



邢国庆 庞俊华 陈智建 编著

UNIX

从入门到精通

(第二版)

经典 实用 权威

UNIX从入门到精通 (第二版)

本书由浅入深，逐步阐述了UNIX系统的基本概念与原理，同时给出了大量的应用实例。内容丰富、语言流畅，是学习、使用、管理与维护UNIX系统的一本不可多得的工具书，可以作为学习UNIX操作系统的主要参考书。

Windows 7中文版从入门到精通

Windows Server 2008从入门到精通

Mac OS X 10.5中文版从入门到精通

UNIX从入门到精通 (第二版)

Ubuntu 9 Linux从入门到精通

Oracle 11g从入门到精通

SQL从入门到精通

SQL Server 2008中文版从入门到精通

Crystal Reports 2008水晶报表从入门到精通

Java SE 6从入门到精通



责任编辑：李红玉
文字编辑：杨 昆 姜 彦



ISBN 978-7-121-10295-0



9 787121 102950 >

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。

定价：58.00元

内 容 简 介

本书从UNIX的基本命令入手,由浅入深,逐步阐述UNIX系统的基本概念与设计原理,同时给出了大量的应用实例。在此基础上,对UNIX的核心精华部分,如Shell编程、进程管理、用户管理、磁盘空间管理、文件系统内部组织、文件系统维护、系统启动与关机、TCP/IP网络维护与应用、软件包的制作与安装等方面进行了深入的讨论。本书内容丰富、重点突出、文字简练、语言流畅、实用性强。

本书可作为大专院校师生UNIX操作系统课程的教学参考书,也可作为IT从业者的UNIX自学手册。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

UNIX从入门到精通/邢国庆,庞俊华,陈智建编著.—2版.—北京:电子工业出版社,2010.3
ISBN 978-7-121-10295-0

I. U… II. ①邢…②庞…③陈… III. UNIX操作系统 IV. TP316.81

中国版本图书馆CIP数据核字(2010)第016235号

责任编辑:李红玉

文字编辑:姜 影 易 昆

印 刷:北京天竺颖华印刷厂

装 订:三河市鑫金马印装有限公司

出版发行:电子工业出版社

北京市海淀区万寿路173信箱 邮编:100036

北京市海淀区翠微东里甲2号 邮编:100036

开 本:787×1092 1/16 印张:29.75 字数:740千字

印 次:2010年3月第1次印刷

定 价:56.00元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系。
联系及邮购电话:(010) 88254888。

质量投诉请发邮件至zlts@phei.com.cn,盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线:(010) 88258888。

再 版 前 言

《UNIX从入门到精通》自出版以来，得到了广大读者的极大关照，值此再版之际，本人深表谢意。根据出版社的要求和读者的反馈意见，再版时，本书从内容、文字及章节编排等方面做了全面的梳理与更新，力求内容丰富、重点突出、文字简练，兼顾知识的系统性与实用性。同时也纠正了第一版中存在的问题。

目前存在许多不同品牌的UNIX，如IBM公司的AIX、Sun公司的Solaris，以及HP公司的HP-UX等。那么，学习UNIX系统究竟应如何下手呢？这是一个令许多UNIX初学者感到困惑和迷茫的问题。事实上，这些UNIX系统之间大同小异，具有很多共性。因此，只需任意选用一种UNIX系统，创建一个上机学习环境，选购一本有效参考书，从最基本的命令行开始（最好不要走图形界面的捷径），逐步深入其中，快速掌握一门UNIX技术。等到得心应手时，再学习其他UNIX，就会达到触类旁通的效果。

UNIX的强大功能完全表现在命令行中，体现在命令的充分发挥与灵活运用方面。故本书从UNIX的基本命令入手，由浅入深，逐步阐述UNIX系统的基本概念与设计原理，同时给出大量的应用实例。在此基础上，对UNIX的精华部分，如Shell编程、进程管理、用户管理、磁盘空间管理、文件系统内部组织、文件系统管理、系统启动与关机、TCP/IP网络的管理与应用、软件包的制作与安装等方面进行了深入的讨论。

在内容编排方面，本书采用概念与实例相结合，系统性与实用性相结合的原则，力求循序渐进，兼顾初学者与资深人员，尽可能提供更多的知识点。书中给出的实例与代码都是精心选择的，既有助于理解基本概念，也可为日后的系统维护提供参考。

本书主要介绍UNIX系统最基本的核心内容，尽可能不涉及不同UNIX品牌的功能扩充部分。由于UNIX没有统一的版本，同样的命令，在不同的UNIX系统中，其选项、参数与输出的结果也不尽相同，这为编写一本通用的UNIX书籍带来很大的难度。为了保持本书所述内容的普遍性，在讨论的过程中，我们尽量介绍大多数系统都提供的命令和功能。在介绍具体的命令时，也尽量不讲特定系统独有的内容，即使很有特点的命令选项也是如此。本书给出的命令语法，也只是常用的命令选项或几个主流UNIX厂商均有的内容，而非命令的全部。这一点需要特别说明。

在本书给出的例子中，需要用户输入的命令均以黑体字符形式给出。其中，命令提示符为“#”者，表示只有超级用户才能使用的命令；命令提示符为“\$”者，表示普通用户也可以使用的命令。

本书是作者多年学习UNIX系统的经验与体会，如果能使读者在学习UNIX操作系统时有所裨益，将是本人的莫大荣幸。但由于时间仓促，且限于作者的水平和能力，如有不当或者错误之处，恳请广大读者给予批评指正。

在本书的写作过程中，从写作的宗旨与原则，到章节的选定与编排，都得到了电子工业出版社及美迪亚电子信息有限公司各位编辑的热情鼓励和全力帮助。仇鹏涛、赵东江、黄辰、曾伟玲、刘琦、王群、曹雷、王薇、刘楠、高如欣、伊晓强、魏成明、张秋慧、李宗玉、梁志强、袁伟、田宇以及我的家人邸静和邢梦可等也给予了大力协助，在此一并表示感谢。

目 录

第1章 UNIX概述与安装1	2.8.5 命令行补充.....47
1.1 UNIX早期发展过程概述.....1	2.9 命令别名.....48
1.1.1 UNIX的缘起.....2	2.10 作业控制.....51
1.1.2 UNIX的交替发展.....4	2.11 会话记录.....53
1.1.3 UNIX的战国时代.....5	2.12 使用man命令查询系统参考手册...55
1.1.4 策略与标准之争.....6	
1.1.5 UNIX的黑暗时期.....7	第3章 文件系统基础知识57
1.1.6 AT&T UNIX System V Release 4.0.....7	3.1 文件系统的层次结构.....57
1.1.7 后UNIX时代.....8	3.1.1 树形层次结构.....57
1.2 UNIX的层次组织结构.....9	3.1.2 路径名.....58
1.3 UNIX的逻辑组织结构.....10	3.2 文件系统的组织结构.....59
1.3.1 进程管理子系统.....11	3.3 文件的类型.....62
1.3.2 内存管理子系统.....12	3.3.1 普通文件.....63
1.3.3 文件管理子系统.....12	3.3.2 目录文件.....64
1.3.4 I/O管理子系统.....13	3.3.3 特殊文件.....66
1.3.5 硬件系统.....13	3.3.4 链接文件.....70
1.4 安装Solaris操作系统.....13	3.3.5 符号链接文件.....71
1.4.1 硬件要求.....14	3.3.6 管道文件.....73
1.4.2 安装步骤.....14	3.4 文件的安全保护机制.....73
	3.4.1 显示文件的访问权限.....74
第2章 命令行基础知识25	3.4.2 修改文件的访问权限.....75
2.1 命令行结构.....25	3.4.3 设置文件的访问权限.....76
2.2 后台进程.....28	3.4.4 其他访问权限设置.....78
2.3 标准输入、标准输出与标准错误 输出.....28	
2.4 输入输出重定向.....29	第4章 文件和目录操作80
2.5 管道.....33	4.1 创建文件.....80
2.6 元字符与文件名生成.....35	4.2 显示文件列表.....81
2.7 转义与引用.....38	4.2.1 使用ls命令显示文件列表.....81
2.8 命令历史.....40	4.2.2 利用通配符显示文件.....83
2.8.1 fc命令.....41	4.2.3 显示隐藏文件.....84
2.8.2 history命令.....43	4.2.4 递归显示目录与文件.....85
2.8.3 重复执行先前的命令.....44	4.3 显示文件内容.....86
2.8.4 编辑并执行校正后的命令.....46	4.3.1 使用cat命令显示文件.....86
	4.3.2 使用more命令分页显示文件....86

4.3.3 使用head命令显示文件前几行内容	88	5.2.2 命令模式	110
4.3.4 使用tail命令显示文件最后几行内容	88	5.3 保存编辑的文件并退出vi	111
4.4 复制文件	89	5.4 vi编辑器的基本命令	112
4.5 移动文件	89	5.4.1 移动光标位置	112
4.6 删除文件	91	5.4.2 输入文本	113
4.7 显示当前工作目录	92	5.4.3 修改与替换文本	113
4.8 改换目录	92	5.4.4 撤销先前的修改	114
4.9 创建目录	94	5.4.5 删除文本	115
4.10 移动目录	94	5.4.6 复制、删除与粘贴文本	115
4.11 复制目录	95	5.4.7 重复执行指定次数的命令	116
4.12 删除目录	96	5.5 使用ex命令	116
4.13 比较文件之间的差别	96	5.5.1 显示行号	116
4.13.1 使用diff命令比较两个文件	96	5.5.2 多行复制	117
4.13.2 使用diff3命令比较三个文件	97	5.5.3 移动文本行	117
4.14 从系统中检索文件	98	5.5.4 删除文本行	118
4.14.1 简单检索	100	5.6 检索与替换	118
4.14.2 使用逻辑运算符	100	5.6.1 检索字符串	118
4.14.3 利用find命令本身实现其他处理功能	101	5.6.2 模式检索	119
4.14.4 利用管道实现其他处理功能	101	5.6.3 替换字符串	120
4.15 检索文件内容	102	5.7 编辑多个文件	120
4.15.1 利用grep检索文件内容	102	5.7.1 编辑多个文件	120
4.15.2 过滤其他命令的输出数据	103	5.7.2 合并文件与合并文本行	121
4.15.3 使用grep检索多个文件	103	5.8 定制vi编辑器的运行环境	121
4.15.4 检索不包含特定字符串的文本行	104	5.8.1 临时设定vi的运行环境	121
4.15.5 在grep中使用正则表达式	104	5.8.2 永久性地定制vi的运行环境	123
4.15.6 检索元字符本身	107	5.9 其他特殊说明	124
4.15.7 在命令行中使用引号	107	5.9.1 删除或替换特殊字符	124
4.16 排序	107	5.9.2 在编辑期间运行UNIX命令	124
第5章 编辑文件	109	5.10 vi编辑器命令总结	125
5.1 启动vi编辑器	109	第6章 Shell基础知识	129
5.1.1 创建文件	109	6.1 shell与Shell脚本	129
5.1.2 状态行	109	6.1.1 为什么需要Shell编程	129
5.2 vi编辑器的两种工作模式	110	6.1.2 什么是Shell脚本	130
5.2.1 输入模式	110	6.1.3 运行Shell脚本	131
		6.1.4 退出与出口状态	131
		6.1.5 调用适当的Shell解释程序	133
		6.1.6 位置参数	135
		6.2 变量与变量替换	137
		6.2.1 变量分类	137

6.2.2	变量赋值	138	7.5	until循环语句	193
6.2.3	内部变量	138	7.6	select循环语句	194
6.2.4	变量引用与替换	141	7.7	嵌套的循环	195
6.2.5	变量的间接引用	143	7.8	循环控制与辅助编程命令	197
6.2.6	特殊的变量替换	144	7.8.1	break和continue命令	197
6.2.7	变量声明与类型定义	147	7.8.2	true命令	199
6.3	命令与命令替换	148	7.8.3	sleep命令	199
6.3.1	Shell内部命令	148	7.8.4	shift命令	200
6.3.2	部分命令介绍	151	7.8.5	getopt命令	200
6.3.3	命令替换	161	7.8.6	getopts命令	202
6.4	test语句	163	7.9	循环语句的I/O重定向	204
6.4.1	文件测试运算符	164	7.9.1	while循环的I/O重定向	205
6.4.2	字符串测试运算符	165	7.9.2	until循环的I/O重定向	206
6.4.3	整数值测试运算符	167	7.9.3	for循环的I/O重定向	206
6.4.4	逻辑运算符	168	7.10	here文档	207
6.5	命令行的解释执行过程	169	7.11	Shell函数	212
6.5.1	读取命令行	170	7.12	逻辑与和逻辑或并列结构	218
6.5.2	命令历史替换	171	7.12.1	逻辑与命令并列结构	218
6.5.3	命令别名替换	171	7.12.2	逻辑或命令并列结构	219
6.5.4	花括号扩展	171	7.13	Shell数组	220
6.5.5	波浪号替换	172	7.14	信号的捕捉与处理	225
6.5.6	I/O重定向	173	7.15	其他Shell课题	229
6.5.7	变量替换	174	7.15.1	子Shell	229
6.5.8	算术运算结果替换	174	7.15.2	Shell脚本的调试	230
6.5.9	命令替换	174	7.15.3	系统性能考虑	235
6.5.10	单词解析	175			
6.5.11	文件名生成	175	第8章	进程管理	237
6.5.12	引用字符处理	176	8.1	ps命令概述	237
6.5.13	进程替换	177	8.2	查询进程及其状态信息	239
6.5.14	环境处理	178	8.2.1	查询当前活动的进程	239
6.5.15	执行命令	178	8.2.2	查询系统中所有的进程	240
6.5.16	跟踪执行过程	179	8.2.3	显示进程的重要状态信息	241
6.5.17	实例验证	179	8.2.4	显示进程的详细状态信息	241
			8.3	监控进程及系统资源	242
第7章	Shell高级编程	181	8.4	终止进程的运行	245
7.1	if条件语句	181	8.5	调整进程的调度类别及优先级	248
7.1.1	if语句的表现形式	181	8.5.1	显示进程的调度类别与优先级	248
7.1.2	嵌套的if-then条件测试	183	8.5.2	按照指定的调度类别与优先级	249
7.2	case分支语句	185		运行进程	249
7.3	for循环语句	187			
7.4	while循环语句	191			

8.5.3 调整进程的调度类别与优先级	250	用户	283
8.5.4 设置实时进程的时间片	250	9.4.3 利用w命令查询系统中的用户活动	283
8.6 调整分时进程的优先级	251	9.4.4 向注册用户发送消息	284
8.6.1 nice命令	251	9.5 以不同用户的身份访问系统	284
8.6.2 renice命令	252		
8.6.3 调整进程优先级的作用	252	第10章 软件包的制作与管理	287
8.7 定时运行系统任务和用户程序	253	10.1 软件包组成简介	287
8.7.1 cron守护进程的调度过程	253	10.1.1 基本组成部分	287
8.7.2 调度定时重复执行的任务	255	10.1.2 选用的信息文件	288
8.7.3 提交一次性定时执行的任务	255	10.1.3 选用的Shell脚本文件	288
8.8 调度重复执行的任务	255	10.2 软件包的相关文件和命令	288
8.8.1 crontab的工作原理	256	10.2.1 pkginfo文件	289
8.8.2 创建和编辑crontab文件	257	10.2.2 prototype文件	290
8.8.3 显示crontab文件	258	10.2.3 pkgmap文件	293
8.8.4 删除crontab文件	258	10.2.4 copyright文件	293
8.8.5 crontab命令的访问控制	259	10.2.5 depend文件	294
8.8.6 应用实例——数据库定时备份	260	10.2.6 space文件	294
8.9 调度一次性执行的作业	261	10.2.7 compver文件	295
8.9.1 提交at作业	261	10.2.8 软件包的相关工具	295
8.9.2 显示at作业及作业队列	263	10.3 制作软件包	295
8.9.3 删除at作业	263	10.3.1 制作软件包的步骤	296
8.9.4 at命令的访问控制	263	10.3.2 创建pkginfo文件	301
8.9.5 应用实例——系统定时关机	264	10.3.3 利用pkgproto命令创建prototype文件	301
第9章 用户管理	266	10.3.4 利用pkgmk命令制作软件包	304
9.1 增加与删除用户	266	10.3.5 pkgtrans命令	308
9.1.1 /etc/passwd文件	266	10.4 安装软件包	310
9.1.2 /etc/shadow文件	267	10.5 查询软件包	312
9.1.3 用户管理实例	268	10.6 检测软件包	313
9.2 定制用户的工作环境	272	10.7 卸载软件包	315
9.2.1 选择命令解释程序	272		
9.2.2 设置用户初始化文件	273	第11章 磁盘空间管理	318
9.2.3 定制Shell工作环境	275	11.1 查询磁盘空间信息	318
9.3 增加与删除用户组	280	11.1.1 常用的磁盘空间管理工具	318
9.4 监控用户	281	11.1.2 使用df命令检查存储空间的使用情况	318
9.4.1 利用who命令查询系统中的用户	282	11.1.3 使用du命令检查存储空间占用情况	321
9.4.2 利用finger命令查询系统中的			

11.1.4	使用quot命令查询每个用户占用的存储空间	323	第13章 TCP/IP网络应用	367
11.1.5	使用find命令找出超大容量的文件	324	13.1 OpenSSH	367
11.1.6	使用find命令找出长期闲置的文件	324	13.1.1 sshd_config配置文件	367
11.1.7	使用find命令找出并删除core文件	325	13.1.2 ssh_config配置文件	370
11.1.8	使用ls命令检测文件的大小	325	13.1.3 使用SSH注册到远程系统	372
11.2	采用标准工具备份与恢复数据	326	13.1.4 使用SSH执行远程系统中的命令	373
11.2.1	利用cpio实现备份和恢复	327	13.1.5 使用SCP替代FTP	373
11.2.2	利用tar实现备份和恢复	333	13.1.6 使用SFTP替代FTP	374
11.2.3	利用dd实现数据的复制	337	13.1.7 SSH与SCP的无密码注册	375
11.3	文件系统限额管理	339	13.1.8 OpenSSH的安全考虑	377
11.3.1	限额概述	339	13.2 Telnet远程系统注册	378
11.3.2	设置限额	341	13.3 FTP文件传输	379
11.3.3	限额的维护	343	13.3.1 连接FTP服务器	381
第12章 TCP/IP网络管理	347		13.3.2 FTP应用	382
12.1 TCP/IP简介	347		13.3.3 FTP访问控制	384
12.1.1 TCP/IP协议的层次结构	347		13.3.4 FTP自动注册	384
12.1.2 TCP/IP协议如何处理数据通信	349	第14章 网络文件系统	386	
12.2 网络接口设置	351	14.1 NFS简述	386	
12.3 主机名字解析	353	14.2 配置NFS服务器	387	
12.4 网络路由设置	354	14.3 配置NFS客户系统	390	
12.4.1 静态路由	354	14.3.1 安装远程文件系统	390	
12.4.2 动态路由	355	14.3.2 设置/etc/vfstab文件	392	
12.5 配置网络服务	356	14.4 NFS自动安装	392	
12.6 网络管理与维护	358	14.4.1 主映射文件	393	
12.6.1 使用ifconfig命令维护网络接口	358	14.4.2 直接映射文件	394	
12.6.2 使用netstat命令监控网络状态	359	14.4.3 间接映射文件	394	
12.6.3 使用ping命令测试远程主机的连通性	363	第15章 系统启动与关机	396	
12.6.4 使用ftp命令检测网络主机的传输性能	364	15.1 磁盘分区与初始引导	396	
12.6.5 使用traceroute命令跟踪路由信息	365	15.1.1 磁盘分区	396	
		15.1.2 初始引导过程	399	
		15.1.3 系统初始化	400	
		15.2 init进程与系统生成	402	
		15.2.1 运行级	403	
		15.2.2 /etc/inittab文件	405	
		15.2.3 处理方式	405	
		15.2.4 /etc/inittab文件举例	406	
		15.2.5 启动用户定义的应用程序	409	

15.3 用户注册过程	409	第17章 文件系统管理	432
15.3.1 用户注册的处理过程	409	17.1 创建文件系统	432
15.3.2 utmpx和wtmpx文件	410	17.1.1 使用mkfs命令创建UFS文件 系统	432
15.4 系统关机过程	411	17.1.2 使用newfs命令创建文件系 统	436
15.4.1 使用shutdown命令关闭系 统	411	17.2 使用labelit命令命名文件系统	438
15.4.2 使用init命令关闭系统	412	17.3 安装、卸载文件系统	438
15.4.3 使用其他命令关机	412	17.3.1 安装文件系统	438
15.5 应用实例	413	17.3.2 /etc/vfstab文件	439
第16章 文件系统内部组织	415	17.3.3 安装文件系统	441
16.1 文件系统的组织结构	415	17.3.4 卸载文件系统	442
16.2 超级块	417	17.4 确定文件系统的类型	444
16.3 信息节点	418	17.5 检测与修复文件系统	446
16.3.1 特权标志位	418	17.5.1 何时需要检测文件系统	447
16.3.2 数据块地址数组	420	17.5.2 文件系统检测的内容	448
16.4 数据区与空闲数据存储块的组 织	421	17.5.3 交互检测与修复UFS文件系 统	452
16.5 信息节点的分配与释放	421	17.5.4 自动检测与修复UFS文件系 统	454
16.6 数据块的分配与释放	422	17.5.5 恢复严重受损的超级块	454
16.7 信息节点与目录和文件的关系 ...	424	17.5.6 解决fsck命令无法修复的UFS 文件系统问题	455
16.8 UFS文件系统	424	17.5.7 fsck的阶段处理方式	456
16.8.1 UFS文件系统的组织结构	424	17.5.8 利用其他工具修复文件系 统	463
16.8.2 引导块	425	参考文献	466
16.8.3 超级块	425		
16.8.4 柱面组信息块	426		
16.8.5 信息节点区与信息节点	428		
16.8.6 数据块区	429		
16.8.7 数据块的分配与释放过程 ...	429		
16.8.8 信息节点的分配与释放过 程	430		

第1章 UNIX概述与安装

UNIX之所以能够取得如此巨大的成功，这与其早期的发展历史有着极大的关系。作为开始，本章首先将简单地介绍UNIX早期的发展过程，以及UNIX版本的变化。从总体上直观地介绍一下UNIX的层次组织结构与逻辑组织结构，简述UNIX核心的四个主要组成部分，使读者对UNIX有一个全面的印象。

1.1 UNIX早期发展过程概述

UNIX的成功与其早期的发展历程及功能特点有着极大的关系，概括起来主要有以下三个原因：

- 简洁性。UNIX最初的成功并不在于它在技术上有多先进，主要是由于UNIX对操作系统概念和技术的极大简化。以当时的硬件技术水平和软件实践而言，这确实是一项明智之举。

- 开放性。由于与美国联邦政府司法部签订的法令限制，AT&T公司当时不能介入计算机及其相关技术产品的商业活动，这使得UNIX成为一个公开的、并非与硬件一起捆绑销售的软件产品。贝尔实验室向大学和研究机构公开其源代码，供教学与研究用。贝尔实验室的“不宣传、不支持、不负责纠错”策略也促使UNIX的研究人员与爱好者互帮互助，互相交流经验，最终促成UNIX用户组的建立。

- 可移植性。UNIX系统的简洁性和开放性，吸引了许多人的关注，进而产生了极大的兴趣，许多人相信可以把UNIX移植到自己的计算机上，籍此建立一个自己的操作系统与开发环境，提供一个自助的研究与开发环境。

此外，下述四个事件或元素也是UNIX能够取得成功的重要原因。

- C语言。C语言的出现与成功反过来也促进了UNIX系统的成功。UNIX与C语言相辅相成，互为依托，使得UNIX系统成为有别于其他操作系统的一种独特发展现象。使用C语言编写操作系统既提高了开发效率，易于阅读和理解，更有利于把UNIX移植到不同的硬件平台上。

- TCP/IP。在UNIX系统中，TCP/IP的出现使得ARPANET及后来的Internet成为现实。同时，Internet与TCP/IP的日益流行也促使UNIX系统得到广泛的应用，从而促进了UNIX系统的蓬勃发展。

- C语言标准I/O库。C语言标准I/O库的出现进一步奠定了UNIX系统可移植性的基础。使得系统开发人员完全可以抛开底层硬件实现的细节，只需专注于高层功能的实现，缩短了操作系统的研发周期。

- 公开源代码。向大学和研究结构公开源代码，其效果有二：前期激发了人们研究和移植UNIX系统的兴趣，导致UNIX成为操作系统的新宠；及至后期，UNIX成为大学操作系统课程的代名词。许多大学均以UNIX作为操作系统课程的研究对象，从而出现了“UNIX操作系统设计”等著名的UNIX教材，同时也培养了一大批潜在的UNIX系统准用户。

1.1.1 UNIX的缘起

1965年，麻省理工学院（MIT）和通用电器公司（GE）开始联合开发一个名为Multics的新型操作系统。其设计目标是开发一个多用户并发访问的交互式分时操作系统。具有足够强的计算与数据存储能力，使用户能够方便地共享数据。按当时主要以批处理为主的操作系统技术水平而言，这确实是一项雄心勃勃的计划。不久之后，AT&T贝尔实验室的计算科学研究中心也加入了这个联合开发项目。

经过多年的努力，到1969年初，Multics系统的初始版本开始在GE-645计算机上投入运行，可以同时支持三个用户。但与原来的设想相距甚远，既没有提供预期的通用计算服务，也不清楚何时才能达到其原先的设计目标。鉴于Multics项目前途渺茫，贝尔实验室决定退出这一工程项目。

一切都始自1969年，一个闷热而潮湿的夏天，而且始自一个游戏程序。

在开发Multics系统期间，Ken Thompson在一个运行GECOS操作系统的Honeywell 635计算机上，用FORTRAN语言编写了一个“空间旅行”的游戏程序，仿真太阳系中各行星的相对位置和运行轨迹，同时设计了一个宇宙飞船，使之穿梭飞行于各行星之间。

由于结束了Multics的研发工作，贝尔实验室计算科学研究中心的技术人员失去了“方便的交互计算服务”。为此，Ken Thompson、Dennis Ritchie和其他曾参与过Multics项目的贝尔实验室同事开始尝试改善自己的编程环境。期间，Ken Thompson萌发了编写一个多用户文件系统的想法，以便每个人都能够按一定的目录层次组织和存储自己的文件。经与D.Ritchie和Rudd Canaday讨论，最终在纸上草拟了一个文件系统设计方案。不久以后，Thompson写了一个内核和一个模拟其文件系统设计方案的仿真程序，并在GE-645计算机上实现了这个文件系统。

这个新的文件系统使Thompson能够以类似于Multics系统的层次目录结构存储他开发的游戏程序源文件。但Thompson的“空间旅行”游戏程序在GE-645机器中的运行结果却并不令人满意。GE-645是一个分时系统，忽停忽动的响应速度，使得Thompson很难控制飞船的飞行。由于运行结果不理想，且运行费用太贵（运行一次游戏需要花费75美元，这在当时是一笔不小的开支），促使Thompson想要尽快寻找一个替代系统。

很快，Thompson在一个角落里找到了一台几乎无人使用的PDP-7计算机，这是一个16位字长的机器，具有16KB内存，并配有一台图形显示终端。对于Thompson来说，这显然是一个好得不能再好的替代品。但由于需要借助GECOS开发环境，在GE-645与PDP-7之间进行交叉编译，并需要使用纸带在两个计算机之间交换数据，因而极感不便。

不久之后，Thompson完成了其游戏程序的开发。为了建立一个更好的开发和运行环境，Thompson和Ritchie开始实现其早先设计，并经过仿真验证的文件系统。

由此催生了初始的UNIX系统。尽管这个新的小型操作系统当时尚没有名字，但其中却包含一个小的系统内核、初始的UNIX文件系统、进程管理子系统以及内存管理子系统，提供一个小型的命令解释程序（当时尚未出现Bourne Shell），及一组有限的实用程序与文件系统操作命令，是一个能够同时支持两个用户的分时系统环境，而且第一次引入了信息节点的概念。

在此期间，GE-645一直用做开发环境，在两个机器之间进行交叉编译，但很快便在PDP-7上开发了汇编程序。经过不断完善，这一新的系统最终不再需要GECOS作为开发环境也能够自我支持。

UNIX的名字来自贝尔实验室另外一名参与UNIX开发的研究人员Brian Kernighan。作为Multics (Multiplexed Information and Computing System) 系统的双关语, 他把这一支持两个用户的小型操作系统取名为UNICS (Uniplexed Information and Computing System)。不久之后, UNICS又进一步演变成现在熟悉的UNIX。

虽然新生的UNIX系统受到很多人的赏识, 但直至得到实际应用时, 其潜力才为人们所认知。UNIX系统的第一个真正用户是贝尔实验室的专利部门, 作为文档管理系统提供文字处理功能。但此时却存在一个问题, 由于PDP-7是一个借来的机器, 机器性能也不强, 很难满足计算机研究小组不断增长的需求。到了1971年, 第一个汇编版本的UNIX系统开始移植到PDP-11/20机器 (56KB内存, 2个2.4MB的硬盘) 上。此时, UNIX可以支持较多的用户了, 同时也增加了一个称做roff的文本格式处理系统。

但直至1971年11月, 才真正公布了UNIX系统第一版, 并在文档上开始正式命名为UNIX。同时也提供了第一版的“UNIX程序员手册” (作者K. Thompson和D. M. Ritchie)。其中包括60多个命令, 如cat (显示文件内容)、chmod (修改文件访问权限)、chown (修改文件属主)、cp (复制文件)、ls (列举目录内容)、mv (移动或重新命名文件)、wc (计数文件的字符、字和行数) 以及who (查询系统中的用户) 等。

在获得了早期的成功之后, Thompson准备为UNIX开发一个FORTRAN编译器, 但却由此产生了一个称之为B的解释语言 (简化版的BCPL语言——Basic Combined Programming Language)。

1972年6月, 采用B语言重写的UNIX系统内核增加了管道功能, 用于连接一个程序的输出和另一个程序的输入, 以便组合各种命令, 构建新的功能, 从而出现了UNIX第二版。但由于受B语言的限制 (数据没有类型之分, 仅支持单一的机器字长, 其他数据类型只能通过函数或特殊的操作符实现), 且由于B是一种解释性的语言, 使系统的性能受到了影响。

于是, 擅长编程语言的Ritchie在B语言的基础上开发了一种新的C语言。1973年2月, Thompson和Ritchie利用C语言重写了UNIX操作系统。尽管这次重写并没有功能性的重大变革, 但由于使用高级语言编写操作系统, 改变了依赖汇编语言开发操作系统的历史, 使UNIX系统具有了可移植性的条件, 这对UNIX的发展有着极其重大的影响。

此时, UNIX在贝尔实验室内部的安装数量已增加到25个。因而成立了一个UNIX系统小组, 为贝尔实验室提供内部技术支持。在此期间, 又于1973年11月和1974年6月, 相继推出了UNIX的第四版和第五版。

1974年7月, Thompson和Ritchie在一份计算机科学年鉴——《ACM通讯》上公开发表了UNIX操作系统。这篇论文的发表标志了UNIX时代的开始, 使得UNIX系统开始日益流行, 世界各地的研究机构也开始关注这一新的操作系统, 从而也促使贝尔实验室开发出UNIX第五版。

由于1956年与美国联邦政府签署的法令限制, AT&T不能销售计算机及其软件产品。根据这个法令, AT&T既不能登广告为UNIX做宣传, 又不能在市场上销售, 也无法支持这一系统。尽管如此, UNIX系统还是越来越流行。

于是, 贝尔实验室开始向大学免费提供UNIX, 以便用于教学。其中包括加州大学的伯克利分校和澳大利亚悉尼的新南威尔士大学。同时, UNIX系统在电话营运公司也十分流行, 安装的数量不断增长。因为UNIX为程序开发、网络通信提供了良好的运行与开发环境。

1975年5月，UNIX第六版开始发行并通过西电（WE）公司第一次向贝尔实验室之外的用户提供UNIX。此时，UNIX的代码绝大部分都是用C语言编写的，而在当时，为了提高计算机硬件的性能，大多数操作系统都是采用汇编语言编写的，几乎没有听说操作系统能够用高级语言开发。

因此，C语言很快成为众所周知的可以代替汇编语言的系统开发工具。C语言编译器很小，很容易移植到各种计算机硬件系统中。另外，C语言的丰富功能，易于阅读和学习，高效的运行速度，使得系统程序员很快就放弃使用汇编语言开发系统程序。

1.1.2 UNIX的交替发展

1. BSD版的UNIX

1975年，Thompson作为一个访问学者回到其母校——加州大学伯克利分校，在学校的PDP-11/70计算机上安装了UNIX第六版，开始进行UNIX的教学和研究工作。几乎与此同时，Bill Joy和Chuck Haley作为研究生也来到伯克利，从此拉开了BSD版UNIX的序幕。

此后，UNIX的发展开始变得比较复杂。有时，贝尔实验室增加了某个新的特性，然后即以源码的形式向外分发。伯克利分校改善或增加了另外一个新的特性，这个新的特性最终又合并到贝尔实验室的下一个版本中，使得UNIX系统的功能不断地得到提升和广泛应用。

随着Thompson及其他贝尔实验室成员在伯克利分校教授UNIX，许多学生和教授对UNIX产生了极大的兴趣。Bill Joy把在此期间开发的Pascal编译器、ex（一个新的行编辑器）等软件，连同移植的UNIX V6捆绑到一起，称之为BSD（Berkeley Software Distribution），从1977年的年底开始向外分发。1978年推出的2BSD则是重写UNIX V6的重大尝试，其中包括了csh、mail，以及Bill Joy开发的著名vi编辑器等。

后来，伯克利分校计算机科学系又开始把UNIX V7移植到32位的VAX机器上，为UNIX的功能扩展铺平了道路。同时在UNIX内核中引入了页面调度和虚拟内存管理的概念，从而导致BSD 3.0的出现。

1981年6月，伯克利又推出了4BSD，其中包括快速文件系统、作业控制、可靠的信号处理，以及自动重新引导系统等功能。UNIX的这些功能特性，促使美国国防部高级研究项目局（DARPA）对VAX机器中的UNIX系统及其内存管理，以及VAX自己的操作系统VMS进行评估，同时投资在UNIX系统中做进一步的研究，尤其是网络通信方面，更是作为研究的重点。从而促成1981年6月推出4.1BSD，第二年的4月又首次开发出ARPANET网站的测试版，且在UNIX中实现了著名的TCP/IP和套接字。从此时开始，UNIX的发展进入了一个关键的起点。

