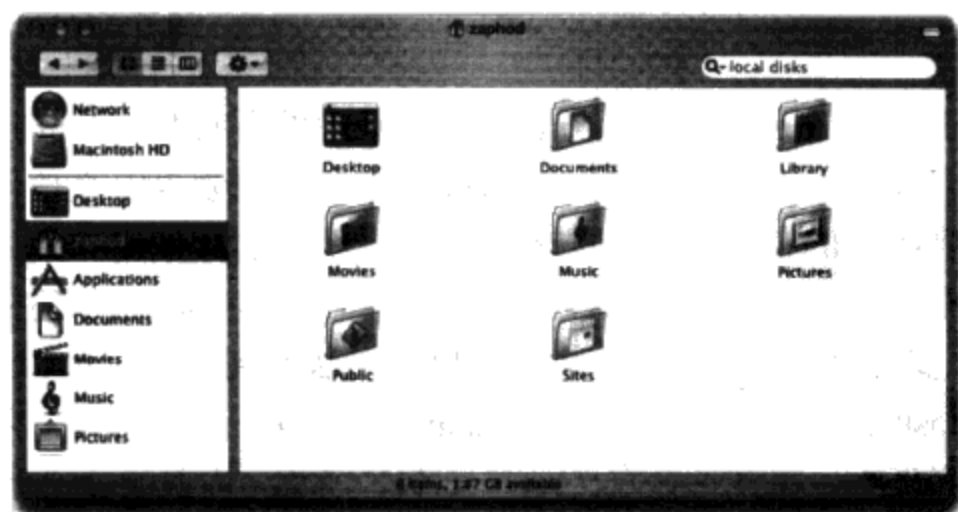


图 20-2

尽管这些文件夹都是默认安装的, 但是可以在主目录中执行任何喜欢的操作, 可以忽略所有这些文件夹。在这个目录中, 可以创建其他文件夹, 并根据自己的偏爱命名它们。

20.3.3 管理

再次强调，旧版的 Mac OS 和 Mac OS X 之间最大的差别是，旧版的 Mac OS 从根本上说是一个单用户系统。这意味着，只要能够访问计算机，就可以执行任何操作。例如可以修改网络设置、配置打印机、设置文件共享、以及查看和修改计算机上任意文件夹中的文件。由于 Mac OS X 的多用户特性，在修改系统设置时将受到更多限制。

Apple 使用基于角色管理的概念。系统允许任何已获得授权的用户确认系统执行维护任务和其他特权功能，而无需使用专用的管理账户。默认情况下，在系统安装时创建的用户具有管理权限，然而，这个用户可以将管理权限赋予其他任何用户。

在打开大多数系统预置的面板时，可以看到在窗口的左下角有一个锁形图标。为了修改这个设置，必须提供一个具有管理权限的用户的账户和密码，如图 20-3 所示。



图 20-3

大多数传统的 Unix 软件都是通过编辑配置文件来配置的，有时需要管理权限才能修改配置。Mac OS X 应用程序传统上是基于 GUI 的，每个程序通常在 Application 菜单中都会提供一个 Preferences 菜单项。通过设置 GUI 中的首选项，就可以配置应用程序。下一节讨论预置 (preference) 文件。

20.3.4 预置文件

应用程序通常把与用户相关的数据保存到硬盘的一个文本文件中，通常把这些数据称为设置、默认行为和其他与用户相关的信息。应用程序以后就可以在需要的时候参考这个文本文件以便查看以前设置的数据，而不是在每个应用程序启动的时候要求用户设置他们偏爱的定制信息。

典型的 Macintosh 操作系统一直把这些文件保存在一个特定的目录中，这就是硬盘 System 目录的 Preferences 文件夹。一直以来，系统都把预置文件保存为二进制数据文件，这些文件对人类是不可读的，或者没有专用的软件(例如 ResEdit)甚至都不能修改这些文件。Mac OS X 同时修改了这些文件的格式和存放位置。

Mac OS X 应用程序利用普通文本文件保存预置信息，这些文件由带有键的值组成。这些键/值对保存在名为 Property Lists(属性列表)的特殊格式的文件中，这些文件以.plist 为扩展名。属性列表文件把数据组织成命名数据和数据列表。将这些文件格式化成 XML，通过这种格式，可以使用统一的方式组织、存储和访问数据。

.plist 文件和 Mac OS 9 中的预置文件的功能一样。然而，通过标准化文件格式，完全可以使用文本编辑器编辑这些文件，或者使用可以解析 XML 的应用程序，这些应用程序提供一种编辑这些文件的更简单的方式。Apple 提供了一个基于 GUI 的工具，作为开发人员工具的一部分，这个工具可以编辑.plist 文件。如果已经安装了 Developer Tools 包，可以在/Developer/Applications/Utilities 找到这个工具。可以把这个工具和修改 Mac OS 9 预置文件所需要的过程比较一下，这个过程涉及到使用 ResEdit 应用程序修改十六进制的预置文件，并决定使用哪种由不同的供应商选择的格式。

20.4 Unix 和 Mac OS X/Mac OS 9 命令与 GUI 的对等命令

Mac OS X 以 Termianl 应用程序的形式提供了系统的命令行接口。很多可以在 Mac OS GUI 下执行的操作都可以通过 Unix CLI 执行。表 20-1 显示了怎样把 Mac OS 9(或更早的版本)GUI 命令转换成 Mac OS X GUI 下对等的操作、对等的 Unix 命令是什么、该命令的简短描述、以及这些命令出现在哪一章。

在标准的 Macintosh 键盘上，Command 键由一个 Apple 标志和交叉符号作为标记。人们通常把这个键成为 Apple 键。然而，命令键更常用，下表中把 Apple 键称为 Command 键。

表 20-1

Mac OS 9 GUI 操作	Mac OS X GUI 操作	对等的 Unix 命令	说 明	所 在 章 节
在 Finder 中选中文件，并从 File 菜单选择 Get Info 菜单，或者按下 CMD+I	在 Finder 中选中文件，并从 File 菜单选择 Get Info 菜单，或者按下 CMD+I	chmod	修改文件属性	第 6 章
使用 Finder 打开文件夹	使用 Finder 打开文件夹	cd	切换文件夹	第 4 章
查看窗体的标题栏	查看窗体的标题栏	pwd	显示当前的工作目录	第 4 章
磁盘急救	磁盘实用工具	fsck	扫描硬盘检查文件中的错误，扫描硬盘损坏或执行其他管理任务	第 4 章
CMD+D	CMD+D	cp	复制文件	第 6 章
选项+单击屏幕右上角的菜单条	单击屏幕右上角的菜单条并选择 Open Date & Time	date	修改日期	第 11 章
CMD+DELETE (单个文件)	CMD+DELETE (单个文件)	rm	删除文件	第 6 章

(续表)

Mac OS 9 GUI 操作	Mac OS X GUI 操作	对等的 Unix 命令	说 明	所 在 章 节
CMD+DELETE(文件夹)	CMD+DELETE(文件夹)	rm -R	以递归方式删除文件或文件夹	第 6 章
双击文件夹图标	双击文件夹图标	ls	显示目录中的内容	第 4 章
简单文本	文本编辑	vi(还有很多其他编辑器, vi 是系统上最常用的)	执行文件的文本编辑	第 7 章
N/A	从 Apple 菜单中选择 Log Out(Shift+CMD+Q)	exit	结束 shell	第 2 章
硬盘安装	磁盘实用工具	fdisk	硬盘分区	第 4 章
CMD+F	CMD+F	grep(Unix 的 find 命令只查找文件名而不在文件内部查找)	在文件中搜索作为参数给出的单词	第 8 章
在 Finder 中从 Help 菜单选择 Help Center	在 Finder 中从 Help 菜单选择 Mac Help	man	显示在线帮助文件	第 2 章
选择 Control Panels 菜单中的 TCP/IP	在 System Preferences Application 中选择	ifconfig	显示并修改网络接口(以及 IP 地址)	第 16 章
N/A	活动监控器 (activity monitor)	vmstat(Solaris), top, free(Linux)	显示内存统计信息	第 14 章
CMD+N	SHIFT+CMD+N	mkdir	创建目录	第 4 章
拖动文件夹到新的位置	拖动文件夹到新的位置	mv	移动文件	第 4 章
N/A	LoginWindow	who	显示当前已登录到系统上的用户的信息	第 3 章
N/A	活动监视器 (activity monitor)	netstat	显示网络统计信息	第 16 章
N/A	网络实用工具	nslookup	查找主机名	第 16 章
N/A	网络实用工具	ping	向指定的主机传送网络包	第 16 章
CMD+P	CMD+P	lpr	打印命名文件	—
选择文件并按下 Enter 键	选择文件并按下 Enter 键	mv	重命名文件	第 4 章

(续表)

Mac OS 9 GUI 操作	Mac OS X GUI 操作	对等的 Unix 命令	说 明	所 在 章 节
在 Finder 的 View 菜单中选择 List, 然后选择描述条 (description bar) 以便按描述排序	在 Finder 的 View 菜单中选择 List, 然后选择描述条 (description bar) 以便按描述排序	sort	以字符或数字为序排列数据	第 8 章
N/A	网络实用工具	tracert	显示指向特定主机的网络路由	第 16 章
选项+在 Finder 的列表视图 (List View) 中单击文件夹后面的 Disclosure Triangle	选项+在 Finder 的列表视图 (List View) 中单击文件夹后面的 Disclosure Triangle	ls -R	以递归的方式显示目录的内容	第 4、6 章
用相应的程序打开文件	用相应的程序打开文件	cat	显示文件或目录的内容	第 6 章
从 Apple Menu 中选择 About This Computer	从 Apple Menu 中选择 About This Mac	uname -a	显示当前操作系统的版本信息	—

20.5 Mac OS X 和其他 Unix 系统之间的差别

大家都知道, Mac OS X 是一个基于 Unix 的系统。这对每个人都有很大的好处, 从即不使用 Terminal 应用程序也不会向系统输入 CLI 命令的用户(他们将受益于系统稳定性和软件可用性的提高), 到那些 Unix 老用户, 这些用户能够第一次体验到 Apple 研究和开发的丰硕成果。Mac OS X 是学习使用 Unix 的优秀平台, 在 Mac OS X 上适用的所有内容几乎都可以移植到 Unix 系统上, 例如可以在 ISP、工作场所或其他位置找到的内容。

虽然 Mac OS X 系统的核心是 FreeBSD 的一个变种, 但是 Mac OS X 和其他 Unix 系统之间存在着很多差别。最重要的差别在系统管理方面, 特别是账户的创建和维护以及系统启动方面。

20.5.1 目录服务和 NetInfo

Mac OS X 从它的前一个操作系统 NeXTStep 继承了目录服务的概念。由于 NeXTStep 从一开始就设计成以网络为中心的操作系统, 每个客户端系统都会通过挂钩(hook)和位于中心的网络资源商量关于验证、资源访问权限、打印机配置等事情。根据本地配置, 每个系统既可以单独运作, 也可以作为网络客户端, 因为每个系统都有 NetInfo 数据库, 可以把这个数据库配置成一个独立的数据库, 或者配置成运行在另一台主机上的 NetInfo 数据库的客户端。虽然当前版本的 Mac OS X 仍在使用 NetInfo 数据库, 但 Apple 正打算删除这个私有数据库, 并转向开

放的标准，例如 LDAP，使用 OpenLDAP。

目录服务的影响是，系统在普通的操作中不再参考/etc 目录下的文件(`passwd`、`group` 和 `protocols` 等)。只有在极少数情况下，例如以单用户模式启动系统进行维护时，才有可能参考这些文件，但这并不是主要的形式，在正式的产品环境中不会使用这种方法。

一直以来，Unix 管理员都会查看/etc 目录中的文件，检查系统的配置情况或者修改配置。例如，`/etc/group` 文件包含了标准 Unix 系统上的组列表，如果希望将某个用户添加到一个组中，可以直接修改这个文件，在该组所在的行中加入该用户，修改就完成了。要修改 Mac OS X 中的组，必须在 NetInfo 数据库中完成。可以通过由 Apple 提供的基于 GUI 的应用程序 NetInfo Manager 进行修改，如图 20-4 所示。

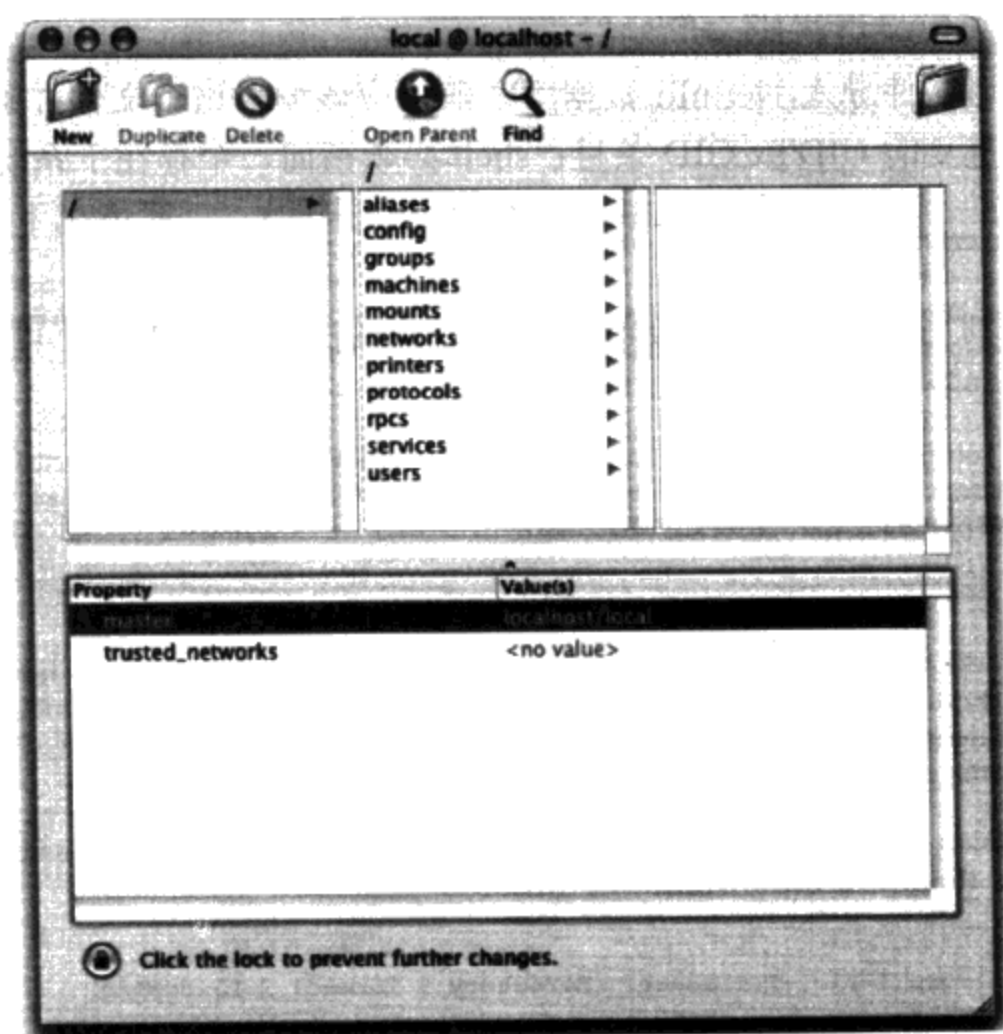


图 20-4

Apple 也提供了 NetInfo 数据库的命令行接口，可以在命令行显示、添加、编辑和删除 NetInfo 数据库中的任何条目。

在标准的 Unix 安装中，配置文件存放在/etc 目录。Mac OS X 也有一个存放标准文件的目录/etc/directory。然而，如果目录服务的配置没有发生变化，系统只会在单用户模式下参考这些文件，而在通常的操作中将忽略它们。