1983年9月推出的4.2BSD是一个主要版本，其中包括完整的TCP/IP、快速文件系统、改善的系统接口，以及新的信号机制等功能。1986年6月推出的4.3BSD则包括了XNS网络软件、系统参数调整、目录文件名缓存机制，以及Internet名字服务器等功能。

2. 贝尔实验室版的UNIX

1976年，Ritchie开发了一个可移植的标准I/O库函数（stdio），这是一个令UNIX系统和C语言能够继续发展的关键步骤。标准I/O库函数的出现，使得程序员不必编写依赖于硬件的代码，这意味着采用C语言编写的程序，只需重新编译就可以移植到另一个系统。也就是说，只要系统支持C语言，不管底层的硬件结构如何，即可实现UNIX系统的移植。

此前, UNIX系统一直是在16位的计算上开发和移植的。为了把UNIX移植到32位的机器上, 促使Ritchie扩展C语言, 增加了union结构和显式的short、long和unsigned int等数据类型, 以支持程序员开发可运行于32位硬件系统上的UNIX操作系统。

另外, 由于UNIX主要是在PDP计算机上开发的, 且与PDP的内存管理单元(MMU)结合得比较紧密。为了移植到其他硬件系统上, 因而重写了内存管理例程以及设备驱动程序接口, 以便UNIX能够更容易地移植到其他结构的系统上, 最终导致UNIX第七版于1979年1月推出。UNIX第七版是第一个可移植的UNIX系统。

UNIX第七版包括一个具有完整功能特性的C编译器, 通称可移植的C编译器(PCC), 同时还增加了一个著名的命令解释程序, 通称sh或Bourne Shell (以作者Steve Bourne的名字命名), 及一整套新的设备驱动程序。

1982年, 贝尔实验室向外推出了第一个商业版本的UNIX系统——UNIX System III。其中合并了UNIX第七版和其他UNIX派生体的各种功能特性, 包括管道文件和运行队列。之后, SCO也推出了XENIX System III。

1983年, AT&T推出UNIX System V (System IV并未对外正式公布)。令业界感到惊奇的是, AT&T声称将对此UNIX及未来的版本提供支持。尤为重要的是, AT&T声称未来的版本将与现在的System V保持兼容。

UNIX System V内核引入了许多新的功能特性, 且通过数据和信息节点缓冲区, 以及散列表等技术改善了系统的性能。作为系统服务之一, 第一次引入了信号灯、消息队列和共享内存等技术。

1984年, AT&T推出UNIX System Release 2.0 (SVR 2)。其中引入了记录和文件锁的概念, 并在传统的内存交换技术之外又增加了页面调度等功能 (不同于Berkeley的页面调度技术)。此时, UNIX已安装了约10万套, 很快成为操作系统的首选。

1987年, AT&T推出了UNIX System V Release 3.0 (SVR3), 其中引入了进程通信(IPC) 机制、远程文件共享(RFS)、增强的信号操作、共享函数库、文件系统切换(FSS), 以及传输层接口(TLI) 等, 而且首次推出了STREAMS通信机制, 同时也在系统性能和安全方面做了改善与增强, 尽管这些安全增强仍属于用户级的。到1987年底, UNIX系统的安装数量达到75万套, 约有450万用户。

1.1.3 UNIX的战国时代

从UNIX第七版开始, UNIX进入了诸侯纷争的战国时代。

由于容易移植到不断发展的各种硬件平台, 商业利益驱使一些公司和研究机构开始关注UNIX系统。同时, 随着微处理器芯片技术的发展, 导致微型计算机的快速发展, 政府部门也开始对UNIX系统感兴趣, 使得销售UNIX计算机系统成为潜在的商机。许多公司开始移植UNIX系统, 其中, Microsoft与SCO公司联合推出了XENIX系统, 使得UNIX开始出现在数千个微型机系统中, 包括Zilog Z8000、Intel 808X和Motorola 68000等。

在此期间, 不同版本和变体的UNIX系统开始广泛应用于各个领域。BSD版的UNIX占有绝大部分的图形工作站市场份额, Microsoft XENIX也拥有大量的Intel 286/386微型机用户及市场份额。MIT的X Window系统, Sun公司的网络文件系统(NFS) 仍然在基于BSD版本的UNIX系统上盛行。

UNIX System V的成功也令许多计算机厂商开始在4.3BSD的基础上进行仿真。基于System V的厂商也在其计算机系统中仿真4.2BSD系统。有的计算机厂商甚至同时提供两个环境。而且，System V也采纳了BSD版的许多功能特性，如vi编辑器和curses软件包等。

因此，用户开始转向标准化组织，希望能够制定一个单一版本的UNIX操作系统。完全兼容4.xBSD、XENIX和UNIX System V，使用户不至于在选用UNIX系统时感到无所适从。

1.1.4 策略与标准之争

从上述UNIX操作系统的发展历史可以知道，UNIX已兼收并蓄，成为一个功能强大的可移植操作系统，能够很容易地移植到各种不同体系结构的硬件系统。但这也为用户带来一个很大的困惑，究竟应当在哪一个版本或哪一个派生体上开发和运行他们自己的应用。

当UNIX系统已开始成为一个事实上的多用户操作系统标准时，并不意味着UNIX是一个公共的标准。直至20世纪80年代后期，UNIX仍归AT&T拥有。AT&T既经营自己的计算机业务，又为其他竞争对手发售UNIX许可（尽管UNIX仍归AT&T公司所有，但直至20世纪80年代中后期，AT&T公司的UNIX市场份额仍然很少）。谈到UNIX，IT业界普遍认为UNIX是“AT&T的操作系统”。而对AT&T而言，UNIX只不过是其电信业务的一个支持工具。

此外，当时的UNIX操作系统还存在许多变体，如BSD版的UNIX系统、基于BSD版的SunOS系统以及微机版的XENIX系统等，也出现了许多标准，如AT&T的SVID、X/Open和POSIX等。如此一来，用户在选择UNIX系统时面临更多的困惑。UNIX System V、4.xBSD，还是XENIX？SVID、X/Open，还是POSIX？

1. SVID

1985年，AT&T发布System V接口定义（SVID）。其中定义了UNIX System V命令和库函数的接口标准。除SVID之外，AT&T还提供了一个System V验证套件（SVVS），其中包括一组实用程序，用于验证一个系统是否符合SVID。发布这个SVID的目的是验证操作系统环境，允许用户创建与硬件独立的应用软件。

AT&T对SVID的验证非常严格，每个厂商开发的UNIX系统完全符合SVID标准时，才能够贴上“System V”的标签。这种验证使许多厂商处于困境，因为一旦增强功能或修改操作系统，恐怕很难通过SVVS测试，结果导致UNIX厂商必须在两难中做出选择：要么继续提供兼容的UNIX System V操作系统，而不增加用户需要的功能增强；要么提供不符合SVID标准但能够反映用户需求的UNIX操作系统，而这又会使用户感到被锁定到一个特定厂商的UNIX系统。因此，用户又开始寻求其他标准化组织和真正的UNIX标准。

2. POSIX

1986年，IEEE 1003项目组（P1003）提出了一个可移植操作系统的“试用”标准1003.1。由于此时UNIX的使用权仍属AT&T，故这个标准取名为POSIX（Portable Operating System Interface for UNIX）。1988年，P1003又发布了最终的实用标准，其中包括POSIX操作系统接口、C语言、目录服务、Shell、实用程序、线程、软件管理以及打印机管理等。POSIX标准的覆盖范围涉及系统调用、库函数、实用程序、接口、验证与测试，以及实时与安全等。受到普遍关注的POSIX标准是P1003.1（也称POSIX.1），其中定义了系统接口。另一个重要的POSIX标准是P1003.2，主要涉及Shell与实用程序。P1003.3包含POSIX兼容性的测试方法，P1003.4包含实时扩展功能。

3. X/Open

作为一个非盈利性的国际组织，X/Open成立于1984年，开始仅由5个欧洲计算机厂商组成，后来，美国的大部分计算机公司也陆续加入其中。实际上，X/Open最初并非一个标准化组织，其主要目的是联合有兴趣开发通用应用环境（CAE）的计算机系统厂商，定义一组UNIX操作系统标准，实现软件接口的标准化，以便开发可移植的应用软件。

基于SVID，同时采纳POSIX标准，X/Open于1988年发布了它的X/Open可移植指南（X/Open Portability Guide, XPG），定义了从系统实用程序到网络服务等一系列软件接口规范。其中包括系统调用与库函数、C语言、SQL、IPC、传输层接口、命令与实用程序等。

X/Open相信，一个单一标准的UNIX系统将是IT业界计算机用户的共同愿望。现在的问题是，究竟使用那一个版本的UNIX系统作为共同的标准呢？

1.1.5 UNIX的黑暗时期

1988年~1990年是UNIX历史上最黑暗的时期，UNIX失去了一次统一的机会。

面对IT业界和用户统一UNIX的强烈愿望，UNIX厂商出于自身的利益，并未能达成一致。1987年，为了统一UNIX的各种技术，推出一个标准化的UNIX系统，AT&T与Sun公司签订了一个合约，决定共同开发一个能够容纳现有各主流UNIX系统，符合已有的国际标准，能够反映用户需求，具有通用应用接口的开放式多用户操作系统。

但是，以IBM和HP公司为首的其他厂商出于自身的利益，深恐UNIX系统会危及自己的市场，他们决定联合起来组成一个开放软件基金（OSF, Open Software Foundation），开发一个“新”的开放式操作系统。为了应对这一挑战，以AT&T和Sun公司为首的UNIX厂商成立了UNIX国际。

“UNIX大战”的结果导致UNIX系统厂商最后分裂为两大阵营，即上述的UNIX国际和OSF。此时，X/Open保持中立，且继续致力于开放操作系统规范API部分的标准化。同时把标准的范围拓展到语言、数据库连接、网络与主机互联。最终导致各种XPG版本的出现。从发布XPG4（1992年）之后，根据市场的需求，X/Open继续拓展开放系统规范的范围。许多大的公司和组织开始采用X/Open标准作为系统设计和应用开发的基础。

1.1.6 AT&T UNIX System V Release 4.0

1989年，AT&T开发出UNIX System V Release 4.0。这是一个容纳了三大主流UNIX系统特征，基于当时的所有标准，反映当时用户需求的操作系统。UNIX System V Release 4.0几乎是全面重写了UNIX内核，是一项巨大的软件工程项目。

合并与统一后的UNIX SVR4主要包括虚拟文件系统（VFS）、实时进程支持、新的进程调度类别、虚拟内存管理（VMM）、服务访问机制（SAF）、BSD文件系统（UFS）、文件系统限额、网络文件系统（NFS）、增强的STREAMS通信机制、驱动程序内核接口（DKI）、增强的信号处理、数据结构的动态分配、扩展的文件打开机制、多国语言支持（MNLS），以及Korn Shell等功能特性。如图1-1所示。

次年，AT&T发布了UNIX System V Release 4.0的发展规划路线图及发展纲要，简要概述了UNIX System未来的发展方向。

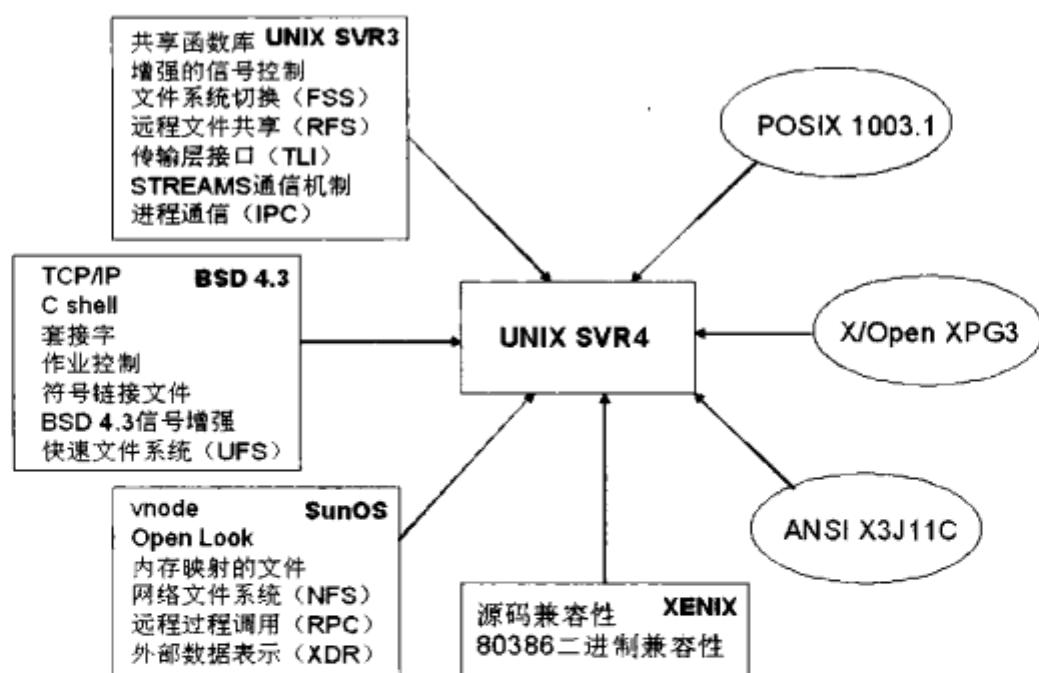


图1-1 UNIX System V Release 4.0

1.1.7 后UNIX时代

由一个相对中立的软件厂商或组织掌控UNIX系统及其规范定义与商标，将会更有利于实现UNIX系统的发展。于是，AT&T于1991年决定转让UNIX系统实验室及其UNIX商标权，以便成立一个独立的软件公司。而Novell此时恰好正在寻求一个重量级的操作系统，以便整合其NetWare软件产品。经过谈判，Novell与AT&T的UNIX系统实验室联合成立了Univel公司，并于次年推出了UnixWare。

1993年，SCO从Novell手中买下UNIX系统业务、UNIX系统源代码及其相关技术，由SCO接替Novell，继续开发UNIX系统。

1995年，SCO公司把UNIX商标转让给X/Open。1996年，X/Open又与OSF合并成立了Open Group。此时，IBM等公司开始转向UNIX SVR4。Open Group陆续制定了单一UNIX规范（Single UNIX Specification）及相关的品牌计划，如UNIX 95、UNIX 98以及UNIX03等品牌标准，确保计算机系统能够满足不同版本的“单一UNIX规范”，以维护IT业界及用户的利益。UNIX系统继续发展。

如图1-2所示的UNIX系统版本树给出了UNIX的版本演化及发展过程。

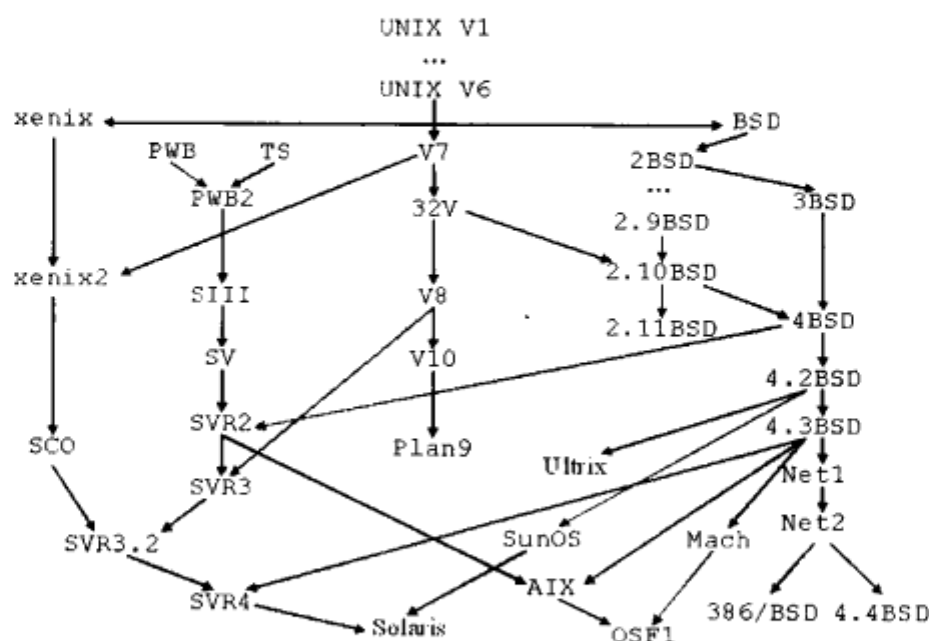


图1-2 UNIX系统版本树

UNIX发展到今天，其实已经背离了当初开发UNIX系统的初衷，背离了UNIX得以成功的简洁性、开放性和可移植性等三个主要特点。可以说，现在的UNIX系统，只有UNIX之名，而无UNIX之实，徒具UNIX的外壳。

尽管成立了各种标准化组织和行业化组织，制定了一大堆所谓的标准，可至今并没有达到真正的开放、标准化和统一，每个厂家仍然各行其是，总要塞一点自己的东西，还总是宣称符合各种标准。既然都符合标准，为什么又不一致呢？是标准定义得不严格，还是根本就没有一个真正的标准？

1.2 UNIX的层次组织结构

从层次结构的观点看，UNIX系统主要包含UNIX内核、系统调用、Shell用户界面（系统命令与实用程序），以及应用程序等组成部分。

如图1-3所示为UNIX系统的层次组织结构。其中的硬件部分为操作系统提供最基本的运算、存储和I/O处理等底层支持，例如执行指令，访问内存，执行数据的I/O通信。硬件上面是UNIX操作系统的核心，其中包括进程管理子系统、内存管理子系统、文件管理子系统以及I/O管理子系统等。UNIX内核的外层是各种实用程序，如各种Shell程序、C语言开发工具以及编辑程序等，这些实用程序都是利用UNIX核心提供的各种底层服务，为用户利用计算机资源提供各种服务。

UNIX采用建筑块式的分层组织形式，由内到外，逐层扩展而成。有一句笑话说，UNIX系统与洋葱头有两个共同点：一是两者均具有由内到外、逐层包裹的分层结构；二是在剥开表层之后，都会惹人流眼泪，愈深入其中流泪愈多。尽管是一个笑话，但第一点确实是正确的，参见图1-3。在未深入研究和学习UNIX之前，第二点也是真的。但随之对UNIX的深入了解，眼泪恐怕也就流干了。

UNIX系统鼓励“利用现有机制，采用各种组合形式，构建或增加新的功能”。基于UNIX提供的管道、I/O重定向以及其他各种机制，可以把许多命令或程序组合在一起，构成新的处理工具。而且，即使采用单个命令行，也可以建立复杂的处理机制，提供功能齐备的各色服务与支持。

UNIX内核构建在硬件系统之上，其功能是控制与协调硬件系统各部件之间的数据处理工作，负责系统的内存管理、进程调度及I/O数据通信。充分发挥底层硬件的计算、存储和I/O通信处理能力，实现计算机资源的共享。

系统调用（包括库函数）则基于UNIX内核提供的支持，为用户开发各种应用程序提供帮助，使应用程序能够访问UNIX操作系统内核提供的各种服务。UNIX系统提供的系统调用或函数库存储在内存或文件系统中，供用户随时调用。

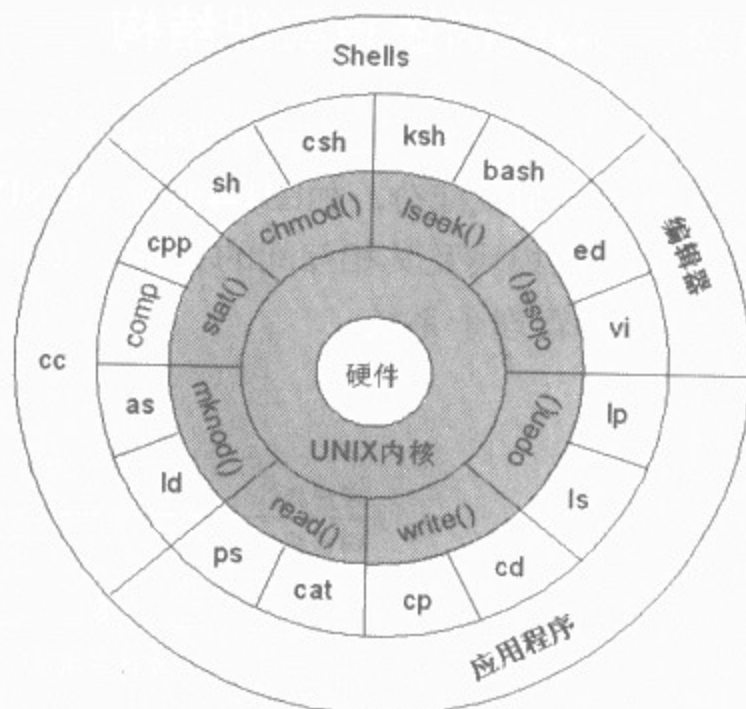


图1-3 UNIX的层次组织结构

库函数是在系统调用的基础上，为简化开发人员的编程而提供的。这些库函数或者是单个系统调用的进一步抽象，或者是组合若干系统调用提供的新功能。

Shell、系统命令和实用程序则基于系统调用，隐藏了底层的处理细节，为用户提供友好的人机界面和丰富的工具，为完成用户的各种处理需求提供支持。

处于层次结构最外层的应用程序（也包括Shell）都是可执行的二进制代码文件，以文件形式存储在文件系统中，且具有一定的组织结构，使得UNIX系统能够确定这些文件是包含机器指令的可执行程序。

应用程序通常是由源程序经编译后生成的目标程序。编译程序则是一种特殊的程序——读取由高级语言编写的源程序，然后组合应用其他工具，最终把源程序翻译成可执行的机器指令代码，生成新的程序。例如，C编译器cc还需调用其他工具，如C预处理器cpp、汇编程序as以及链接程序ld等，最终生成一个可运行的程序。

1.3 UNIX的逻辑组织结构

UNIX核心通常包含进程管理子系统、文件管理子系统、内存管理子系统，以及I/O管理子系统等主要组成部分，如图1-4所示为UNIX系统内核的逻辑组织结构，展示了UNIX内核各子系统、各功能模块及其相互间的关系。其中包括进程管理子系统、文件管理子系统、内存管理子系统与I/O管理子系统。

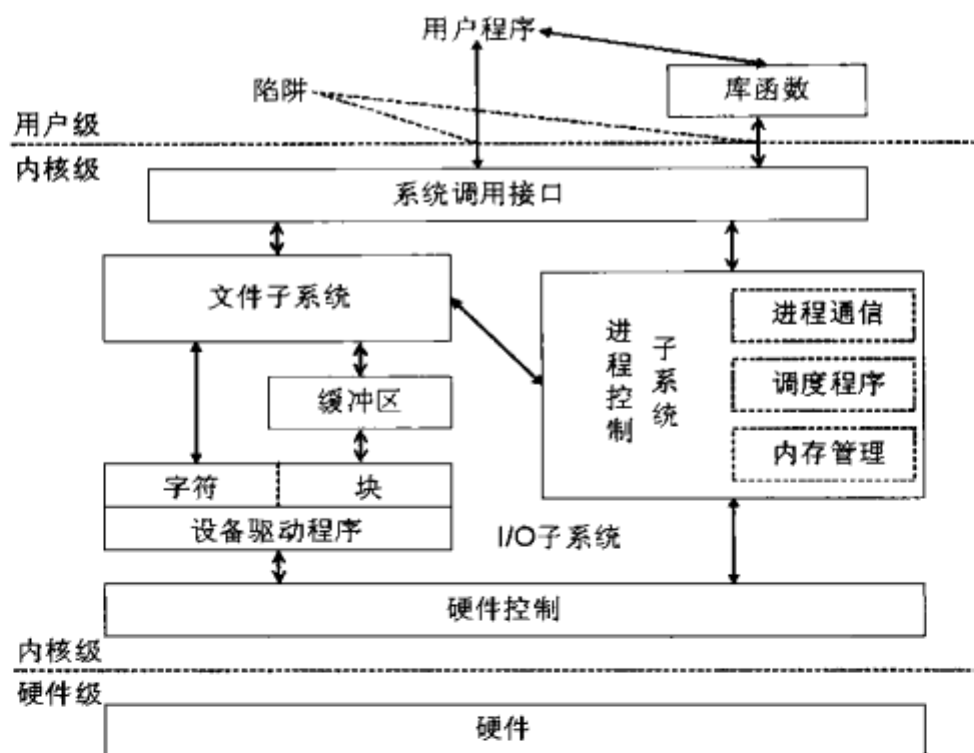


图1-4 UNIX的逻辑组织结构

文件和进程是UNIX系统模型中的两个中心概念，相应的管理子系统也是UNIX系统中的两个主要核心组成部分。在UNIX系统中，“文件占有‘地方’，进程具有‘生命’”。文件用于存储各种数据，包括文本数据和二进制数据（机器指令代码）。程序或命令就是存储在文件系统中的二进制数据文件。进程是程序或命令的执行。一个进程可以经历其生命周期中从创建、运行到终止的三个主要阶段。

如图1-4所示，UNIX操作系统可以分为三个层次：用户、内核及硬件。其中，系统调用和库函数接口是用户程序与UNIX内核之间的分界线，是UNIX内核提供的基本接口。汇编语言程

序可以直接引用系统调用，访问UNIX内核，实现复杂的功能。作为一种高级语言，使用C语言编写的应用程序既可以直接引用系统调用，也可以使用库函数，包括标准I/O函数库等，从而实现各种应用。

1.3.1 进程管理子系统

进程管理子系统是UNIX操作系统的重要组成部分之一，是UNIX系统的核心。负责控制、调度和处理人机提交的各种处理任务。进程管理子系统用于控制进程的同步、进程间的通信以及进程调度，以便控制、协调和管理进程的整个生命周期。包括进程的创建、运行、休眠和终止，以及进程的其他中间状态，如图1-5所示。因为UNIX是一个多用户、多任务的操作系统，UNIX核心必须确保每一个运行的进程能够公平合理地共享系统资源，包括CPU的执行时间，以便所有的进程能够并发地执行。

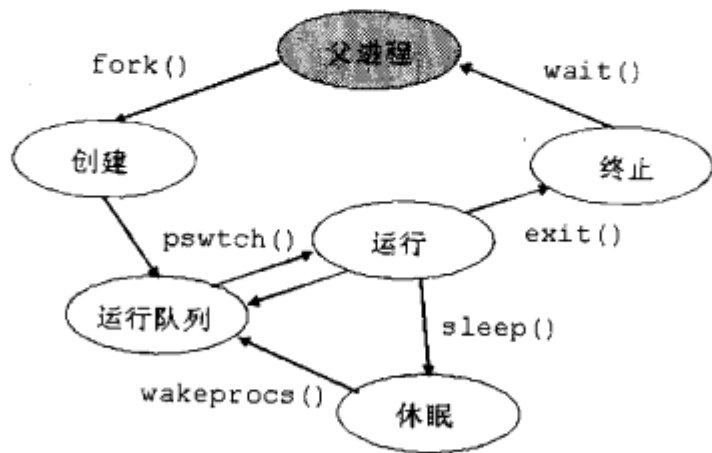


图1-5 进程的生命周期

与进程管理有关的系统调用包括fork（创建一个新的子进程）、exec（使用新的程序覆盖原有进程的映像）、exit（结束一个进程的执行）、wait（与一个使用fork系统调用创建的子进程同步，等待子进程以exit语句结束）、brk（分配进程内存空间）以及signal（控制进程对意外事件的响应）等。

进程管理子系统的基本功能如下：

- 调度。当创建一个新的进程时，进程管理子系统的进程调度模块将会基于进程的调度类别与优先级，把进程置于相应的运行队列。同时，根据系统的调度原则，为进程分配CPU，调度进程的运行。进程调度模块将会依次调度位于运行队列中的每一个进程，把进程投入运行，直至进程运行结束，或因等待资源而自动放弃CPU，或因运行时间超出预定时间片而强制放弃CPU。然后按照同样的方式，选择下一个具有最高优先级的进程，使之开始运行。

- 切换。基于进程的调度类别、优先级和时间片，每个进程都会获得一个公平使用CPU的机会。在进程的运行过程中，进程管理子系统必须确定进程占用CPU的时间，以及何时需要切换另一个进程使用CPU。

- 时间控制。进程管理子系统必须跟踪、记录进程的运行时间，监控作为系统心跳的系统时间。

- 内存使用。进程管理子系统也必须与内存管理子系统进行协调，在创建和结束进程时实现内存的分配与释放。

- 文件子系统。进程管理子系统还必须与文件子系统进行协调，以便能够加载程序文件，使进程能够访问数据文件。当需要把一个二进制数据文件加载到内存，并准备执行时，进程管理子系统需要与文件管理子系统进行交互，由文件管理子系统把可执行文件读入内存。

- 异常处理。在进程运行期间执行各种异常处理，向进程发送信号，而且，正在运行的进程也可以向其他进程发送信号，由进程管理子系统确保实现进程间的通信。

1.3.2 内存管理子系统

任何一个进程只有位于物理内存时才能运行，但是，进程的地址空间通常受限于计算机硬件实际安装的内存容量。为了解决这一矛盾，UNIX内核采用虚拟内存方式，利用页面调度和进程交换技术，实现虚拟内存与物理内存的转换与映射，使进程的内存地址空间不受物理内存的限制。

内存管理模块用于控制内存的分配。在任何时候，如果系统没有足够的物理内存可供进程使用，UNIX内核将会采用页面调度和进程交换的形式，在内存和外部存储设备之间调度相应的页面，或交换整个进程，使所有的进程都能获取公平执行的机会。

1.3.3 文件管理子系统

对于任何操作系统而言，其中的一个重要功能是提供一个一致的、有序的和易于访问的文件系统。同时还要提供一定的安全控制机制，确保数据的完整性与安全。这就是文件管理子系统的责任，其功能包括：

- 数据的完整性。文件管理子系统必须提供一个操作系统维护的文件视图。控制数据文件的访问，确定用户或进程访问的文件数据已经读入或写入内存，还是仍在磁盘设备中，跟踪进程打开和关闭的文件。

- 文件管理。记录每个文件的属性以及分配的数据块，管理整个磁盘空间、磁盘分区及其包含所有文件数据的文件系统。

- 访问权限。文件管理子系统必须确保每个用户都能够读、写和执行与其身份相适应的文件，防止非法访问，从而确保文件和数据的安全。

- 系统调用。对于任何文件和文件系统，确保所有系统调用都能够正确地实施相应的操作。任何进程均可通过一组特定的系统调用与文件管理子系统进行交互，实现文件的读写访问。因此，文件管理子系统必须确保open（打开文件）、close（关闭文件）、read（读文件）、write（写文件）、stat（查询文件属性）、chown（修改文件属主）以及chmod（修改文件的访问权限）等系统调用都能够正确地操作相应的文件。

- 并发文件访问的协调。文件管理子系统必须维护一个打开文件表，采用文件和记录锁保护，确保多个进程能够同时访问同一个或不同的文件。

- 磁盘文件与外部设备的统一管理与操作界面。UNIX文件系统的最大特点是，所有的设备均作为一种特殊文件由文件系统统一管理，同时，所有的普通文件操作方式也都可以不加区分地直接施加到设备文件中。

文件是UNIX系统组织、管理和维护数据的基本单位。UNIX系统支持不同类型的文件，如普通文件（包括ASCII文本和二进制可执行文件）、目录文件、管道文件、符号链接文件，以及表示硬件设备的特殊文件。文件系统——从用户的角度来看——是一种按树形结构组织的目录文件树。文件系统通常构建在磁盘等海量存储介质上。而文件管理子系统则用于管理文件、分配文件空间、管理存储与空闲空间、控制文件的访问和数据检索。

另外，通过虚拟文件系统，UNIX系统还可以同时支持不同类型的文件系统，使得用户能够方便地访问和存储各种存储介质中的数据。

用户或应用程序利用文件管理子系统与I/O管理子系统进行交互，通过块I/O设备驱动程序

随机访问块特殊文件，通过“原始”I/O设备驱动程序直接访问字符特殊文件中的数据，或进行数据通信，实现从UNIX内核到存储设备之间的数据传输。

1.3.4 I/O管理子系统

UNIX内核提供许多I/O通信机制，例如进程间的通信，进程与设备间的通信，机器与机器间的网络通信。I/O管理子系统的功能目的就是实现各种数据通信与传输要求。

I/O处理过程可以分为两个阶段：一是在用户地址空间与内核地址空间之间执行高级的数据传输；二是在内核地址空间与物理硬件设备之间执行低级的数据传输。

另外，I/O子系统也必须与其他子系统进行交互。例如，当需要把数据写入磁盘时，I/O子系统的高级I/O功能负责把数据从用户地址空间复制到内核地址空间，然后把数据传递到文件管理子系统，由文件系统确定把数据写至哪一个磁盘分区的那一个文件位置。最后，再由文件系统调用I/O子系统的低级I/O功能，实现磁盘数据的实际写操作。

UNIX System V Release 4.0提供三种不同类型的I/O处理：

- 文件I/O。实现文件系统中各种数据的读写。在数据的低级I/O处理阶段，数据的传输是由块设备驱动程序实现的。
- 字符I/O。从设备中直接读写数据。数据直接传递到字符设备驱动程序，由驱动程序负责与物理设备直接对话。但字符设备也必须利用高级I/O功能，在用户和内核地址空间之间复制数据。
- STREAMS I/O。类似于字符I/O，但STREAMS机制为驱动程序开发人员提供了更大的灵活性。

1.3.5 硬件系统

底层的计算机硬件系统主要提供算术与逻辑运算、数据的处理与存储，I/O中断处理及数据通信等功能。

1.4 安装Solaris操作系统

学习UNIX莫过于自己安装一个操作系统，一边看书一边上机。为了安装一个UNIX学习环境，可以从Sun公司的官方网站中，下载一个x86版的Solaris 10 UNIX操作系统。利用下载的映像文件，制作一个DVD安装介质。

最新的Solaris 10系统既可以单独安装，也可以与Windows系统安装在同一台计算机上，把Solaris系统安装到Windows系统未用的磁盘分区中。注意，在安装Solaris与Windows双引导系统时，应首先安装Windows，然后再安装Solaris系统，否则无法正常启动Solaris系统。

在安装的Windows系统中，必须为Solaris系统预留出磁盘空间。注意，Solaris只认主磁盘分区，不认逻辑分区及扩展分区，因此只能安装在主分区中。如果Windows系统已经分为多个逻辑盘，如C和D两个逻辑盘，需要事先利用Windows系统的管理工具，删除D盘，并确保把预留的磁盘空间设置为主分区。

1.4.1 硬件要求

在制作好DVD安装之后，下一步就是选择一个适当的计算机。在选择机器时，可以参照表1-1中的基本要求。

表1-1 Intel x86系列计算机硬件系统要求

硬件系统要求	简单说明
内存	最小配置384MB，建议配置512MB或更多的内存
硬盘	最小必须提供10GB的磁盘空间
swap交换分区	默认值为512MB。理论上，交换分区应为实际内存的两倍
CD/DVD驱动器	从Sun公司网站中下载Solaris 10系统软件后自制CD/DVD安装介质
显示器	1024×768显示器

注意：UNIX系统采用交换分区提供虚拟内存支持，因此，磁盘交换分区的设置非常重要。在确定交换分区的大小时，通常应以系统配置的内存为基准。如果系统配置的内存小于1GB，可以把交换分区的容量设为内存容量的两倍。如果内存大于等于1GB，交换分区的大小可以等于物理内存的容量。但在一个32位的计算机系统中，交换分区不能超过2GB。如果确实需要使用更多的交换分区，可以设置多个交换分区，如果可能，最好把每个交换分区分布到不同的磁盘中。这种解决方案既克服了交换分区的容量限制，又可以实现负载平衡，从而提高系统的性能。

1.4.2 安装步骤

下面以配有512MB内存的Intel x86计算机，刻录的Solaris 10 DVD安装介质为例，说明UNIX系统的初始安装过程，并以此作为UNIX系统的学习环境。

- (1) 检查系统的BIOS设置，确保计算机能够利用DVD引导系统。插入DVD安装介质，重新引导系统。
- (2) 在引导过程中，当出现如图1-6所示的引导界面时，按下Enter键。

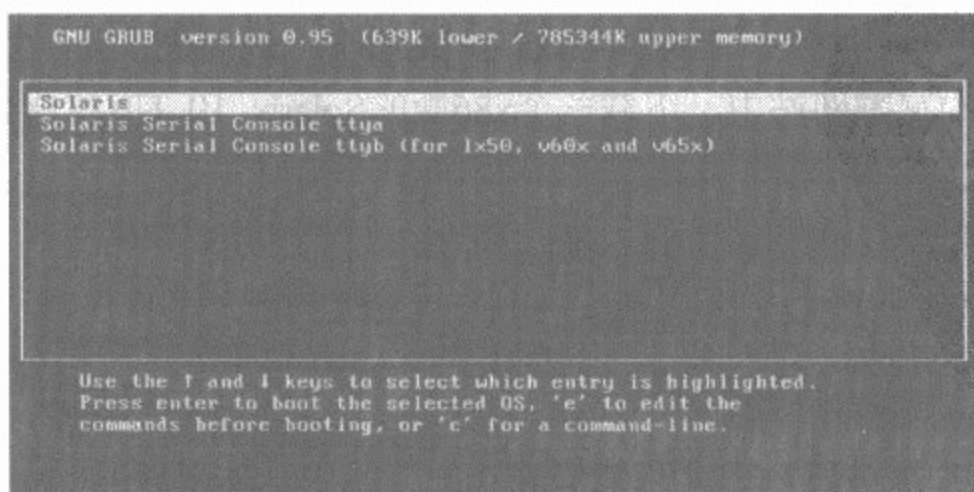


图1-6 初始安装界面

- (3) 当出现如图1-7所示的安装选择菜单时，输入“1”，选择交互安装形式（也是默认的安装方式）。

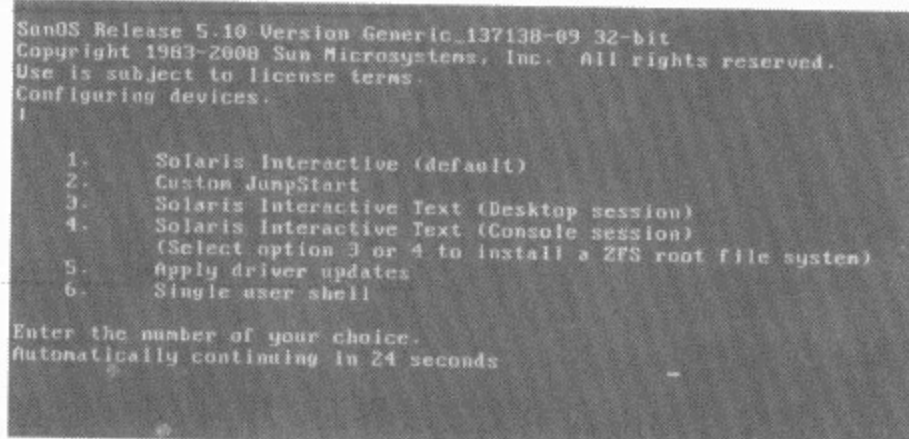


图1-7 安装方式选择界面

(4) 在如图1-8所示的键盘布局选择界面，按下F2键，选用默认的美式键盘。

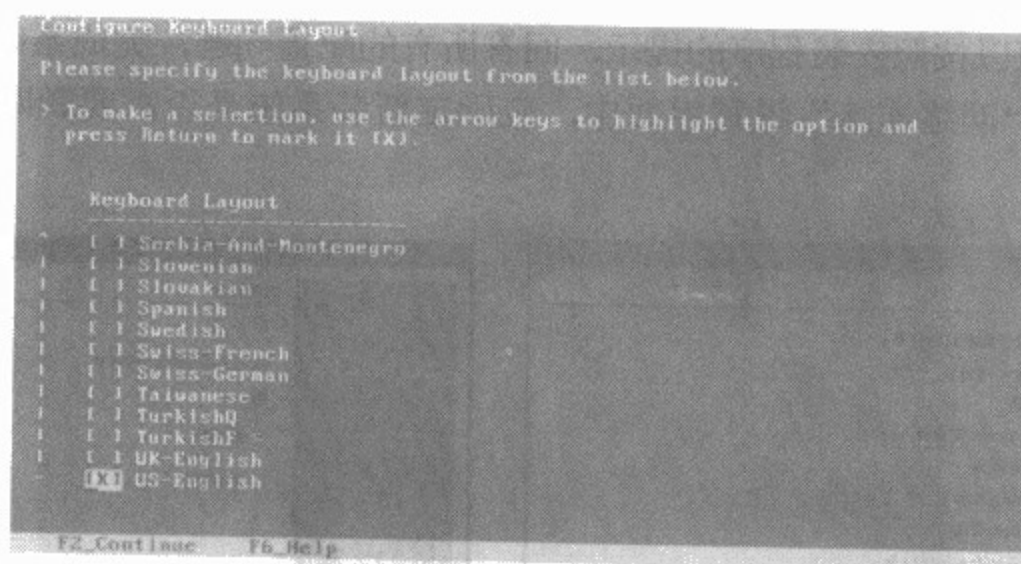


图1-8 键盘布局选择界面

(5) 之后，安装程序开始检测内存配置，如果内存大于384MB，将会提示用户当下一个屏幕出现时，应在30秒之内决定究竟采用图形界面还是文本界面安装系统，如图1-9所示。注意，如果当前系统的内存配置低于384MB，安装程序将会采用文本方式安装Solaris系统。

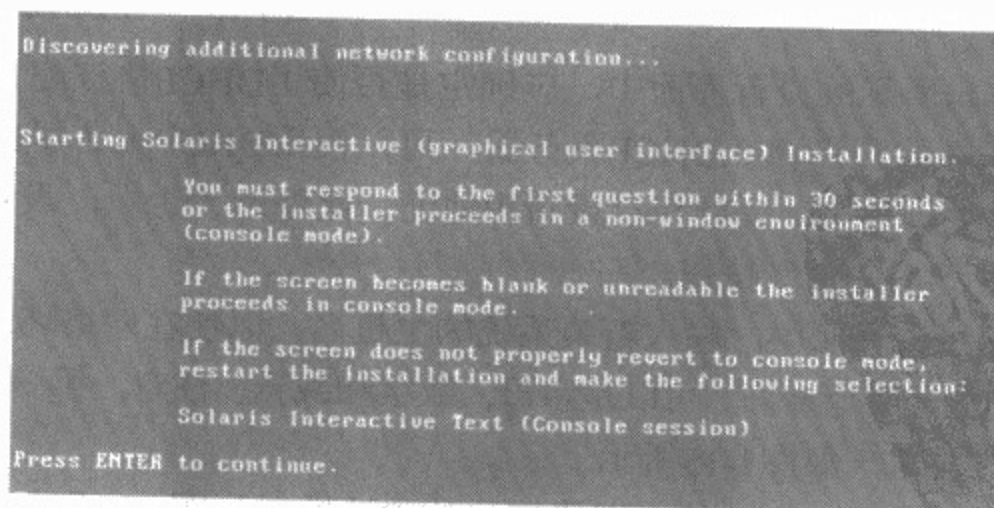


图1-9 硬件系统配置界面

(6) 当屏幕出现如图1-10所示的提示信息时，如果想要采用图形界面安装系统，必须及时按下Enter键。注意，此时一定要在30秒之内按下Enter键，否则将会自动进入文本界面安装方式。

(7) 在出现如图1-11所示的语言选择界面中，输入“6”，选择中文作为安装过程使用的语言（即选用中文安装程序），完成之后的所有安装任务。

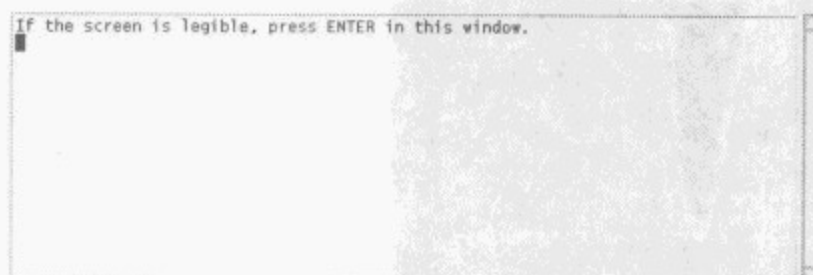


图1-10 确定安装方式界面

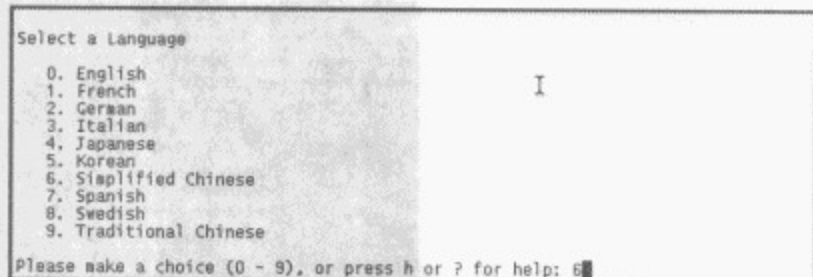


图1-11 安装程序语言选择界面

(8) 选择语言之后，将会开始出现中文安装界面，其中第一个就是“欢迎”界面，如图1-12所示。单击“下一步”按钮之后，安装程序将会通过一系列设置界面，提示用户输入或选择必要的配置参数，引导用户完成整个安装过程。配置参数包括网络连接、Kerberos安全机制、名字服务、日期与时间以及超级用户密码等。

(9) 此后，可以按照安装程序的提示，回答所有的配置问题，完成系统安装任务。例如，在如图1-13所示的“网络连接”设置界面中，应确定新装系统是否需要联网，然后单击“下一步”按钮。

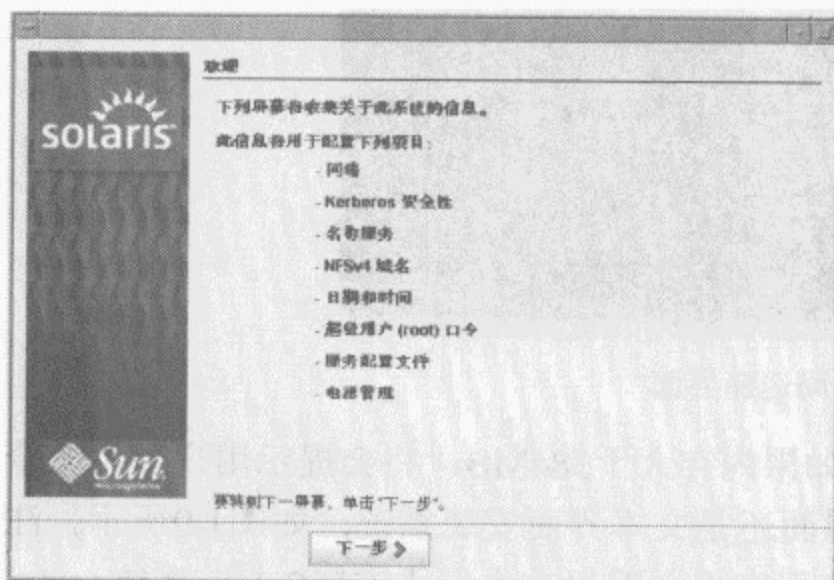


图1-12 “欢迎”界面

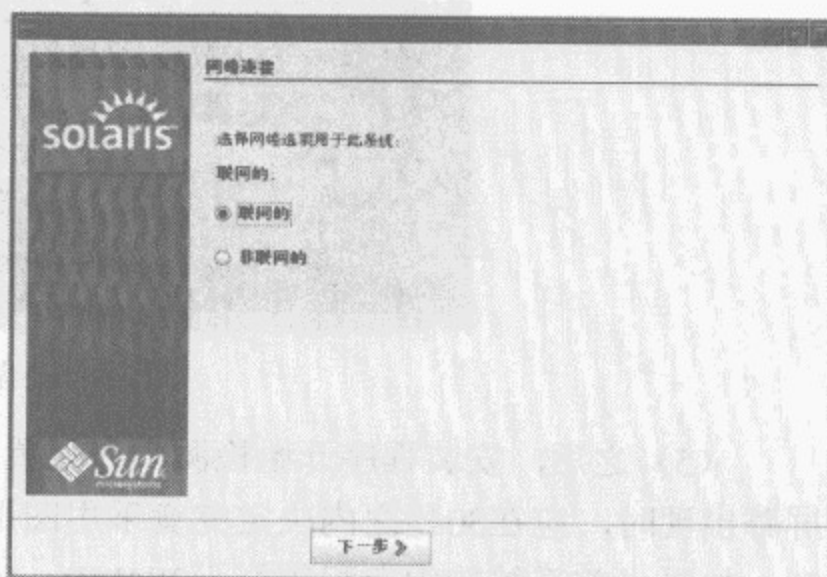


图1-13 “网络连接”选择界面

(10) 在如图1-14所示的设置界面中，应确定是否使用DHCP协议配置当前系统的网络接口。如果当前系统所在的网络存在DHCP服务器，可以使用DHCP服务器自动配置新装系统网络接口的IP地址等参数。否则，需要自己设置IP地址。此处选“否”，表示自己设置IP地址等网络参数，然后单击“下一步”按钮。

(11) 在主机名设置界面中，输入新装系统的主机名，以便在网络环境中能够区别每一个系统。主机名至少应包含两个字符，可以是字母、数字和减号。注意，第一个字符最好是字母。然后单击“下一步”按钮，如图1-15所示。

(12) 在IP地址设置界面中，输入网络接口的IP地址，如图1-16所示。

(13) 在网络掩码设置界面中，输入子网掩码，然后单击“下一步”按钮，如图1-17所示。

(14) 在IPv6设置界面中，确定是否启用IPv6。如果想要学习IPv6协议，或网络中其他系统也使用IPv6协议，可以启用IPv6。否则，选用默认的IPv4，即传统的TCP/IP协议，然后单击“下一步”按钮，如图1-18所示。

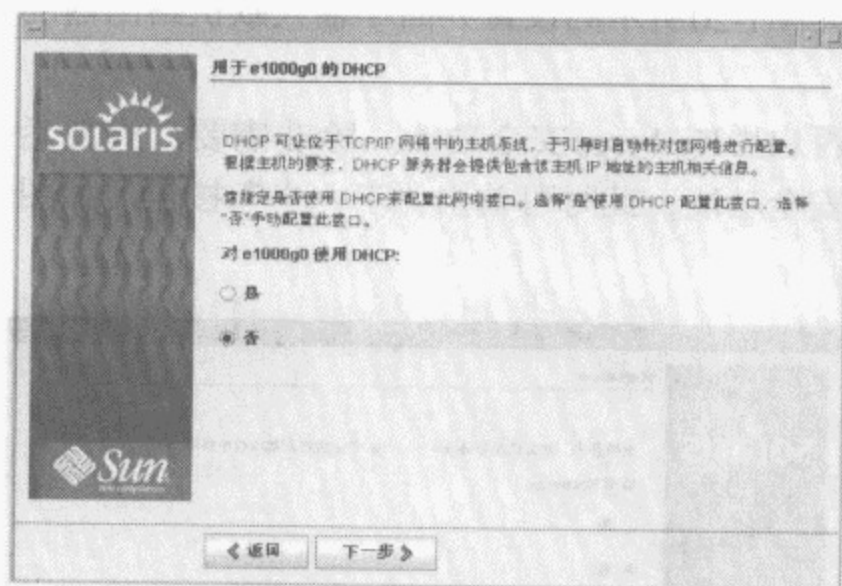


图1-14 DHCP配置界面

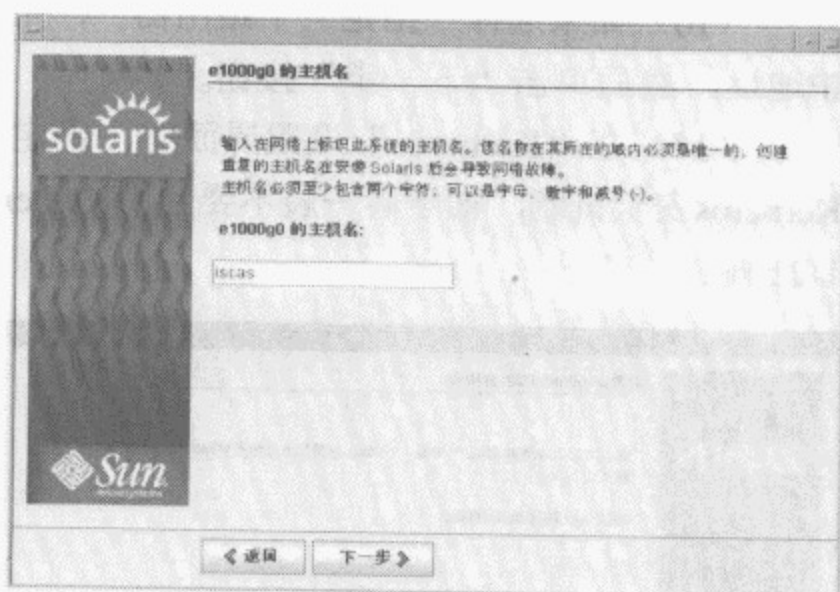


图1-15 主机名设置界面

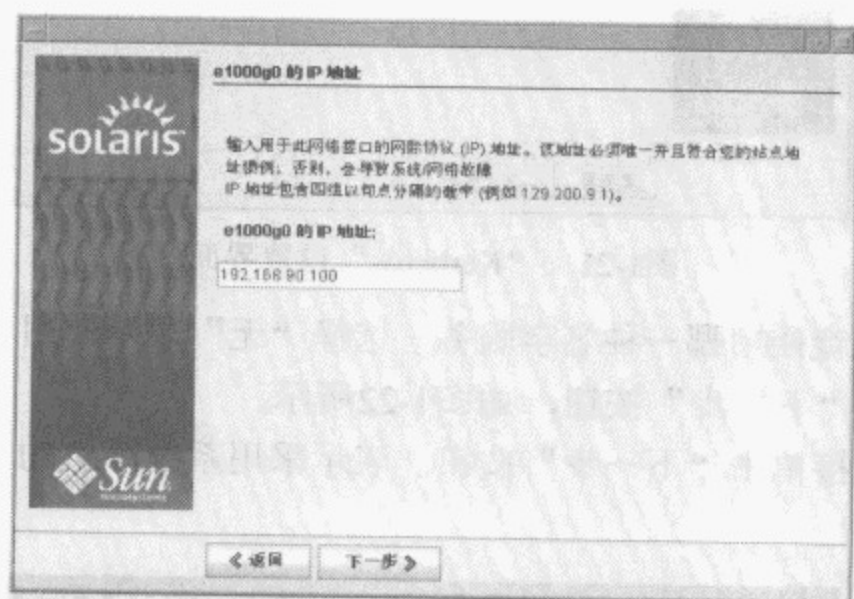


图1-16 IP地址设置界面

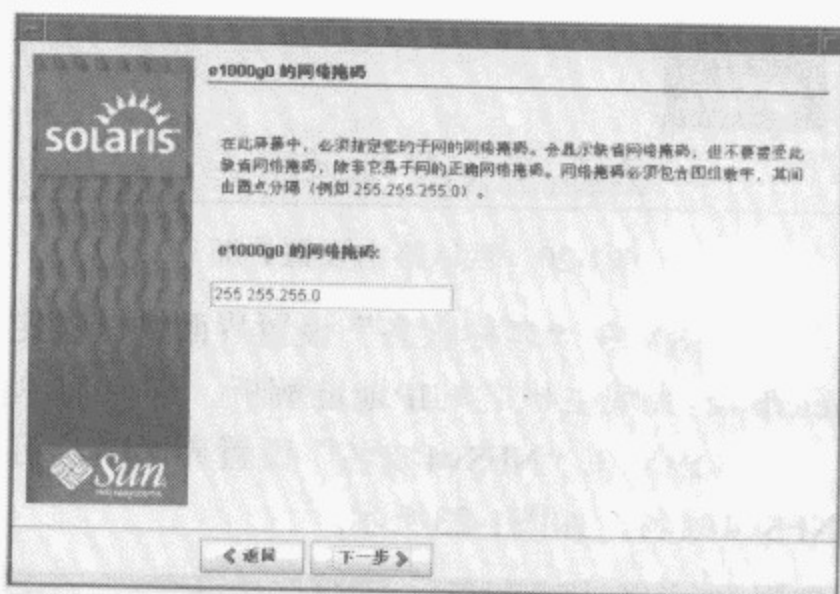


图1-17 网络掩码设置界面

(15) 在默认路由设置界面中, 确定是由安装程序从所在的网络中检索一个路由器, 指定一个路由器, 还是暂不指定。除非确定新装系统所在的网络确实存在路由器, 否则一般不要选用“检测一个路由器”。即使存在, 最好还是 选用“指定一个路由器”。然后单击“下一步”按钮, 如图1-19所示。

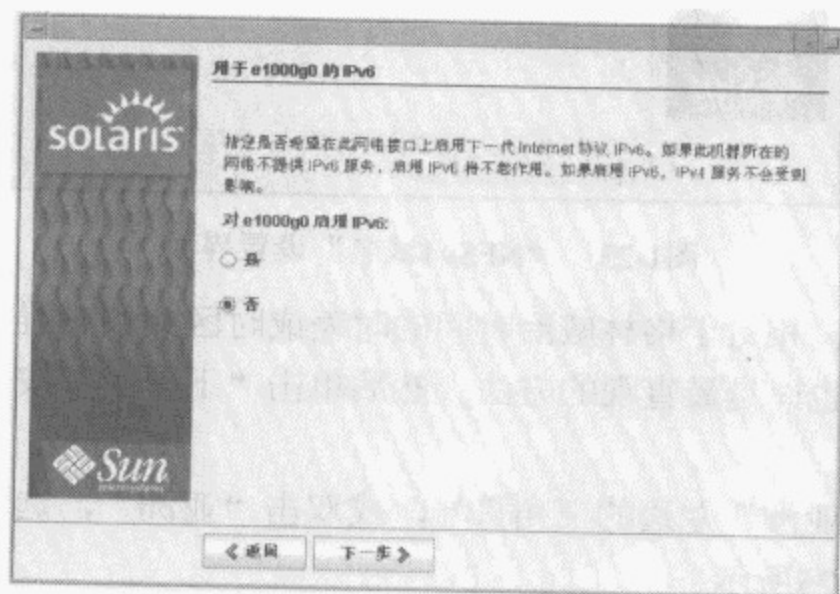


图1-18 IPv6设置界面

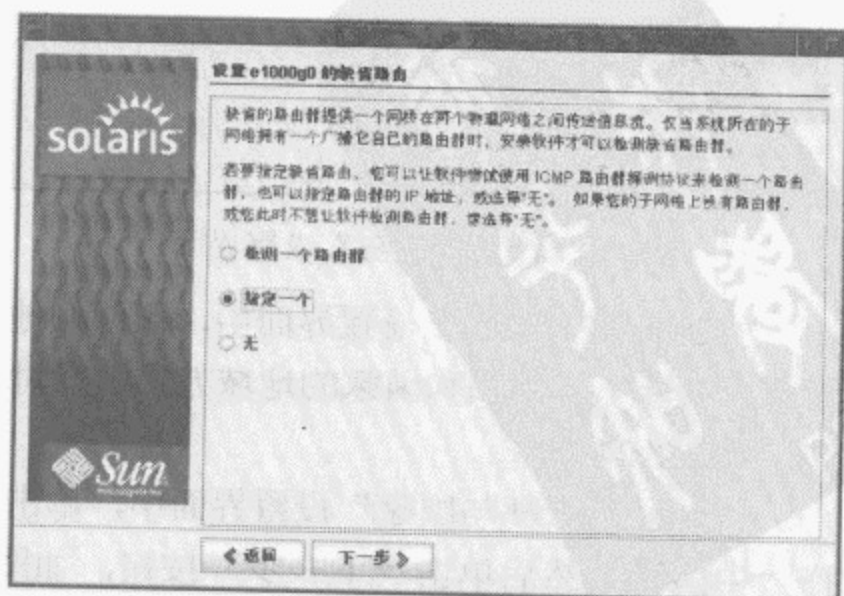


图1-19 默认路由选择界面

(16) 如果选择“指定一个路由器”，可在如图1-20所示的设置界面中输入默认路由器的IP地址，然后单击“下一步”按钮。

(17) 在“Kerberos”设置界面中，确定是否启用Kerberos安全功能。除非需要，或熟悉Kerberos安全机制，初学者一般不要启用Kerberos安全功能。此时可单击“下一步”按钮，如图1-21所示。

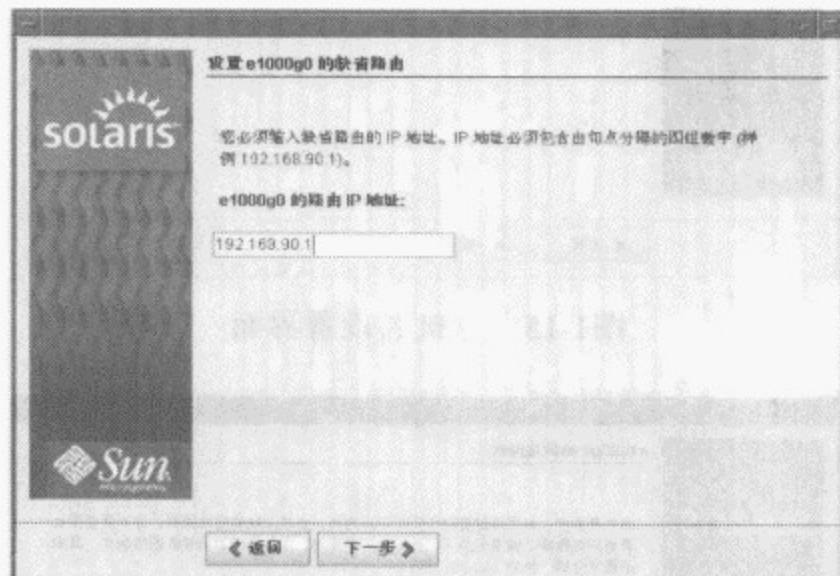


图1-20 默认路由设置界面

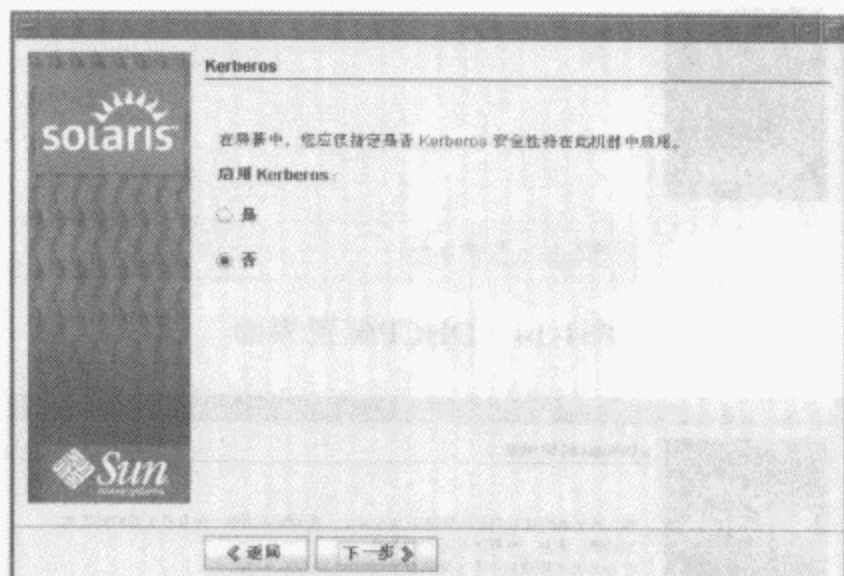


图1-21 “Kerberos”设置界面

(18) 在“名称服务”设置界面中，确定究竟使用哪一种名字服务。选择“无”表示使用/etc/hosts实现主机名的IP地址解析。确定后单击“下一步”按钮，如图1-22所示。

(19) 在“NFSv4域名”设置界面中，可直接单击“下一步”按钮，表示采用系统派生的NFSv4域名，如图1-23所示。

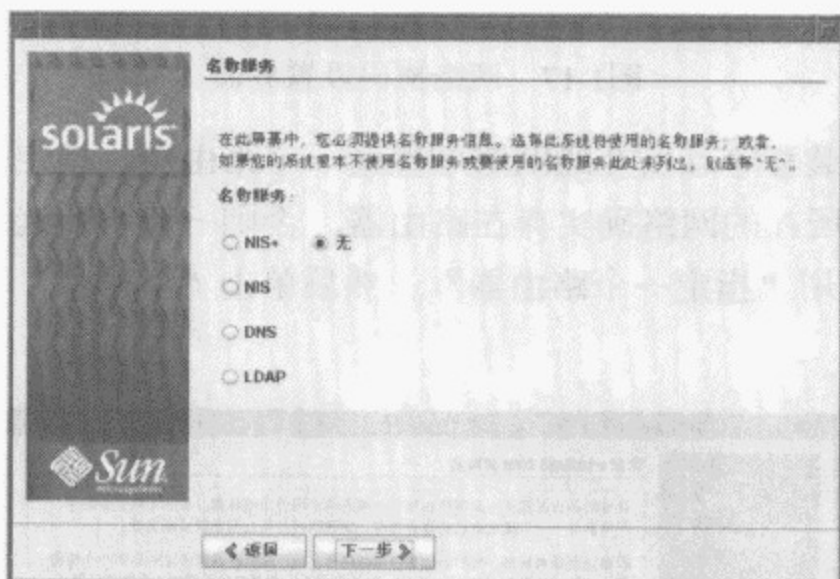


图1-22 “名称服务”设置界面

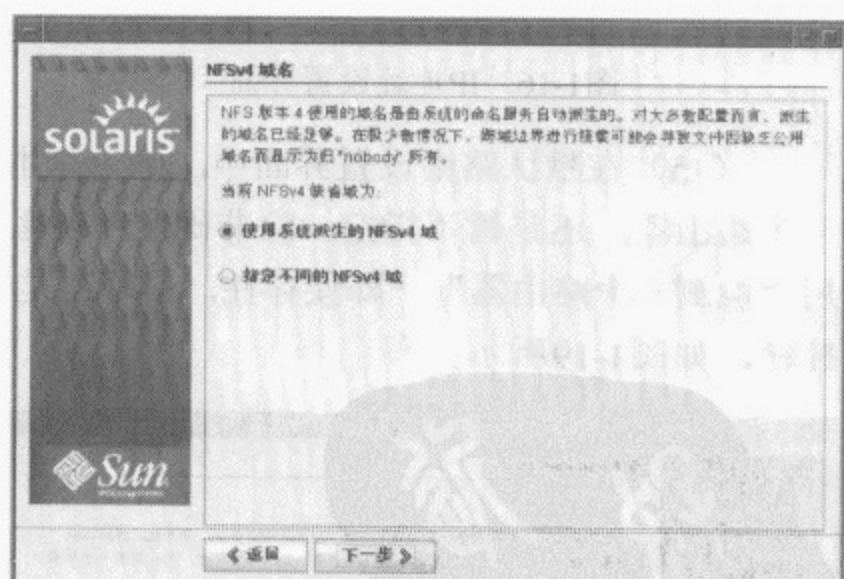


图1-23 “NFSv4域名”设置界面

(20) 在“时区”设置界面中，可以按地域、相对于格林威治时间的时差或时区文件设置时区。实际上，按洲和国家的地域方式选择时区也许是最直观的方法。然后单击“下一步”按钮，如图1-24所示。