在 Mac OS X 的说法中，将标准的 Unix 文本文件称为普通文件(flat file)。如果希望通过普通文件来处理保存在 NetInfo 中的信息，可以使用 `nidump` 实用工具将数据导出到相应的普通文件中。

20.5.2 `nidump` 和 `niload`

例如，如果希望利用已经存在的账户信息创建/etc/passwd 文件，可以输入如下命令序列：

```
$ sudo nidump passwd . > /tmp/passwd
$ sudo mv /etc/passwd /etc/passwd.old
$ sudo mv /tmp/passwd /etc/passwd
```

第一步以恰当的格式在/tmp 目录中创建了一个密码文件。为了安全起见,第二步为已经存在的/etc/passwd 文件创建一个副本,接着最后一步用新创建的文件覆盖了原来的/etc/passwd 文件。第一条命令中的点号(.)表示要操作的 NetInfo 域,这里表示本地域,而不是根域,这个域用正斜线(/)字符表示。

反过来,可以使用 niload 实用工具将数据从普通文件导入到 NetInfo 数据库中。为了将位于/tmp/passwd.txt 的标准密码文件中的数据导入 NetInfo 数据库,可以执行下面的命令:

```
niload passwd .< /tmp/passwd.txt
```

在将数据从普通文件导入 NetInfo 数据库之前,应该编辑这个文件并确保以下几个条件:在这个文件中没有重复的 UID 或 GID 条目;shell 必须指向一个存在于系统上的 shell;主目录位于/Users 目录中(如果这个文件是从其他系统复制过来的)。一般情况下,主目录位于/home 目录下。

为了查看可以利用 niload 和 nidump 导入或导出的所有数据类型,可以单独执行 nidump 命令:

```
$ nidump
usage: nidump [-r] [-T timeout] {directory | format} [-t] domain
known formats:
    aliases
    bootptab
    bootparams
    ethers
    exports
    fstab
    group
    hosts
    networks
    passwd
    printcap
    protocols
    resolv.conf
    rpc
    services
```

普通文件所使用的标准格式通常都可以在 Unix 系统的/etc 目录中找到,例如/etc/passwd、/etc/group 和/etc/hosts 等。这些文件都是标准的格式,Apple 将它们称为已知格式(known format)。这种功能使得从其他 Unix 系统导入用户和其他通用资源、以及将这些信息导出到其他系统变得非常容易。一个具体的示例是,在很多大型网络中,有一个通用的主机文件,这个文件将被分配到所有的系统上以用于主机名解析。为了把这些信息保存到 Mac OS X 的 NetInfo 数据库,可以利用 niload 导入这个文件,而已知的主机文件格式可以称为 hosts。

20.5.3 NetInfo 数据库的备份和恢复

随着不断的收集,保存在 NetInfo 数据库中的关键信息会越来越多。为了让系统正常运行,这个数据库绝不能出现任何问题。出于这个原因,这个数据库每天都通过 cron job /etc/daily 备份。/var/backups/local.nidump 文件是由 cron job /etc/daily 创建的备份。

关于 NetInfo 的更多信息,可以查看 niutil、nidump 和 niload 的联机帮助文档,并从 <http://docs.info.apple.com/article.html?artnum=106416> 下载“Understanding and Using NetInfo”文档。

关于 Open Directory(开放目录)的更多信息,请访问网页 http://developer.apple.com/documentation/Networking/Conceptual/Open_Directory/Chapter1/chapter2_section_1.html。

20.5.4 系统启动

第 2 章讨论了 Unix 系统的启动过程、系统启动时运行的脚本以及服务是怎样启动的。Mac OS X 遵循了标准 Unix 系统类似的启动过程,然而,这个过程的组织方式有些不一样。实际上,Mac OS X 和其他 Unix 系统的启动过程很可能存在很大的差别。

在给机器加电的时候,计算机执行固件中的命令,类似于标准 PC 上的 BIOS。固件在完成工作后将控制权转移给系统的 BootX 加载器,这个加载器负责引导 Mach 微内核。接下来,系统初始化设备驱动器子程序 I/O Kit,然后安装根目录文件系统。这些操作完成之后,系统将加载 mach_init。mach_init 负责在 Mach 微内核中处理内务。最后,BSD init 进程开始运行,这个进程的 PID 是 1,和 Unix 系统的标准一样。

到此为止,系统已完全加载,所有指定的启动服务都已启动,并准备接收交互式的登录。

默认情况下,系统以图形模式启动,在屏幕上提供的信息非常少。如果出于排查问题或收集信息的目的希望观察启动过程的详细内容,可以在给系统加电的时候按下 CMD+V 键使系统进入详细模式(verbose mode)。

Unix 系统使用/etc/rc.d 或/etc/init.d 目录中的文件作为启动脚本,并在系统引导时在加载过程中使用/etc/inittab 文件导向 init 进程。但 Mac OS X 的启动过程与 Unix 系统的完全不一样,如前所述。启动的内容将在/System/Library/StartupItems/和/Library/StartupItems 两个目录中处理。

由 Apple 安装的服务安装在/System/Library/StartupItems 目录中,这些服务是作为 Mac OS X 的一部分安装的。和 System 目录中的所有内容一样,不应该修改这些文件,因为如果在下次运行 Software Update 应用程序的时候覆盖了这些文件,那么所做的任何修改都将丢失。在 Library 目录中有一个 Startup Items 文件夹,可以在这个文件夹中配置本地安装的服务,以便在系统启动时启动这些服务。/Library/StartupItems 文件夹可以用于所有的本地服务。

关于 I/O Kit 的更多信息,请访问站点 <http://developer.apple.com/hardware/iokit/>。

关于 Mac OS X 引导过程的更多信息,请访问站点 <http://developer.apple.com/documentation/MacOSX/Conceptual/BPSystemStartup/Concepts/BootProcess.html>。

20.5.5 文件结构上的差别

一直以来,Macintosh 都使用一种称为分叉(forked)的文件结构在系统上保存文件。在 Macintosh 上创建的所有文件都有两个分叉:数据分叉和资源分叉。并不需要同时使用两个分

叉，很多程序都把所有的数据保存到数据分叉中。资源分叉用于保存诸如图标、字体和类似附加信息等数据。

Windows 和 Unix 文件只由一个分叉组成，那就是数据分叉。Mac OS X 基本上只处理资源分叉，虽然它保留了与旧文件格式的兼容性。Mac OS X 中的应用程序以不同的方式保存资源，而且文件不再需要资源分叉。

在终端利用传统的实用工具 `mv` 和 `cp` 处理 Mac OS X 中的文件的难点在于，在使用这两个工具移动或复制文件时，资源分叉可能会和文件分离。Apple 提供了 `ditto` 工具(`/usr/bin/ditto`)，在命令行使用这个工具复制文件可以保留资源分叉。另外，如果安装了开发工具，那么在 `/Developer/Tools` 目录中就安装了 `MvMac` 和 `CpMac` 工具。

关于 Macintosh 上资源分叉的更详细的解释，请访问 Wikipedia 网站的在线页面：http://en.wikipedia.org/wiki/Resource_fork。

20.5.6 根用户账户

默认情况下，Mac OS X 中不能使用根账户。最好使用第 12 章中描述的 `sudo` 实用工具执行管理任务。如果发现需要使用 root shell，可以根据自己的偏爱执行 `sudo /bin/bash` 命令或 `sudo /bin/tsch` 命令。如果希望激活根账户以便进行交互式操作，可以通过执行命令 `sudo passwd root` 为这个账户设置一个密码，然后打开 NetInfo Manager 并从 Security 菜单中选择 Enable Root User 菜单从而激活根账户，如图 20-5 所示。

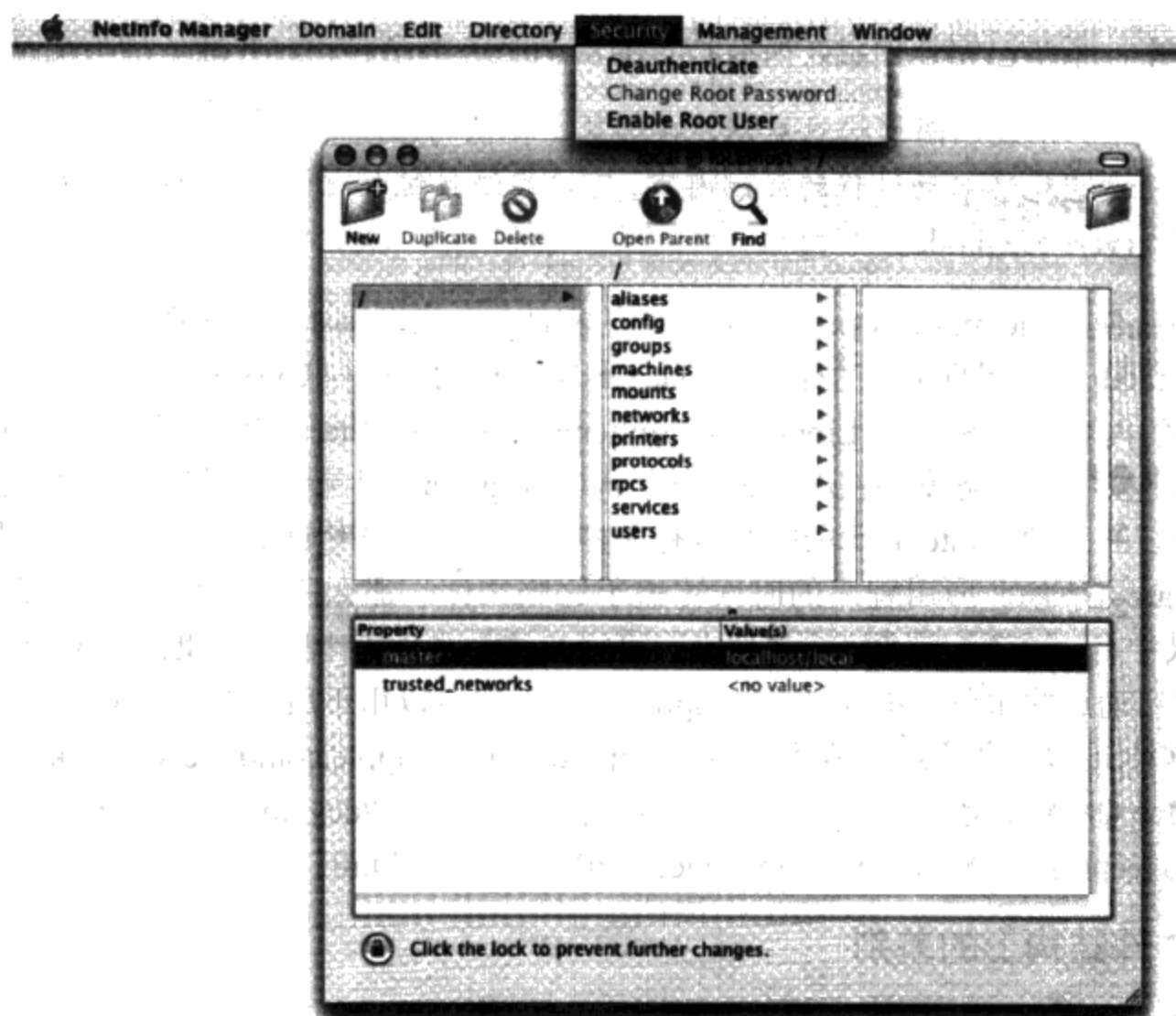


图 20-5

然后可以通过 3 种不同的方式登录到根账户：

- 使用传统的 `su -`。
- 输入命令 `sudo su -`。
- 在 Accounts Panel of System Preferences 的 Login Options 区域把 “Display Login Window as:” 设置成 “Name and Password”。然后登出，再以 root 作为用户名并输入为 root 设置的密码。

前两种方法允许在 Terminal 窗口中以根用户的身份执行命令，而第三种方法提供了一种完全的根用户登录方式，登录完成之后系统将显示桌面和停靠栏。记住，在使用第一个命令的时候，必须输入系统根用户密码；在使用第二个示例中的命令的时候，必须输入自己账户的密码。

20.6 小结

本章讨论了与 Apple 操作系统相关的问题，包括 Mac OS X——当前的 Macintosh 操作系统，这是一个基于 Unix 的系统。本章介绍了 Mac OS X 的历史，讨论了 Mac OS 9 和 Mac OS X 之间的差别，并讨论了 Mac OS X 和其他 Unix 操作系统之间的差别。

20.7 练习

- (1) 希望把 Mac OS X 的用户导出到 `passwd` 文件中以便在另一个 Unix 系统中使用，应该怎样做？
- (2) 如果希望在系统上安装只有一个用户能够使用的应用程序，应该在何处安装这个应用程序以便其他用户不能使用它？

第 21 章 转换：适用于 Windows 用户的 Unix

由于 Microsoft Windows 在桌面机/个人计算机市场上份额非常大，因此它成了很多人使用的第一个操作系统。大多数学习 Unix 的用户都是在理解了 MS-DOS/Windows 环境之后才学习 Unix 的。本章介绍 Unix 和 Windows/DOS 之间的相似之处，如果要从某个版本的 DOS 或 Windows XP(大多数 Windows 2000 中的命令和 Windows 98 中的一些命令)转向 Unix，本章将使这个转变过程变得更容易。

21.1 结构上的比较

在 Windows 和 Unix 的方法学之间存在一些基本的差别，特别是在两者的文件系统之间。在 Unix 中，正斜线(/)表示分隔符，并用在命令中表示一个新的目录层级。在 Windows 中，反斜线\具有相同的作用。例如，在 Unix 中切换目录到目录/var/log 的 cd 命令为：

```
cd /var/log
```

在 MS-DOS/Windows 中，导航到 c:\windows\system 目录可以输入：

```
cd c:\windows\system
```

在 Unix 中，前一条命令中的两个目录 var 和 log(在这个示例中 log 目录是 var 目录的子目录)由/分隔符标记出来。前面的/(称为根)是最顶层的目录，Unix 中的所有其他目录都是从它开始的。Windows 的目录是 windows 和 system(在这个示例中 system 是 windows 目录的子目录)。c:\表示这个文件系统的起点。

Windows 的顶层目录类型(与 Unix 中的根目录类似)有所不同，一般称为 c:\目录，它通常是这个文件系统的顶层目录。其他设备，例如辅助硬盘、网络驱动器、软盘驱动器、CD-ROM 驱动器和 USB 键驱动器通常都是它们自己的顶层目录，相当于 c:\。例如，典型的 Windows 系统使用 a:\表示软盘驱动器，用 d:\表示 CD-ROM 驱动器，所有这些目录和 c:\具有相同的层级。

在 Unix 中，/是最顶层的目录，没有对等目录。辅助设备，例如 CD-ROM、USB 和软盘驱动器都是/(根)目录的子目录，通常可以通过/mnt 导航到这些设备。所以软盘驱动器可能在 /mnt/floppy 目录，CD-ROM 可能在/mnt/cdrom 目录，等等(安装点(mountpoint)在不同的 Unix 实现之间有很大差别，第 4 章讨论了这个内容)。其他硬盘也是/(根)的子目录，通常将其附加(安装)在/mnt 目录下。

MS-DOS/Windows 的文件系统不是大小写敏感的，对系统而言，文件名 TESTFILE.txt 和 TestFile.txt 以及 testfile.txt 是一样的。可以输入这 3 个名字中的任何一个或其他大小写形式，

但它仍然指向同一个文件。在 Unix 中则不是这样，它是大小写敏感的。文件 TESTFILE.txt 和 TestFile.TXT 是两个完全不一样的文件，而 testfile.txt 在系统中又是另一个不同的文件。目录也是这样，所以在引用文件和目录的时候必须注意大小写。

Windows 中的 Administrator 账号基本上相当于 Unix 中的根账号。在 Unix 中，根(也称为超级用户(superuser)账号可以在系统上执行任何操作，而 Windows 的 Administrator 账号在执行可能毁坏系统的任务时将被系统阻止。

Windows 操作系统会保护用户和管理员，如果用户和管理员试图删除系统中的某个关键文件，Windows 就会阻止这样的错误。例如，如果管理员试图删除 c:\windows\explorer.exe，它将收到如图 21-1 所示的错误信息。

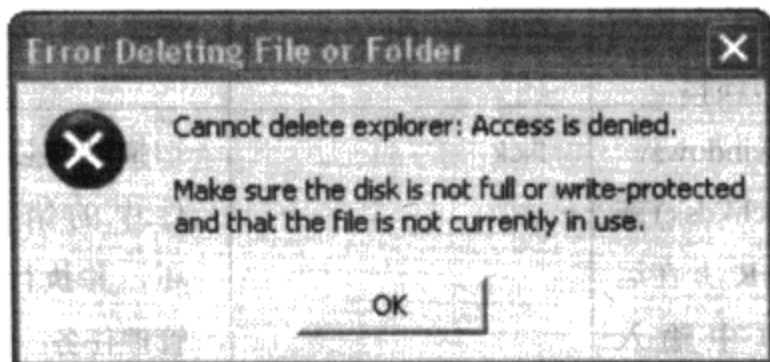


图 21-1

在 Unix 中，可以设置 rm 命令提供类似的功能(将 rm -i 设置成 rm 的别名，如第 5 章讨论的那样)，但默认情况下，系统将执行删除操作或执行任何所需的功能而没有任何提示。

Windows 的回收箱(Recycle Bin)在大多数 Unix 系统中都没有对等的程序，虽然在很多 Unix 的图形用户界面中实现了类似的功能。这个回收箱用于恢复“删除的”文件。例如，Linux KDE、Solaris GNOME、CDE 窗口管理器和 Mac OS X Aqua 都包含与回收箱对等的组件，但是大多数命令行 Unix shell 都不提供恢复删除文件的功能。

在 Unix 中，如果试图以根用户的身份删除/bin/sh，系统将毫不犹豫地删除它。Unix 假定如果以根用户登录，您应该明白自己在做什么，即使将完全删除系统，而 Windows 则试图保护系统，如图 21-2 所示。

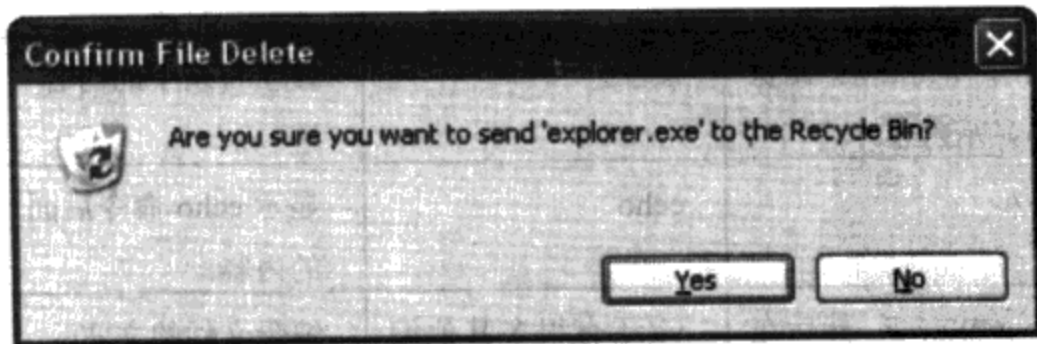


图 21-2

Windows 提供 MS-DOS 界面。要使用这个界面，按下 Win+R 运行对话框，并在该对话框中输入 cmd。表 21-1 显示了怎样从 Windows/Dos 命令行转变到 Unix、怎样在 Windows GUI 中执行操作、对等的 Unix 命令是什么、命名的简单描述，以及这些命令出现在哪一章。

表 21-1

Windows/Dos 命令	Windows GUI 操作步骤	Unix 命令	说 明	所 在 章 节
at	使用任务调度器(在控制面板中)	at 或 cron	在特定的时刻执行命令	第 11 章
attrib	Alt+Enter	chmod	修改文件或目录的属性	第 4 章
cd	使用 Windows 浏览器导航左边的目录	cd	切换目录	第 4 章
chdir 或 cd	查看浏览器的左边部分或窗口的标题栏	pwd	显示当前的工作目录	第 4 章
chkdsk	chkdsk(c:\windows\system32\cchkdsk) 或按下 Win+R 并在运行对话框中输入 chkdsk	fsck	扫描硬盘以检查文件中的错误和损坏, 并执行其他的管理任务	第 4 章
cls	Win+D	clear	清屏	—
copy	选中文件的时候按下 Ctrl+C, 然后按下 Ctrl+V 在新位置存放文件	cp	复制文件	第 6 章
date 和 / 或 time	在屏幕的右下角左键双击时间	date	显示时间	第 11 章
del 或 erase	Delete 键	rm	删除文件	第 6 章
rmdir /S	Delete 键	rm -R	以递归的方式删除文件和目录	第 6 章
dir	利用 Windows 浏览器显示内容	ls	显示目录中的内容	第 4 章
echo	N/A	echo	显示 echo 命令后面的内容	第 4 章
edit 或 edlin	按下 Win+R, 然后在对话框的文本框中输入 notepad	vi(还有很多其他编辑器, vi 是系统上最常用的)	编辑文件的文本	第 7 章
exit	Alt+F4	exit	结束 shell	第 2 章
fc 或 comp	N/A	diff	比较两个文件之间的差别	—

(续表)

Windows/Dos 命令	Windows GUI 操作步骤	Unix 命令	说 明	所 在 章 节
diskpart 或 fdisk	按下 Win+R, 然后输入 diskmgmt.msc	fdisk	硬盘分区	第 4 章
find	Win+F	grep(Unix 的 find 命令只查找文件名而不在文件内部查找)	在文件中搜索作为参数给出的单词	第 8 章
help 或 command?	F1	man	显示在线帮助文件	第 2 章
ipconfig	网络连接 (Network Connections)(在控制面板中)	ifconfig	显示和修改网络接口	第 16 章
mem	按下 Ctrl+Shift+Esc, 然后选择进程选项卡	vmstat(Solaris), top, free(Linux)	显示内存统计信息	第 14 章
mkdir	单击右键, 选择 New, 然后选择 Folder	mkdir	创建目录	第 4 章
more	N/A	more	按页显示输出(而不是滚动屏幕)	第 6 章
move	选中文件, 按下 Ctrl+X, 然后按下 Ctrl+V 把文件存放到新位置	mv	移动文件	第 4 章
net session	按下 Win+R, 然后输入 fsmgmt.msc	who	显示当前已登录到系统上的用户的信息	第 3 章
netstat	按下 Ctrl+Shift+Esc, 然后选择 Networking 选项卡	netstat	显示网络统计信息	第 16 章
nslookup	在 GUI 中没有对应的操作	nslookup	查看主机名	第 16 章
path	按下 Win+Break, 选择 Advanced 选项卡, 然后选择 Environment Variables 按钮修改 PATH 变量	echo \$PATH	显示可执行文件的目录和顺序	第 4 章
ping	在 GUI 中没有对应的操作	ping	向指定的主机传送网络包	第 16 章

(续表)

Windows/Dos 命令	Windows GUI 操作步骤	Unix 命令	说 明	所 在 章 节
print	Ctrl+P	lpr	打印命名文件	——
prompt	N/A	取决于使用哪个 shell	修改命令行提示符	第 5 章
rename	选中文件然后按下 F2	mv	重命名文件	第 4 章
rmdir	Delete	rmdir	删除目录	第 4 章
set	按下 Win+Break, 选择 Advanced 选项卡, 然 后选择 Environme nt Variables, 并查看系 统变量(System Varia bles)	set	显示环境变量	第 5 章
sort	在 Windows 浏览器 中 选 择 描 述 条 (description bar)以便 按描述排序	sort	以字符或数字为序 排列数据	第 8 章
tracert	在 GUI 中没有对应 的操作	tracert	显示到达特定主机 的网络路由	第 16 章
tree	选择文件夹图标, 浏 览器将显示目录树	tree 或 ls -R	以递归的方式显示 目录的内容	第 4 章
type	用相应的程序打开 文件	cat	显示文件或目录的 内容	第 6 章
ver	Win+Break	uname -a	显示当前操作系统的 版本信息	第 2 章
xcopy	选中目录, 按下 Ctrl+C, 然后按下 Ctrl+V 把文件夹复 制到新的位置	cp -R	以递归的方式复 制指定目录中的 所有文件	第 6 章

21.2 主要管理工具之间的比较

Unix 和 Windows 操作系统可以执行相似的功能, 或者拥有执行相似功能的文件。表 21-2 显示了 Windows 的管理工具和这些工具在大多数 Unix 系统中的对等工具。

表 21-2

Windows 文件或命令	Unix 文件	说 明
autoexec.bat 和 config.sys	/etc/init.d 或/etc/rc.number	与系统启动相关
C:\Documents and Settings \username	/home/ username 或/export/home/ username	系统上的用户空间, 用于存放个人文件
C:\Program Files	/opt(通常是这个目录, 但取决于 具体的 Unix 版本)	保存安装文件的默认位置
C:\windows\explorer.exe 或 C:\windows\system32\cmd.exe	/bin/sh、/bin/ksh、/bin/csh 等	系统的 shell
C:\ Windows\system32 和 C:\ windows\system	/etc、/bin、/sbin 等	保存所有对系统操作很关键的文件
C:\ Windows\system32\Config	/etc	系统的配置文件
C:\ Windows\system32\Spool	/var/spool	打印信息(printing information)
C:\Windows\Temp	/tmp	保存临时文件的目录
事件查看(Event View)(控制 面板管理工具)	/var/adm/messages(在不同的 Unix 系统上可能会有所不同)	查看系统产生的消息
网络连接(在控制面板中)	/etc/hostname.interface_name、 /etc/hosts、/etc/resolv.conf、/etc/nss witch.conf、/etc/nodename	用于设置网络连接
性能监视器(控制面板, 管理 工具)	top、netstat(命令)	监视系统的性能
任务计划(在控制面板中)	/var/cron/crontab 和/var/cron/at(在 不同的 Unix 系统上可能会有所 不同)	设置程序在某个特定的时间运行
服务(控制面板, 管理工具)	/etc/inetd.conf 和/etc/services	设置系统上可用的服务
用户账户(在控制面板中)	/etc/passwd 和/etc/shadow	设置用户账号

21.3 流行应用程序的比较

Windows 用户使用的大多数程序在 Unix 中都有对等的程序, 而且这些 Unix 程序通常都是免费的。表 21-3 列出了一些 Microsoft Windows 中比较常用的程序和这些程序在 Unix 中的对等程序。

表 21-3

Windows 程序	Unix 程序	是否包含在 Unix 中
Internet Explorer	Mozilla Firefox(http://mazilla.org), Netscape Navigator (http://netscape.com), Safari(包含在 Mac OS X 中), Konqueror(http://konqueror.org)	有时
Microsoft FrontPage	Quanta Plus(http://quanta.sourceforge.net), NVUC(http://nvu.com), Netscape Composer(和 Netscape Navigator 一起提供, http://netscape.com)和 OpenOffice.org(http://openoffice.org), 也包含了一个 Web 编辑器	否
Microsoft IIS	Apache(http://apache.org)	有时
Microsoft Money 或 Quicken	GnuCash(http://gnucash.org), MoneyDance(http://moneydance.com)	否
Microsoft Office	OpenOffice(http://openoffice.org), KOffice(http://koffice.org), GNOME Office(http://gnome.org/gnome-office)	是
Microsoft Outlook	Ximian Evolution(现在位于 Novell 下, http://novell.com/products/desktop/features/evolution.html)	否
Notepad 或 WordPad	vi	是
PGP	GnuPG(http://directory.fsf.org/gnuPG.html)	有时
Photoshop	GNU Image Manipulation Program(GIMP)(http://gimp.org)	有时
telnet	telnet	是
Winamp	XMMS(http://xmms.org)	否
Winzip 或 PKZIP	tar 和 gzip	是
WS FTP	GFTP(http://gftp.seul.org)	有时

要完全列出 Unix 的所有软件是不可能的, 绝大多数 Microsoft Windows 程序都有对等的 Unix 程序。下面的网站包含了大量的 Unix 软件文档, 利用这些文档有助于寻找所需的软件:

- **Sourceforge**——www.sourceforge.net
- **Freshmeat**——<http://freshmeat.net>(该网站非常偏向 Linux, 但也有其他 Unix 系统的软件)
- **Free Software Foundation**——www.fsf.org
- **SunFreeware**(Sun 特有的)——www.sunfreeware.com
- **Apple**(Mac OS X 特有的)——www.apple.com/support/downloads

还可以在其他网站找到 Unix 软件, 可以使用最喜欢的搜索引擎进行搜索。

21.4 在 Windows 中使用 Unix

有时候可能想尝试一下 Unix，却又必须使用 Microsoft Windows 环境，或者希望在完全放弃 Microsoft Windows 系统之前看看 Unix。可以通过免费软件或商业软件以多种方法实现这个想法。下面列出其中的一些软件：