(21) 在“洲与国家”设置界面中，单击“亚洲”左边的三角符号，或双击“亚洲”，选择“中国”，然后单击“下一步”按钮，如图1-25所示。

(22) 在“日期和时间”设置界面中，可以接受或调整安装程序提供的日期和时间。然后单击“下一步”按钮，如图1-26所示。

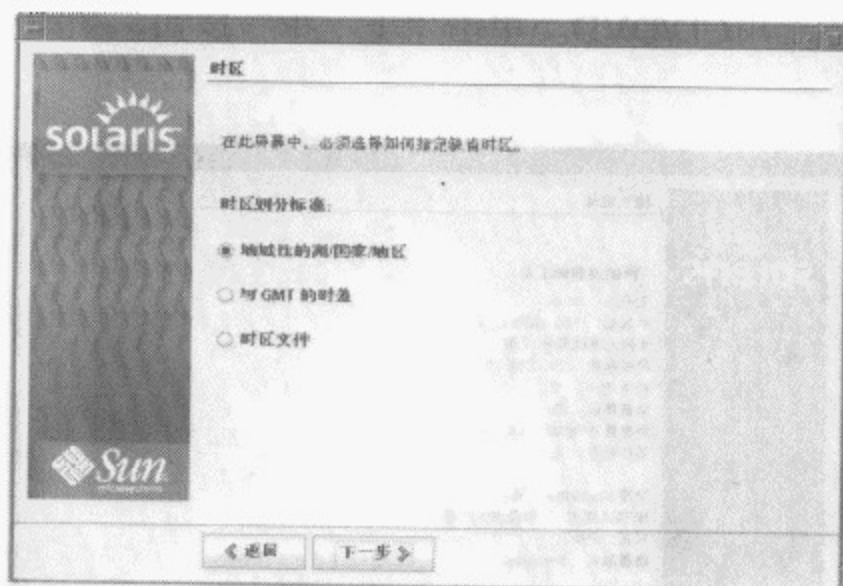


图1-24 “时区”设置界面

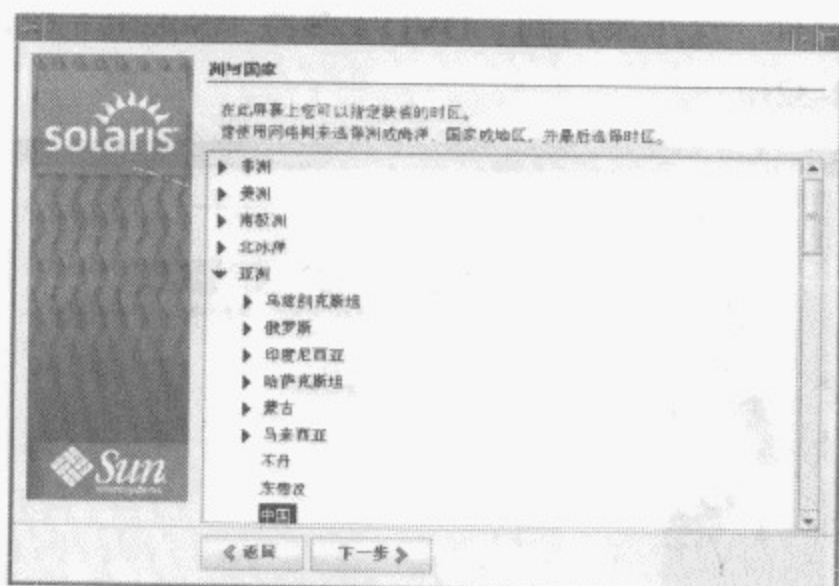


图1-25 “洲与国家”设置界面

(23) 在超级用户密码设置界面中，输入并记住超级用户root的密码。否则，无法注册访问新安装的Solaris系统。然后单击“下一步”按钮，如图1-27所示。

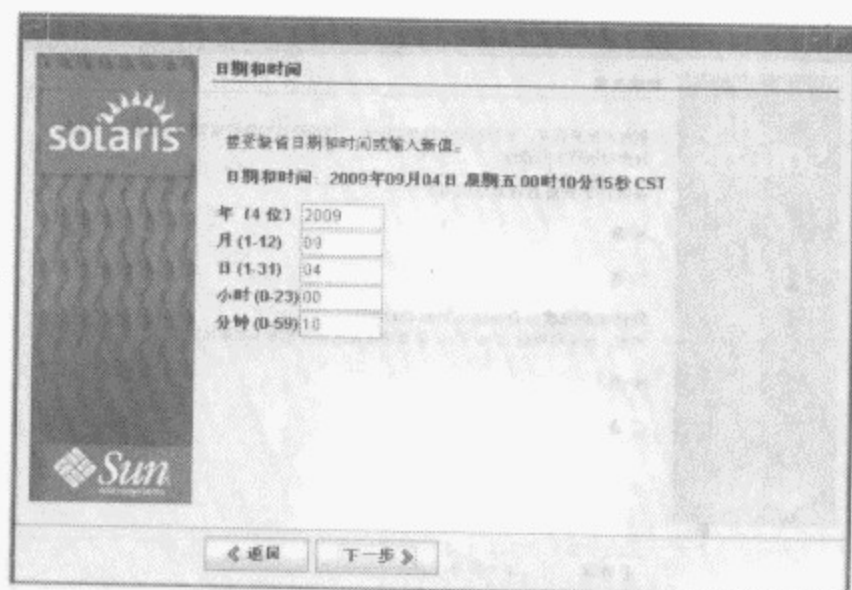


图1-26 “日期和时间”设置界面

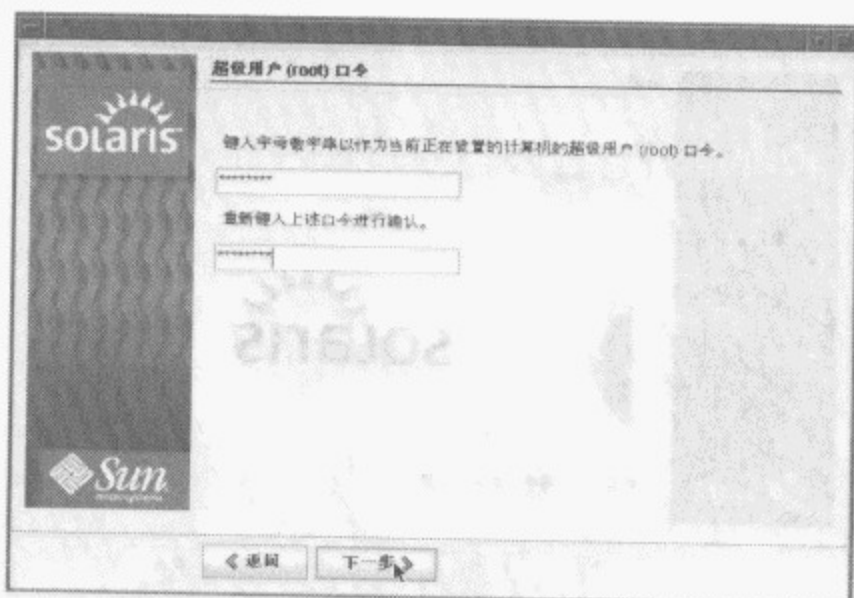


图1-27 超级用户密码设置界面

(24) 在远程服务设置界面中，取决于是否启用新装系统中的Telnet与FTP服务器，可以选择“是”或“否”。选择“是”表示启用telnetd及ftpd等传统网络服务，允许用户使用telnet远程注册，使用ftp访问本地系统。然后单击“下一步”按钮，如图1-28所示。

(25) 在回答或提供了所有的配置参数之后，安装程序已经完成了系统信息的初步收集过程。在“确认信息”界面提供的汇总信息中，如果发现之前的输入信息有误，可以返回先前的界面，重新设置。如果一切正常，可以单击“确认”按钮继续，如图1-29所示。

(26) 至此，安装程序即可开始执行实际的安装步骤。单击“下一步”按钮继续，如图1-30所示。

(27) 在“安装选项”设置界面中，可以选择在软件全部安装完成之后是否重新引导系统，以及在软件安装完成之后是否自动弹出CD/DVD（注意，引导CD或DVD需人工弹出）。单击“下一步”按钮继续，如图1-31所示（注意，在随后弹出的“通知”窗口中单击“确定”按钮）。

(28) 在“指定媒体”设置界面中，需要指定其他安装介质或安装方式。除了引导CD/DVD之外，其他安装介质可以是CD（或同一DVD），也可以是位于NFS等服务器中的CD/DVD映像

文件。这里采用同一DVD安装整个Solaris系统，故选用CD/DVD。单击“下一步”按钮，如图1-32所示。

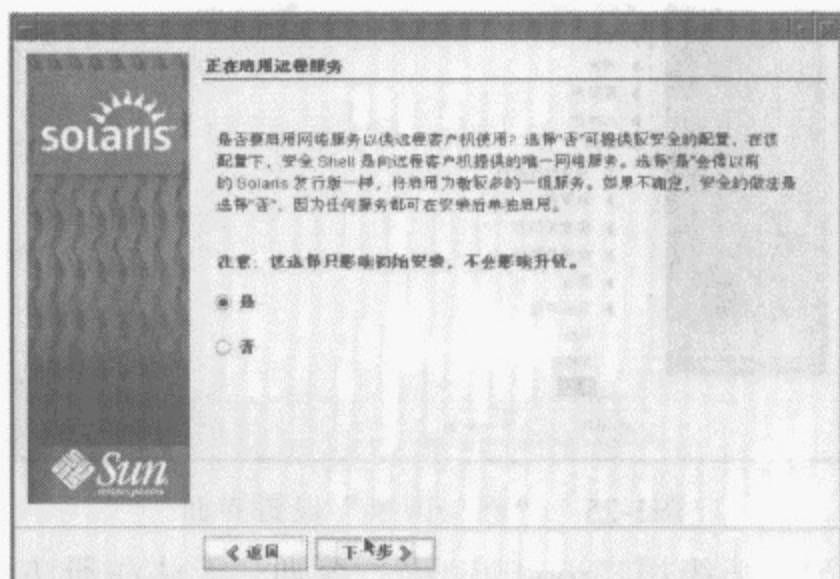


图1-28 远程服务设置界面

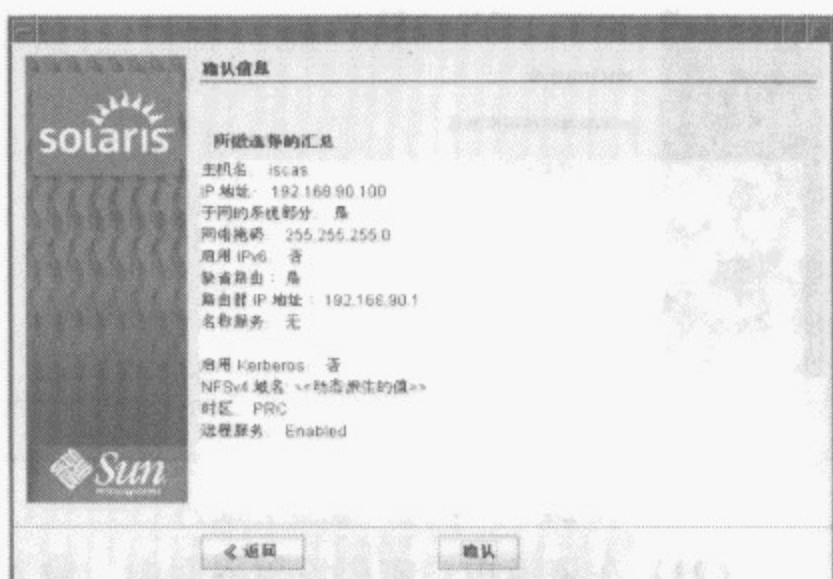


图1-29 “确认信息”界面



图1-30 “欢迎”界面

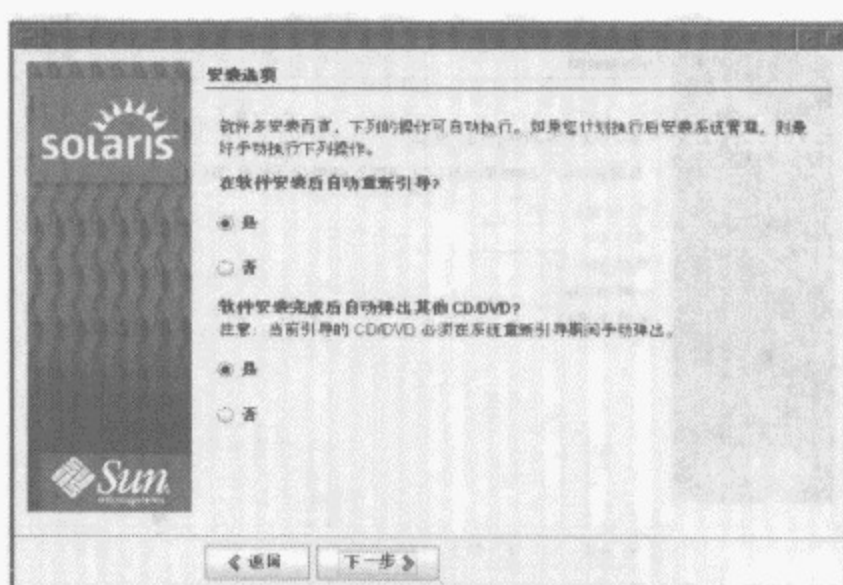


图1-31 “安装选项”设置界面

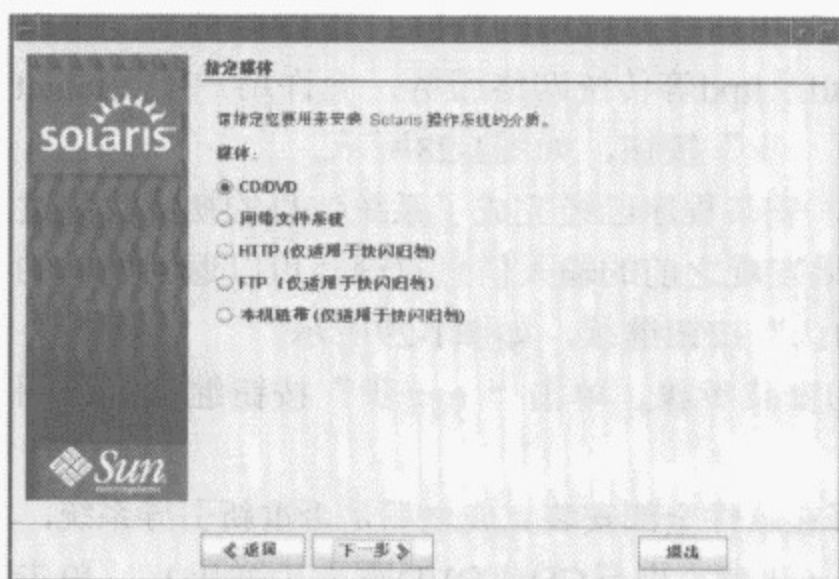


图1-32 “指定媒体”界面

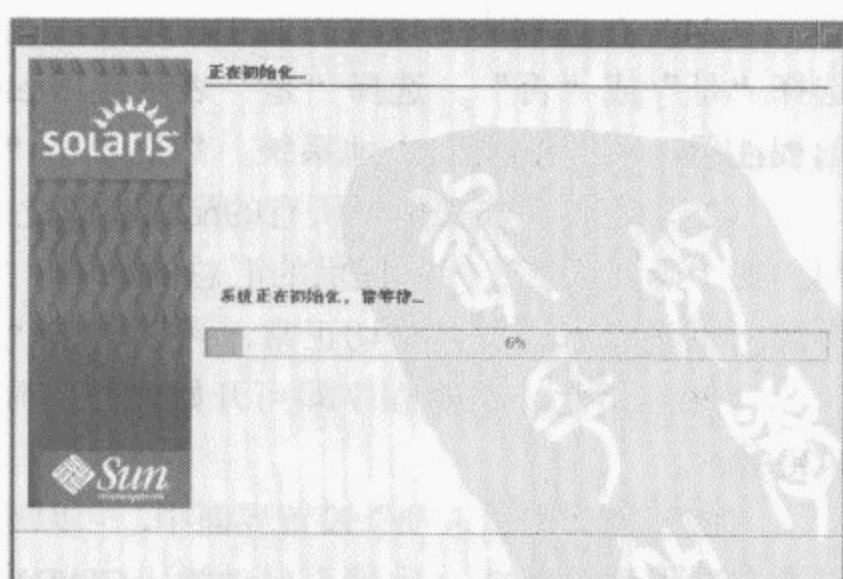


图1-33 系统初始化界面

(29) 之后，系统开始执行初始化（此时无需用户干预），如图1-33所示。

(30) 在“许可证”界面中，需勾选“接受”复选框，然后单击“下一步”按钮，如图1-34所示。

(31) 在“选择安装类型”界面中，可以选择系统的软件安装方式：即默认安装或自定义安装。默认安装方式意味着按照默认的文件系统布局，安装完整的Solaris系统。选择自定义安装方式可由用户定制文件系统，选择Solaris定义的软件组。通常不建议选择默认安装方式，因为其磁盘划分并不合理。选择“自定义安装”后单击“下一步”按钮，如图1-35所示。

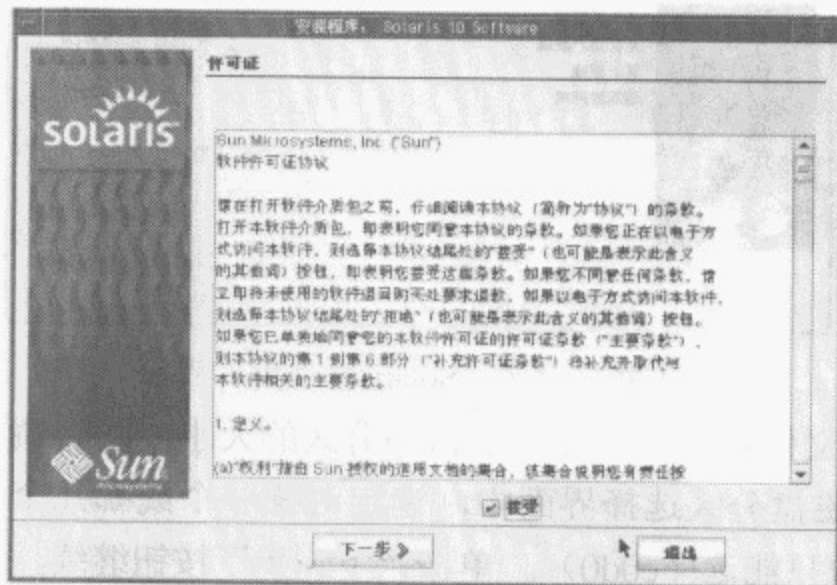


图1-34 “许可证”界面

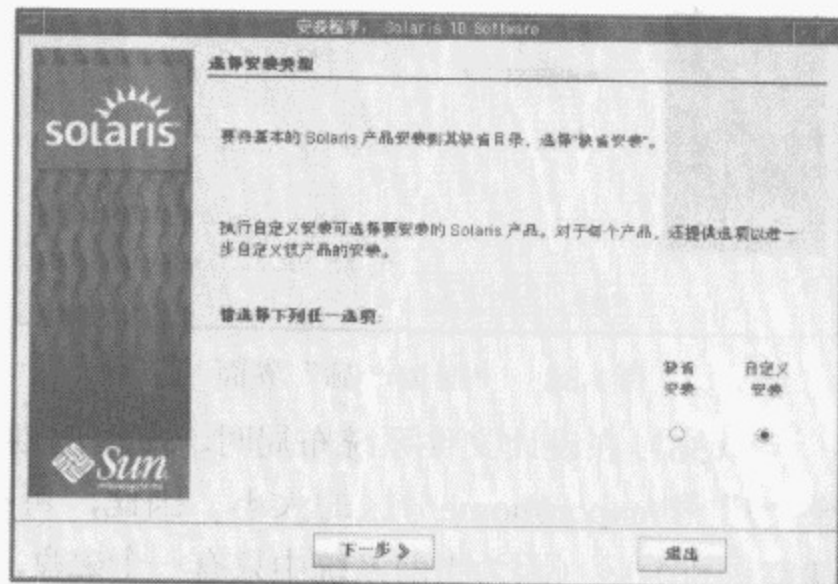


图1-35 “选择安装类型”界面

(32) 在两个选择语言环境设置界面中，可以直接单击“下一步”按钮，表示选择默认的“亚洲”及“中文(zh)”语言环境，如图1-36和图1-37所示。

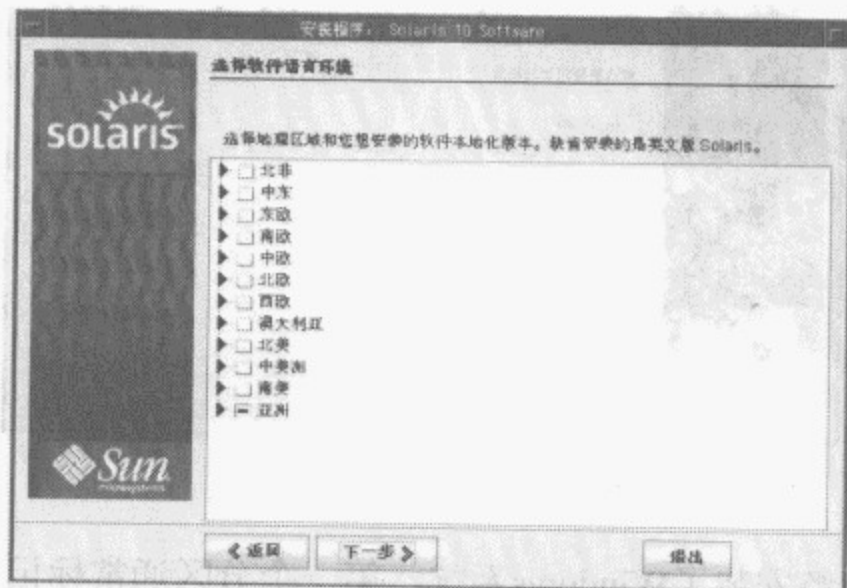


图1-36 “选择软件语言环境”界面

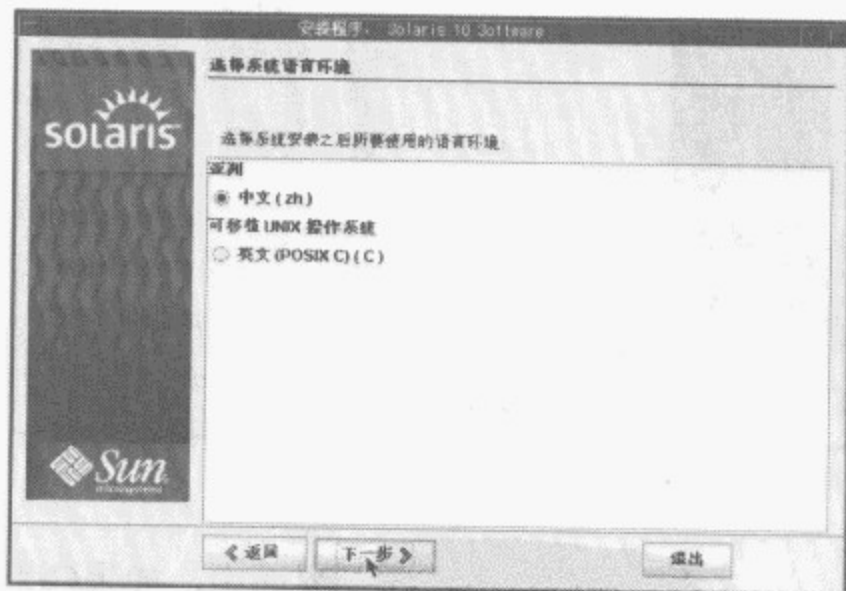


图1-37 “选择系统语言环境”界面

(33) 在“附加产品”设置界面中，可以直接单击“下一步”按钮，表示不安装任何附加产品，如图1-38所示。

(34) 在“选择Solaris软件组”界面中，可以选择默认值“整个群组”，也可以选择“整个软件群组以及OEM”，以便安装OEM支持软件。然后单击“下一步”按钮，如图1-39所示。

(35) 如果硬件系统中配有多个磁盘，且需要把Solaris系统分散安装到多个磁盘，可以在“选择磁盘”设置界面中，把需要安装Solaris系统的磁盘依次加到“选定的磁盘”一列中。由于当前系统只配有一个磁盘，且已自动选中，故“可用磁盘”一列为空。此时可直接单击“下一步”按钮，如图1-40所示。

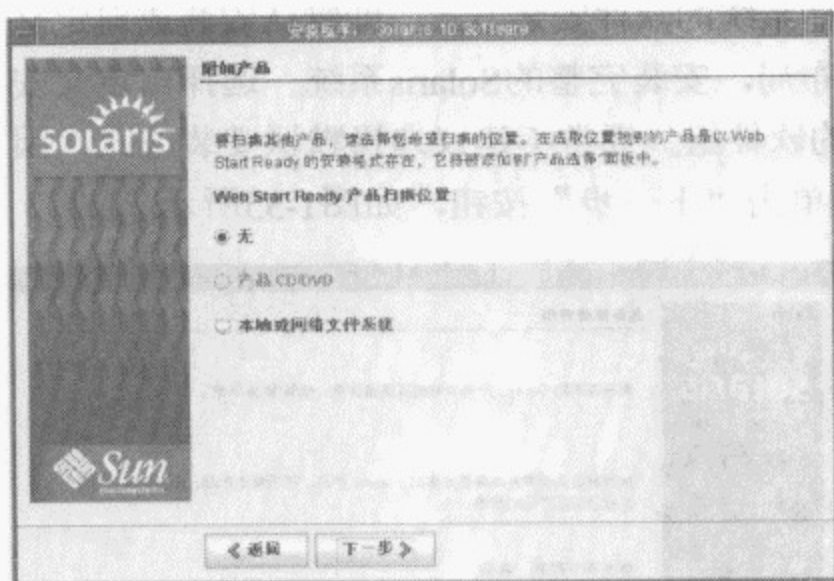


图1-38 “附加产品”界面

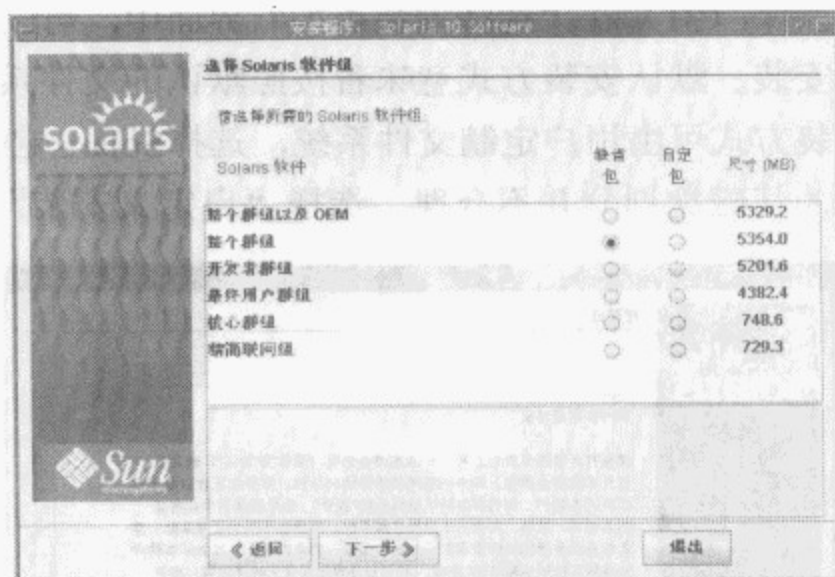


图1-39 “选择Solaris软件组”界面

(36) 在设计文件系统布局时，应自己划分文件系统，自己设定swap分区的大小，同时调整“/”和/export/home分区的大小。因此，可在磁盘分区选择界面中，选定对哪一个或哪几个磁盘进行分区（因为当前系统中只有一个磁盘，故只能选择c0d0）。单击“下一步”按钮继续，如图1-41所示。

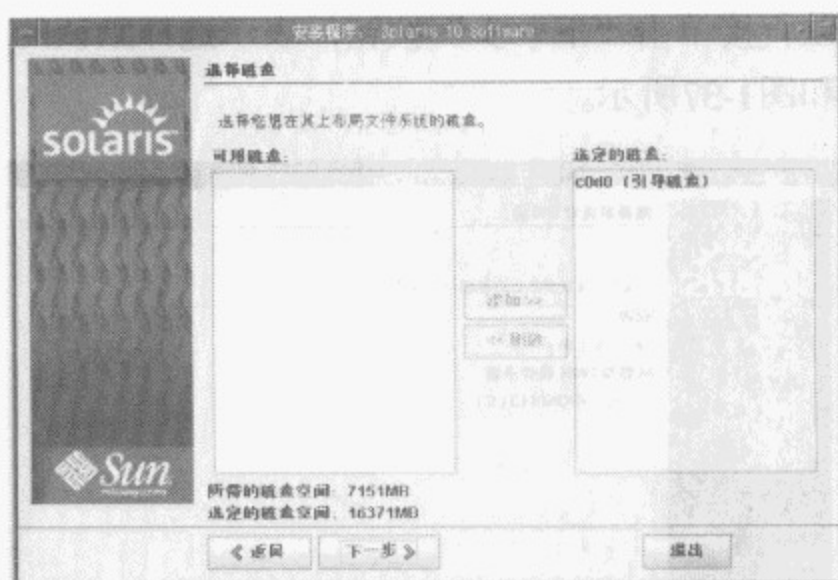


图1-40 “选择磁盘”界面

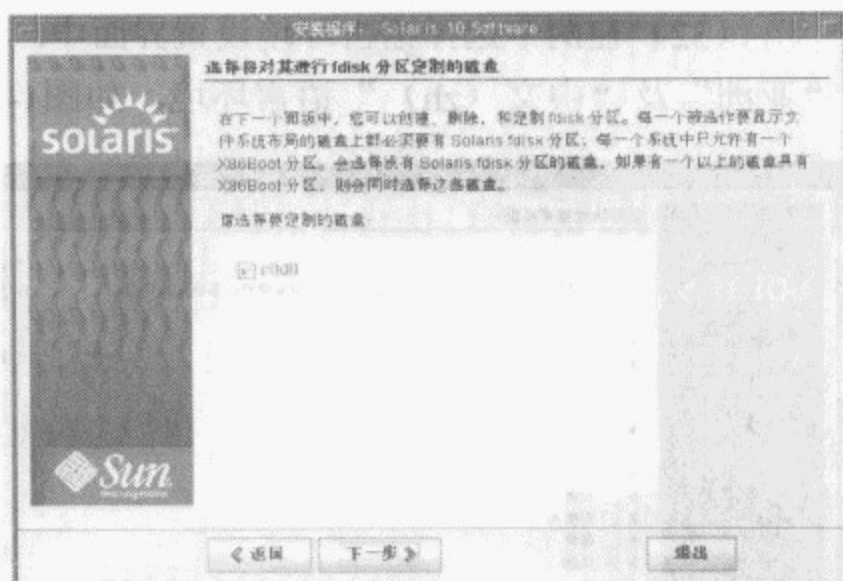


图1-41 磁盘分区选择界面

(37) 在磁盘分区定制界面中，如果C盘已经安装了Windows系统，第一个分区通常标记为“Unknown”，同时在右边一列给出其占用的存储空间。此时可以选择标记为“Unused”，且给出了分区大小的其他空闲分区，并从下拉列表框中选择Solaris。如果系统中仅安装一个Solaris，可直接单击“下一步”按钮，如图1-42所示。

(38) 在选定的Solaris磁盘分区中，安装程序将会给出默认的文件系统分布建议。如果想要调整，修改默认的文件系统空间分配，或增加单独的文件系统（如/usr等），可以单击“修改... (Modify...)”按钮，如图1-43所示。注意，安装程序给出的磁盘空间分配并不一定适当。

(39) 如果想要重新划分文件系统，或增加新的文件系统，可在“文件系统”一列中逐一列出希望创建的每一个文件系统，并为其分配磁盘空间。在此例子中，我们保持原有的文件系统划分，但对文件系统的磁盘空间进行调整（扩大了“/”文件系统和swap交换空间的容量，但缩小了/export/home文件系统的容量）。然后单击“应用 (Apply)”和“确定 (OK)”按钮，如图1-44所示。

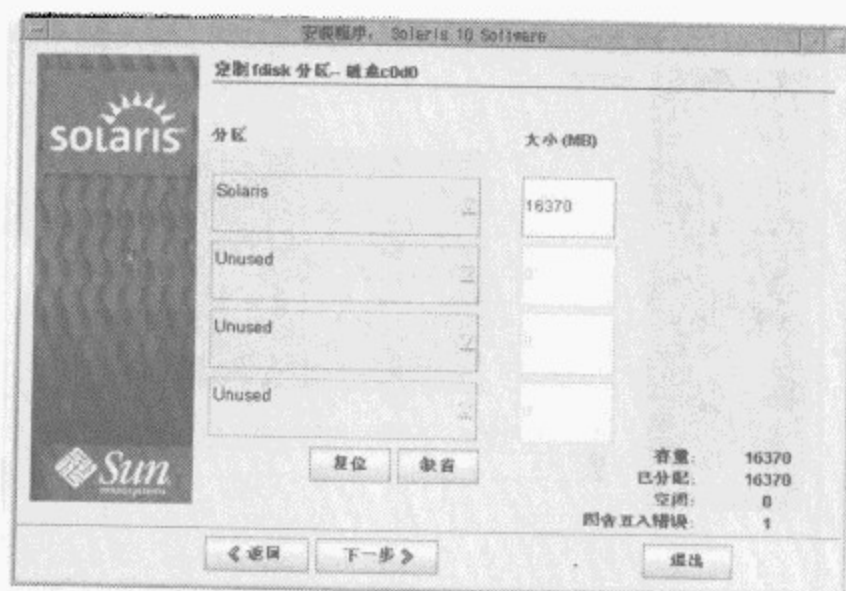


图1-42 磁盘分区定制界面

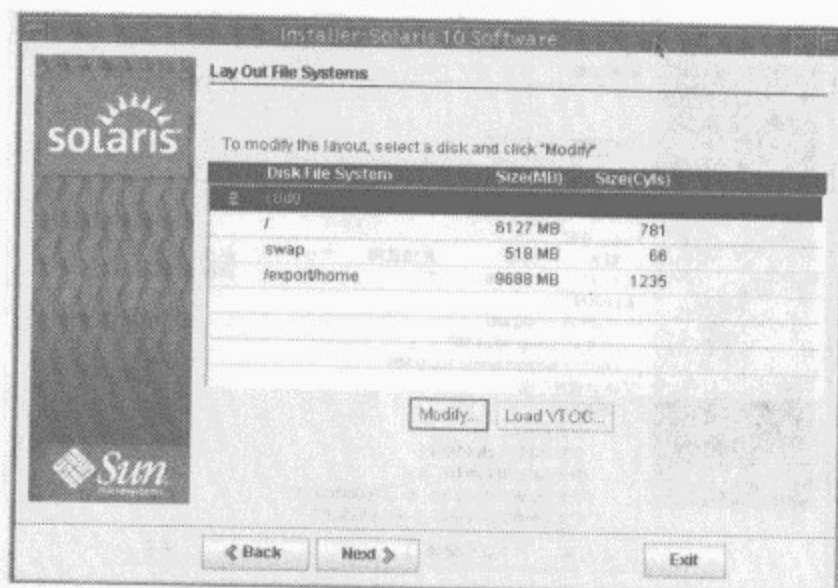


图1-43 默认的文件系统分区布局界面

(40) 完成文件系统的划分和空间分配之后, 安装程序将会重新显示修改后的文件系统分布情况。如果上述操作有误, 可以重覆先前的步骤。在确认磁盘空间分配正确无误之后, 可直接单击“下一步”按钮, 如图1-45所示。