- **VMWare**(www.vmware.com)——这个程序允许您在主操作系统(可能是 Windows)内部运行多个操作系统。它在程序内部模拟另一个系统以便允许安装一个新的操作系统，这个新系统和本地(主)操作系统是完全分隔开的。这个程序的价格相当昂贵。
- **Microsoft Virtual PC**(www.microsoft.com/windows/virtualpc/default.msp)——这个程序允许在主操作系统(可能是 Windows)内部运行多个操作系统。它在程序内部模拟另一个系统以便允许安装一个新的操作系统，这个新系统和本地(主)操作系统是完全分隔开的。这个程序的价格相当昂贵。
- **Bochs**(<http://bochs.sourceforge.net>)——这个程序允许在主操作系统(可能是 Windows)内部运行多个操作系统。它在程序内部模拟另一个系统以便允许安装一个新的操作系统，这个新系统和本地(主)操作系统是完全分隔开的。这个程序是开放源代码，是个免费软件。
- **Knoppix**(www.knopper.net/knoppix/index-en.html)——这是一个在 CD-ROM 中的、自包含的 Linux 完整发布。可以从 CD-ROM 启动，无需覆盖或影响原来的 Windows 或其他操作系统，就可以运行一个完整的 Linux 操作系统。这允许在系统上预先查看 Linux 而不用进行任何长期的修改，只要取出 CD-ROM 重启计算机，就可以完全恢复 Windows 的功能，而不会产生任何影响。这个程序是开放源代码，是个免费软件。
- **Cygwin**(www.cygwin.com)——这个程序允许在 Windows 中执行很多 Unix(Linux)命令而无需要重新启动。甚至可以运行完整的 Unix 窗口环境而维持普通的 Microsoft Windows 登录对话。这个程序是开放源代码，是个免费软件。
- **U/Win**(www.research.att.com/sw/tools/uwin)——这个程序允许在 Windows 中执行很多 Unix(Linux)命令而无需要重新启动。这个程序是开放源代码，是个免费软件。

很多 Windows 用户都使用 Cygwin，这个软件允许用户在维持 Windows 环境的同时获得一些与 Linux 相同的功能，并运行一些为 Linux 创建的软件。下面的“实战”安装 Cygwin 以便运行在本书中学到的 Unix 命令，同时不会影响 Windows 环境的舒适性。

实战

使用 Cygwin

这个练习演示使用其中一个程序，这些程序允许在 Windows 中执行 Unix 命令。该程序的使用非常简单。

- (1) 从 Web 站点 <http://cygwin.com/setup.exe> 下载 Cygwin 安装程序。
- (2) 双击这个文件，开始 Internet 安装，此时应该看到与图 21-3 所示的窗口类似的弹出窗口。

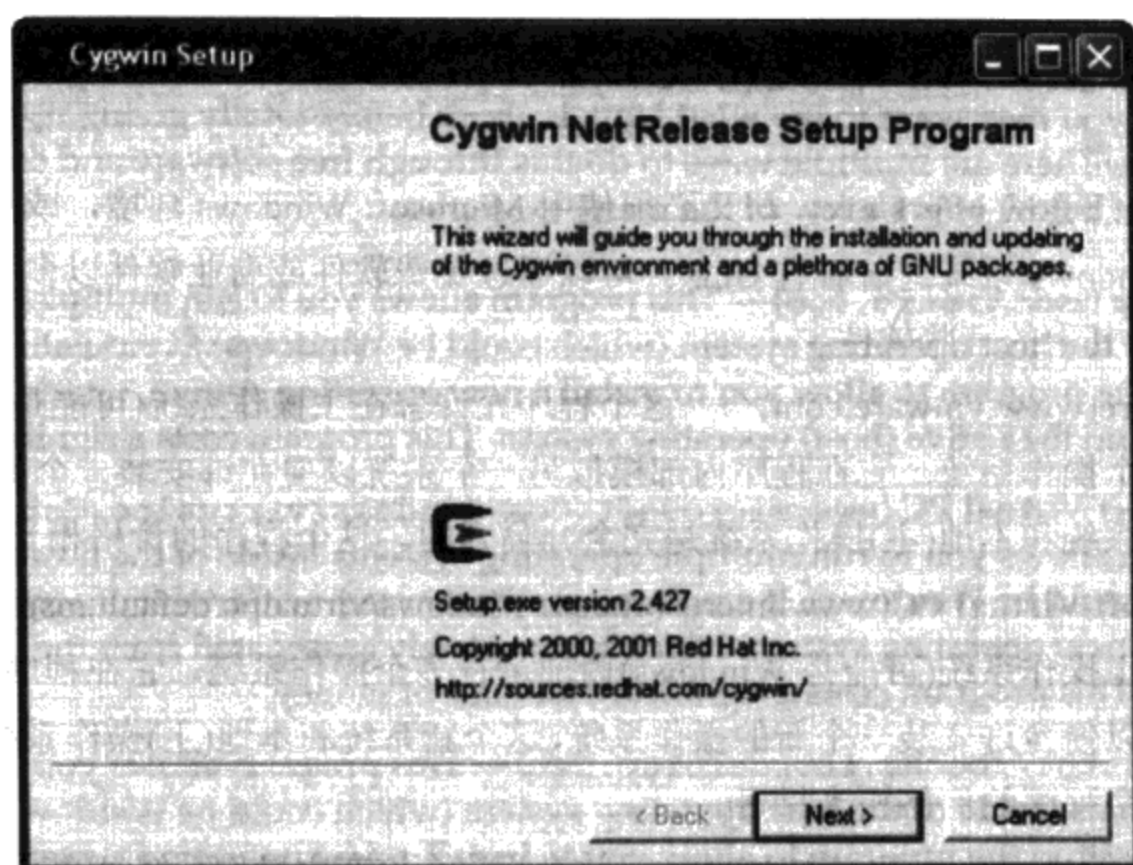


图 21-3

3. 单击 Next 按钮，将看到与图 21-4 类似的窗口。

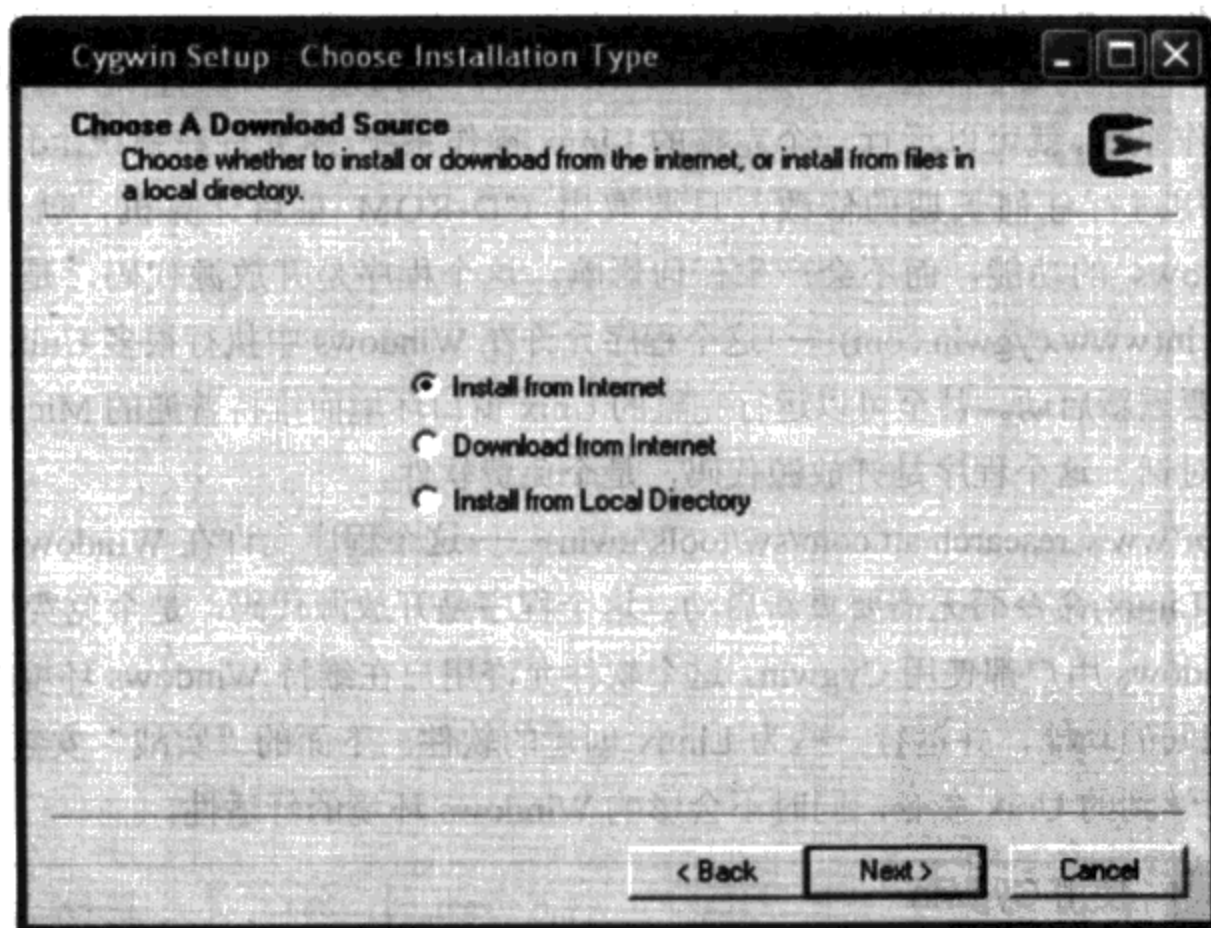


图 21-4

4. 选择 Install from Internet 以下载所需的文件。现在应该看到一个与图 21-5 类似的窗口。

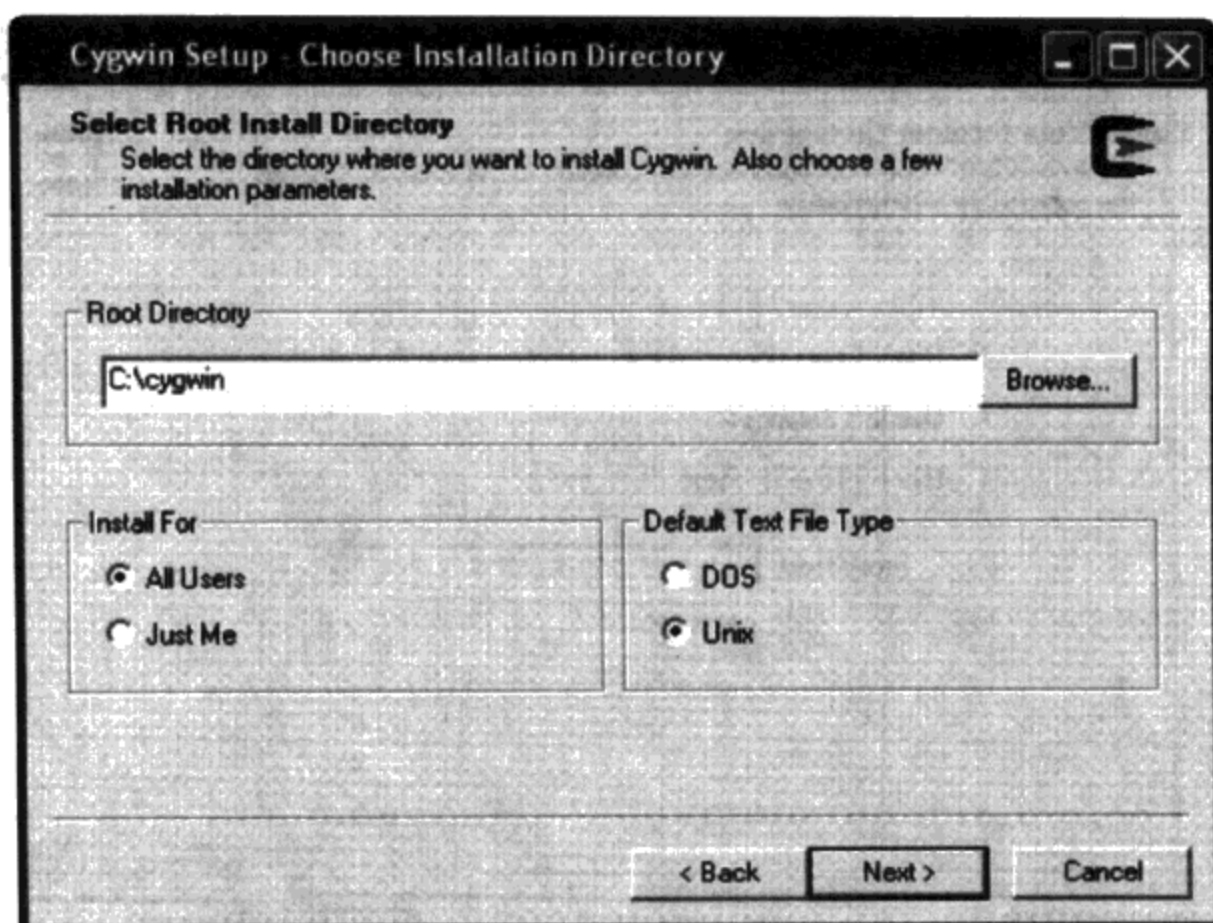


图 21-5

5. 应该选择默认的设置，但如果希望选择其他选项，可以在这里选择。单击 Next 按钮继续安装。此时屏幕上将出现与图 21-6 类似的对话框，在这里可以选择保存安装文件的本地包目录(Local Package Directory)。

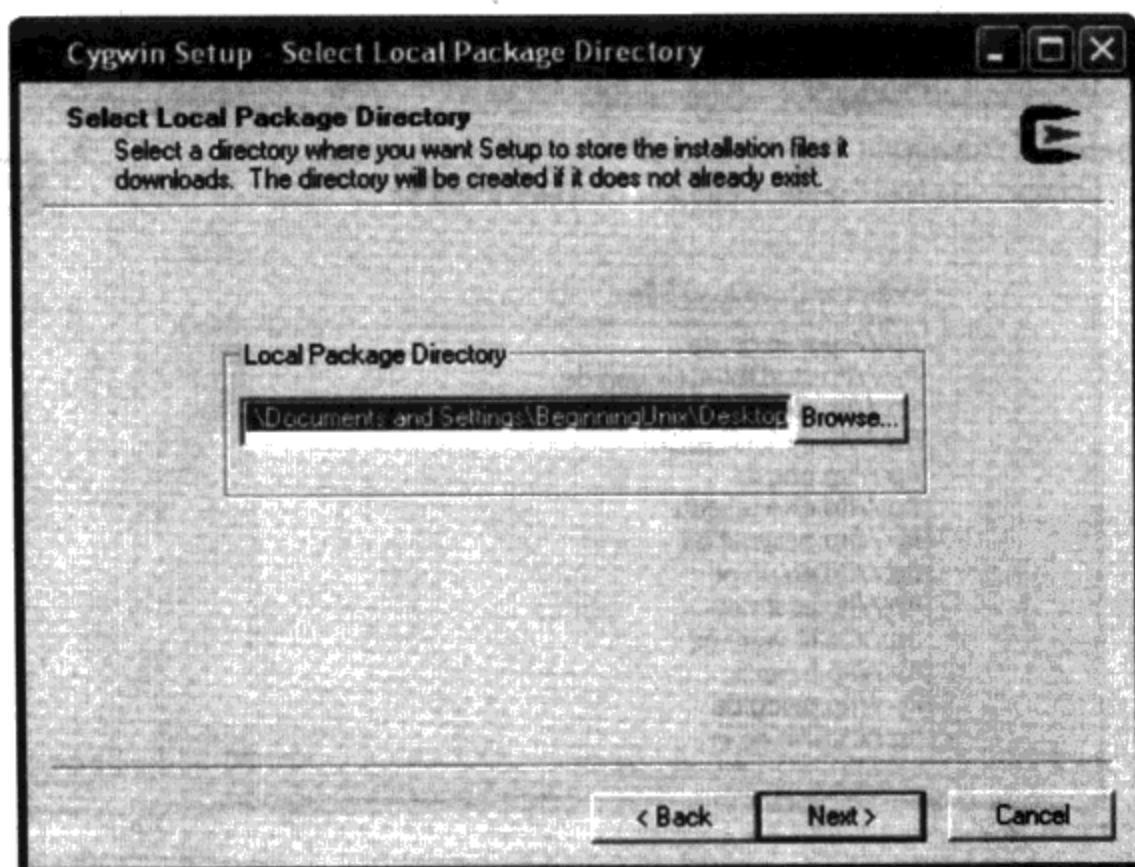


图 21-6

6. 在接下来的对话框中(如图 21-7 所示)，安装程序会提示选择所需的连接类型，一般情况下，直接使用默认设置。

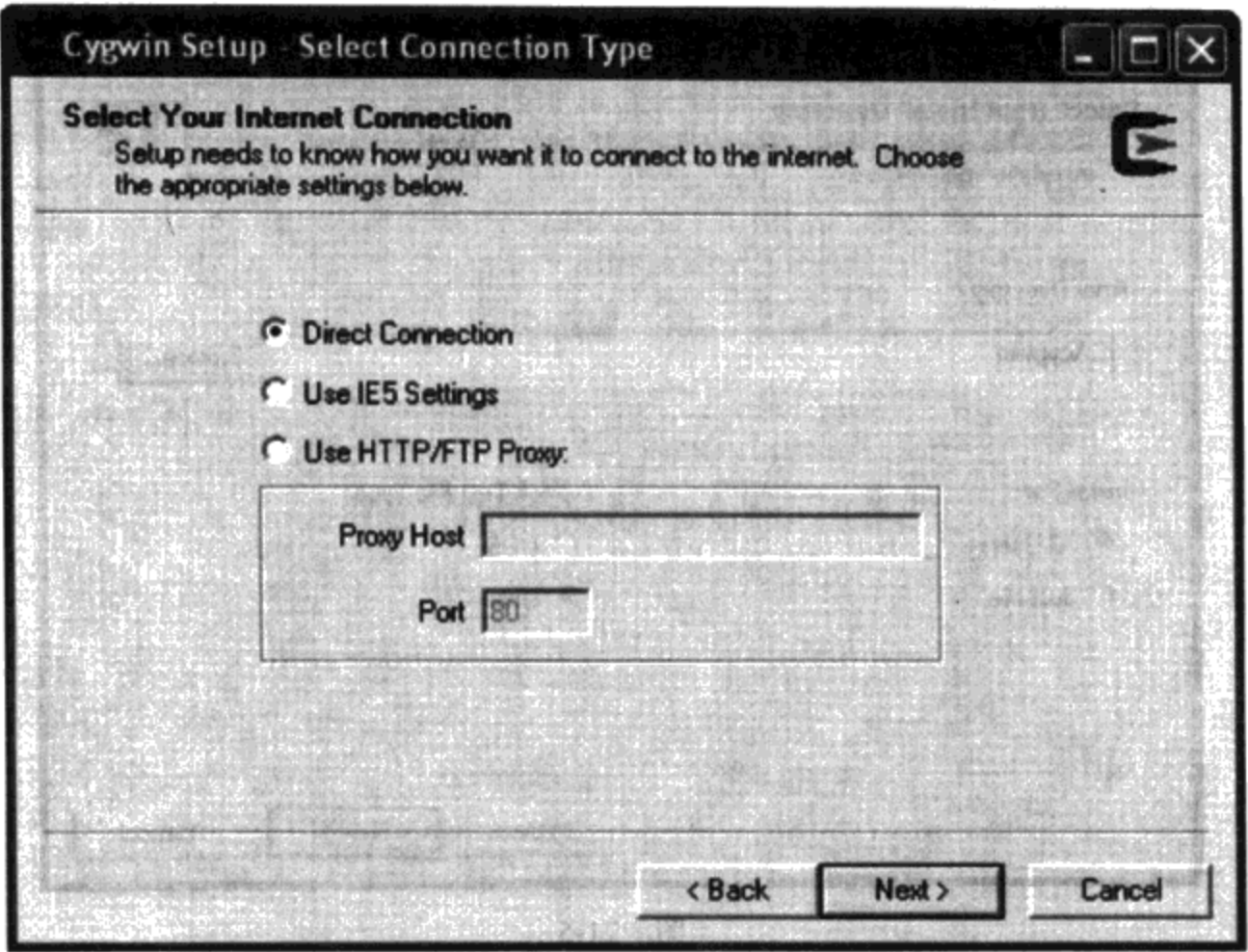


图 21-7

7. Cygwin 接下来显示一组下载站点供用户选择。可以选择偏爱的下载站点，如图 21-8 所示。

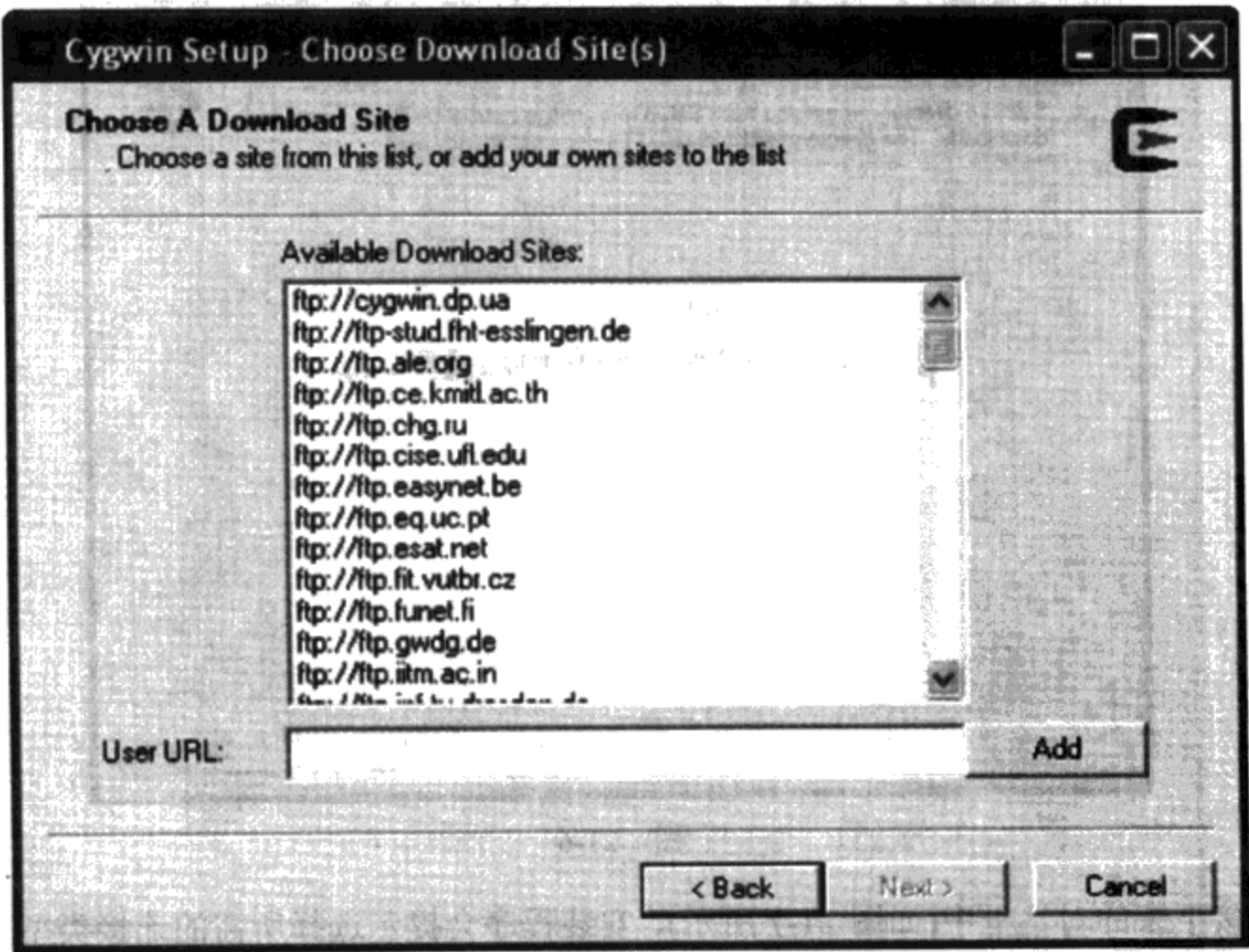


图 21-8

8. 下载开始，如图 21-9 所示。安装程序将在后台创建一组包，并在创建完成之后显示这些包。

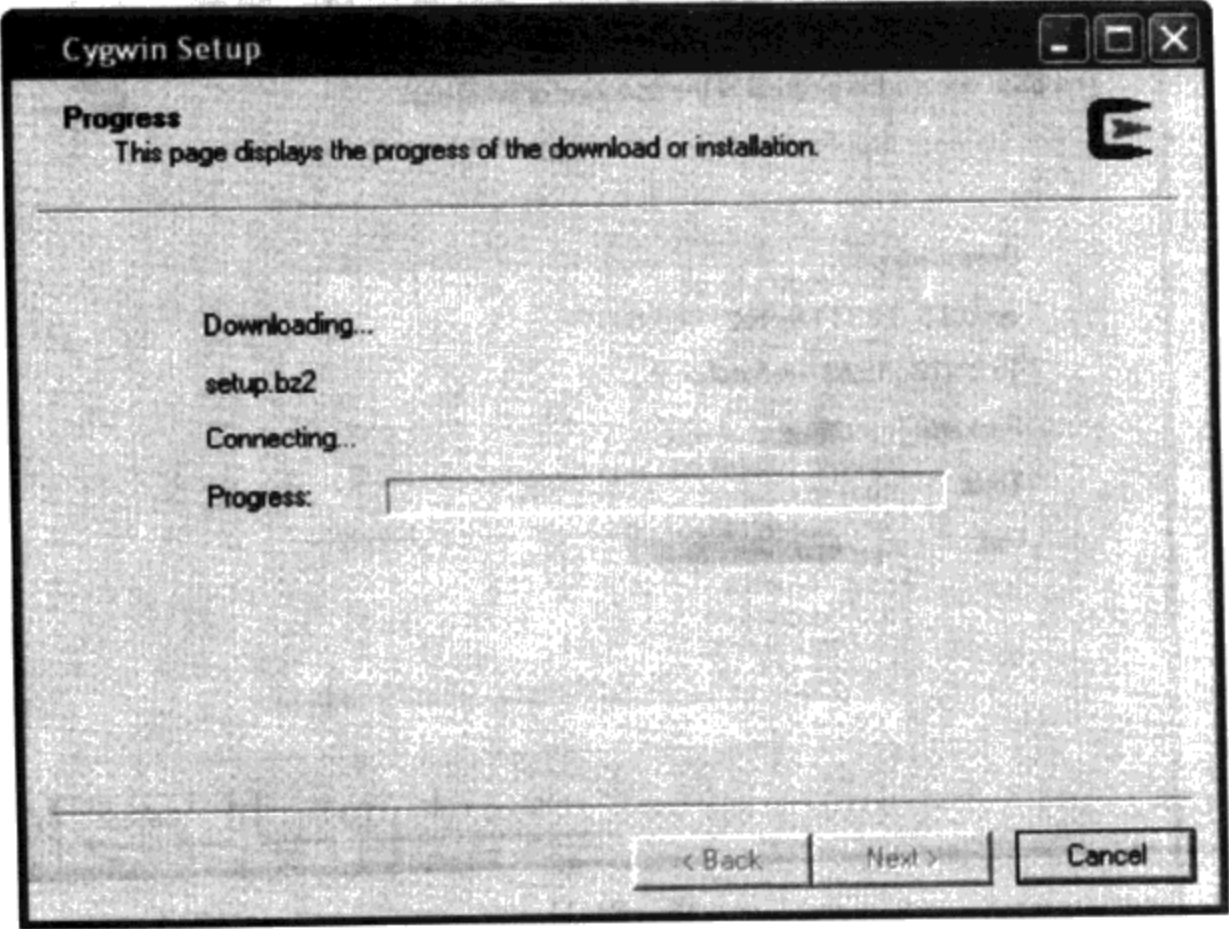


图 21-9

9. 图 21-10 显示了包选择界面。在这个练习中，如果访问 Internet 的速度很快，可以选择所有的包以便获得最多的功能；否则，应该只选择感兴趣的包。在这里，可以指定需要安装的包组或单个包(或程序)。