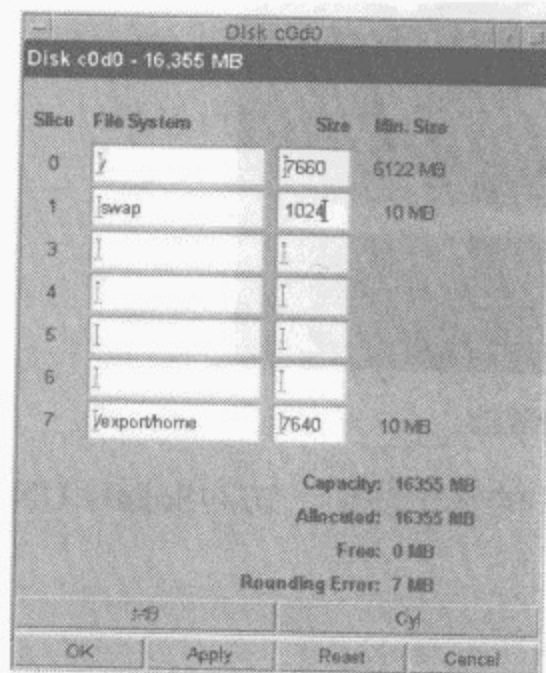


图1-44 文件系统分区调整界面

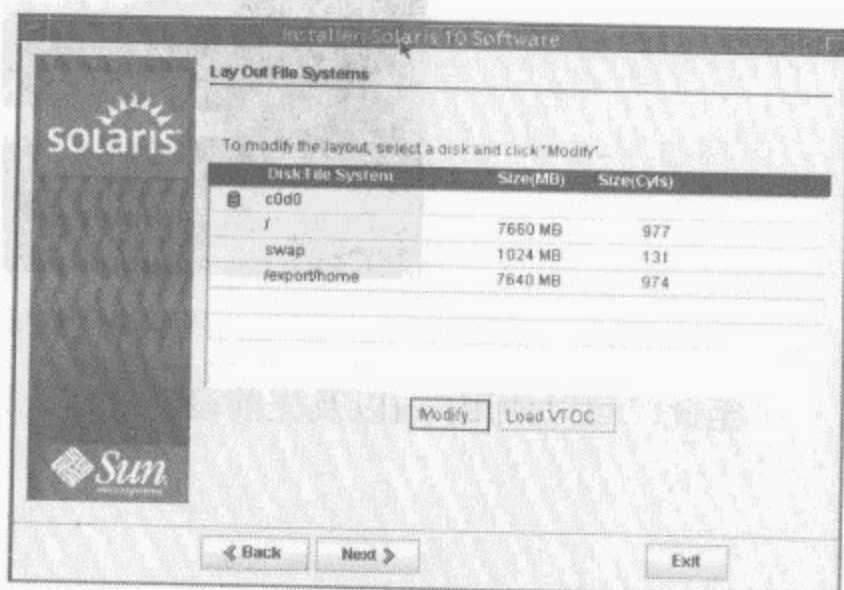


图1-45 文件系统分区布局界面

(41) 在“准备安装”设置界面中, 确认先前提供或选择的信息是否有误。如果需要, 可以单击“返回”按钮进行修改。注意, 这是最后一次校正错误(包括磁盘分区和文件系统布局等)的机会。如果一切正常, 单击“立即安装”按钮, 即可开始安装Solaris系统软件, 如图1-46所示。

(42) 然后, 安装程序将开始创建文件系统, 以及复制软件包等, 如图1-47。注意, Solaris系统的安装过程需要较长的时间(从人工干预的角度来讲, Solaris的安装过程至此实际上已经结束, 尽管后面还会出现几个对话框, 基本上只需按照屏幕上的提示操作即可, 如单击“下一步”按钮或“重新引导”按钮, 也可以任由安装程序自动执行下一步, 包括重新引导系统)。

在软件包全部安装结束之后, 安装程序将会提供整个软件安装的汇总信息, 然后提示用户重新引导系统。在重新启动系统之后, 将会出现一个Solaris系统的注册界面, 如图1-48所示。



图1-46 “准备安装”界面

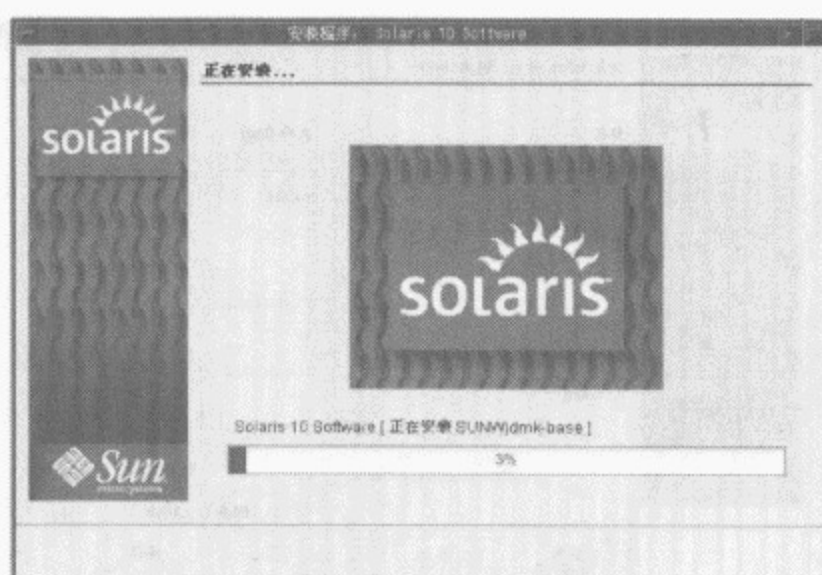


图1-47 软件正式开始安装界面

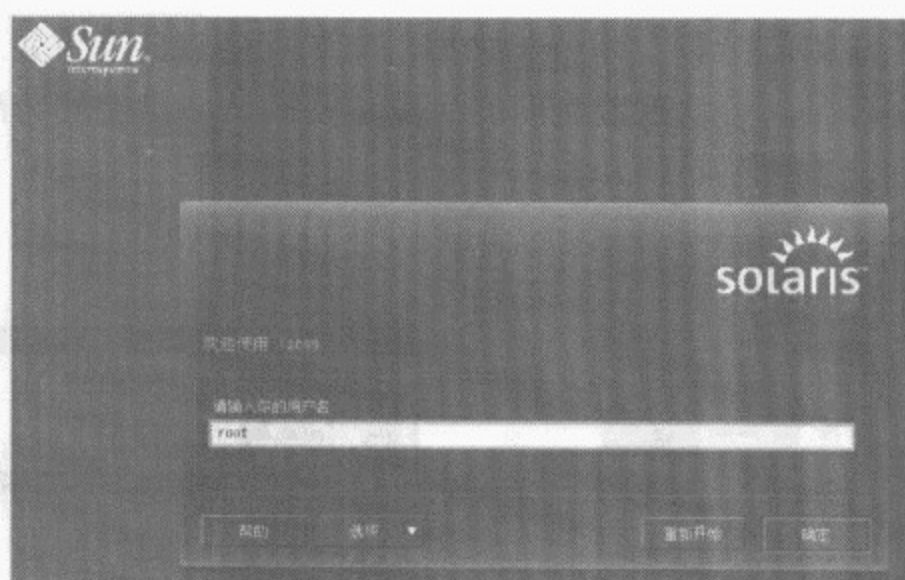


图1-48 Solaris系统注册界面

至此，可以使用root以及先前设置的密码以超级用户的身份注册，访问Solaris UNIX系统。



第2章 命令行基础知识

本章从最基本的命令行界面入手，介绍Shell的命令行结构、标准输入与标准输出、输入输出重定向、管道、命令历史与命令别名，以及作业控制等。

UNIX系统提供大量的命令和工具，如能熟练地掌握最基本的命令，灵活地运用系统提供的各种机制，组合运用UNIX系统的命令和工具，就能够充分地发挥UNIX系统的潜能。UNIX系统的强大功能完全体现在其命令行环境中，图形界面（如CDE桌面环境）提供的所有功能实际上也是利用基本命令和工具实现的。因此，熟练地掌握、灵活自如地运用一定数量的常用命令和工具是每个学习UNIX系统的人都应当具有的基本功。

在Solaris系统中，为了进入命令行环境，可在注册到Java桌面系统之后，点击左下角的启动图标，选择“启动→应用程序→实用程序→终端”，打开一个终端仿真窗口；也可以在注册到CDE桌面环境之后，点击屏幕底部主面板“cpu disk”上方的上箭头，从弹出的菜单中选择“控制台”或“本主机”，进入终端仿真窗口，利用命令行界面访问UNIX系统。此外，在终端窗口的命令行操作过程中，可以随时使用鼠标选中屏幕上的文本数据，利用右键的上下文菜单，复制、剪切和粘贴选中的文本数据。

在学习本章之前，最好先为自己创建一个普通用户帐号，然后以普通用户的身份注册到UNIX系统中，在自己的主目录中学习UNIX系统的各种命令和工具，一开始尽量不要动用系统文件，等到有了一定的基础时，再以超级用户root的身份访问UNIX系统，以免在无意中破坏系统。

2.1 命令行结构

在UNIX系统中，一个命令通常由命令名、命令选项和命令参数等三部分内容组成，中间以空格或制表符等空白字符隔开。命令形式如下：

<命令名> <命令选项> <命令参数>

其中，命令选项通常是以减号“-”开始的单个字符。命令选项主要用于限定命令的具体功能，同时也决定了命令的最终运行结果。在UNIX系统中，每个命令通常均提供大量的选项，因而具有丰富的功能。选项可以单独给出，也可以组合使用。如果选项后面有参数，选项与参数必须单独列出。

顾名思义，命令选项是可以省略的。同样，命令参数也可以省略。也就是说，在命令行结构中，只有命令名通常是必须提供的。一个最简单的命令可以仅仅包含命令名本身。在这种情况下，命令选项和参数均采用默认值。下面就是一个最简单的命令，其中的命令选项和参数均采用默认值，即列出系统的当前日期和时间：

```
$ date
2009年06月16日 星期二 16时15分40秒 CST
$
```

在实际应用过程中，可以根据具体要求，视情况选用或省略命令选项和命令参数。而且，

命令选项和命令参数可与命令名以任意形式组合使用。例如，下列命令仅由命令名和一个命令选项“-n”组成，省略了命令参数，其作用是列出系统的名字：

```
$ uname -n
iscas
$
```

下列命令由命令名和命令参数组成，省略了命令选项，其作用是以简单的输出形式列出指定目录下的文件：

```
$ ls /etc/skel
local.cshrc  local.login  local.profile
$
```

取决于命令本身，命令参数可以是目录、文件或其他内容。例如，在下列命令中，ls是命令的名字，“-l”是命令的选项，/etc/profile文件是命令的参数：

```
$ ls -l /etc/profile
-rw-r--r--  1 root      sys          712 Aug 25  2008 /etc/profile
$
```

下列命令使用了“-l”和“-a”两个命令选项，但省略了命令参数。其作用是列出当前目录中的所有文件，包括隐藏文件：

```
$ ls -la
总数 12968
drwxr-xr-x  2 gqxing   root          512  6月 16日 16:19 .
drwxr-xr-x  7 root      root          512  6月 15日 15:55 ..
-rw-r--r--  1 gqxing   other         169  6月 15日 15:49 .profile
-rw-----  1 gqxing   other        2322  6月 16日 16:19 .sh_history
.....
$
```

如前所述，如果选项本身也带有参数，这样的选项及参数必须单独列出。在下列排序命令中，因为“-k”和“-o”等命令选项本身也要求提供参数，故需分别给出：

```
$ sort -k 5 -n -o sorted tobesorted
$
```

其中，“-k 5”中的5就是“-k”选项的参数，表示以第5个字段为关键字进行排序。“-n”选项表示按数值的大小排序。“-o sorted”中的sorted是选项“-o”的参数，表示存储最终排序结果的输出文件。最后的tobesorted是命令参数，表示需要排序的输入文件。

在后续章节介绍的UNIX命令语法格式中，凡是以方括号“[]”形式给出的命令选项均为可选项，因而可根据具体需求选择使用或忽略。

在UNIX系统的命令提示符下，一次通常仅输入一个命令。如果愿意，也可以一次输入多个命令，命令之间用分号隔开。例如，下列两个命令的作用是首先进入指定的目录/etc/skel，然后列出其中的文件。

```
$ cd /etc/skel; ls -l
总数 6
-rw-r--r--  1 root      sys          136 2008    8月 25 local.cshrc
-rw-r--r--  1 root      sys          157 2008    8月 25 local.login
```

```
-rw-r--r--  1 root    sys          174 2008   8月 25 local.profile
$
```

另外，也可以使用圆括号把若干命令合并在一起，使之构成一个组合命令。例如：

```
$ (cd /etc/skel; ls -l)
总数 6
-rw-r--r--  1 root    sys          136 2008   8月 25 local.cshrc
-rw-r--r--  1 root    sys          157 2008   8月 25 local.login
-rw-r--r--  1 root    sys          174 2008   8月 25 local.profile
$
```

除了括号之外，上述的两个组合命令完全一样，有时其效果也完全一样。但两者的意义却大不相同。第一种命令形式只是在一个逻辑行上并列输入了多个命令，其效果同一次输入一个命令基本上没有区别，而且都是在当前Shell中运行。而第二种命令形式则把多个命令看做一个组合命令，在一个子Shell中运行，所有命令的输出数据将会合并为一个输出流，其差别在管道操作中尤为明显。

例如，下列两组命令的最终结果是完全不同的（其中，“wc -l”命令用于计算读入的行数）。首先让我们分别观察date与who两个命令的输出。

```
$ date
2009年06月16日 星期二 16时19分58秒 CST
$ who
root      console      6月 16日 16:10 (:0)
root      pts/3           6月 16日 16:11 (:0.0)
root      pts/4           6月 16日 16:11 (:0.0)
$
```

然后观察两个并列命令的输出结果：

```
$ date; who
2009年06月16日 星期二 16时19分58秒 CST
root      console      6月 16日 16:10 (:0)
root      pts/3           6月 16日 16:11 (:0.0)
root      pts/4           6月 16日 16:11 (:0.0)
$
```

接着再用管道把两个并列命令与计算输入数据行数的wc命令连接起来，观察其输出结果。可以看到，wc命令仅仅计数了who命令的输出结果——3行。

```
$ date; who | wc -l
2009年06月16日 星期二 16时19分58秒 CST
3
$
```

最后，再利用圆括号把两个命令组合到一起，通过管道连接wc命令，观察其结果。可以看到，两个命令各自的输出数据已合并到一起，wc命令计数的最终结果是4行。

```
$ (date; who) | wc -l
4
$
```

如果命令较长，超出一个物理行的宽度，可以使用反斜线“\”把命令写到多个物理行上。

也可以继续输入，由系统自动延伸至后续行上。例如，下列`read`命令提示用户依次输入名字、电话号码和电子邮件地址，然后分别存于`name`、`phone`和`email`三个变量中。由于命令行较长，其中采用了反斜线“\”，把超长部分延续到下一行：

```
$ read -p "Please input name, phone and email address in order: " \
name phone email
$
```

2.2 后台进程

在UNIX系统中，Shell通常以前台形式解释执行用户输入的命令。在Shell的命令提示符（超级用户默认的命令提示符为“#”，普通用户默认的命令提示符为“\$”）下，系统将会等待用户输入命令，直至按下Enter键。然后由Shell解释命令行，创建一个新的进程，执行用户提交的命令，最后给出命令的执行结果。

在Shell解释执行命令期间，用户需要等待命令执行的完成，中间不能做任何事情。即使命令的执行时间过长，中间不需要输入任何信息，不需要监控命令的执行过程时，也只能静待命令执行的完成。

为了解决此问题，Shell提供了后台进程机制，允许用户以后台进程的方式运行命令，且无需等待命令执行的完成。在解释命令行，以后台进程方式执行命令的同时，Shell将会立即输出命令提示符，允许用户输入新的命令，从而并发地运行多个命令。

如果想要采用后台进程方式运行命令，只需在命令的后面增加一个“&”符号即可。例如，为了在系统中检索究竟存在多少个名为`core`的文件，可以使用下列形式的命令：

```
$ find / -name core -print 2>/dev/null &
[1] 919
$
```

上述命令行后面的“&”符号告诉Shell，提交的命令应以后台进程的方式执行，无须等待命令执行的完成。因此，在给出作业号和进程ID之后，系统将会立即输出命令提示符，提示用户进一步输入其他命令。

在上述例子中，方括号中的数字“1”是以后台作业方式运行的`find`命令的作业号，“919”是`find`命令的进程ID（PID）。为了跟踪与控制后台作业，可以利用作业控制命令控制作业的运行状态。例如，可以使用作业控制命令`fg`把后台作业转为前台进程继续运行。有关后台作业控制的详细介绍，参见本章“2.10 作业控制”一节。利用进程ID（或根据`ps`命令获取的其他进程信息），也可以采用进程控制命令（如`kill`等命令）控制进程的运行行为。有关进程控制的介绍，详见第8章“进程管理”。

如果后台进程有输出数据，其输出信息将会随时出现在用户的终端屏幕上。注意，后台进程的输出信息有可能会出现在交互会话期间的任何时刻，从而造成屏幕输出的混乱。例如，如果用户正在使用`vi`编辑文件，后台进程的输出很可能会干扰编辑器的正常工作。

2.3 标准输入、标准输出与标准错误输出

在UNIX系统中，任何命令，包括Shell本身，通常总是读取来自终端键盘输入的数据，这

种数据输入源称做标准输入（**stdin**），其文件描述符为0。命令的运行结果通常总是输出到用户终端的屏幕上，这一输出目的称做标准输出（**stdout**），其文件描述符为1。另外，在命令运行期间，如果出现问题，相应的错误信息也将输出到用户终端的屏幕上，这种输出目的通常称做标准错误输出（**stderr**），其文件描述符为2。

一旦注册到系统中，系统总是为用户打开三个默认的文件：标准输入（键盘）、标准输出（终端屏幕）和标准错误输出（也是终端屏幕，用于输出错误信息），如图2-1所示。

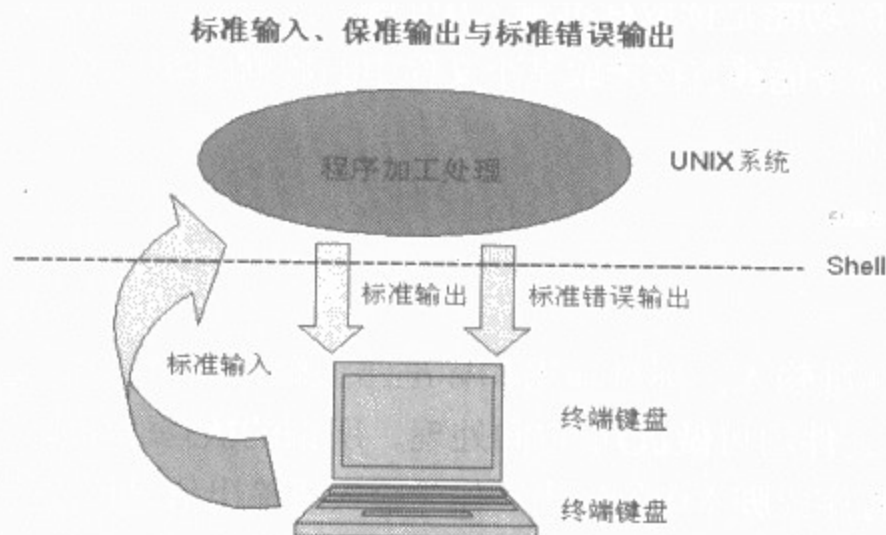


图2-1 标准输入、标准输出和标准错误输出与终端键盘和屏幕间的关系

从键盘上输入任何命令时，其输出数据，包括系统回送的输入信息与错误信息将会出现在用户的终端屏幕上。例如，当输入下列不带文件参数的**cat**命令时，**cat**命令将会读取来自标准输入的数据，并逐字予以显示。也就是说，**cat**命令将会等待用户使用键盘输入数据。当按下**Ctrl-D**键，表示输入结束时，**cat**命令将会把读取的数据再次显示在终端屏幕上。

```
$ cat -v
input from keyboard
Ctrl-D
input from keyboard
$
```

2.4 输入输出重定向

为了仔细分析命令的处理结果，有时需要把命令的标准输出保存到某个文件中。这就需要用到**Shell**的输入输出重定向机制。例如，利用输出重定向符号“>”，下列命令行可以把“ls -l”命令的输出结果保存到指定的文件中：

```
$ ls -l > fname
$
```

在上述命令中，“> fname”意味着把“ls -l”命令的输出数据重定向并写到指定的文件**fname**中。如果指定的文件不存在，**Shell**将会创建一个新的文件，然后把输出数据保存到其中。如果文件已经存在，文件中原有的内容将会被清除，并代之以命令的输出数据。

为了保留文件中原有的数据，把命令的输出数据附加到文件的后面，可以使用重定向符号“>>”。

```
$ ls -l >> fname
$
```

其中，“>> fname”意味着把“ls -l”命令的输出数据重定向并附加到指定文件fname的后面。同样，如果指定的文件不存在，Shell将会创建一个新的文件，并把输出数据保存到其中。如果文件已经存在，Shell将会把命令的输出数据附加到文件的后面，以保留文件中原有的内容。

任何命令（包括Shell本身）的标准输入也可以重定向，使命令直接读取某个文件而不是键盘输入。例如，wc命令的功能是读取标准输入中的输入数据，分别计数输入数据中的字符数、字数和行数。为了使wc命令能够直接读取某个文件中的数据内容，可以使用重定向符号“<”，使之直接读取指定的文件。

```
$ wc -l < fname
42
$
```

在UNIX系统中，标准输入、标准输出和标准错误输出三个文件通常总是打开的。这三个文件，包括其他打开的文件均可做I/O重定向处理。所谓的I/O重定向，只不过是Shell读取或捕捉来自文件、命令、程序或脚本中的输出，作为输入或输出信息传递给另外一个文件、命令、程序或脚本。

在UNIX系统中，系统将会为每个打开的文件分配一个文件描述符，每个文件都与一个文件描述符相关联。文件描述符是一个数字，便于UNIX系统跟踪打开的文件（可以把文件描述符看做简化的文件指针）。标准输入、标准输出和标准错误输出的文件描述符分别是0、1和2。

作为一种临时性的双向机制，把其他打开文件的描述符分配到标准输入、标准输出和标准错误输出，然后利用这些文件描述符执行输入输出，有时是非常有用的（注意，使用文件描述符5可能会引起问题，这是因为当Shell利用exec命令创建子进程时，子进程将会继承文件描述符5。因此，最好避开这个特殊的文件描述符）。

如表2-1所示为Shell支持的各种I/O重定向的规定及其说明。

表2-1 Shell I/O重定向

I/O重定向	简单说明
<fname	使用指定的文件作为标准输入（其文件描述符为0），以便从指定的文件中接收输入数据
>fname	使用指定的文件作为标准输出（其文件描述符为1）。如果文件不存在，则创建命名的文件。如果文件存在，且noclobber标志已经设置，将会产生错误；否则，将会清除文件中原有的数据内容。参见第6章“Shell基础知识”中的set命令介绍
>!fname	除了忽略noclobber标志之外，其功能与“>fname”相同
>>fname	使用指定的文件作为标准输出。如果文件存在，则把输出内容附加到文件后面；否则，创建指定的文件
<>fname	以读写方式打开指定的文件，并使之作为标准输入
<<[-]fstr	以指定的标志字符串fstr之后的文本（称做Here文档）作为标准输入，从fstr之后逐行读取数据，直至遇到第二个fstr（或EOF）标志。此时，第一个fstr是标准输入的起始标志，第二个fstr（或EOF）是标准输入的结束标志。如果“<<”后面附带减号“-”标志字符，则Shell将会忽略后随文本行前面的制表符
<&digit	使用指定的文件描述符复制一个标准输入

(续表)

I/O重定向	简单说明
>&digit	使用指定的文件描述符复制一个标准输出
<&-	关闭标准输入。而“n<&-”表示关闭输入文件描述符n
>&-	关闭标准输出。而“n>&-”表示关闭输出文件描述符n
<&j	把标准输入重定向到文件描述符j表示的输入文件中
>&j	把标准输出重定向到文件描述符j表示的输出文件中

如果I/O重定向符号“<”或“>”前面有一个数字，则表示相应的文件描述符（默认值分别为0或1）对应的文件，如表2-2所示。

表2-2 Shell I/O重定向

I/O重定向	简单说明
0<fname	把标准输入重定向到指定的文件中
1>fname	把标准输出重定向到指定的文件中
1>>fname	把标准输出重定向并附加到指定的文件中
2>fname	把标准错误输出重定向到指定的文件中
2>>fname	把标准错误输出重定向并附加到指定的文件中
i>&j	把文件描述符i表示的输出文件重定向到文件描述符j表示的文件
[j]<>fname	以读写方式打开指定的文件，并把文件描述符j分配到指定的文件。如果文件不存在，则创建该文件。如果未指定文件描述符j，则表示默认的文件描述符0，即标准输入

采用下列命令形式，可以把标准错误输出重定向到标准输出，使命令的错误信息和命令的实际输出数据均写到标准输出中，即在终端屏幕上显示。

```
command 2>&1
```

在下面的例子中，前三个echo命令的标准输出已重定向到script.log文件，因而其输出信息将会写到文件中，而不是出现在终端屏幕上。

```
$ LOGFILE=script.log
$ echo "This line is written to log file." > $LOGFILE
$ echo "This line is appended to log file." >> $LOGFILE
$ echo "This line is appended to log file also." >> $LOGFILE
$ echo "This line is echoed to stdout."
This line is echoed to stdout.
$ cat script.log
This line is written to log file.
This line is appended to log file.
This line is appended to log file also.
$
```

在下面的例子中，前两个命令（变量赋值语句除外）的标准错误输出已重定向到script.errors文件，因此，命令的错误信息将会记录到指定的文件而不是写到终端屏幕上。第三个命令的错误信息将会写到标准错误输出，即终端屏幕上，而不会出现在script.errors文件中。


```
$ ERRFILE=script.errors
$ bad_command1 2>$ERRFILE
$ bad_command2 2>>$ERRFILE
$ bad_command3
```

采用下列形式，可以把多个I/O重定向组合到一个命令行中。

```
command <input-file>output-file
```

采用下列形式，可以把标准输出和标准错误输出重定向到同一个文件中。

```
command >command.log 2>&1
```

其中，命令的任何错误信息将会写到文件command.log文件中，因为标准错误输出已经重定向到该文件。

注意：I/O重定向的顺序是非常重要的。Shell将会根据文件描述符（文件）出现的顺序决定I/O重定向的关联关系。例如，下列I/O重定向命令意味着把命令的标准输出和标准错误输出均重定向到给定的文件中。

```
command 1>fname 2>&1
```

假定I/O重定向之前命令的标准输入、标准输出和标准错误输出对应的文件描述符如图2-2所示。

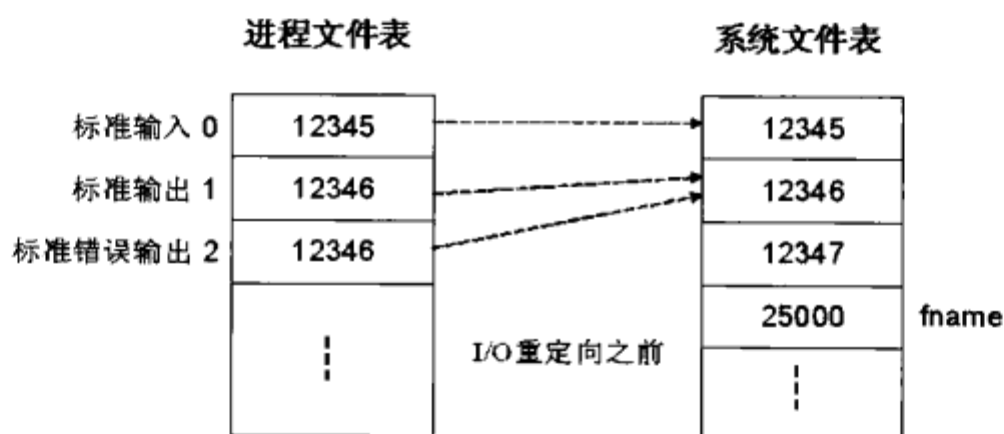


图2-2 I/O重定向之前

Shell将首先建立文件描述符1（即终端屏幕）与文件fname之间的重定向关系，即把fname对应的文件描述符25000写到进程文件表中的标准输出位置。然后建立文件描述符2与文件描述符1表示的文件（即fname）之间的重定向关系，即把标准输出对应的文件描述符25000写到进程文件表中的标准错误输出位置。最终的结果是，命令的标准输出和标准错误输出内容均写到文件fname中，如图2-3所示。

如下面的命令所示，如果I/O重定向的顺序相反，其最终结果将会是另外一个样子。

```
command 2>&1 1>fname
```

假定I/O重定向之前命令的标准输入、标准输出和标准错误输出对应的文件描述符仍如图2-2所示。文件描述符2首先与文件描述符1建立重定向关系。由于两者对应的文件描述符相同（均为12346），故标准错误输出对应的文件描述符保持不变。然后，文件描述符1再与给定的文件fname建立重定向关系，把fname对应的文件描述符25000写到进程文件表中的标准输出位置。

最终结果是，命令的标准错误输出将会写到标准输出，即在终端屏幕上显示。而命令的标准输出则会写到给定的文件**fname**中，如图2-4所示。

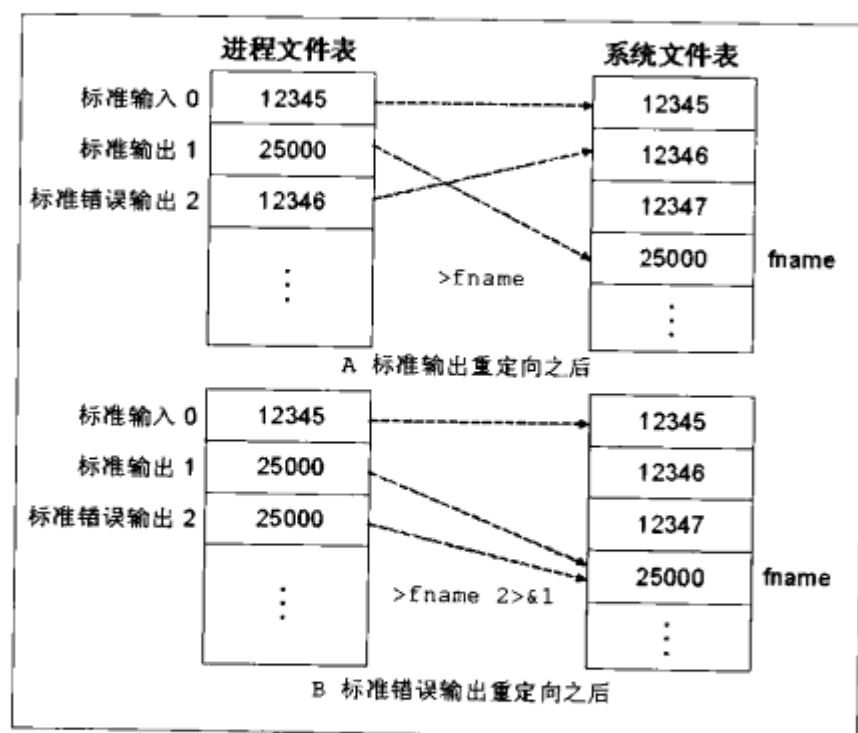


图2-3 I/O重定向之后 (1)

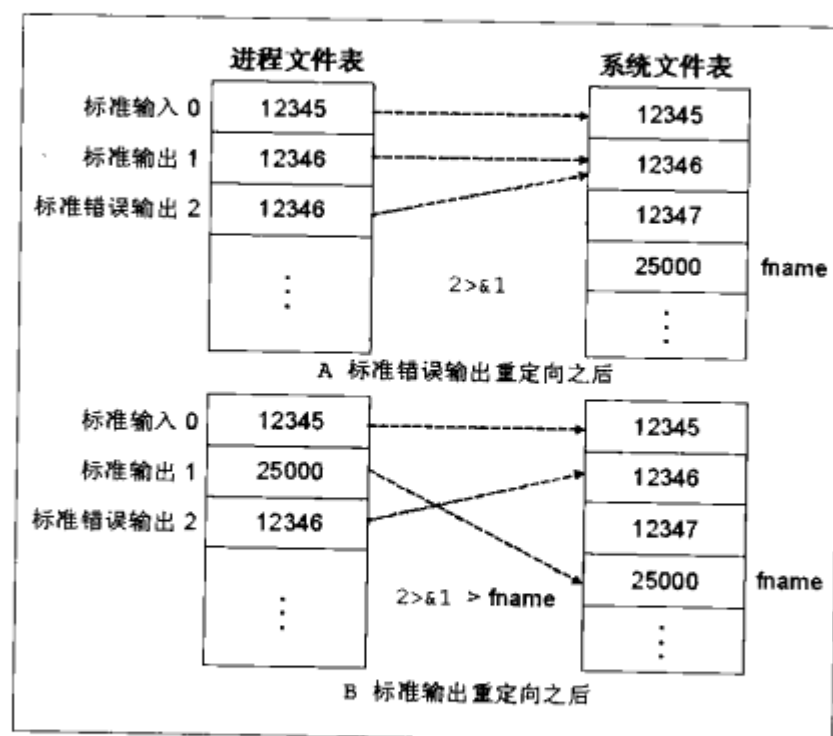


图2-4 I/O重定向之后 (2)

由此可见，如果I/O重定向的顺序不同，其最终的结果将会完全不一样。例如，执行下列命令时，错误信息将会输出到标准输出，即终端屏幕上，而不会写到文件**command.log**文件中。

```
$ ls -yz 2>&1 >> command.log
ls: illegal option -- y
ls: illegal option -- z
usage: ls -lRaAdCxmnlhogrtuvVcpFbqisfHLeE@ [files]
$
```

利用下列命令和I/O重定向的方法，可以创建一个新的空文件。如果文件存在，则清空文件**fname**中原有的内容。

```
$ > fname
$
```

利用I/O重定向的方法，还可以创建一个新文件，使之包含当前目录下的目录文件树列表。例如：

```
$ ls -lR > dir-tree
$
```

注意：子进程能够继承已经打开的文件描述符。为了防止文件被子进程继承，应注意随时关闭不再需要的文件描述符。

2.5 管道

在UNIX系统中，管道是一种先进先出的单向数据通路。利用管道符号“|”，可以把一个命令的标准输出连接到另一个命令的标准输入。例如，利用管道把ls和wc两个命令连接到一起，可以获知指定目录下的文件数量（“-w”选项表示以字为单位进行计数）。

```
$ ls /usr | wc -w
53
$
```

从上述命令的最终执行效果看，可以把组合命令分解为以下两个命令：

```
$ ls /usr > fname; wc -w < fname
53
$
```

由此可以看出，当使用管道方式连接两个命令时，Shell将会把两个进程连接起来，利用管道的单向通信特征，把一个进程的标准输出传递到另一个进程的标准输入。Shell将会协调两个进程的同步，使两个进程能够并发地运行，这样就可以省略存储中间处理结果的临时文件。实际上，管道是一种特殊的I/O重定向。