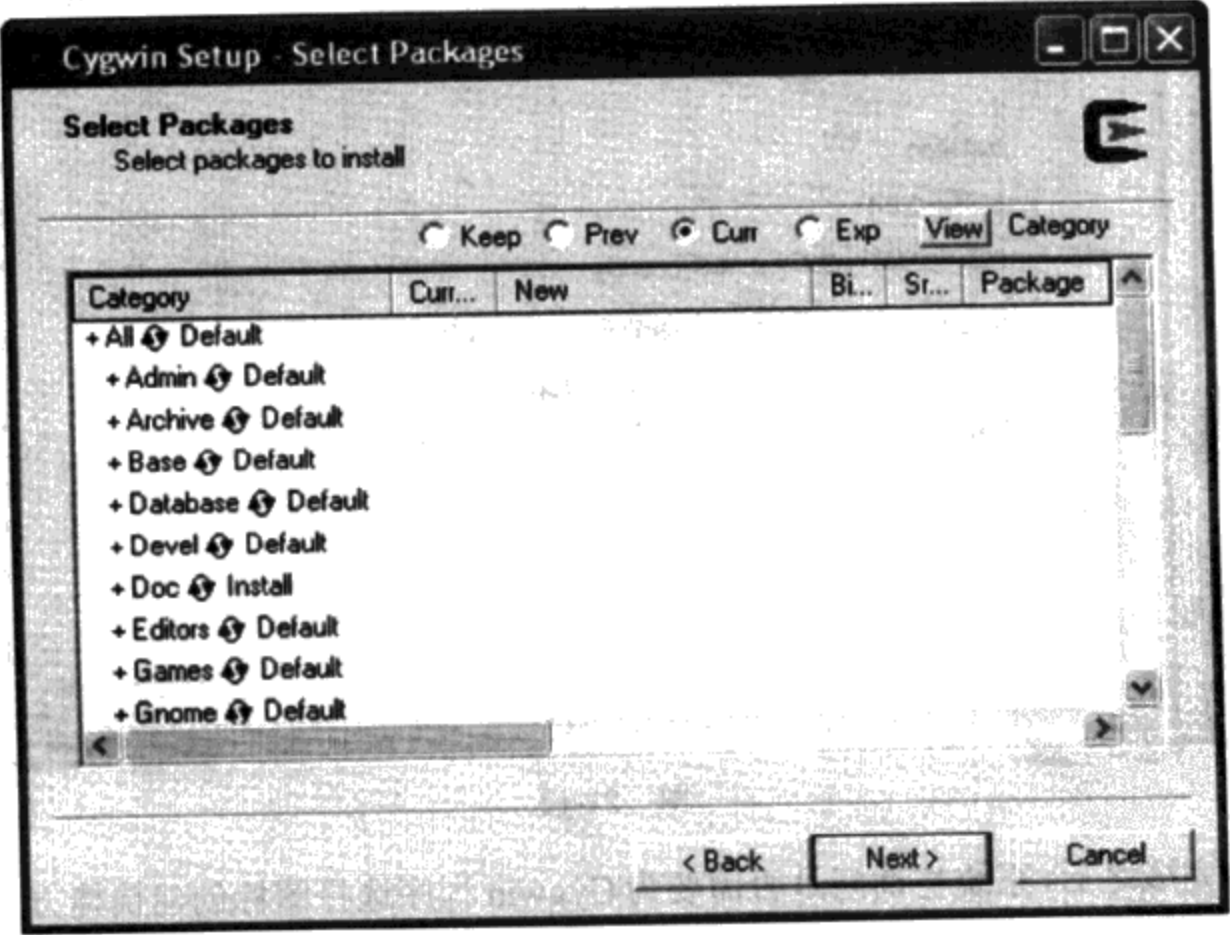


图 21-10

10. 在选择所需的包之后，将看到如图 21-11 所示的下载进程对话框。

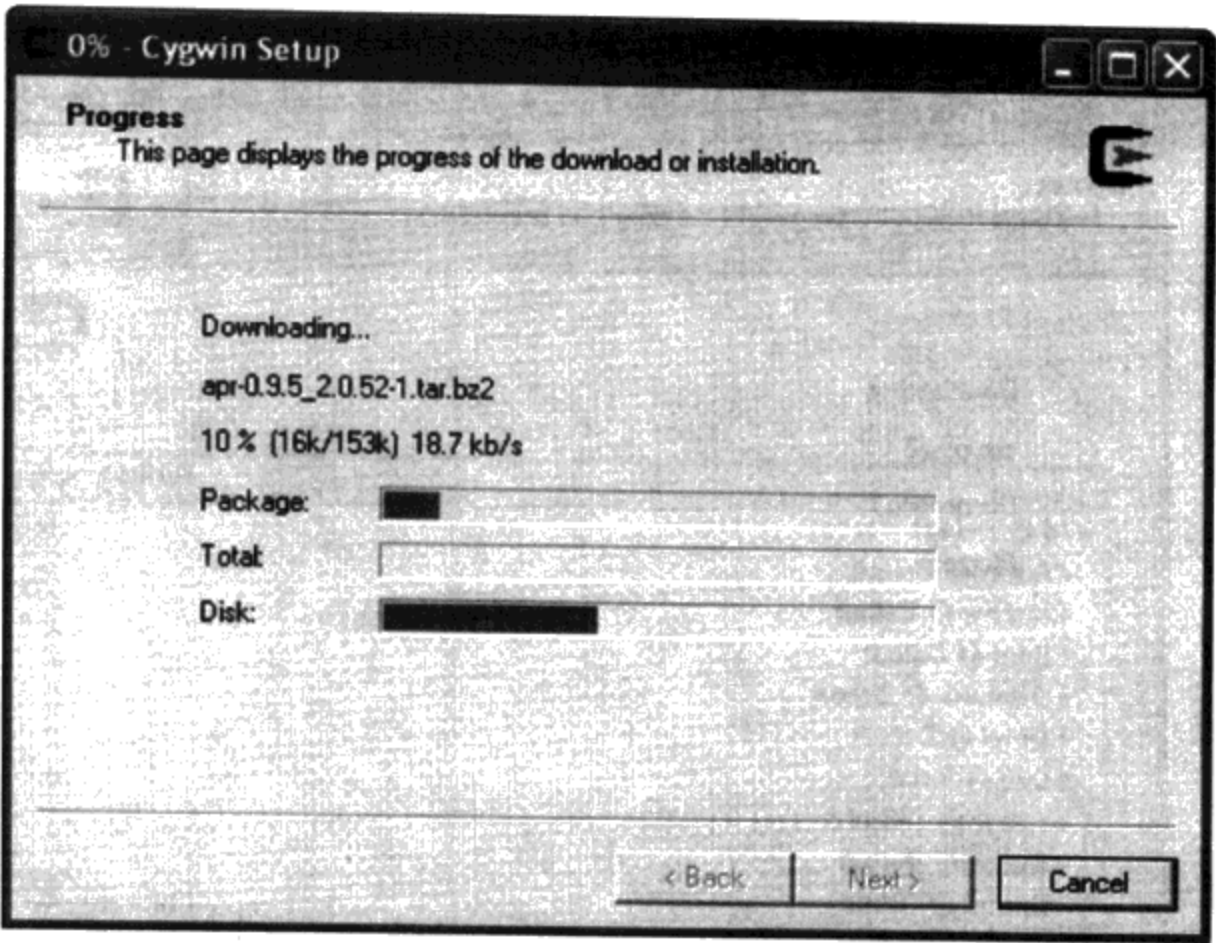


图 21-11

11. 在下载完包之后，对话框将显示安装进度(如图 21-12 所示)。

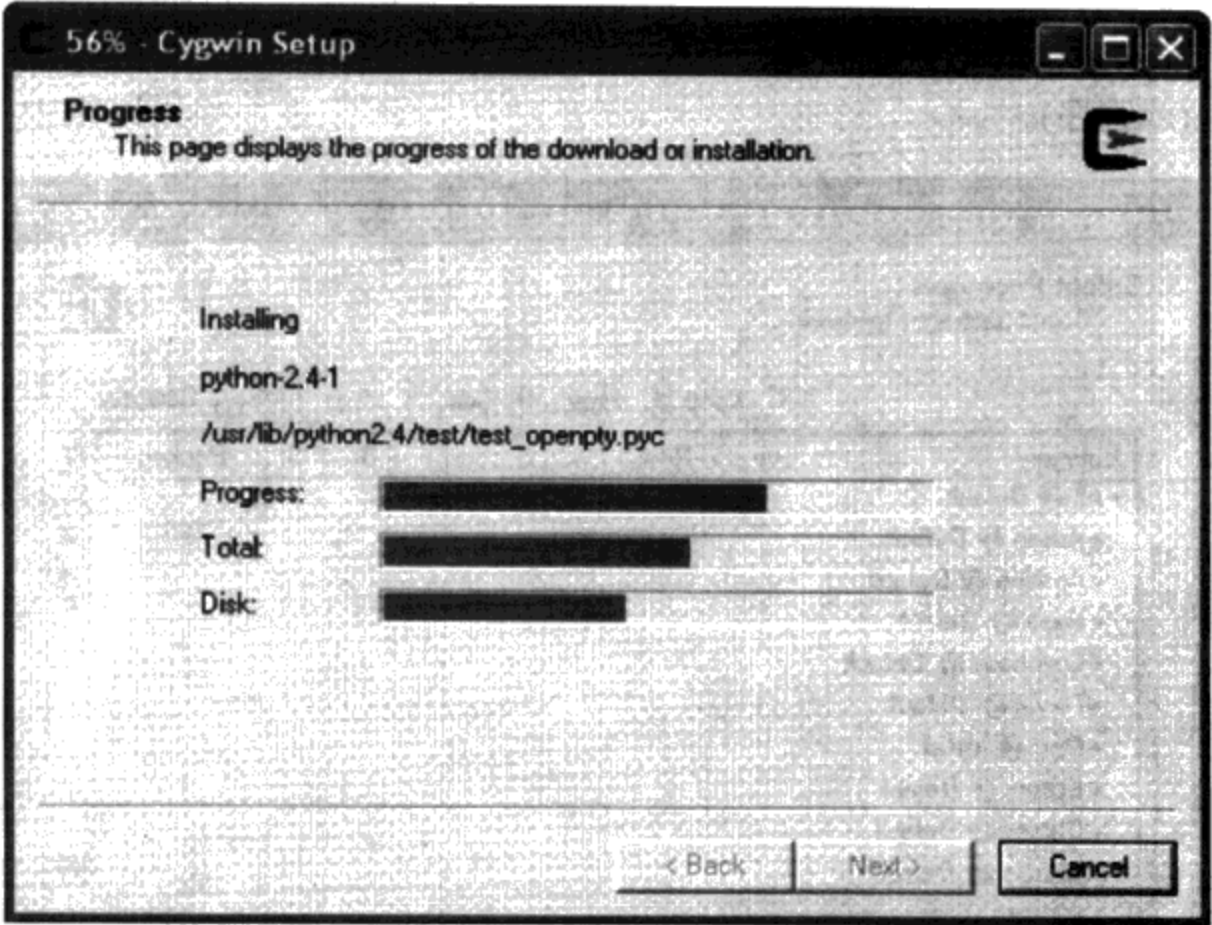


图 21-12

12. 安装完成之后，系统将询问是否需要为 Cygwin 程序选择图标创建快捷方式，如图 21-13 所示。

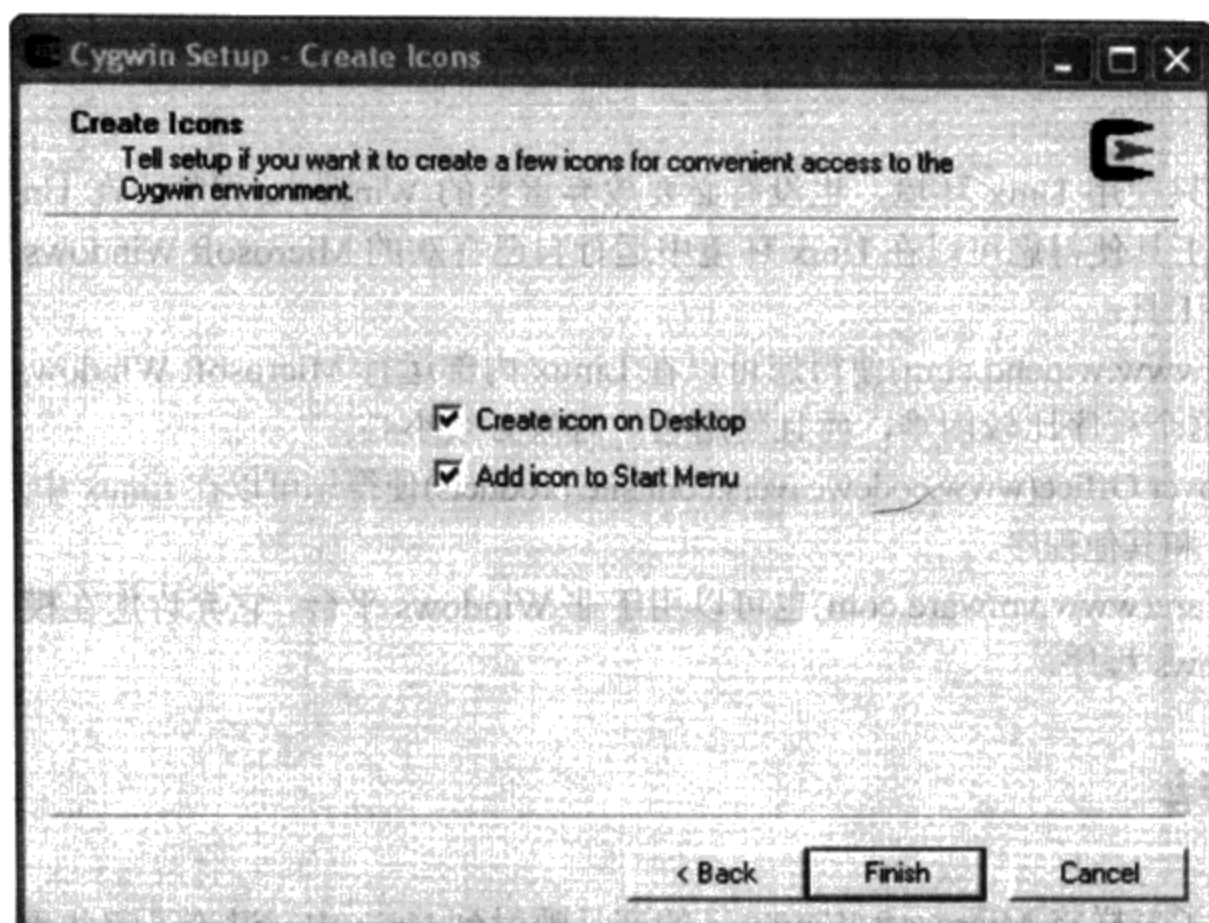


图 21-13

13. 安装结束后，选择桌面上的 Cygwin 图标启动这个程序。系统上将显示 shell 提示符(与图 21-14 所示的窗口类似)。在该提示符上，可以运行本书前面已讨论的大多数命令。

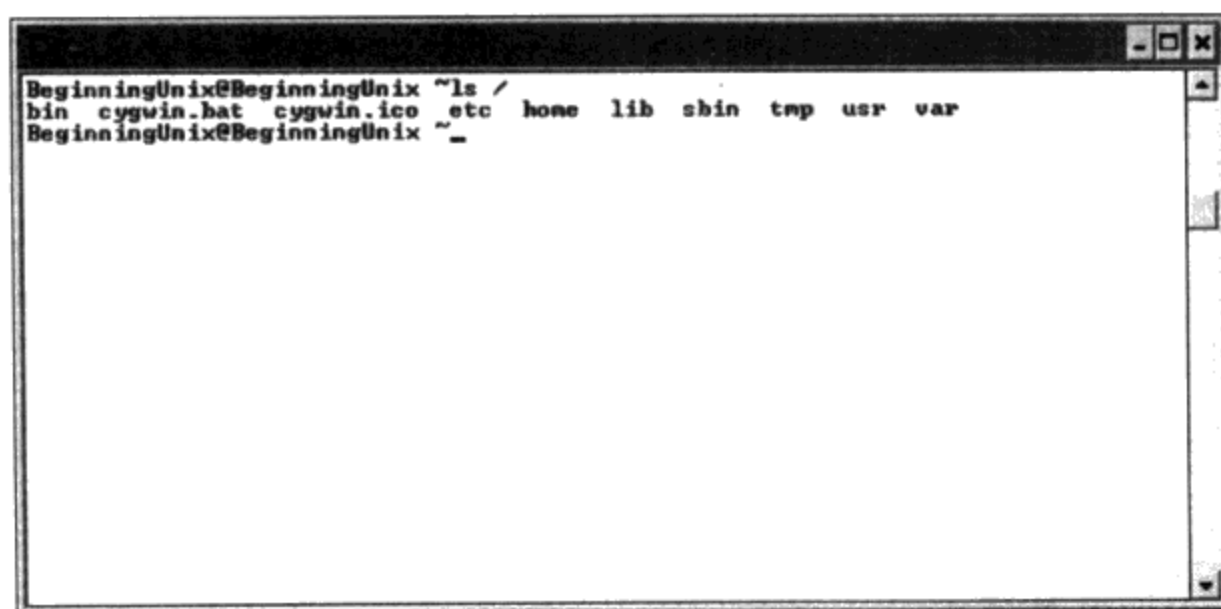


图 21-14

工作原理

Cygwin 允许您模拟 Linux API，通过 Microsoft Windows DLL 提供类似 Linux 的功能。可以在当前的 Windows 会话中运行为 Linux 创建的图形界面和很多程序。