管道的常见用法是为滤通程序提供原始数据，由滤通程序读取来自标准输入的数据，按照指定的检索原则和模式，从输入数据中提取期望的、包含给定字符串的数据。在UNIX系统中，grep就是这样一个常见的滤通程序。

例如，为了从ps命令输出的众多进程中找出某个特定的进程，可以使用管道连接ps和grep命令。

```
$ ps -ef | grep cron
  root          213      1 0 16:08:35 ?                0:00 /usr/sbin/cron
  gqxing        914    878 0 16:23:32 pts/4            0:00 grep cron
$
```

另外一个常见的用法是利用管道把进程的输出数据传递给sort命令，使之按照一定的排序原则进行排序，最终输出排序后的结果。例如，下列组合命令最终将会按照字符顺序输出注册的用户。

```
$ who | sort
cathy          pts/6          Jul 20 16:01          (localhost)
gqxing         pts/5          Jul 20 15:59          (localhost)
root           console        Jul 20 15:54          (:0)
root           pts/4          Jul 20 15:54          (:0.0)
root           pts/7          Jul 20 16:00          (:0.0)
root           pts/8          Jul 20 16:01          (:0.0)
$
```

利用管道，可以把多个命令组合到一起，把命令的标准输出依次传递到下一个命令的标准输入，最终得到经过多个命令依次处理的结果。

```
command1 | command2 | command3 > output-file
```

为了依次加工处理多个命令、脚本和程序的输出数据，管道是非常有用的。例如，为了获取cron的进程ID，以便使用kill命令终止该进程，可以利用管道连接下列命令，首先从进程列表中提取与cron有关的进程，然后删除可能存在的“grep cron”进程，最后再使用awk命令截取位于第2个字段的进程ID（参见第8章“进程管理”介绍的简化版的pgrep命令）。

```
$ ps -ef | grep cron | grep -v grep | awk '{print $2}'
213
$
```


若想复制和备份一个完整的目录，也可以利用管道，组合使用`find`和`cpio`两个命令，把当前目录下的所有目录和文件按照原有的目录层次结构复制到一个新的目录位置。

```
$ cd sourcedir
$ find . -print | cpio -pcdmu newdir
$
```

UNIX系统还提供一个相当于三通管的实用程序`tee`。`tee`命令的主要功能是通过标准输入接收并显示数据，同时把数据存储在指定的文件中。因此，可以利用`tee`命令，在显示一个命令输出数据的同时，把输出结果存储到一个指定的文件中，以便将来再检查。例如，为了显示并保存当前所有注册用户的列表文件，可以使用下列命令（其效果如图2-5所示）。

```
$ who | tee userlist
root      console      Jun 16 16:10      (:0)
root      pts/3              Jun 16 16:11      (:0.0)
root      pts/4              Jun 16 16:11      (:0.0)
$ ls -l userlist
-rw-r--r--  1 gqxing  other          130 Jun 16 16:26 userlist
$
```

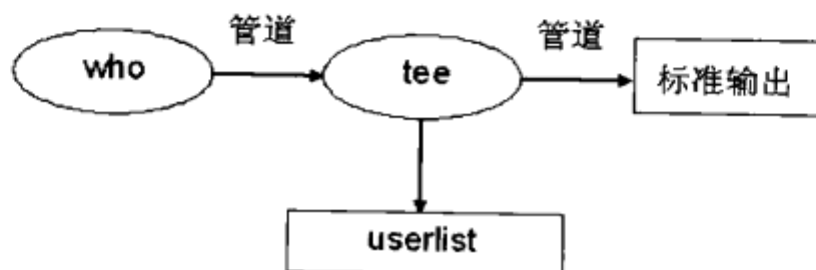


图2-5 `tee`命令的“T”型三通功能

2.6 元字符与文件名生成

在UNIX系统中，很多命令均使用文件作为命令参数。例如，下面的`ls`命令用于列出指定文件参数`atmmon.c`的访问权限、文件大小及文件属主等有关属性。

```
$ cd /export/home/gqxing/src
$ ls -l atmmon.c
-rw-r--r--  1 gqxing  other          26282 Jun 18 15:31 atmmon.c
$
```

当需要处理一组具有共同属性的文件时，怎样指定文件名参数呢？Shell提供一种文件名生成机制，使用户能够利用元字符（或称通配符）实行模式匹配，最终生成一个具有同一属性的文件列表。

为了简化手工输入，按照某种模式选择具有同一属性的文件，Shell的文件名生成机制是非常有用的。例如，可以使用下列命令列出当前目录中的所有C程序文件。

```
$ ls -l *.c
-rw-r--r--  1 gqxing  other          25852 Jun 18 15:31 atmcom.c
-rw-r--r--  1 gqxing  other          26282 Jun 18 15:31 atmmon.c
-rw-r--r--  1 gqxing  other          52125 Jun 18 15:33 handler.c
-rw-r--r--  1 gqxing  other          30625 Jun 18 15:32 listener.c
$
```

Shell命令使用生成的文件名作为参数，依次处理每一个文件。上述命令的处理结果就是依次列出当前目录下每一个以“.c”为文件名后缀的C程序文件。其中的星号“*”就是一个元字符，可以匹配任何字符或字符串，包括空字符串。表2-3给出了Shell支持的与文件名生成有关的元字符及其说明。注意，元字符可以组合使用。

表2-3 与文件名生成有关的元字符

元字符	简单说明
*	可以匹配任何数量的字符或字符串，包括空字符串。例如，“ boc* ”表示任何一个以“ boc ”为起始字符的字符串，“ *.c ”表示任何一个以“.c”为文件名后缀的C程序文件
?	可以匹配相应位置的任何一个字符。例如，“ file? ”表示任何一个以“ file ”为起始字符串，后面附加单个字符的字符串
[...]	由方括号定义的字符集或字符范围，可以使用其中任何一个字符匹配文件名相应位置的一个字符。方括号中的字符集可以由任何字符组成，数量不限。字符可以一一列举，也可以在两个字符之间加一个减号“-”，表示一个字符范围。例如， [a-z] 表示所有的小写字母， [0-9] 表示任何数字
[!...]或[^...]	如果方括号中的第一个字符是感叹号“!”或上箭头“^”（后者仅适用于Bash），其意义恰好相反，表示可以匹配任何一个不属于给定字符集范围的字符

假定当前目录存在下列文件，现在通过例子，进一步说明怎样使用元字符匹配文件名解释Shell的文件名生成机制。

```
$ ls -l
total 12
-rwxr-xr-x 1 gqxing other 524 Jun 22 12:40 Bubble
-rwxr-xr-x 1 gqxing other 188 Jun 22 12:42 bigfile
-rwxr-xr-x 1 gqxing other 941 Jun 22 12:41 ftpmget
-rwxr-xr-x 1 gqxing other 333 Jun 22 12:41 ftpmput
-rwxr-xr-x 1 gqxing other 604 Jun 22 12:43 prime
-rwxr-xr-x 1 gqxing other 640 Jun 22 12:43 whatday
$
```

按照表2-3的说明，**[a-z]**可以匹配任何小写字母，“*”可以匹配任何字符串，故下面的命令可以列出当前目录下任何以小写字母为起始字符的文件名。

```
$ ls -l [a-z]*
-rwxr-xr-x 1 gqxing other 188 Jun 22 12:42 bigfile
-rwxr-xr-x 1 gqxing other 941 Jun 22 12:41 ftpmget
-rwxr-xr-x 1 gqxing other 333 Jun 22 12:41 ftpmput
-rwxr-xr-x 1 gqxing other 604 Jun 22 12:43 prime
-rwxr-xr-x 1 gqxing other 640 Jun 22 12:43 whatday
$
```

由于问号“?”可以匹配任何一个字符，故下列ls命令可以列出当前目录中文件名前四个字符为ftpm，后三个字符为任何字符的所有文件。

```
$ ls -l ftpm???
-rwxr-xr-x 1 gqxing other 941 Jun 22 12:41 ftpmget
-rwxr-xr-x 1 gqxing other 333 Jun 22 12:41 ftpmput
$
```

为了列出当前目录中以p或w为首字符的所有文件，可以使用下列ls命令：

```
$ ls -l [pw]*
-rwxr-xr-x  1 gqxing  other          604 Jun 22 12:43 prime
-rwxr-xr-x  1 gqxing  other          640 Jun 22 12:43 whatday
$
```

上述命令也可以改写如下：

```
$ ls -l p* w*
-rwxr-xr-x  1 gqxing  other          604 Jun 22 12:43 prime
-rwxr-xr-x  1 gqxing  other          640 Jun 22 12:43 whatday
$
```

为了列出当前目录中首字符为大写字母（或其他非小写字母）的所有文件，可以使用下列三种ls命令形式之一（注意，最后一个命令仅适用于Bash）：

```
$ ls -l [A-Z]*
-rwxr-xr-x  1 gqxing  other          524 Jun 22 12:40 Bubble
$ ls -l [!a-z]*
-rwxr-xr-x  1 gqxing  other          524 Jun 22 12:40 Bubble
$ ls -l [^a-z]*
-rwxr-xr-x  1 gqxing  other          524 Jun 22 12:40 Bubble
$
```

如上所述，问号“?”可以匹配当前目录中以单个字符命名的所有文件名。星号“*”能够匹配当前目录中的所有文件名（隐藏文件除外）。如果不存在能够匹配检索模式的文件名，指定的检索模式将会不加修改地作为参数直接传递给相应的命令。例如（仅当当前目录为空时，下面第二个例子才会出现如此输出结果）：

```
$ ls -l ?
?: No such file or directory
$ ls -l *
*: No such file or directory
$
```

元字符也可用于检索文件。例如，可以使用下列命令检索并列出/export/home/gqxing目录下任何子目录中名为core的文件。

```
$ echo /export/home/gqxing/*/core
```

注意：任何元字符都不能匹配以句点“.”为首字符的隐藏文件名。换言之，以句点“.”为起始字符的隐藏文件名必须采用明显的匹配形式。例如，下列命令只能列出当前目录中的所有非隐藏文件：

```
$ echo *
bin conf doc incl script src
$
```

为了匹配以句点“.”为起始字符的隐藏文件名，可以采用下列命令形式，列出所有的隐藏文件：

```
$ echo .*
. . . .profile .sh_history
$
```


注意：set命令的“-f”选项（即“set -f”命令）能够禁止文件名的生成。当Shell无法解释元字符时，应注意检查是否设置了这个标志。

2.7 转义与引用

转义和引用是两个截然相反的概念。在Shell中，为了处理具有特殊意义的元字符，如“<”、“>”、“*”、“?”、“|”和“&”等，使之作为普通字符，可以采用转义符号“\”、单引号和双引号引用元字符，而引用的元字符则失去其特殊意义。

根据上述说明，本身具有特殊意义的元字符，如果在前面加上转义符号“\”，则失去其特殊意义。而对于某些普通字符，如果前面加上转义符号“\”，则具有特殊的意义，这些字符称做转义字符。表2-4给出了Shell支持的，具有特殊意义的部分转义字符。

表2-4 Shell支持的部分转义字符

转义字符	简单说明
\a	生成声音提示
\b	退格符
\f	换页符
\n	换行符
\r	回车符
\t	制表符
\v	竖向制表符
\\	反斜线
\ON	采用1、2或3位八进制数值表示的等价ASCII字符

表2-4中的转义字符可用于echo等命令，以便控制数据的输出或显示格式等。下面的例子说明了怎样在一个echo命令中使用转义字符显示一个选择菜单。

```
$ echo "\n\t\t === Command Menu ===  
\t1. Data processing  
\t2. Print report  
\t3. end"  
  
    === Command Menu ===  
1. Data processing  
2. Print report  
3. end  
$
```

根据上一节的介绍可知，Shell中的元字符都具有特殊的含义，无法直接使用。而为了引用元字符本身，可以采用三种形式。一种是在元字符前面加转义符号“\”，用以表示字符文字本身，而非具有特殊意义的元字符。

例如，下列两个echo命令的最终结果大不相同。

```
$ echo *  
Bubble bigfile ftpmget ftpmput prime whatday
```

```
$ echo \*
*
$
```

转义符号“\”是一种引用单个（特殊）字符的最佳方法。一个（特殊）字符前如果增加了转义符号，等于告诉Shell，随后的字符应按普通文字解释，例如，“*”、“\\$”、“\\”、“\”以及“\”等分别表示“*”、“\$”、“\”、“”以及“”等字符本身，参见下面的例子：

```
$ echo "Hello"
Hello
$ echo "\"Hello\"", he said."
"Hello", he said.
$ echo "\$var"                                     # “\$”后面的变量不作解释
$var
$ echo "\\ "
\
$
```

由此可见，为了引用单个字符，使用转义符号“\”是最方便的方法。但当需要引用多个字符时，“\”就显得非常笨拙了。因此，Shell提供第二种引用方式，即采用单引号的方式引用元字符。在此情况下，单引号之间的所有字符（包括元字符）均按普通文字本身解释，例如，为了引用一个字符串“**”，我们可以在“**”前后增加一对单引号。

```
$ echo xx'**'yy
xx**yy
$
```

这意味着，单引号中的所有元字符（反向单引号“`”和上述的转义字符除外）均作为文字常量处理。因此，利用单引号可以同时引用多个元字符或字符串，如下所示：

```
$ echo '*.c $var "testing"'
*.c $var "testing"
$
```

从上述例子中还可以看出，单引号“'”不允许使用美元符号“\$”引用变量的值，其中的\$var仅表示其本身。

在实际应用中，使用单引号引用参数或变量，其作用是防止Shell提前解释或扩展字符串中的特殊字符。例如，下面的命令表示列出所有以大写字母B或小写字母b为起始字符的文件。

```
$ ls -l [Bb]*
-rwxr-xr-x  1 gqxing  other          524 Jun 22 12:40 Bubble
-rwxr-xr-x  1 gqxing  other          188 Jun 22 12:42 bigfile
$
```

如果在“[Bb]*”前后增加单引号，可以理解为引用元字符本身，也就是把“[Bb]*”作为一个字符串解释。在下面的例子中，单引号中的“[Bb]*”就是作为一个文件名解释的，因而找不到匹配的文件。

```
$ ls -l '[Bb]*'
[Bb]*: No such file or directory
$
```

在UNIX系统中，某些命令或实用程序会重新解释或扩展使用引号传递的字符串中的特殊字符。使用引号引用参数或变量的一个重要用途就是确保能够把其中含有的特殊字符原封不动地传递给被调用的实用程序，由实用程序进行解释或扩展，例如：

```
$ grep '[Ff]irst' [Bb]*
Bubble:# First line should be #!/bin/sh
btree:# The first line is just a comment
$
```

假定存在下列文件，怎样使用元字符匹配并删除“test file”呢？

```
$ ls -l test*
-rw-r--r--  1 gqxing  other      1080 Jun 22 12:55 test file
-rw-r--r--  1 gqxing  other      1010 Jun 22 12:55 test.file
$
```

如果使用问号“?”或星号“*”，将会误删test.file文件。此时可以使用转义符号加空格的形式匹配“test file”，表示test与file之间的空格并非字段分隔符。

```
$ rm test\ file
$ ls -l test*
-rw-r--r--  1 gqxing  other      1010 Jun 22 12:55 test.file
$
```

实际上，也可以使用“rm test" "file”或“rm test' 'file”（使用双引号或单引号括住空格字符）的命令形式删除名字中间包含空格的文件。

第三种方式是利用双引号引用字符串，防止部分（但并非全部）元字符提前解释。在此情况下，除了“!”、“\$”、“”、“\”和“{”字符之外，其他元字符均按文字本身处理，如：

```
$ echo "The current working directory of **$LOGNAME** is `pwd`"
The current working directory of **gqxing** is /export/home/gqxing
$
```

表2-5总结了三种元字符引用方式的不同处理效果。

表2-5 三种元字符引用的效果

元字符 引用方式	\	\$	*	?	"	'	`
\	不解释	不解释	不解释	不解释	不解释	不解释	不解释
'	不解释	不解释	不解释	不解释	不解释		不解释
"	解释	解释	不解释	不解释		不解释	解释

2.8 命令历史

除了Bourne Shell之外，C Shell、Korn Shell和Bash等大多数Shell均支持命令历史机制，以便维护用户输入的命令。Shell的命令历史机制和编辑功能使用户能够重复利用先前输入的命令，提高用户的交互访问能力。利用Shell的命令历史机制，能够不加修改地重复执行先前提交的任何命令，或在先前命令的基础上，经过校正命令中的细小打字错误或稍加编辑后组成一个新的命令，然后再重复执行先前的命令。

本节我们将以Korn Shell为主，兼顾Bash，介绍Shell的命令历史功能。Shell的命令历史机制主要是由下列内部命令和环境变量实现的：

- **fc**命令——用于列出（“-l”选项）、编辑（“-e”选项）或重新执行命令历史缓冲区或文件中记录的命令。

- **history**命令——用于列出命令历史缓冲区或文件中记录的命令。

- **HISTFILE**变量——用于指定命令历史文件。使Shell能够在停止运行之前把缓冲区中的命令历史记录写入指定的文件，以便在下一次启动时Shell能够读回其中保存的、上一次会话期间执行的命令历史记录。如果HISTFILE变量未定义，或定义的文件没有“可写”的访问权限，默认的命令历史文件为\$HOME/.sh_history。

- **HISTSIZE**变量——指定命令历史文件的大小。用于限定当前会话期间需要保存到命令历史文件中的命令数量。如果HISTSIZE变量未定义，文件容量的默认值为128，即保存最近执行的128条命令。

2.8.1 fc命令

利用特殊的内置命令**fc**，可以按照命令序号或利用命令的起始字符（或字符串），显示、编辑或运行先前执行的命令。在使用**fc**命令列举命令历史缓冲区或文件中的命令时，可以指定单个命令，也可以指定一个命令范围。

实际上，Shell的命令历史机制主要是由**fc**内置命令实现的。**fc**命令允许用户显示、不加编辑或稍加编辑地重新执行命令历史缓冲区或文件中保存的命令，其语法格式如下：

```
fc [ -e ename ] [ -nlr ] [ first [ last ] ]
fc -e - [ old=new ] [ command ]
fc -s [ old=new ] [ command ]
```

fc命令的第一种语法格式表示从用户输入的命令历史缓冲区或文件中选择指定范围（从**first**到**last**）的命令。范围的上限不能超过**HISTSIZE**变量指定的值。**first**和**last**既可以是一个字符串，用于匹配最近执行的、以给定的值为起始字符串的命令；也可以是命令在命令历史缓冲区或文件中的序号（如果在序号前加一个负号“-”，则表示相对于当前命令的偏移值）。如果未指定**last**，其默认值为**first**。如果**first**和**last**均未指定，其默认值分为两种情况：在编辑命令历史缓冲区或文件时仅选择前一个命令；在显示命令历史缓冲区或文件时（使用“-l”选项）选择最近执行的16个命令，即从前一个命令开始回溯16个命令。如果使用“-l”选项，将会在标准输出中列出选择的命令。“-n”选项意味着在列出选定范围的命令时，省略命令在命令历史缓冲区或文件记录中的序号。“-r”选项意味着由近至远反向输出选定范围的命令。

在第一种命令形式中，如果“-nlr”三个选项均未指定，则调用指定的或默认的编辑器，编辑命令历史缓冲区或文件中的命令，命令的范围由**first**和**last**确定。

“-e”选项用于定义在校正或编辑先前的命令时使用的编辑器。如果未指定编辑器的名字，则使用**FCEDIT**变量的值作为默认的编辑器。如果未设置**FCEDIT**变量，则使用**EDITOR**变量的值作为编辑器，否则，最终使用**ed**作为默认的编辑器。在完成命令编辑之后，退出编辑器时即可输出并重新执行刚才校正过的命令。注意，如果选择编辑多个命令时，在退出编辑器时将会依次执行选择的所有命令。

例如，为了列出最近执行的命令，可以使用下列命令：

```

$ fc -l
248      LANG=C
249      export LANG
250      uptime
251      ps -ef | grep cron | grep -v grep | awk '{print $2}'
252      ipcs -m
253      ipcs -q
254      swap -l
255      last
256      /usr/lib/acct/fwtmp < /var/adm/wtmpx | more
257      cd /usr/bin
258      ls -il vi
259      find . -inum 525 -exec ls -il {} \+
260      cat /etc/vfstab
261      ncheck -i 525 /dev/dsk/c1d0s0
262      strings /boot/grub/stage1
263      fc -l
$

```

为了列出最近输入的以`cat`为起始字符串的命令，可以使用下列命令：

```

$ fc -l cat
260      cat /etc/vfstab
261      ncheck -i 525 /dev/dsk/c1d0s0
262      strings /boot/grub/stage1
263      fc -l
264      fc -l cat
$

```

为了利用`vi`编辑并执行序号为210至220之间的一组命令，可以使用下列`fc`命令：

```
$ fc -e vi 210 220
```

第二和第三种命令形式表示跳过编辑阶段。如果存在“`old=new`”形式的字符串替换，则由`Shell`直接执行命令行替换，然后重新执行编辑后的新命令。如果未给出命令，则表示执行之前刚执行的命令。例如，为了重复执行先前的`ls`命令，可以使用下列`fc`命令：

```

$ ls -l /etc/profile
-rw-r--r--  1 root      sys          712 Aug 25  2008 /etc/profile
$ fc -s
ls -l /etc/profile
-rw-r--r--  1 root      sys          712 Aug 25  2008 /etc/profile
$

```

为了重复执行先前的第250号命令，可以使用下列`fc`命令（调用`uptime`命令，查询系统的运行时间，以及最近的平均负载等信息）：

```

$ fc -s 250
uptime
 6:48pm up 39 min(s),  1 user,  load average: 0.01, 0.00, 0.01
$

```

又如，在列出`/export/home/gqxing/incl`目录中的文件之后，如果还想查询与`incl`位于同一层次的`src`目录中的文件时，可以使用“`incl=src`”修正先前的`ls`命令，然后再执行替换后的`ls`命令：

```

$ ls -l /export/home/gqxing/incl
total 94
-rw-r--r--  1 gqxing  other    18644 Jun 19 12:57 atm.h
-rw-r--r--  1 gqxing  other    16582 Jun 19 12:58 event.h
-rw-r--r--  1 gqxing  other    10806 Jun 19 12:59 status.h
$ fc -s incl=src
ls -l /export/home/gqxing/src
total 268
-rw-r--r--  1 gqxing  other    25852 Jun 18 15:31 atmcom.c
-rw-r--r--  1 gqxing  other    26282 Jun 18 15:31 atmmon.c
-rw-r--r--  1 gqxing  other    52125 Jun 18 15:33 handler.c
-rw-r--r--  1 gqxing  other    30625 Jun 18 15:32 listener.c
-rw-r--r--  1 gqxing  other      712 Jun 18 15:34 makefile
$

```

2.8.2 history命令

内置命令**history**是**fc**命令的一个特例（在**Korn Shell**中，**history**只是利用**fc**命令定义的一个命令别名，即“**alias history='fc -l'**”），用于读取、显示或清除命令历史缓冲区或文件中记录的命令。例如，为了列出命令历史缓冲区或文件中记录的命令，可以使用下列命令（在**Bash**中，通常会列出命令历史缓冲区或文件中的所有命令，而在**Korn Shell**中，仅列出16条命令）：

```

$ history
548  LANG=C
549  export LANG
550  date
551  sysdef
552  ifconfig
553  netstat -i
554  cd /etc/ssh
555  vi sshd_cnfig
556  vi ssh_cnfig
557  ssh iscas
558  truss find . -print >find.out
559  truss -p -v all 1
560  vmstat
561  iostat
562  prtconf
563  history
$

```

为了列出最近执行的5条命令，可以使用下列命令：

```

$ history -5
560  vmstat
561  iostat
562  prtconf
563  history
564  history -5
$

```

注意：命令历史机制仅适用于交互式Shell，不能在Shell脚本中使用。

2.8.3 重复执行先前的命令

在Korn Shell中，为了重复执行先前的命令，可以利用单字符命令“r”，引用命令历史缓冲区或文件中的命令。r是利用fc命令实现的一个常用命令别名，即“r='fc -e -'或r='fc -s'”，因此，输入“r cc”命令将会执行最近输入的一个以cc为起始字符串的命令。如果仅仅输入一个r，意味着重复执行最近刚执行的命令。而“r old=new cc”则意味着重新执行最近输入的、以cc为起始字符串的命令。在执行之前，先以给定的字符串new替换命令中第一个出现的字符串old。

例如，为了不加修改地重复执行最近刚执行的ls命令，可以直接使用下列“r”命令：

```
$ ls -l /etc/profile
-rw-r--r--  1 root      sys          712 Aug 25  2008 /etc/profile
$ r
ls -l /etc/profile
-rw-r--r--  1 root      sys          712 Aug 25  2008 /etc/profile
$
```

“r string”命令表示重新执行最近执行的，以给定的string为起始字符串的命令。而“!?string[?]”则表示重新执行最近运行的，其中包含给定字符串的命令。因此，如果输入“!cc”命令，将会再次执行最近输入的以cc为起始字符串的命令。例如，为了重复执行最近一次执行的uname命令，可以使用下列命令：

```
$ uname -n
iscas
$ r un
uname -n
iscas
$
```

“r N”表示重复执行命令历史缓冲区或文件中的第N号命令。而“r -N”则表示重新执行最近执行的倒数第N个命令。

```
$ r 36
date
2009年06月16日 星期二 16时15分40秒 CST
$
```

另外，利用“r old=new [cmd]”命令，还可以先修正刚执行的命令，然后再执行。即在执行之前，先以给定的字符串new替换命令中第一个出现的字符串old，然后重新执行校正后的命令。例如，假设在列出/var/spool/cron/atjobs目录中的文件后，还想查询与atjobs位于同一层次的crontabs目录中的文件时，可以使用“r atjobs=crontabs”命令，使Shell在执行命令之前首先以给定的字符串crontabs替换先前的ls命令中第一个出现的字符串atjobs，然后再执行修正后的ls命令。

```
$ ls -l /var/spool/cron/atjobs
total 2
-r-Sr--r--  1 gqxing  other          590 Jun 29 17:39 1246282200.a
$ r atjobs=crontabs ls
ls -l /var/spool/cron/crontabs
```



```
total 10
-rw----- 1 root    sys      190 Aug 25  2008 adm
-r----- 1 root    root     452 Aug 25  2008 lp
-rw----- 1 root    sys      482 Jun 12 17:42 root
-rw----- 1 root    sys      308 Aug 25  2008 sys
-r----- 1 root    sys      404 Jun 12 17:40 uucp
$
```

表2-6是对先前介绍的“r”命令的总结。

表2-6 常用的部分“r”命令

命令	简单说明
r	重复执行先前刚执行的命令。相当于输入“r -1”命令
r N	重复执行命令历史缓冲区或文件中序号为N的命令
r -N	重复执行从当前命令位置开始倒计数的第N个命令
r string	重复执行最近一次执行的，以给定的部分字符string为起始字符串的命令
r old=new [cmd]	重复执行先前的命令。在执行之前，先以给定的字符串new替换命令中第一个出现的字符串old

在Bash中，可以使用“!”命令实现命令行的编辑与重复执行，与Korn Shell相比，其功能更强，表2-7列出了部分常用的“!”命令，其用法类似于r命令。

表2-7 常用的部分“!”命令

命令	简单说明
!	表示引用命令历史缓冲区或文件中的命令（除非后面紧跟着空格、换行符、等号或左圆括号等字符）
!!	重复执行先前刚执行的命令。相当于执行“!-1”命令或Korn Shell中的r命令
!N	重复执行命令历史缓冲区或文件中序号为N的命令
!-N	重复执行从当前命令位置开始倒计数的第N个命令
!str	重复执行最近一次执行的，以给定的字符串str为起始字符串的命令
!?str[?]	重复执行最近一次执行的，包含给定字符串str的命令
!!str	引用前一个命令，并附加给定的字符串str，然后重复执行组合后的命令
!N str	引用第N个命令，并附加给定的字符串str，然后重新执行组合后的命令
!cmd add	引用最近一个以给定的cmd为起始命令的命令，再附加给定的字符串add，然后重复执行组合后的命令
!?str? add	引用最近一个，且其中包含给定字符串str的命令，再附加给定的字符串add，然后重复执行组合后的命令
!#	引用迄今为止已经输入的所有字符
!\$	引用前一个命令的最后一个参数

(续表)

命令	简单说明
!!:g]s/old/new/	重复执行先前的命令。在执行之前，先以给定的字符串new替换命令中第一个出现的或全部的（如果“s”前面有一个字符“g”）字符串old。如果命令行中存在多个匹配的字符串，需要全部替换，可在“s”之前增加一个字符“g”，以替换所有匹配的字符串。另外，命令行中的最后一个斜线字符可以省略
^old^new^	以给定的字符串new替换先前命令中第一个出现的字符串old，然后重复执行校正后的新命令（快速替换形式）

Bash等Shell还提供其他一些替换与重复执行命令的形式，感兴趣的读者可以查阅联机文档有关fc命令的介绍。

2.8.4 编辑并执行校正后的命令

在Korn Shell或Bash等Shell中，用户可以利用Shell提供的命令历史机制和命令行编辑功能，选用熟悉的编辑器，如vi或emacs等，对先前输入的命令进行编辑，从而生成新的命令，然后提交Shell执行。

进入Korn Shell或Bash之后，Shell将按照FCEDIT和EDITOR变量，以及ed编辑器的顺序确定默认的命令行编辑器。如果未设置FCEDIT变量，Shell将使用EDITOR变量的值作为命令行编辑器，如果未设置EDITOR变量，ed将成为默认的命令行编辑器。

如果选用emacs编辑模式，默认情况下总是处于输入模式，可以利用上下箭头键翻阅之前输入的命令（这些命令存储在用户主目录下的.sh_history文件或由HISTFILE变量指定的命令历史文件中），使用左右箭头键左右移动光标，插入任何字符，输入遗漏的数据，或使用退格键删除光标左边的字符，使用Delete键删除光标所在位置的字符，编辑命令行。

如果想用vi编辑命令行，可以采用下列任何一种设置方式：

```
$ FCEDIT=vi; export FCEDIT
或
$ EDITOR=vi; export EDITOR
```

另外，也可以使用set命令设置默认的命令行编辑器：

```
$ set -o vi
```

一旦采取了上述措施之一，即可进入vi命令行编辑模式。命令提示符所在行（通常是终端窗口底部的最后一行）就像一个只有一行数据的编辑窗口。

无论何时，只要按下Esc键，即可进入vi命令模式。当处于vi命令模式时，可以使用上下箭头键翻阅之前输入的命令，或使用减号“-”（或字符“j”）与加号“+”（或字符“k”）上下移动命令历史记录，翻阅期望运行的命令。也可以使用斜线“/”、问号“?”或G等vi命令前后检索命令，使用左右箭头键左右移动光标。使用编辑命令校正命令行，如使用“x”删除单个字符，使用“r”替换单个字符，使用“R”替换字符串，使用“i（或I）”插入字符或字符串，使用“a（或A）”附加字符或字符串等。

在完成命令编辑之后，按下Enter键即可执行校正后的命令。

与vi编辑模式相比，emacs编辑模式使用起来更容易。表2-8给出了两种编辑模式常用的命

令行编辑命令。

表2-8 常用的命令行编辑命令

vi (命令模式)	emacs	简单说明
上箭头、减号或k	上箭头或Ctrl-P	获取前一个命令
下箭头、加号或j	下箭头或Ctrl-N	获取下一个命令
左箭头、Ctrl-H或h	左箭头或Ctrl-B	左移一个字符位置
右箭头、空格键或l	右箭头或Ctrl-F	右移一个字符位置
b	Esc B	左移一个字
w	Esc F	右移一个字
退格键 (编辑模式)	退格键	删除光标位置左边的一个字符
x	Delete或Ctrl-D	删除光标所在位置的一个字符

2.8.5 命令行补充

利用UNIX系统提供的Readline库， Bash还支持命令行补充功能。当输入的命令名、文件名或变量名不完整时，可以使用制表符键“Tab”实现命令的补充，由Bash提供名字的剩余部分，从而节省用户的输入时间。也可以利用这个机制查询和检索记不清的命令。

1. 命令名补充

当输入部分命令名而按下Tab键时， Bash将会按照命令检索路径， 搜寻以给定文字为起始字符串的命令。如果找不到匹配的命令， Bash将会发出鸣叫声。如果恰好发现一个匹配的命令， Bash将会自动补充命令名的剩余部分。如果发现多个匹配的命令， 在vi编辑模式下， Bash不会输出任何信息， 如果在emacs编辑模式中， Bash将会发出鸣叫声。再按下Tab键之后， Bash将会显示一系列其前缀与用户输入部分匹配的命令， 然后允许用户采用同样的方式继续完成命令的选择。

例如， 当输入bz， 然后两次按下Tab键时， Bash将会给出10个以bz为前缀的命令：

```
$ bz→(Tab)→(Tab)
bzcat      bzdiff      bzfgrep    bzip2      bzless
bzcmp      bzegrep      bzgrep     bzip2recover bmore
$ bz■
```

如果继续输入字符c， 接着再按Tab键两次， Bash将会给出两个以bzc为起始字符串的命令。如果接着输入a再按Tab键， Bash将会完成bzcat命令的补充， 因为此时只有一个匹配的命令：

```
$ bzc→(Tab)→(Tab)
bzcat  bzcmp
$ bzca→(Tab)→t■
$ bzcat■
```

至此， 命令的补充即告完成， 可以接着继续输入其他命令选项和参数。

2. 文件名补充

同样， 当用户输入一个文件名的起始部分， 接着按下Tab键之后， Bash将会提供文件名的剩余部分。如果用户提供的文件名前缀足以唯一确定一个文件， Bash将会显示一个完整的文件名。如果存在多个匹配的文件， Bash将会尽可能地补充文件名的后续部分， 直至遇到某个分界

点需要由用户做出进一步的选择。

假定src目录中存在两个文件：atmcom.c和atmmon.c。当用户输入a，接着按下Tab键之后，Bash将会把文件名补充到atm，然后提示用户做出选择。

```
$ vi src/a→(Tab) → vi src/atm
```

此时，用户可以根据文件列表做出适当的选择，如输入“c”或“m”，再按Tab键即可完成文件名的补充。但是，如果接着按Tab键，Bash将会重新刷新屏幕，同时显示一系列可选的文件供用户做出抉择：

```
$ vi src/a→(Tab)→(Tab) → vi src/atm
atmcom.c  atmmon.c
$ vi src/atm
```

当输入足够的信息（如“c”或“m”），且在按下Tab键之后，Bash将会完成文件名的补充：

```
$ vi src/atmc→(Tab)→om.c
```

至此，用户可以接着输入其他的命令行参数，或按下Enter键以执行输入的命令。

3. 变量名补充

如同命令和文件名补充一样，Bash也支持变量名的补充功能。当输入一个变量名的起始部分，接着按下Tab键之后，Bash将会提供变量名的剩余部分，例如：

```
$ echo $HO→(Tab)→(Tab)
$HOME      $HOSTNAME    $HOSTTYPE
$ echo $HOM→(Tab)→E
```

2.9 命令别名

为了照顾用户的使用习惯和对不同操作系统命令的偏好，简化输入的命令，提供默认的命令选项，Bash以及其他大多数Shell（如Korn Shell、TC Shell、C Shell和Z Shell等）等均提供一种别名机制，使用户能够定义自己喜欢使用的命令名字，以便在输入别名时，Shell能够替换并执行实际的命令（包括选项）。

除了分号、“&”符号、括号、管道符号、书名符号、换行符、空白字符、元字符、引号、等号，以及变量和命令替换字符之外，命令别名可以由任何任意数量的字符或字符串组成。

对于用户而言，别名机制主要是通过alias和unalias命令实现的。其中，alias命令用于定义和列出用户设置的（包括系统定义的）命令别名。其语法格式简写如下：

```
alias [name[=value]]
```

在上述语法格式中，如果别名定义的value中包含空格等空白字符，value前后应加单引号或双引号。例如，为了使用一个简单的命令别名替代chmod命令，以便增加Shell脚本的执行权限，可以使用下列alias命令定义一个命令别名：

```
$ alias cx='chmod 755'
$
```

为了使用一个简单的命令别名替代一个复杂的命令，确保使用的ls等命令总是带有一组基

本选项，可以使用下列**alias**命令定义一个命令别名：

```
$ alias ll='ls -l'
$
```

在UNIX系统中，如果没有特意使用**set**命令设置**noclobber**特性，当使用**cp**、**rm**或**mv**命令复制、删除或重新命名文件时，如果目标文件与现有的文件同名，有可能会在无意之间覆盖或删除同名的文件。为此，我们可以定义下列命令别名，以便在遇到此类情况时能够提示用户做出选择。

```
$ alias cp='cp -i'
$ alias mv='mv -i'
$ alias rm='rm -i'
$
```

在定义命令别名时，等号右边可以包含多个命令，中间由分号隔开；也可以使用管道符号，把多个命令连接起来，构成一个组合命令。例如，为了计数当前目录中的文件，我们可以定义下列别名（注意，当文件名中包含空格字符时，这种文件计数的结果并不准确）：

```
$ alias cf='ls | wc -w'
$
```

当执行上述别名命令时，即可给出当前目录中的文件计数：

```
$ cf
      8
$
```

在输入**alias**命令时如果未指定参数，**alias**命令将会列出系统定义以及用户设置的所有命令别名。如果仅仅指定了命令别名，**Shell**将会列出给定命令别名的定义。例如：

```
$ alias
autoload='typeset -fu'
cf='ls | wc -w'
command='command '
cp='cp -i'
cx='chmod 755'
functions='typeset -f'
history='fc -l'
integer='typeset -i'
ll='ls -l'
local=typeset
mv='mv -i'
nohup='nohup '
r='fc -e -'
rm='rm -i'
stop='kill -STOP'
suspend='kill -STOP $$'
$ alias history
history='fc -l'
$
```

在定义命令别名时，如果语法格式的**value**中包含变量，选用单双引号是非常重要的。在定

义命令别名时，如果使用的是双引号，**value**中的任何变量将会在定义过程中进行变量替换。如果使用单引号，**value**中的变量在调用命令别名之前不会进行变量替换。下面的例子解释了这一差异。

PWD变量包含当前工作目录的路径名。假定在位于主目录（**/export/home/gqxing**）时，用户**gqxing**利用双引号定义了命令别名**dir1**：

```
$ echo $PWD
/export/home/gqxing
$ alias dir1="echo Current working directory is $PWD"
$ alias dir1
alias dir1='echo Current working directory is /export/home/gqxing'
$
```

在建立上述命令别名时，等号“=”右边的**\$PWD**变量将会立即执行变量替换。因此，当使用“**alias dir1**”命令显示**dir1**命令别名的定义时，其输出结果表示变量替换已经发生。不管在何处执行**dir1**命令，其显示的当前工作目录都是替换后的**/export/home/gqxing**，而非实际的工作目录：

```
$ cd /etc
$ dir1
Current working directory is /export/home/gqxing
$
```

当使用单引号定义**dir2**命令别名时，将会防止Shell提前解释**\$PWD**变量。下列“**alias dir2**”命令的输出表示**dir2**命令别名仍然保持未替换的**\$PWD**变量：

```
$ echo $PWD
/export/home/gqxing
$ alias dir2='echo Current working directory is $PWD'
$ alias dir2
alias dir2='echo Current working directory is $PWD'
$
```

当使用**cd**命令改换到其他目录时，**dir2**命令显示的工作目录就是正确的：

```
$ cd /etc
$ dir2
Current working directory is /etc
$
```

unalias命令用于删除已经定义的命令别名，其语法格式如下：

```
unalias [-a] [name ...]
```

其中，“-a”选项用于删除已经定义的所有命令别名，包括系统定义的命令别名。为了删除或撤销某个特定的命令别名，可以在**unalias**命令中直接指定。例如，为了删除已定义的命令别名**cf**，可以使用下列命令：

```
$ unalias cf
$ alias cf
cf: alias not found
$
```

下列命令将会删除系统定义或用户设置的所有命令别名:

```
$ unalias -a
$ alias
$
```

注意: 命令别名不能递归地定义。但是, 在别名定义等号右边的value中, 如果最后一个字符是空格(或制表符), 则紧跟命令别名的下一个命令也将做别名检查和替换。此外, 使用Shell函数也可以实现别名机制的所有功能。

2.10 作业控制

现在, 几乎所有的Shell均支持作业控制功能。在Bash中, set命令的“-m”或“-o monitor”选项用于启用Shell的作业控制功能。通常, 作业控制功能总是启用的。如果作业控制功能不正常, 可以检查这个设置是否正确。

除了进程ID之外, Shell还会为每个作业分配一个数字较小的作业号。例如, 当利用&符号启动后台作业, 使之异步运行时, Shell将会输出下列信息, 其中第一个数字表示作业号, 第二个数字是作业的进程ID:

```
$ find / -name "*conf" -print > conf.log 2>&1 &
[2] 1290
$
```

Shell采用作业控制表, 记录和跟踪当前的作业。利用jobs内部命令, 可以显示作业控制表中保存的当前作业。因此, 为了查询系统中的后台作业信息, 可以使用jobs命令列出系统中的所有作业, 其中包括作业号、作业的命令以及正在运行或暂停运行等状态信息。下面的例子说明当前系统中存在两个后台作业, 其中一个作业正在运行, 另外一个作业已处于停止运行状态:

```
$ jobs
[2] + Stopped (SIGTSTP)      cat -v
[1]-  Running                find / -name "*conf" -print > conf.log 2>&1 &
$
```

当运行一个较长时间才能完成的程序时, 为了能够在此期间继续执行其他任务, 可以使用Ctrl-Z键以及bg命令, 把程序放入后台运行。按下Ctrl-Z键时将会向Shell和当前程序发送一个STOP信号。收到此信号之后, Shell将会输出一个表示作业已经停止(“Stopped”)的信息, 接着输出另一个命令提示符, 示例如下:

```
$ overload.sh
^Z[1]+ Stopped (SIGTSTP)      overload.sh
$
```

此时, 用户可以改变作业的运行状态。或者用bg命令把作业放到后台运行, 或者在完成其他任务后, 利用fg命令仍让作业回到前台运行。

例如, 为了把停止运行的作业放到后台运行, 可以使用下列命令:

```
$ jobs
[1]+ Stopped (SIGTSTP)      overload.sh
```

```

[2]-  Running                  find / -name "*conf" -print > conf.log 2>&1 &
$ bg %1
[1]+  overload.sh &
[2]  Exit 1                    find / -name "*conf" -print > conf.log 2>&1
$ jobs
[1]+  Running                  overload.sh &
$

```

为了让一个后台作业回到前台继续运行，可以使用下列命令：

```

$ jobs
[1]+  Stopped (SIGTSTP)        overload.sh
[2]-  Running                  find / -name "*conf" -print > conf.log 2>&1
$ fg %2
find / -name "*conf" -print > conf.log 2>&1
$

```

为了停止一个后台作业，可以使用下列命令：

```

$ jobs
[1]+  Stopped (SIGTSTP)        overload.sh
[2]-  Running                  find / -name "*conf" -print > conf.log 2>&1
$ kill %1
[1]-  Terminated             overload.sh
[2]+  Exit 1                   find / -name "*conf" -print > conf.log 2>&1
$ jobs
$

```

为了等待一个当前正在运行的作业的完成，可以使用下列命令：

```

$ jobs
[1]+  Stopped (SIGTSTP)        overload.sh
[2]-  Running                  find / -name "*conf" -print > conf.log 2>&1
$ wait %2
[2]-  Exit 1                   find / -name "*conf" -print > conf.log 2>&1
$

```

通常，后台作业可以输出数据，但不允许从终端读取数据。如果后台作业需要从终端读取数据，作业将会停止运行。另外，如果使用“**stty tostop**”命令设置终端选项，将会禁止后台作业输出数据。此后，当遇到数据输出或需要从终端读取数据时，后台作业也会停止运行。

除了作业号之外，Shell还提供若干方法，以使用户选择后台作业，详列如下：

- **%number** 使用作业号引用后台作业
- **%string** 使用给定的字符串作为命令行起始字符串引用作业
- **%?string** 引用命令行含有给定字符串的作业
- **%%** 引用当前作业
- **%+** 等价于%%，表示当前作业
- **%-** 引用前一个作业

在UNIX系统中，如果用户提交了后台作业，Shell会随时监控用户提交的后台作业及其运行状态。不管作业的当前运行状态如何，一旦使用**exit**命令或**Ctrl-D**键退出系统，系统将会输出一条警告信息：“**You have stopped (running) jobs**”。

此时，可以利用**jobs**命令查询后台作业，决定是否需要等待作业的完成。如果经确认后仍想退出系统，而不关心作业是否继续运行，可再次运行**exit**命令或按下**Ctrl-D**键，**Shell**将会立即终止用户提交的后台作业，退出当前的**Shell**，或关闭终端窗口，例如：

```
$ overload.sh
^Z[1]+  Stopped                  nohup overload.sh
$ exit
You have stopped jobs
$ exit
```

如果想在退出系统后仍然确保后台作业能够继续运行，直至其正常结束，可以利用**nohup**命令实现。在此情况下，如果提交的作业处于后台运行状态，当使用**exit**命令、**Ctrl-D**键或关闭终端窗口等方式退出系统时，系统不会给出任何警告信息，但后台作业仍会继续运行，例如：

```
$ nohup overload.sh &
[1]      1213
Sending output to nohup.out
$ exit
```

事实上，**nohup**的功能是让调用的进程忽略**SIGHUP**信号，同时把进程的输出生重定向到用户主目录下的**nohup.out**文件中。当需要提交一个运行时间很长，而用户又不想一直待在终端之前静等命令的最终处理结果时，可以借助于**nohup**命令。

在退出**Shell**时，系统将会向用户注册**Shell**的所有子进程发送一个**SIGHUP**信号。通常，这个信号将会引起用户的所有进程终止运行。如果使用**nohup**命令，所有正在运行的进程或后台作业，包括本身已经屏蔽了**SIGHUP**信号的程序，将会忽略**SIGHUP**信号而继续运行。

对于事先未使用**nohup**命令调用的进程，如果在退出系统时仍想保持继续运行，直至进程正常结束，可以使用**nohup**命令的“-p”选项，指定进程的**PID**，使之屏蔽**SIGHUP**和**SIGQUIT**信号的干扰，从而达到此目的。

```
$ nohup -p pid
$
```

当用户退出系统后，上述进程原本输出到控制终端的所有数据将会重定向并写入用户主目录下的**nohup.out**文件中。

2.11 会话记录

script命令是UNIX系统提供的一个非常有用的工具，可用于记录一个用户从注册到退出系统的整个或部分会话过程，其中包括用户的输入和系统的响应信息。实际上，这个工具仅适用于字符终端设备，尽管也能够捕捉**vi**编辑器等会话过程，但由于在**vi**的编辑过程中很可能会使用控制字符，执行诸如光标移动等操作，使得捕捉的内容难于阅读，如果利用**cat**命令显示捕捉的**vi**会话过程有时可能会一闪而过，因而来不及阅读，致使记录的内容意义不大。因此，最好使用编辑器（如**vi**等）等工具查阅会话过程的记录文件。

通常，**script**命令捕捉的会话过程将会记录在当前目录下的**typescript**文件中。为了使用不同的文件存储会话过程，可以在**script**命令之后指定一个不同的文件名。为了依次记录用户的每一个注册会话过程，可以使用“-a”选项。否则，**script**将会覆盖原有的文件内容，删除先前的会

话记录。下面的例子说明了怎样使用**script**命令记录用户的会话过程：

```
$ script
Script started, file is typescript
$ date
Thu Jun 18 15:03:48 CST 2009
$ uptime
 3:03pm  up 30 min(s),  1 user,  load average: 0.01, 0.01, 0.03
$ ulimit -a
time(seconds)                unlimited
file(blocks)                  unlimited
data(kbytes)                   unlimited
stack(kbytes)                  10240
coredump(blocks)               unlimited
nofiles(descriptors)           256
vmemory(kbytes)                unlimited
$ exit
Script done, file is typescript
$
```

为了终止会话记录功能，可以使用**exit**命令退出**script**命令（这里的**exit**命令并不是退出UNIX系统）。之后，可以使用**cat**、**more**或编辑器查阅**typescript**文件。下面的例子就是利用**cat**命令查阅刚才创建的**typescript**文件时的结果：

```
$ cat typescript
Script started on Thu Jun 18 15:03:42 2009
$ date
Thu Jun 18 15:03:48 CST 2009
$ uptime
 3:03pm  up 30 min(s),  1 user,  load average: 0.01, 0.01, 0.03
$ ulimit -a
time(seconds)                unlimited
file(blocks)                  unlimited
data(kbytes)                   unlimited
stack(kbytes)                  10240
coredump(blocks)               unlimited
nofiles(descriptors)           256
vmemory(kbytes)                unlimited
$ exit

script done on Thu Jun 18 15:04:20 2009
$
```

注意：当使用vi等编辑器编辑**typescript**文件时，可以使用Ctrl-V键、Ctrl-M键和“s”等编辑命令删除多余的回车字符，参见第5章“编辑文件”。也可以利用tr等命令，删除文件中每一行后面附加的回车字符“\M”（即“-d”选项指定的“\”字符）。

```
$ cat typescript | tr -d '\r' > newfile
$
```

2.12 使用man命令查询系统参考手册

UNIX系统提供大量的系统命令和用户命令，而许多命令又有众多的选项，一个人不可能记住这么多命令，即使具有多年UNIX系统经验的资深人员也是如此。对于常用的命令，也不可能记住每一个选项。为此，UNIX系统提供一种联机帮助文档，供用户随时查阅命令的说明与用法。

UNIX系统的联机文档也是通过man命令实现的。为了查询UNIX系统提供的任何命令，了解命令的功能、用法、可用的选项以及参数，可以使用下列命令（其中的command可以是任何UNIX命令，包括man命令本身）：

```
$ man command
```

man借助于more命令，逐页显示指定命令的联机文档，并在终端窗口的左下角输出一个冒号“:”提示符。此时，可以使用空格键逐页显示余下的解释内容，或使用“q”子命令退出man命令。因此，当使用man命令查询联机文档时，可以参照第4章“文件和目录操作”介绍的more命令，按照more命令的用法逐页显示，或随机翻阅，如2-6所示。

```

User Commands                                     ls(1)
NAME
  ls - list contents of directory

SYNOPSIS
  /usr/bin/ls [-aAbcCdEfFghHiIlMnOpqrRstuvVx1@] [file]...

  /usr/xpg4/bin/ls [-aAbcCdEfFghHiIlMnOpqrRstuvVx1@]
  [file]...

  /usr/xpg6/bin/ls [-aAbcCdEfFghHiIlMnOpqrRstuvVx1@]
  [file]...

DESCRIPTION
  For each file that is a directory, ls lists the contents of
  the directory. For each file that is an ordinary file, ls
  repeats its name and any other information requested. The
  output is sorted alphabetically by default. When no argument
  is given, the current directory (.) is listed. When several
  arguments are given, the arguments are first sorted
--More-- (2%)

```

图2-6 man命令

对于任何用户而言，利用man命令随时查询命令的联机帮助信息是非常有用的。对于常用的命令，即使忘记不常用的命令选项及其用法，当只知其名，不知其用法，或只知其意，不知其名（参见后面即将介绍的“man -k”命令）时，均可借助于man命令获取帮助信息。

由于UNIX系统的参考手册非常庞大，为了便于快速阅联机文档，UNIX系统把各种参考手册的内容至少分为7节，分别对应下列文档类型：

- (1) 命令，包括系统命令与用户命令；
- (2) 系统调用（系统内核提供的函数）；
- (3) 库函数；
- (4) 系统文件及其语法格式定义；

- (5) 标准、环境以及宏定义；
- (6) 演示与游戏；
- (7) 设备和网络接口文件。

chmod、chown、kill、mount、nice和uname等既是UNIX系统中的命令，也是同名的系统调用。passwd等既是命令名，又是系统文件名（或一类文件的总称）。如果不加区别，man命令通常只会给出命令的说明，而不会输出系统调用或系统文件格式等的说明。因此，为了查询passwd系统文件，可以使用下列命令（在查询对象之前，指定对象所属的文档类别）：

```
$ man 4 passwd
```

当用户需要完成某种任务，而不知道究竟应使用哪一个命令及其确切的名字时，可以在man命令中使用“-k”选项，按照关键字进行检索。“man -k”命令将会利用给定的关键字检索所有联机手册中每个命令的简单描述部分，显示匹配的命令及其简单说明。

下面的例子说明了怎样使用“man -k”命令查询关键字reboot，其输出结果包括每一个与之相关的命令、系统调用、库函数、文件、文档类型，以及简单描述。

```
$ man -k reboot
fastboot      fastboot (1b)  - reboot/halt the system without checking the disks
fasthalt      fastboot (1b)  - reboot/halt the system without checking the disks
metasync       metasync (1m)  - handle metadvice resync during reboot
reboot         reboot (1m)  - restart the operating system
reboot         reboot (3c)  - reboot system or halt processor
$
```

所有的man联机文档均位于/usr/share/man目录中。由于man联机文档是采用特殊的nroff格式实现的，无法直接打开阅读。为了获得文本格式的文档，以便复制或下载到笔记本供随时查阅，可以使用col命令剔除其中的格式控制字符。例如，为了生成一个文本格式的find命令说明文件，可以使用下列命令：

```
$ man find | col -b > find
$
```


第3章 文件系统基础知识

文件系统是UNIX操作系统的重要组成部分之一，承担信息处理的组织、管理和维护等任务。而文件则是UNIX系统中信息存储、读写和执行的基本单位。操作系统正是通过文件系统实现信息的各种处理功能。本章主要介绍文件系统的基础知识，讨论文件系统的层次结构和组织结构，介绍UNIX系统支持的各种文件类型，以及文件的安全保护机制。

3.1 文件系统的层次结构

在UNIX系统中，信息的基本组织单位称做文件。UNIX文件系统采用一种逻辑的方法组织、存储、访问、处理和管理信息。把文件组织在一个层次目录结构的文件系统中，每个目录包含一组相关的文件（或子目录）。UNIX系统的一个重要特性就是提供一种通用的文件处理方式，简化物理设备的访问；按文件方式处理物理设备，允许用户以同样的命令处理普通文件和物理设备。例如，在打印机上打印文件与在终端屏幕上显示文件的处理方式是类似的。

3.1.1 树形层次结构

从理论方面讲，文件系统是文件的一种逻辑组织结构。从用户的角度看，UNIX的文件系统只是一个树形层次组织结构的目录文件树，文件系统的起点是根目录（/）。根目录相当于整个目录文件树的根，如图3-1所示。

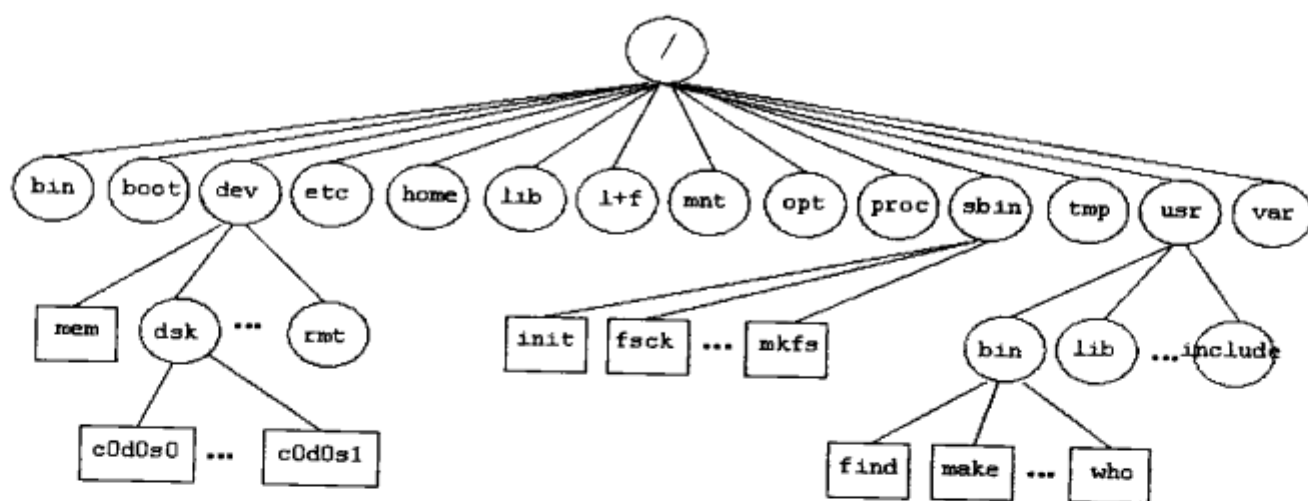


图3-1 目录文件的层次组织结构

子目录是整个目录文件树形层次组织结构中的一个中间节点，是比当前目录层次低一级的目录。文件是整个目录树形层次组织结构中的一个叶子节点。例如，如果/usr目录是当前目录，那么所有位于/usr下面的目录及其子目录都是当前目录的子树。如bin和lib就是/usr下边的子树。除非明确指定了目录路径，大多数UNIX系统命令均把文件参数看做当前目录中的文件。

在文件系统中，若干文件可以组成一个目录，若干目录可以构成一个目录的层次组织结构，而位于目录层次结构顶端的就是一个称之为根目录的特殊目录，根目录包含了各种系统目录和文件，例如，/bin、/dev、/etc、/home、/lib、/proc、/sbin、/tmp、/usr以及/var等标准目录。如图3-1所示为一个简化的文件系统层次组织结构。

在操作系统中，文件系统的设计目的就是要把文件有序地组织在一起。UNIX系统提供一种便于用户从逻辑上组织文件的文件系统。

UNIX系统鼓励用户按照一定的原则建立目录，例如，存储源程序的目录、存储目标程序的目录，以及存储文档的目录等。UNIX文件系统的关键思想是其层次组织结构，不管系统中拥有多少用户，每个用户都可以创建若干目录及其子目录，分类存储自己的文件。

另外，文件的访问权限是多用户计算机文件系统的一个重要组成部分，用于保护用户的数据安全。

UNIX系统的一个重要特性是，所有的I/O设备都与特殊文件联系在一起，用户无需了解硬件设备的读写方式，只需像操作普通文件一样操作特殊文件，即可达到访问I/O设备的目的。例如，读取特殊文件相当于从硬件设备中直接读出数据，写特殊文件则相当于直接向硬件设备发送数据。利用特殊文件，实现了用户程序与硬件设备之间的通信，由文件系统管理硬件设备的I/O处理，使硬件设备对用户是透明的。

3.1.2 路径名

无论何时，当前工作目录中的所有文件都是可以直接存储的。通过名字，可以直接引用文件。而对于非当前目录中的文件，必须在文件名之前加上各级目录路径才能访问。文件的路径名指的就是从某个目录（如根目录或当前目录）开始，穿过整个文件系统，直至到达目标文件而经过的一条目录层次路径。例如，从根目录（/）开始，中间经过usr和bin两级子目录的一条路径就是find文件的路径名：

```
/ → usr → bin → find
```

把上述访问路径写成UNIX文件系统中标准路径名就是/usr/bin/find。

文件的访问路径可以有两个起点：一是当前工作目录；二是根目录。凡是以根目录为起始位置，即以斜线字符“/”为起始字符的路径名称为绝对路径名。绝对路径名指定了文件在文件系统的层次组织结构中从根目录开始的存储位置。

相对路径是指相对于当前工作目录而言的目录。凡以当前工作目录或其他以非斜线字符为起始字符的所有路径名都是相对路径名。相对路径名指定了文件在文件系统中相对于当前工作目录的存储位置。

例如，路径名/usr/include/stdio.h就是一个从根目录开始的绝对路径名。其中，usr是根目录的子目录，include是usr目录的子目录，而stdio.h则是这个目录层次末端的一个文件。

include/stdio.h是相对于/usr目录的一个相对路径名，而stdio.h则是相对于/usr/include目录的一个相对路径名。

此外，每个目录中均包含以句点“.”和双句点“..”命名的两个特殊目录文件，分别表示当前目录及其父目录。这两个特殊目录把文件系统各级目录有机地联结在一起。其中句点“.”是当前目录的别名，期望访问当前目录中的文件时，都可以直接使用句点“.”而不必明确给出当前目录名。双句点“..”是当前目录父目录的别名。从任何目录位置开始，使用双句点“..”形式的父目录，可以逐层攀升到文件系统层次组织结构的最上层。

下面几个简单的规则适用于所有的路径名：

- 如果路径名以斜线字符开始，则说明路径名是从根目录开始的绝对路径名，除此之外，其他所有的路径名都是相对于当前目录的相对路径名。

- 路径名或者是由斜线字符分隔的一系列名字，或者是单个名字。在一串名字中，最后一个名字就是实际的文件，其他名字均为目录。文件可以是任何类型的文件。
- 不管处于任何目录位置，在路径名中使用一个双句点“..”符号即可往上攀升一个目录层次。连续使用多个双句点“..”符号可以逐次往上攀升多个目录层次。在路径名中，除了双句点“..”之外，其他任何名字均为降低目录层次。

3.2 文件系统的组织结构

在UNIX文件系统中，目录和文件的组织结构是有一定规律可循的。这种有序的组织结构有助于用户访问、管理和维护UNIX系统，如图3-2所示为一个典型的文件系统目录组织结构。

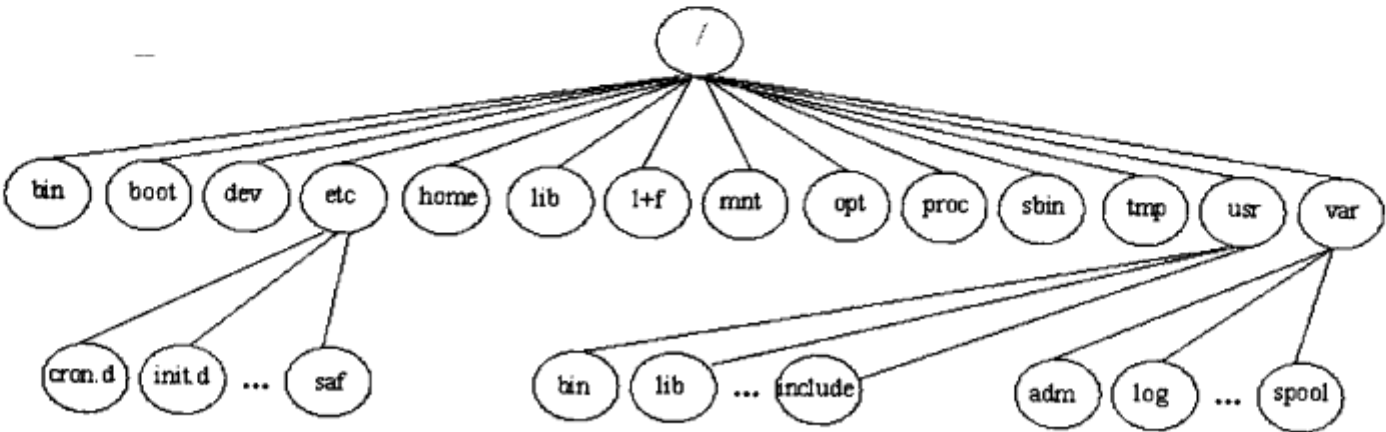


图3-2 UNIX文件系统的典型目录结构

表3-1是对UNIX系统部分主要目录的简要说明。

表3-1 UNIX系统部分主要目录的简要说明

常见目录	典型子目录	简单说明
/bin		其中包含系统管理员和普通用户可以共享的各种通用程序，如cat、cp、mv、rm、ls、ps等常用的基本命令，以及各种Shell
/boot		其中包含系统引导程序、内核文件、内存映像文件以及GRUB初始引导程序和配置文件等
	grub	其中存有GRUB配置文件，以及三种不同类型的初始引导程序等
/dev		在UNIX系统中，任何设备均对应一个或多个特殊文件（也称设备文件），这个目录包含系统支持的所有设备文件。例如，console表示系统控制台，lpX表示打印机，diskette表示软驱，ttyXX表示系统的串口设备，mem表示物理内存，kmem表示系统内核占用的虚拟内存等
	dsk	其中用于定义磁盘等存储设备的块特殊文件
	rdsk	其中用于定义磁盘等存储设备的字符特殊文件，使之能够支持原始数据接口
	rmt	其中用于定义磁带机等设备的特殊文件
	term	其中用于定义终端等异步通信设备的特殊文件
/etc		这个目录是整个UNIX系统的中心，其中包含许多重要的系统文件与配置文件，如hosts、inittab、passwd、group和vfstab等，此外，还有其他大量的配置文件分别位于单独的子目录中。通常应注意备份这个目录中的重要配置文件，以便需要时能够快速地恢复系统

(续表)

常见目录	典型子目录	简单说明
	apache / apache2	Apache配置文件的根目录，其中包含Apache服务器的各种配置文件，如httpd.conf等。Apache是一个通用的、高性能的HTTP服务器，也是目前世界上最流行的Web服务器。Apache采用模块化的设计方式，支持运行时的动态模块选择、虚拟主机，以及服务进程数量的动态调整等
	cron.d	用于存储cron调度运行后台作业所用的控制文件，如at.deny、cron.deny和FIFO等文件
	default	其中的文件用于提供部分软件使用的控制变量及其默认值，其中包括autofs、fs、ftp、init、passwd和tar等变量定义文件
	dfs	NFS等共享资源配置文件所在的目录
	ftpd	其中存有ftpaccess和ftpusers等控制文件
	inet	其中包含TCP/IP的配置与维护文件
	init.d	用于存储系统启动过程中需要由init调度执行的脚本文件
	lp	其中存有lp假脱机打印机的各种配置文件
	ppp	用于存储PPP的配置文件
	rcN.d	用于存储进入相应运行级时由init进程调度执行的脚本文件（其中的N为0、1、2、3、4、5、6和S，表示系统的运行级）
	saf	sac终端访问服务使用的各种配置文件的根目录。用于维护ttymon与listen等终端及网络访问服务
	skel	用于储存最基本的用户初始化文件，如local.cshrc、local.login和local.profile等。每当增加新用户时，都会把其中的初始化文件复制到用户的主目录中
	ssh	OpenSSH网络服务使用的各种配置文件的根目录，其中含有sshd_config等重要配置文件
/export		用于提供目录或文件系统等共享资源
/home		用户主目录的根目录。每当增加一个新用户，都会将/home目录中创建一个形如/home/\$USER的子目录，作为用户的主目录，其中的\$USER为用户名（注意，Solaris系统使用/export/home作为用户主目录的根目录）
/lib		此目录含有系统引导过程，以及运行系统命令所需的内核模块及各种动态链接共享库文件（其扩展名为.so，相当于Windows系统中的.dll文件）
/lost+found		每个文件系统分区都存在一个lost+found目录，用于存储fsck命令在检测与修复文件系统时删除的文件或目录。UNIX系统要求正常地关机，一旦由于电源等硬件故障或软件故障致使系统异常停机，将会导致文件系统损坏，在重新启动系统时，系统需要利用fsck命令检测与修复文件系统。在检测文件系统期间，fsck将会尝试修复任何受损的文件。部分恢复的文件或无法恢复的数据块将会放在每个文件系统的lost+found目录中。如果这个目录中存有受损的数据文件，可以使用fsdb等工具，尝试自行恢复丢失的数据。如果是系统文件，也许需要重新安装相应的软件包

(续表)

常见目录	典型子目录	简单说明
/mnt		文件系统的临时安装点。这是一个通用的安装点，可以临时安装任何文件系统或远程资源。如果愿意，也可以在此目录中创建若干子目录，以便安装不同的设备，如使用/mnt/cdrom和/mnt/usb分别安装CD/DVD和U盘等
/opt		应用程序等附加软件的安装目录
/proc		虚拟文件系统/proc的根目录，其中的数字子目录都是以进程ID命名的，分别对应于当前正在运行的实际进程
/sbin		其中包含与系统引导、系统管理与维护，以及硬件配置等方面的有关命令或脚本文件，如fdisk、ifconfig、init和mount等。在安装了/usr等文件系统之后才需要执行的系统命令通常放置在/usr/sbin目录中。注意，/sbin目录中的命令主要供超级用户使用，普通用户通常无法使用
/tmp		临时文件目录。用于存储系统运行过程中生成的临时文件，也可以供用户存储自己的临时文件。在系统的运行过程中，许多程序均使用这个目录存储临时数据文件，其中的许多文件对于当前正在运行的进程而言是非常重要的，删除这样的文件可能会导致系统瘫痪。因此，一般不要自己删除这个目录中的文件，除非确实有把握。对于大多数UNIX系统，在系统的启动过程中，将会清除这个目录中的所有文件
/usr		/usr既可以作为一个单独的文件系统，也可以作为根目录下的一个子目录，其中存有系统提供的共享数据（如用户命令、库函数、头文件和文档等）
	apache {2}	其中分类存储Apache服务器提供的实用程序、头文件、共享库函数、联机文档以及HTML形式的手册等
	bin	其中包含用户可用的各种命令
	ccs	C开发环境专用目录，其中含有make、as、dis以及ld等工具软件
	include	用于存储各种C语言头文件。这个目录及其子目录中的头文件是C开发人员需要经常引用的文件。其中，sys等子目录中定义的数据结构，对于深入学习、理解和掌握UNIX系统具有极大的参考价值
	lib	其中包含各种共享库函数，可供程序员以静态或动态方式链接自己开发的应用程序
	sbin	其中包含系统引导或启动过程中需要用到的各种系统命令
/var	share	共享目录，其中含有man（联机文档的根目录）和doc（各种软件包特定的文档）等子目录
		/var既可以作为一个单独的文件系统，也可以作为根目录下的一个子目录，用于存储各种可变长的数据文件（如日志文件）、暂存文件或待处理的临时文件等
	adm	用于存储lastlog（其中包含用户上一次注册的时间记录）、messages（其中包含系统内核与程序的日志信息）、sulog、utmpx（其中包含用户的当前活动信息）和wtmpx（其中包含用户的系统注册信息）等文件
	apache / apache2	其中主要用于存储Apache服务器提供的各种内容服务，如文档根目录htdocs、CGI动态内容根目录cgi-bin，以及日志文件等