更多关于使用 Cygwin 的信息请访问网站 <http://cygwin.com/cygwin-ug-net/cygwin-ug-net.html>。

21.5 在 Unix 中使用 Windows

如果决定只使用 Unix 环境，也没有必要放弃重要的 Windows 程序。在 Unix 社区中有一些工具，这些工具使得您可以在 Unix 环境中运行自己喜欢的 Microsoft Windows 程序。下面列出几个流行的工具：

- Wine(www.winehq.com)使得您可以在 Linux 内部运行 Microsoft Windows 程序。新用户使用这个程序比较困难，而且使用它的时候必须小心。
- Crossover Office(www.codeweavers.com/site/products)使得您可以在 Linux 中运行 Microsoft Office 和其他程序。
- VMWare(www.vmware.com)也可以用于非 Windows 平台，它允许您在模拟环境中运行 Windows 程序。

21.6 小结

本章学习了怎样将 Microsoft Windows 的产品映射到 Unix 中，讨论了哪些程序相似、文件结构如何不同以及程序之间有哪些相似之处。另外还学习了在 Windows 环境中怎样使用 Unix 以及利用什么程序在 Unix 环境中使用 Windows。利用这些信息，可以把当前的知识转换到 Unix 系统。

附录 A 练习题答案

第 3 章

练习 1 答案

1. /etc/passwd。包含有关账户名、用户 ID、组 ID、主目录和 shell 的信息。
2. /etc/shadow。包含与/etc/passwd 中的账户名对应的加密密码以及密码和账龄(account-aging)信息。
3. /etc/group。标识账户可用的不同组。

练习 2 答案

```
# adduser -c "C Jane Doe" -d /export/home/jdoe -e 2005/05/31 -f 10 -g employees
-Ginfo_tech_1, info_tech_2, info_tech_5 -m -s /bin/ksh -u 1000 jdoe
# passwd jdoe
Changing password for user jdoe.
New UNIX password:
Retype UNIX password:
passwd: all authentication tokens updated successfully.
#
```

第 4 章

练习答案

```
chmod 755 samplefile
```

第 5 章

练习 1 答案

```
PS1="[\d /u]$ "
```

练习 2 答案

找到该程序的可执行文件。将它的目录添加到 PATH 变量中, 该命令类似于: `PATH=$PATH:/newdirectory`

练习 3 答案

将这个命令添加到所选 shell 的恰当的运行控制文件中。

第 6 章

练习答案

```
ls /home/k* | wc > k-users-files
```

第 7 章

练习 1 答案

如果按下 Esc 键两次，就将进入命令模式。

练习 2 答案

在命令模式下，输入 / computer 或 ? computer 分别向前或向后搜索单词 computer(如果希望只查找完整的 computer 实例，一定要在“computer”字符串的前面和后面各加上一个空格)。如果要在相反的方向上搜索，输入大写的 N。

练习 3 答案

在命令模式下，在要复制的 5 行文本的第一行输入 5yy，然后输入 10j 向下跳 10 行。输入小写的 p 在当前行的下面以粘贴前面选择的 5 行文本。

练习 4 答案

```
:1,$s/person/human/g
```

第 8 章

练习 1 答案

下面的两个命令都可以：

```
cat /etc/inetd.conf | grep telnet.d
grep telnet.d /etc/inetd.conf
```

练习 2 答案

```
find /tmp -size +5000000k -atime 4
```

第 9 章

练习 1 答案

```
awk -f: '{print $4 "" $1""$2""$3}' | sort > addresses-sorted.txt
```

练习 2 答案

```
s/St\./Street/g
s/Ave\./Avenue/g
s/Rd\./Road/g
```



```
s/Dr\./Drive/g
s/N\./North/g
s/S\./South/g
s/E\./East/g
s/W\./West/g
```

练习 3 答案

```
sed 's/\(NY\), \(NEW York\)/\2, \1/' addresses.txt > addresses-new.txt
```

第 11 章

练习答案

```
crontab -e
** /2**userid ls | mail -s "Directory Listing for Home Directory" userid
```

第 12 章

练习答案

```
# Host alias specification
Host_Alias      Backup_Host=linux_backup
# User alias specification
User_Alias      Administrator=jdoe
# Cmnd alias specification
Cinnd_Alias     Backup_Script=/bin/su backupuser
# User privilege specification
Administrator   Backup_Host=Backup_Script
```

第 13 章

练习 1 答案

```
#####
# filescript.sh
#
# Dec. 9, 2004
#
# A program to find certain types of files, and report certain types
# of information, selectable by the user.
#
# CHANGELOG:
#
# 12/9/04 -- This is version 1. No changes at this point
#
#####

#!/bin/sh

# Get directory from command line. Otherwise, directory is current.

if [ $1 ] # NOTE: This form of the 'test' command returns
```

```

        # true, as long as the value specified actually
        # exists.
then
    DIR=$1
else
    DIR='pwd'
fi

cd $DIR
echo "Working directory is $DIR"

# Prompt for an option from the keyboard.
echo "What information would you like to know (size, permission, owner,
group, all)?"

read OPTION

# If the option is "size", find all ".txt" files in directory and print
# their names and file sizes.
#
# If the option is "permission", print the permissions held by the file
#
# If the option is "owner", print the owner of the file
#
# If the option is "group" print the group
#
# If the option is "all", print all of the above info
if [ $OPTION = "size" ]
then
    ls -la *.txt | awk '{print $9": "$5}'

elif [ $OPTION = "permission" ]
then
    ls -la *.txt | awk '{print $9": "$1}'
elif [ $OPTION = "owner" ]
then
    ls -la *.txt | awk '{print $9": "$3}'
elif [ $OPTION = "group" ]
then
    ls -la *.txt | awk '{print $9": "$4}'
elif [ $OPTION = "all" ]
then
    ls -la *.txt | awk '{print $9": "$1", "$3", "$4", "$5}'
else
    echo "Must be size, permission, owner, group, or all."

fi

```

练习 2 答案

虽然练习 1 答案中的 `elif` 语句链可以运行，但是看起来有些笨重。使用 `case` 可以提供一个较简洁的解决方案，如下所示：

```
#####
# case $OPTION in
#     "size")
#         ls -la *.txt | awk '{print $9": "$5}'
#     ;;
#     "permission")
#         ls -la *.txt | awk '{print $9": "$1}'
#     ;;
#     "owner")
#         ls -la *.txt | awk '{print $9": "$3}'
#     ;;
#     "group")
#         ls -la *.txt | awk '{print $9": "$4}'
#     ;;
#     "all")
#         ls -la *.txt | awk '{print $9": "$1", "$3", "$4", "$5}'
#     ;;
#     *)
#         echo "Must be size, permission, owner, group, or all."
#     ;;
# esac
#
#####
```

第 14 章

练习 1 答案

下面的函数将传递给它的两个值相加。编写代码验证传递给该函数的参数数量正确，而且是正确的数据类型。将该函数放入 `.bashrc` 中以便所有的脚本都可以使用它：

```
#!/bin/bash

total () {
    sum='expr $1 + $2'
}

exit 0
```

练习 2 答案

按如下方式使用 `tail` 命令和 IO 重定向操作符：

```
#!/bin/bash

tail /var/log/some_file filename

exit 0
```

将 `some_file` 替换成 `/var/log` 目录中需要监视的日志文件。

练习 3 答案

```
#!/bin/bash
```

```

badsum() {
    result = 'expr $1 / $2'
}

function cleanup() {
    echo "You have attempted to divide by 0. You are either very brave, or very
        silly... "
}

exit 1

}
trap cleanup 8

badsum 3 0

exit 0

```

练习 4 答案

作为根用户创建并运行如下命令：

```

#!/bin/bash
echo "The date is: $(date +%c)"
echo
echo "The following users are logged on: $(who | more)"
echo
echo "The following files are being used: $(lsof | more)"

wall «End
Good day! The Administrator is now in. If you have any queries please give me a
shout!
End

exit 0

```

第 15 章

练习 1 答案

在 syslog.conf 中添加如下文本：

```
kern.alert                                /dev/console
```

练习 2 答案

```
mail.debug                                @horatio
```

练习 3 答案

```

watchfor /INVALID/
echo
bell
mail addresses=root@localhost, subject=" Login attempted with incorrect username"

```

第 16 章

练习答案

首先，创建一个名为 `top_process.awk` 的 `awk` 脚本并输入如下代码：

```
/^ *[0-9]/      {
top_cpu_usage = 0
top_command = 0
top_user = 0

for (line = 0; line <= NR; ++line) {
if (top_cpu_usage > $3) {
    top_cpu_usage = $3
    top_command = $11
    top_user = $1
}
}

printf "Current Top Process: " top_command ", by user: " top_user ", at:
" top_cpu_usage " \n"
}
```

接下来创建一个名为 `top_process.sh` 的 `shell` 脚本并输入如下代码：

```
#!/bin/bash
# run traceroute and parse the data with top_process.awk
ps auwx > process_results.data
awk -f top_process.awk process_results.data
```

最后，在本地 `crontab` 文件中输入如下代码以自动执行 `shell` 脚本：

```
# run our shell script for testing the network
*/2 * * * * userid cd ~/bin; ./top_process.sh | mail -s "System Performance Data"
userid
```

第 17 章

练习 1 答案

```
#!/usr/bin/perl
use warnings;
open (FILE, '/etc/passwd');
    @lines = <FILE>;
close (FILE);
foreach $line (@lines) {
    (@fields) = split /:/, $line;
    print "Username: ".$fields[0];
    print "Userid: ".$fields[2];
    print "\n\n";
}

exit (0);
```

练习 2 答案

调试器将在不同的代码行结束执行，因为每次调用都提供了不同的代码分支。

练习 3 答案

子程序测试中的 else 测试应该从：

```
} else {  
    # There was a error in the input; generate an error message  
    warn "Unknown input";  
}
```

修改成：

```
} else {  
    return ('no numeric');  
}
```

第 18 章

练习答案

```
tar -cvzf /tmp/etc_backup /etc
```

或

```
tar -cvf /tmp/etc_backup /etc  
gzip /tmp/etc_backup
```

第 19 章

练习 1 答案

1.
 - a. `http: //catb.org/~esr/fetchmail/fetchmail-6.2.5.tar.gz`
 - b. `tar xvfz fetchmail-6.2.5.tar.gz`
 - c. `cd fetchmail-6.2.5`
`more INSTALL`
 - d. `./configure`
 - e. `make`
 - f. `su root -c "make install"` (在命令行输入超级用户密码)
 - g. `man fetchmail` (该命令的后面显示联机帮助文档)

练习 2 答案

2.
 - a. `www.acme.com/software/micro_httpd/micro_httpd_14dec2001.tar.gz`
 - b. `tar xvfz micro_httpd_14dec2001.tar.gz`
 - c. `cd micro_httpd`
`more README`
 - d. `make`
 - e. `cp micro-httpd desired-location`
 - f. `man inet.d` 或 `man xinet.d` (接下来的说明与具体安装的操作系统有关)

第 20 章

练习 1 答案

```
nidump passwd . passwd >> /tmp/passwd
```

练习 2 答案

应该将应用程序安装到/Users/username/Applications 目录中，将 username 替换成用户的简短用户名，该应用程序将提供给这个用户。

附录 B 一些有用的 Unix 网站

Unix 基础知识

- Bell Labs Unix 概述——www.bell-labs.com/history/unix/tutorial.html
- Darwin ——<http://developer.apple.com/darwin/projects/darwin/>
- GNU——<http://gnu.org>
- Linux 文档项目——<http://en.tldp.org/>
- Linux.com —— <http://linux.com>
- Linux.org——www.linux.org
- Mac OS FAQ——<http://osxfaq.com/Tutorials/LearningCenter/>
- Mac OS X 提示——www.macosxhints.com/
- Mac OS X 基础知识——http://apple.com/pro/training/macosx_basics/
- SAGE——<http://sageweb.sage.org/>
- Unix Guru Universe——<http://ugu.com>
- Unix Rosetta Stone ——<http://bhami.com/rosetta.html>

Unix 历史

- Dennis Ritchie 的主页——www.cs.bell-labs.com/who/dmr/
- Ken Thompson 的主页——www.bell-labs.com/about/history/unix/thompsonbio.html
- The Unix Heritage Society —— www.tuhs.org/
- Unix Timeline——<http://levenez.com/unix/>

Unix 安全

- Unix 安全——http://secinf.net/unix_security/
- Linux 安全——<http://linuxsecurity.com>
- SANS.org Top 20 Vulnerabilities——<http://sans.org/top20/#ul>
- Security Focus——<http://securityfocus.com/unix>
- Unix 网络与安全工具——<http://csrc.nist.gov/tools/tools.htm>
- Unix Security Checklist——www.cert.org/tech_tips/usc20_full.html

提供商站点

- Debian GNU/Linux——<http://debian.org/>
- FreeBSD——<http://freebsd.org/>
- Hewlett Packard HP-UX——<http://hp.com/products1/unix/operating/>
- IBM AIX——www.ibm.com/servers/aix/
- KNOPPIX——<http://knopper.net/knoppix/index-en.html>
- Mac OS X——<http://apple.com/macosx/>
- NetBSD——<http://netbsd.org/>
- OpenBSD——<http://openbsd.org/>
- OS/390 Unix——www.ibm.com/servers/s390/os390/bkserv/r8pdf/uss.html
- Plan 9——www.cs.bell-labs.com/plan9dist/
- Red Hat Enterprise Linux——<http://redhat.com>

- Red Hat Fedora Core—<http://fedora.redhat.com/>
- SGI IRIX—<http://sgi.com/products/software/irix/>
- Sun Microsystem 的 Solaris Unix—<http://sun.com/software/solaris/>
- SUSE Linux—www.novell.com/linux/suse/index.html
- Yellow Dog Linux (用于 Apple 系统)—<http://yellowdoglinux.com/>

软件资源

- Apple(Mac OS X 专用)—<http://apple.com/support/downloads/>
- MacUpdate(Mac OS X 专用)—www.macupdate.com/
- Fink Project (Mac OS X 专用)—<http://fink.sourceforge.net/>
- 自由软件联盟—www.fsf.org
- Freshmeat (Linux 偏向于使用该软件, 该软件在其他的 Unix 上也可用)—www.freshmeat.net
- RPM Find (Linux 专用)—<http://rpmfind.net/>
- Sourceforge—<http://sourceforge.net>
- SunFreeware(Sun 专用)—<http://sunfreeware.com>

Unix 杂志

- Linux Journal—<http://linuxjournal.com>
- Linux Magazine—<http://linux-mag.com/>
- MacAddict—<http://macaddict.com/>
- Mac Tech—<http://mactech.com/>
- MacWorld—<http://macworld.com/>
- Sys Admin—<http://sysadminmag.com/>

Unix 新闻和常用信息

- BSD News—<http://bsdnews-com/>
- Linux Format—<http://linuxformat.co.uk/>
- Linux Gazette—<http://linuxgazette.com/>
- Lmx Insider—<http://linuxinsider.com/>
- Linux Today—<http://linxtoday.com/>
- Linux Weekly News—<http://lwn.net/>
- Linux.org—www.linux.org
- Mac Minute—<http://macminute.com/>
- Mac News Network—<http://macnn.com/>
- NewsForge—<http://newsforge.com/>
- Slashdot.org—<http://slashdot.org>
- Solaris Central—<http://solariscentral.org/>
- Sun News—<http://sun.com/software/solaris/news/>
- Unix Review—<http://unixreview.com/>
- Unix World—<http://networkcomputing.com/unixworld/unixhome.html>

娱乐资源

- KDE 与 GNOME—<http://freshmeat.net/articles/view/179/>
- Unix Haters 参考手册—<http://research.microsoft.com/~daniel/unix-haters.html>
- vi 全页—<http://thomer.com/vi/vi.html>
- vi 与 Emacs—<http://newsforge.com/article.pl?sid=01/12/04/0326236>

[G e n e r a l I n f o r m a t i o n]

书名= U N I X入门经典

作者= (美) P a u l L o v e , (美) J o e M e r l i n o等著；张楚雄，许文昭译

页数= 3 5 5

出版社= 北京市：清华大学出版社

出版日期= 2 0 0 6

S S号= 1 1 5 6 3 0 5 6

D X号= 0 0 0 0 0 6 0 0 5 2 8 1

U R L = h t t p : / / b o o k . s z d n e t . o r g . c n / b o o k D e t a i l . j s
p ? d x N u m b e r = 0 0 0 0 0 6 0 0 5 2 8 1 & d = E 9 4 3 A D 8 0 B A 2 9 5 3 5 3 C
D B 9 B F 5 D F C 2 4 B E F A

封面
版权
前言
目录

第 1 章	U n i x 基础
1 . 1	简史
1 . 2	U n i x 的版本
1 . 3	操作系统组件
1 . 3 . 1	U n i x 内核
1 . 3 . 2	s h e l l
1 . 3 . 3	其他组件
1 . 4	小结
第 2 章	起步
2 . 1	系统启动
2 . 2	登录和退出 U n i x
2 . 2 . 1	G U I 登录
2 . 2 . 2	命令行登录
2 . 2 . 3	远程登录
2 . 2 . 4	s h e l l
2 . 2 . 5	退出
2 . 3	关闭系统
2 . 4	使用联机帮助页
2 . 5	小结
第 3 章	用户和组
3 . 1	账户基础知识
3 . 1 . 1	根账户
3 . 1 . 2	系统账户
3 . 1 . 3	用户账户
3 . 1 . 4	组账户
3 . 2	管理用户和组
3 . 2 . 1	/ e t c / p a s s w d
3 . 2 . 2	/ e t c / s h a d o w
3 . 2 . 3	/ e t c / g r o u p
3 . 2 . 4	M a c O S X 的不同之处
3 . 3	管理账户和组
3 . 3 . 1	账户管理
3 . 3 . 2	组管理
3 . 3 . 3	使用图形用户界面工具进行用户管理
3 . 4	变成另一个用户
3 . 5	与用户和组相关的命令
3 . 6	小结
3 . 7	练习
第 4 章	文件系统
4 . 1	文件系统基础
4 . 1 . 1	目录结构
4 . 1 . 2	根的基本目录
4 . 2	路径和大小写
4 . 3	文件系统导航
4 . 3 . 1	p w d
4 . 3 . 2	c d
4 . 3 . 3	w h i c h 和 w h e r e i s
4 . 3 . 4	f i n d
4 . 3 . 5	f i l e
4 . 3 . 6	l s
4 . 4	文件类型
4 . 5	链接

	4 . 6	文件和目录权限
	4 . 7	修改权限
	4 . 7 . 1	以符号模式使用 <code>chmod</code>
	4 . 7 . 2	以绝对模式使用 <code>chmod</code>
	4 . 8	查看文件
	4 . 9	创建、修改和删除文件
	4 . 9 . 1	删除文件
	4 . 9 . 2	创建和删除目录
	4 . 1 0	基本的文件系统管理
	4 . 1 1	使文件系统可访问
	4 . 1 2	小结
	4 . 1 3	练习
第 5 章		定制工作环境
	5 . 1	环境变量
	5 . 1 . 1	<code>PS1</code> 变量
	5 . 1 . 2	其他环境变量
	5 . 2	路径
	5 . 2 . 1	<code>PATH</code> 环境变量
	5 . 2 . 2	相对路径和绝对路径
	5 . 2 . 3	切换文件系统
	5 . 3	选择 <code>shell</code>
	5 . 3 . 1	临时修改 <code>shell</code>
	5 . 3 . 2	修改默认的 <code>shell</code>
	5 . 3 . 3	各种 <code>shell</code>
	5 . 4	配置 <code>shell</code>
	5 . 4 . 1	运行控制文件
	5 . 4 . 2	环境变量
	5 . 4 . 3	别名
	5 . 4 . 4	选项
	5 . 5	动态共享库路径
	5 . 5 . 1	<code>LD_LIBRARY_PATH</code>
	5 . 5 . 2	<code>LD_DEBUG</code>
	5 . 6	小结
	5 . 7	练习
第 6 章		深入 <code>Unix</code> 命令
	6 . 1	命令的剖析
	6 . 2	查找命令的相关信息
	6 . 2 . 1	<code>man</code>
	6 . 2 . 2	<code>info</code>
	6 . 2 . 3	<code>apropos</code>
	6 . 3	命令的修改
	6 . 3 . 1	元字符
	6 . 3 . 2	输入和输出重定向
	6 . 3 . 3	管道
	6 . 3 . 4	命令置换
	6 . 4	操作文件和目录
	6 . 4 . 1	<code>ls</code>
	6 . 4 . 2	<code>cd</code>
	6 . 5	常用的文件操作命令
	6 . 5 . 1	<code>cat</code>
	6 . 5 . 2	<code>more / less</code>
	6 . 5 . 3	<code>mv</code>
	6 . 5 . 4	<code>cp</code>
	6 . 5 . 5	<code>rm</code>
	6 . 5 . 6	<code>touch</code>
	6 . 5 . 7	<code>wc</code>

	6 . 6	文件所有权和权限
	6 . 6 . 1	文件所有权
	6 . 6 . 2	文件权限
	6 . 6 . 3	u m a s k
	6 . 6 . 4	执行文件
	6 . 7	保持文件系统配额
	6 . 8	小结
	6 . 9	练习
第 7 章		用 v i 编辑文件
	7 . 1	使用 v i
	7 . 2	在文件中移动
	7 . 3	搜索文件
	7 . 4	退出并保存文件
	7 . 5	编辑文件
	7 . 5 . 1	删除字符
	7 . 5 . 2	修改命令
	7 . 5 . 3	高级命令
	7 . 6	帮助
	7 . 6 . 1	运行命令
	7 . 6 . 2	替换文本
	7 . 7	v i 的版本
	7 . 8	小结
	7 . 9	练习
第 8 章		高级工具
	8 . 1	正则表达式和元字符
	8 . 1 . 1	理解元字符
	8 . 1 . 2	正则表达式
	8 . 2	使用 S F T P 和 F T P
	8 . 3	更高级的命令
	8 . 3 . 1	g r e p
	8 . 3 . 2	f i n d
	8 . 3 . 3	s o r t
	8 . 3 . 4	t e e
	8 . 3 . 5	s c r i p t
	8 . 3 . 6	w c
	8 . 4	小结
	8 . 5	习题
第 9 章		高级 U n i x 命令：S e d 和 A W K
	9 . 1	s e d
	9 . 1 . 1	使用 - e 选项
	9 . 1 . 2	s e d 文件
	9 . 1 . 3	s e d 命令
	9 . 2	A W K
	9 . 2 . 1	用 A W K 提取数据
	9 . 2 . 2	使用模式
	9 . 3	利用 A W K 编程
	9 . 4	小结
	9 . 5	练习
第 1 0 章		作业控制和进程管理
	1 0 . 1	进程
	1 0 . 2	s h e l l 脚本
	1 0 . 3	正在运行的进程
	1 0 . 3 . 1	p s 语法
	1 0 . 3 . 2	进程状态
	1 0 . 4	系统进程
	1 0 . 5	进程属性

10.6	停止进程
10.6.1	进程树
10.6.2	僵死进程
10.7	top命令
10.8	/proc文件系统
10.9	SETUID和SETGID
10.10	shell作业控制
10.11	小结
第11章	在指定时间运行程序
11.1	系统时钟
11.1.1	使用date检查和设置系统时钟
11.1.2	在Linux上利用hwclock同步时钟
11.1.3	利用NTP同步系统时钟
11.2	安排将来运行的命令
11.2.1	利用cron执行程序
11.2.2	使用at命令进行一次性执行
11.3	小结
11.4	练习
第12章	安全性
12.1	安全性的基础知识
12.1.1	资产价值保护
12.1.2	潜在的问题
12.2	保护Unix系统
12.2.1	口令的安全性
12.2.2	口令破译程序
12.3	限制管理访问
12.3.1	UID 0
12.3.2	根用户管理选项
12.3.3	设置sudo
12.4	系统管理的预防性任务
12.4.1	删除不需要的账户
12.4.2	修补、限制或删除程序
12.4.3	禁用不需要的服务
12.4.4	监控并限制对服务的访问
12.4.5	实现内置防火墙
12.4.6	其他的安全程序
12.5	小结
12.6	练习
第13章	基本shell脚本编程
13.1	注释脚本
13.2	开始脚本编程
13.2.1	调用shell
13.2.2	变量
13.2.3	从键盘读取输入
13.2.4	特殊变量
13.2.5	退出状态
13.3	流程控制
13.3.1	条件流程控制
13.3.2	迭代流程控制
13.4	选择脚本编程shell
13.5	小结
13.6	练习
第14章	高级shell脚本编程
14.1	高级脚本编程的概念
14.1.1	输入和输出重定向
14.1.2	命令替换：反引号和圆括号扩展

	1 4 . 1 . 3	使用环境变量和 s h e l l 变量
1 4 . 2	s h e l l 函数	
	1 4 . 2 . 1	返回值
	1 4 . 2 . 2	嵌套函数和递归
	1 4 . 2 . 3	作用域
	1 4 . 2 . 4	函数库
	1 4 . 2 . 5	信号和陷阱
	1 4 . 2 . 6	文件处理
	1 4 . 2 . 7	数组
1 4 . 3	s h e l l 的安全性	
	1 4 . 3 . 1	攻击可能来自何处
	1 4 . 3 . 2	采取预防措施
	1 4 . 3 . 3	受限 s h e l l
1 4 . 4	系统管理	
	1 4 . 4 . 1	收集信息
	1 4 . 4 . 2	执行任务
	1 4 . 4 . 3	调试脚本
1 4 . 5	小结	
1 4 . 6	练习	
第 1 5 章	系统日志	
	1 5 . 1	日志文件
	1 5 . 2	s y s l o g d
		1 5 . 2 . 1 s y s l o g . c o n f
		1 5 . 2 . 2 消息
		1 5 . 2 . 3 日志记录器
	1 5 . 3	轮循日志
	1 5 . 4	监视系统日志
		1 5 . 4 . 1 l o g w a t c h
		1 5 . 4 . 2 s w a t c h
	1 5 . 5	小结
	1 5 . 6	练习
第 1 6 章	U n i x 网络互联	
	1 6 . 1	T C P / I P
		1 6 . 1 . 1 T C P
		1 6 . 1 . 2 I P
		1 6 . 1 . 3 与 T C P / I P 一起使用的其他协议
		1 6 . 1 . 4 网络地址、子网、子网掩码和 T C P / I P 路由选择
	1 6 . 2	为 U n i x 系统设置 T C P / I P 网络
		1 6 . 2 . 1 T C P / I P 网络请求配置
		1 6 . 2 . 2 动态设置
		1 6 . 2 . 3 发送 T C P / I P 网络请求
		1 6 . 2 . 4 回应 T C P / I P 网络请求
		1 6 . 2 . 5 i n e t d
	1 6 . 3	网络管理工具
		1 6 . 3 . 1 通过 T r a c e r o u t e 跟踪网络的性能
		1 6 . 3 . 2 防火墙
		1 6 . 3 . 3 例行检查网络延迟
	1 6 . 4	小结
	1 6 . 5	练习
第 1 7 章	P e r l 编程实现 U n i x 自动化	
	1 7 . 1	P e r l 的优点
	1 7 . 2	一些有用的 P e r l 命令
		1 7 . 2 . 1 变量
		1 7 . 2 . 2 运算符
		1 7 . 2 . 3 基本函数
	1 7 . 3	更多 P e r l 代码的示例

- 17.4 检修Perl脚本
- 17.5 小结
- 17.6 练习

第18章 备份工具

- 18.1 备份基础知识
 - 18.1.1 决定备份什么数据
 - 18.1.2 备份介质类型
 - 18.1.3 备份类型
 - 18.1.4 备份时间
 - 18.1.5 验证备份
 - 18.1.6 保存备份
- 18.2 备份命令
 - 18.2.1 tar
 - 18.2.2 使用gzip和bzip2压缩
 - 18.2.3 cpio
 - 18.2.4 dump、backup和restore
 - 18.2.5 其他备份命令
- 18.3 备份套件
- 18.4 小结
- 18.5 练习

第19章 从源代码安装软件

- 19.1 源代码
- 19.2 开放源代码许可证
 - 19.2.1 BSD许可证
 - 19.2.2 GNU公共许可证
- 19.3 寻找和下载Unix软件
 - 19.3.1 选择软件
 - 19.3.2 下载文件
 - 19.3.3 验证源代码
- 19.4 编译和安装
 - 19.4.1 提取文件
 - 19.4.2 开始编译
- 19.5 make、Makefile和make目标
 - 19.5.1 Makefile
 - 19.5.2 帮助创建Makefile的工具
 - 19.5.3 GNU编译工具
 - 19.5.4 diff和patch
- 19.6 利于维护的安装技术
- 19.7 排查编译问题
- 19.8 预编译软件包
- 19.9 小结
- 19.10 练习

第20章 转换：适用于Mac OS用户的Unix

- 20.1 Mac OS X简史
- 20.2 Mac OS 9与Mac OS X之间的差别
- 20.3 文件夹也是目录
 - 20.3.1 必需的文件夹
 - 20.3.2 主目录
 - 20.3.3 管理
 - 20.3.4 预置文件
- 20.4 Unix和Mac OS X/Mac OS 9命令与GUI的对等命令
- 20.5 Mac OS X和其他Unix系统之间的差别
 - 20.5.1 目录服务和NetInfo
 - 20.5.2 nidump和nload
 - 20.5.3 NetInfo数据库的备份和恢复
 - 20.5.4 系统启动

	2 0 . 5 . 5	文件结构上的差别
	2 0 . 5 . 6	根用户账户
	2 0 . 6	小结
	2 0 . 7	练习
第 2 1 章	转换：适用于W i n d o w s 用户的U n i x	
	2 1 . 1	结构上的比较
	2 1 . 2	主要管理工具之间的比较
	2 1 . 3	流行应用程序的比较
	2 1 . 4	在W i n d o w s 中使用U n i x
	2 1 . 5	在U n i x 中使用W i n d o w s
	2 1 . 6	小结
附录 A	练习题答案	
附录 B	一些有用的U n i x 网站	