(续表)

常见目录	典型子目录	简单说明
	cron	其中存有cron守护进程的日志文件log
	log	系统守护进程日志文件的目录位置，其中包括authlog和syslog等重要日志文件。位于/var/log目录中的文件会不断地增长，因而要求定期地备份或清除。通常，UNIX系统会定时执行例行检查，以循环截取的方式，删除过时的数据，保留一定时间范围内的最新数据，使文件的大小能够保持一个适中的规模
	lp	其中存有lp打印服务的日志文件等
	run	系统运行信息文件的根目录，其中的部分.pid文件存有相应守护进程的PID
	samba	其中存有Samba资源共享系统的日志文件，及其使用的封锁文件等
	sadm	用于存储软件包或软件补丁的安装记录等
	saf	其中包含终端与网络访问服务的日志文件
	spool	用于缓存各种待处理的文件，如后台作业、软件包以及打印任务等。通常，每一类等待处理的缓存文件均位于各自的子目录中，如/var/spool/pkg等
	tmp	用于存储各种临时文件

3.3 文件的类型

从理论上讲，文件是由一系列连续的字节流组成的，最后以一个EOF字符结束。但从物理实现来讲，文件实际上是由磁盘（或其他存储介质）上的一系列数据块组成的，而组成文件的数据块，并不一定是连续的。

UNIX系统可以同时支持许多不同类型的文件系统，以及不同类型的文件。本章所述的文件仅指UNIX文件系统支持的基本文件。也就是说，所谓文件乃指驻留在本地磁盘上的，用户经常与之打交道的各种基本文件。

在UNIX系统中，文件是最基本的数据组织单位，所有的输入输出操作都是通过文件实现的，UNIX系统处理的任何数据和设备均可归结为对文件的操作。从理论上讲，能够读写普通文件的任何程序都可以读写任何I/O设备。

从用途方面划分，一个文件可以是：

- 文档——也即普通的文本文件，包括脚本、程序源代码、配置数据和日志等。
- 命令——大多数命令都是可执行的二进制数据文件。也就是说，命令是可以执行的程序文件。例如，date命令就是一个可以执行的程序文件，其功能是输出和设置当前的系统日期与时间。
- 目录——简言之，目录是一个包含文件名列表的数据文件。
- 设备——包括终端、磁盘、CD/DVD、磁带机和打印机等。

在UNIX系统支持的各种文件系统中，如最流行的UFS或日志文件系统，通常均支持普通文件、目录文件、特殊文件、链接文件、符号链接文件，以及管道文件等6种不同类型的常规文件（套接字文件不在本章讨论之列），现分述如下。

3.3.1 普通文件

在UNIX系统中，普通文件简称文件。所谓文件，实际上是一个命名的数据集合，是一组信息的基本存储单位。通常，每个文件都拥有一个名字，通过名字，可以对文件的数据内容进行处理。

在传统的System V文件系统中，文件名最多不能超过14个字符。而在UFS等文件系统中，文件名可以长达255个字符。在不同的目录中，文件可以同名，但在同一个目录中，则不允许同名的文件存在。原则上，文件名可由任何字符组成，但若包含不可打印字符、空白字符及Shell元字符，在实际应用过程中将是非常麻烦的。

普通文件可以存储任何数据，存储的内容可以是ASCII文本、源代码、Shell脚本、配置数据及各种文档等，也可以是二进制程序代码等。由此可见，普通文件是UNIX文件系统中应用最多的一种文件类型。

此外，UNIX系统中的文件并没有所谓的记录、结构以及内容之分，所有文件都是由一维的字符流组成的，存储在磁盘等存储设备中，在文件系统中占有零个或多个数据块。而且，数据也并非一定驻留在连续的磁盘块中。但操作系统隐藏了这一内部存储形式，用户看到的是一系列连续的字节流。对于用户而言，无需考虑文件的底层存储结构。

当然，也可以根据数据内容把文件分类为C源代码、Shell脚本、文本数据以及二进制可执行程序等。但UNIX系统却不这样认为，对于存储不同数据的普通文件，UNIX操作系统不加任何区别，也不要求一个普通文件必须采用什么记录结构。

但是，为了便于维护与处理，UNIX系统也定义了若干标准数据文件格式，例如，二进制可执行文件必须采用a.out文件格式，一些档案文件前部通常都会包含一个文件格式标识代码（magic code），用于标识文件的内容。例如，cpio命令生成的档案文件，其标识代码为070701（参见/etc/magic文件）。此外，应用程序也可以按一定的记录格式组织和存储自己的文件。实际上，数据库等应用程序也是这样组织和使用文件的。

由于UNIX系统并不刻意区分文件的类型，不像Windows系统那样，以扩展名区分文件的类型。因此，单从文件名本身来看，UNIX系统中的大部分文件都无从知道其类型。UNIX文件系统也没有对文件的命名强加任何约定，但由于应用程序的需要及用户的使用习惯，通常以“.c”作为C源程序文件名的后缀，以“.sh”作为Shell脚本文件名的后缀。按照惯例，可执行程序的文件名通常不加任何后缀。

当需要确定文件的内容，判断文件的类型时，可以选用一定的工具，其中最典型的就是ls和file命令。例如，下列file命令可用于确定部分文件的内容与类型（实际上，标识代码也是file命令用于确定文件内容与类型的方法之一）：

```
$ cd /etc
$ file hosts magic default initpipe TIMEZONE
/etc/hosts:      ascii text
/etc/magic:      English text
/etc/default:    directory
/etc/initpipe:   fifo
/etc/TIMEZONE:   commands text
$ file /var/adm/utmpx /sbin/mount /sbin/mountall
/var/adm/utmpx:  data
```

```

/sbin/mountall:    executable /sbin/sh script
/sbin/mount:       ELF 32-bit LSB executable 80386 Version 1, dynamically
linked, stripped
$

```

从用户的角度来看，普通文件可以分为两种类型：即文本文件和二进制数据文件。文本文件只包含可打印字符，如ASCII字符、中文字符等，而二进制数据文件中的每个字节允许有256种数值。

在UNIX文件系统中，通常提供下述文件操作：打开文件（open）、创建文件（create）、读文件（read）和写文件（write）等。UNIX系统提供大量的文件操作命令，用于创建、编辑和处理文本文件。例如，为了显示一个文本文件，可以使用下列cat命令（有关文件的各种操作与处理方法，详见第4章“文件和目录操作”）：

```

$ cat /etc/motd
Sun Microsystems Inc.    SunOS 5.10        Generic January 2005
$

```

对于二进制数据文件来说，其中的内容无法直接显示，因为每个字节的256种取值大部分都是不可打印字符。若想显示（而不是执行）二进制数据文件的内容，可以使用专门的命令，如od命令，以ASCII字符、八进制数据或十六进制数据等形式显示二进制数据文件。例如，为了检查一个二进制数据文件的内容，尤其是头部是否包含特定的数据标识码，可以使用下列命令：

```

$ od -c file.cpio
0000000  0  7  0  7  0  1  0  0  0  0  0  0  1  0  0  0
0000020  0  0  4  1  e  d  0  0  0  0  0  0  6  4  0  0
.....
$

```

3.3.2 目录文件

目录文件简称目录。目录也是一种文件，是一种特殊类型的文件，其中存储的内容不是普通意义上的数据，而是一系列文件名及其信息节点号。除了存储的内容不同之外，目录文件与普通文件在文件系统中的存储方式是一样的。目录用于提供文件名、信息节点与文件数据之间的关联关系。因而可以说，目录的结构也决定了文件系统的组织结构。

严格地讲，目录文件是由一系列目录项组成的，而每个目录项又主要是由两个不同的字段组成的：一个字段为信息节点号，用于引用信息节点，另一个字段为文件的名称。“ls -ai”命令可用于列出目录中的文件名及其信息节点号，下面是这个命令的一个输出结果（其中第一列为信息节点号，第二列为文件名）：

```

$ ls -ai | sort -n
2 .
2 ..
3 lost+found
4 export
6 var
28 usr
.....
$

```


在目录文件中，每个目录项通过信息节点号实现了文件名与文件数据的映射。通过文件名可以找到其信息节点，通过信息节点最终又可以找到文件中的实际数据内容。在上述例子中，文件名`/lost+found`的相应信息节点号是3，是继根目录（`/`）之后创建的第一个目录。

如前所述，UNIX文件系统中的每个目录均包含两个特殊的目录，即以句点“`.`”和双句点“`..`”表示的当前目录及其父目录。因此，为了进入当前目录或父目录，用户可以相应地引用“`.`”或“`..`”。例如，如果当前的工作目录为`/usr/include/sys`，输入“`cd ..`”命令后即可进入`/usr/include`目录：

```
$ pwd
/usr/include/sys
$ cd ..
$ pwd
/usr/include
$
```

“`usr`”是一个目录文件，虽然其中包含的数据与普通文件并无实质的差别，但与普通文件不同的是，目录文件是由UNIX文件系统直接管理的，用户只能查询其中的文件列表，不能使用编辑器等工具直接修改目录文件，也就是说，用户只能读取目录文件，只有操作系统才能写目录文件。

尽管超级用户可以利用UNIX系统提供的若干实用程序维护目录文件，但普通用户绝对不能直接写目录文件。例如，超级用户可以利用文件系统调试器`fsdb`，直接操作文件系统的任何组成部分。但应了解的是，用于维护目录（或文件系统）的大多数实用程序都是具体文件系统特定的。

为了保证目录的完整性，操作系统不允许用户直接修改目录文件。用户可以在目录中创建文件，由操作系统把文件加到目录中。当用户删除文件时，由操作系统从目录文件中删除相应的文件名，目录文件始终是由操作系统负责维护的。

这是因为目录文件毕竟不是普通文件，如果不了解目录的组织结构，不了解文件系统的工作原理，如果处理不当，将会造成无法预料的后果。即使熟悉文件系统，也不能直接操作目录文件。因为目录操作涉及一系列内部的处理步骤和过程，不是一般用户所能胜任的。

在任何目录中，可以存储普通文件、管道文件、特殊文件、符号链接文件，也可以创建目录文件，称做子目录。为了创建一个目录，可以使用下列命令：

```
$ mkdir dirname
```

为了进入不同的目录，可以使用下列命令：

```
$ cd dirname
```

在UNIX系统中，每个用户都拥有一个属于自己的目录，称做用户主目录。一旦注册到UNIX系统，系统将会自动地把用户引导至自己的主目录。用户可以在主目录中创建自己的文件，创建自己的子目录。在与UNIX交互会话期间，用户也可以利用`cd`命令自由地从一个目录转移到其他目录中（如果具有足够的访问权限）。例如，如果想进入`/usr/bin`目录，可以输入下列命令：

```
$ cd /usr/bin
$
```

在系统访问期间，用户所在的目录称为当前目录、工作目录或当前工作目录。在UNIX系

统中，通过使用不加分任何选项的cd命令，即可返回到自己的主目录。例如，任何用户都能够使用下列命令进入自己的主目录：

```
$ cd
$
```

为了了解自己当前所处的目录，可以输入pwd命令。下列命令的输出结果表示用户的当前工作目录为/usr/include：

```
$ pwd
/usr/include
$
```

尽管目录文件通常是由系统维护的，用户不能直接写目录文件，但任何用户进程均可利用下列与目录操作有关的系统调用访问目录，自由地读取目录文件，只要具有相应的访问权限：

- mkdir() 创建一个新目录；
- rmdir() 删除一个空目录；
- getdents() 获取目录文件的内容。

最初，S5文件系统中的每个目录项仅为16个字节，其中前两个字节为信息节点号，后14个字节为文件名，这就是早期UNIX系统中文件名的长度为14的由来。其结构定义如图3-3所示。

而在UFS文件系统中，目录项（或文件名）的长度是可变的。但不管长度如何，每个目录项均存储在一个512字节的目录块中。按照定义，每个目录项均包含一个信息节点号、目录项的总长、文件名长度，以及文件的名称。最后再附加4个NULL填充字符，保证文件名均以一或多个NULL字符结束，如图3-4所示。

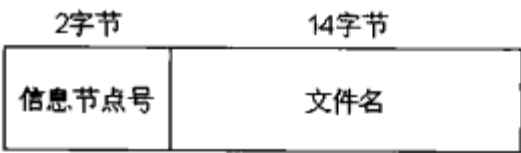


图3-3 S5文件系统目录项的结构定义

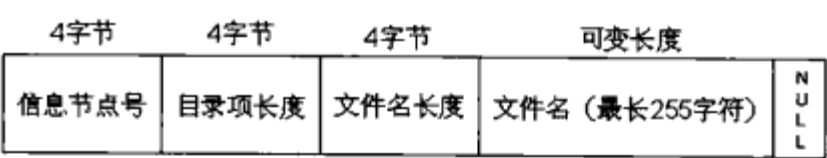


图3-4 UFS文件系统目录项的结构定义

在UFS文件系统中，每个目录都是由上述结构信息组成的。除了存储的数据内容不同之外，目录文件与普通文件在文件系统中的存储方式是一样的。尽管从理论上讲，用户也能够访问目录，但目录文件通常是由系统维护的。

3.3.3 特殊文件

特殊文件是UNIX系统中最具特色的重要特性之一。特殊文件也称设备文件。为了便于用户访问外部设备而不必涉及各种I/O设备的具体操作细节，UNIX系统利用特殊文件作为用户与I/O设备之间的接口，使用户能够像读写普通文件一样访问外部设备。每个特殊文件均对应一个I/O设备，由I/O设备驱动程序实现用户与设备之间的数据通信。因此，用户可以像处理普通文件一样，通过打开、读写特殊文件等操作，实现访问外部设备的I/O处理要求。

特殊文件不包含任何数据。相反，特殊文件只是提供一种机制，即在文件系统中建立一个物理设备与文件名之间的映射。系统支持的每个设备，包括内存，都与至少一个特殊文件相关联。特殊文件是利用下列mknod命令或系统调用创建的，而且必须提供相应的驱动程序，并集成到系统内核中。否则，即使创建了特殊文件，也无法访问相应的设备。

mknod special type [major minor]

其中, **special** 为特殊文件名, 如 **/dev/console** (即系统控制台)。**major** 为主设备号, 表示按设备类型组织的设备驱动程序指针数组的索引。**minor** 为次设备号, 表示同类设备中的某个子设备, 可以用做调用相应驱动程序的参数。**type** 为特殊文件的类型, 合法的特殊文件类型如下:

- **c** 字符特殊文件;
- **b** 块特殊文件;
- **p** 管道文件。

例如, 为了创建一个管道文件, 可以使用下列命令:

```
# mknod mypipefile p
# ls -l mypipefile
prw-r--r--  1 root    root          0 Jun 15 19:00 mypipefile
#
```

当提交一个读写特殊文件的请求时, 系统将会直接调用相应的设备驱动程序。驱动程序则负责在控制进程与相关的物理设备之间传输数据。例如, 假定存在下列两个命令:

```
# cp /etc/passwd /tmp/garbage
# cp /etc/passwd /dev/console
root:x:0:0:Super-User:/:/bin/ksh
daemon:x:1:1:/:/
bin:x:2:2:/:/usr/bin:
sys:x:3:3:/:/
adm:x:4:4:Admin:/var/adm:
.....
#
```

第一个命令只是把 **/etc/passwd** 文件的内容复制到 **/tmp/garbage** 文件中, 创建或复制一个相应的文件。第二个命令则意味着把 **/etc/passwd** 文件复制到 **/dev/console** 文件中, 而后者实际上是一个与控制台终端相关联的特殊文件。因此, **/etc/passwd** 文件的内容将会显示在控制台终端的屏幕上。

UNIX 系统支持两种特殊文件: 即字符特殊文件和块特殊文件。除了一个目录项和一个信息节点之外, 设备特殊文件并不占用文件系统的任何空间, 而且这个特殊文件只是一个设备驱动程序的访问点。

UNIX 系统中的每一个设备或者是块设备, 或者是字符设备, 二者必居其一。通常, 块设备主要用于存储数据, 字符设备主要用于传输数据。例如, 磁盘和 **CD/DVD** 等均属于块设备, 连接到串口和并口的设备等均属于字符设备。

设备除有块设备和字符设备等类型之分, 还有主次设备号之分。主设备号用于区分不同的设备, 次设备号用于区别同类设备的特定设备或分区。例如, 假定磁盘设备的主设备号为 8, 则次设备号 1、2、... 分别表示磁盘设备的第一个、第二个、... 磁盘分区。

1. 块特殊文件

块特殊文件与采用数据块组织结构和处理方式的设备 (如磁盘) 相关联。所谓数据块组织结构的设备实际上指的是能够以固定长度的数据块传输数据, 也能够随机访问其中任何数据块的存储设备, 而且, 访问每个数据块的时间差别也不大。

磁盘就是典型的数据块组织结构的设备。在 **UNIX** 系统中, 磁盘与系统内存之间通常是以数据块的方式传输数据的, 以数据块为单位读写数据。借助于文件系统, 可以在磁盘的任何位

置读写任意字节数量的数据。

在UNIX系统中，一个典型的数据块是由512、1024、4096或8192个字符组成的。因为文件系统通常都是驻留在块设备中的，故许多文件系统的维护命令均要求使用块特殊文件名作为文件参数。

例如，下列设备文件名就是一个对应于数据块组织结构的磁盘特殊文件（其中第一个字段的起始字符为b）：

```
$ ls -l /dev/dsk/c2t0d0s6
brw-rw-rw-  1 root      sys      35,  6 Dec 22 08:18 /dev/dsk/c2t0d0s6
$
```

当通过块特殊文件接口传输数据时，系统通常会在其内存（或高速缓冲区）中缓存数据。在指定的时间间隔内，系统将会把内存中的数据写入外部存储设备，从而更新存储设备中的数据。在交互访问或应用程序中，用户可以利用sync命令或fsync()系统调用，强制系统把内存中的数据写入存储设备，实现存储设备与内存之间的数据同步。

采用这种设计方式的一个问题是，如果在内存与存储设备进行数据同步之前，系统突然瘫痪，数据的完整性将会遭到破坏。因此，如果UNIX系统出现故障，内存中尚未写到存储设备上的最新数据很可能会丢失。

更重要的是，内存中存有文件系统超级块中的最新数据，如果之前没有及时同步，文件系统（内存超级块与磁盘超级块之间）将会处于数据不一致的状态。如果再严重一点，文件系统有可能无法修复。

2. 字符特殊文件

任何非数据块组织结构的设备均为字符设备，而字符特殊文件就是与非数据块组织结构的任何设备相关联的特殊文件。与数据块组织结构的设备相反，字符设备无法使用定长的数据块，也不能随机访问，其最底层的I/O接口一次只能处理一个字符，这与块设备I/O接口一次能够处理一个完整的数据块的情况全然不同。

控制台终端、打印机、流式磁带机和其他串行设备均属于字符设备，因而具有相应的字符特殊文件名。当然，为了用于其他目的，数据块组织结构的设备（如磁盘）也具有相应的字符特殊文件名。

通过字符特殊文件接口传输的数据将会在设备驱动程序与控制进程之间直接传递，中间并不经过系统的数据缓冲区。因此，这种形式的设备接口经常称做原始接口。例如，下列设备文件名就是一个对应于磁盘的字符特殊文件（其中第一个字段的起始字符为c）：

```
$ ls -l /dev/rdisk/c2t0d0s6
crw-rw-rw-  1 root      sys      35,  6 Dec 22 08:18 /dev/rdisk/c2t0d0s6
$
```

3. 定义特殊文件

根据前述说明，每个设备都对应于一个特殊文件。通过读写特殊文件，即可实现设备的I/O处理要求。实际上，这些特殊文件只是一个接口，设备的I/O处理都是由设备的驱动程序完成的。因此，每当在系统配置中增加一个新的设备时，都需要完成下列任务：

（1）获取或编写相应设备的驱动程序；

- (2) 更新系统配置表, 描述新加的设备及相应的设备驱动程序;
- (3) 重新生成新的UNIX内核, 使其包含新的设备驱动程序;
- (4) 利用mknod命令, 创建新增设备的特殊文件。

4. 特殊文件实例——4个字符特殊文件

UNIX系统提供的mm驱动程序采用4个不同的次设备号, 支持4个不同的字符特殊文件, 作为用户与系统内存之间的设备接口。在安装UNIX操作系统时, 系统会自动地创建mm驱动程序支持的4个不同的特殊文件。这4个特殊文件的作用简述如下:

- /dev/null是一个数据回收站或漏斗文件, 也是一个无底洞, 只要写入这个文件, 数据就会消失。这个文件经常用于过滤程序的输出目的。注意, 当读取此文件时, 其功能等同于读取/dev/zero文件, 用于返回指定数量的数值0。

- /dev/zero可以提供任意数量的数值0。例如, 当程序从这个文件中读取256个字节时, 将会收到256个0。如果写入这个文件, 数据将会消失, 其作用等同于/dev/null。

- /dev/mem提供计算机的物理内存接口, 是UNIX系统的内存映像。读取/dev/mem文件的字节地址将被解释为物理内存地址。如果从头开始(偏移值为0)读取这个文件, 将会读取物理内存中从第一个字节开始的数据。

- /dev/kmem提供系统内核的虚拟内存接口。如果从头开始(偏移值为0)读取这个文件, 将会读取系统内核从第一个字节开始的数据。如果没有这个接口, 开发人员必须知道系统内核代码及其数据在物理内存中的起始位置。许多系统程序就是利用这个特殊文件访问系统内核, 获取各种系统变量的内核地址, 维护系统数据的。

在上述4个文件中, /dev/null最为常用。当我们不关心程序的标准输出, 或标准错误输出, 而只是关注程序的出口状态时, 经常会使用下列命令形式:

```
command > /dev/null
command > /dev/null 2>&1
```

从Solaris UNIX系统中可以清楚地看到, 上述特殊文件的驱动程序均为mm。但由于这些文件是符号链接文件, 故还需要进入相应的目录, 进一步验证这4个文件确为具有不同次设备号的字符特殊文件:

```
$ cd /dev
$ ls -l kmem mem null zero
lrwxrwxrwx 1 root other 27 6月 12日 17:22 kmem -> ../devices/pseudo/mm@0:kmem
lrwxrwxrwx 1 root other 26 6月 12日 17:22 mem -> ../devices/pseudo/mm@0:mem
lrwxrwxrwx 1 root other 27 6月 12日 17:22 null -> ../devices/pseudo/mm@0:null
lrwxrwxrwx 1 root other 27 6月 12日 17:22 zero -> ../devices/pseudo/mm@0:zero
$ cd /devices/pseudo
$ ls -l mm@0:*
crw----- 1 root sys 13, 3 6月 12日 17:22 mm@0:allkmem
crw-r----- 1 root sys 13, 1 6月 12日 17:22 mm@0:kmem
crw-r----- 1 root sys 13, 0 6月 12日 17:22 mm@0:mem
crw-rw-rw- 1 root sys 13, 2 6月 16日 16:11 mm@0:null
crw-rw-rw- 1 root sys 13, 12 6月 16日 16:08 mm@0:zero
$
```

3.3.4 链接文件

UNIX系统提供一种机制，使之能够采用不同的文件名引用同一数据或程序，也即把同一数据或程序赋予不同的文件名，这种文件称做链接文件，也称硬链接文件。当调用不同目录中的同名程序时，或以不同的名字调用同一个程序时，执行的是同一程序副本。

采用链接文件的好处是，只需在文件系统中保存一份数据或程序副本，多个文件名均指向同一数据内容，更新任何一个文件，即可反映到其他文件中。既能节省磁盘存储空间，又可保证数据的一致性以及文件存储位置的灵活性。其中的一个例子就是vi编辑器。依据调用时的文件名，vi能够以5种不同的模式运行。这5种运行模式分别对应于ex、edit、vi、view和vedit等5个程序文件。

作为单独的文件，上述5个程序文件存在于文件系统的同一目录位置，只不过与之对应的文件数据（存储在磁盘中的数据块）只有一个副本。通过以下两个命令，即可验证上述说法：

```
$ ls -li /usr/bin/vi
525 /usr/bin/vi
$ find /usr/bin -inum 525 -exec ls -li {} \+
525 -r-xr-xr-x 5 root bin 193968 Sep 14 2007 /usr/bin/edit
525 -r-xr-xr-x 5 root bin 193968 Sep 14 2007 /usr/bin/ex
525 -r-xr-xr-x 5 root bin 193968 Sep 14 2007 /usr/bin/vedit
525 -r-xr-xr-x 5 root bin 193968 Sep 14 2007 /usr/bin/vi
525 -r-xr-xr-x 5 root bin 193968 Sep 14 2007 /usr/bin/view
$
```

由于上述5个文件具有相同的信息节点号（525），且其链接计数（第3列）为5，说明这5个文件是采用硬链接的方式链接在一起的，因而拥有同一数据副本，但具有5个不同的文件名字。注意，非链接文件（包括符号链接文件）的链接计数为1。在UNIX系统中，信息节点与数据是一一对应。如果文件的信息节点号相同，其引用的就是同一个信息节点，因而拥有同一数据副本，如图3-5所示。

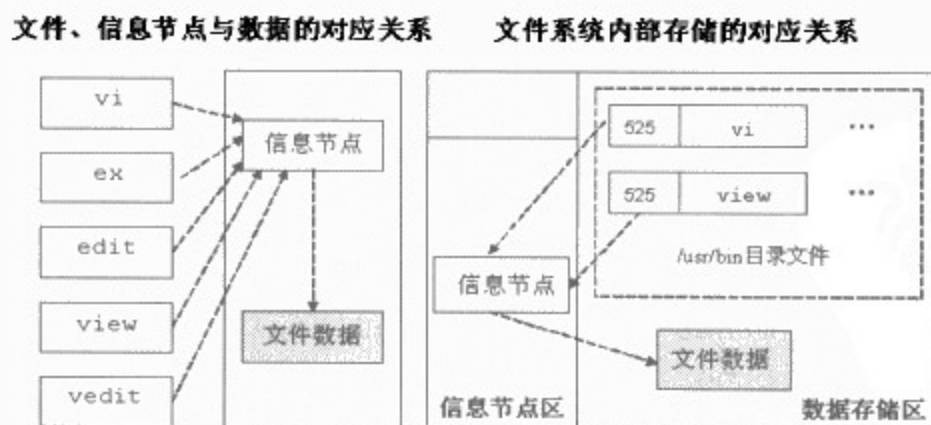


图3-5 硬链接文件的数据引用关系

如想创建硬链接文件，可以使用ln命令或link()系统调用实现。创建硬链接文件时，文件引用的是同一信息节点号和文件数据，只是在同一目录或不同目录中增加一个新的文件名而已。注意，硬链接文件的唯一局限是链接的两个文件必须位于同一个物理文件系统中。

在UNIX系统中，对于同一个数据内容，出于应用的需要，经常会赋予不同的文件名。例如，/etc/init.d目录中的Shell脚本文件主要用于启动或关闭系统守护进程。同一个文件又可用于不同的运行级，因而需要放置到不同的目录中，如放置到/etc/rc0.d、/etc/rc2.d或/etc/rc3.d等目录中。

例如，假定我们制作了一个Oracle数据库的启动与关闭脚本，其名字为oracle，存放在/etc/init.d目录中。为了把文件的副本分别存放到/etc/rc3.d和/etc/rc0.d两个目录中，我们可以使用下列ln命令，创建两个链接文件：

```
# cd /etc/init.d
# ln oracle ../rc3.d/S90oracle
# ln oracle ../rc0.d/K10oracle
# ls -l oracle
-rwxr--r-- 3 root sys 856 Jun 19 2009 oracle
# ls -l /etc/rc3.d/S90oracle
-rwxr--r-- 3 root sys 856 Jun 19 2009 /etc/rc3.d/S90oracle
# ls -l /etc/rc0.d/K10oracle
-rwxr--r-- 3 root sys 856 Jun 19 2009 /etc/rc0.d/K10oracle
#
```

如此一来，我们就建立了三个文件，三个文件共享同一个数据副本。也就是说，三个文件名均指向同一数据内容。

3.3.5 符号链接文件

从System V Release 4.0开始，UNIX系统提供一种新的文件链接机制，以使用户能够跨越不同的物理文件系统建立链接文件，这种新的链接文件称做符号链接文件。为了创建符号链接文件，可以使用“ln -s”命令或symlink()系统调用实现。

与硬链接文件的实现方式不同，符号链接文件的本身也是一种数据文件，而且是一个单独的文件，只不过文件中的内容并非真正的数据，而是一个指向目的文件（或目录）的路径名而已。由此可见，利用符号链接机制，将会创建新的文件名和新的信息节点。而且，符号链接文件也会拥有自己的数据块，其中包含的就是其引用的目的文件（或目录）的完整路径名，如图3-6所示。

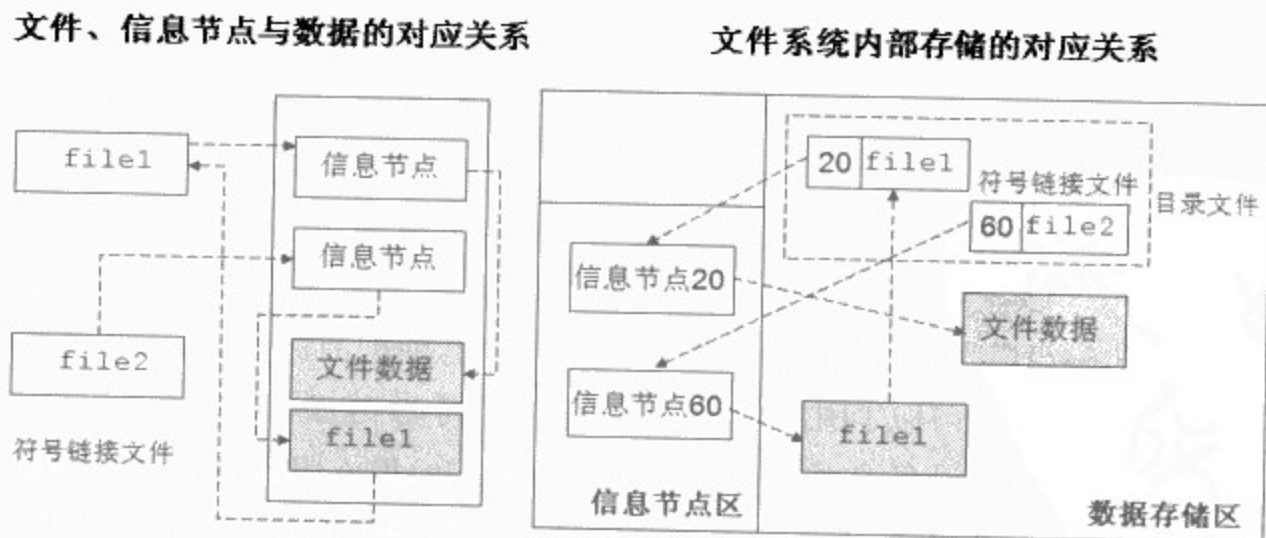


图3-6 符号链接文件的数据引用关系

当引用符号链接文件时，系统首先需要打开符号链接文件本身，然后再根据文件中的内容，打开其指向的文件。因此，当操作一个符号链接文件时，系统将会按照符号链接文件中的路径名进行二次检索，直至找到实际的文件。此外，符号链接文件与其链接的目的文件不必位于同一物理文件系统中，其引用的文件甚至也可以不存在。符号链接文件可用于引用任何文件，尤其是目录，但硬链接不能链接目录，以免出现目录树引用的死循环。

采用符号链接文件的最大好处是能够确保目录文件结构的兼容性。在开发UNIX System V Release 4.0时，UNIX系统的目录结构做了较大的调整，文件组织也发生了很大的变化。例如，原来位于根目录中的/bin和/lib分别移至/usr/bin、/usr/sbin和/usr/lib等目录中。/etc目录中的系统维护命令均移至/sbin或/usr/sbin等目录中。各种系统文件也分门别类，移至不同的目录。如/etc/hosts等与TCP/IP有关的文件均移至/etc/inet目录中。

当需要调整文件系统的目录结构或文件位置时，为了保持文件系统的兼容性（这也是产生符号链接文件的主要原因之一），可以采用符号链接文件的形式，继续保持原有的目录结构或文件位置。例如，Solaris UNIX系统把/bin目录迁移至/usr/bin，但仍然保持/bin的目录位置，只是把这个目录变成了符号链接文件：

```
$ ls -l /bin
lrwxrwxrwx 1 root root          9 Jun 12 17:21 /bin -> ./usr/bin
$
```

利用符号链接文件，还可以照顾用户以往的上机习惯，把之前常用的命令名赋予新增的命令，实现命令名的借用或间接引用。例如，halt、poweroff与reboot引用的实际上是同一个程序。

在符号链接文件中，文件大小字段中的数值恰好等于符号链接文件引用的目的文件的路径名的长度（观察第5列与“->”符号后的路径名长度）。

```
$ ls -l /etc/reboot
lrwxrwxrwx 1 root root      16 Jun 12 17:21 /etc/reboot -> ../usr/sbin/halt
$
```

UNIX中也存在3个与符号链接文件有关的系统调用，其目的是访问符号链接文件本身及其引用的目的文件或目录的相关信息。这些系统调用如下：

- readlink() 用于读取符号链接文件引用的目的文件或目录的路径名。这些信息存储在符号链接文件的数据块中。

- lstat() 其功能类似于stat系统调用，除用于获取目的文件的属性信息外，还可用于获取符号链接文件本身的有关信息。

- lchown() 类似于chown系统调用，既可用于修改目的文件的属主属性，也可用于修改符号链接文件本身的属主属性。

在实际应用中，链接文件具有重要的用途。例如，在构建互为备份的双机系统中，通常采用两个系统共享同一个磁盘阵列的形式，实现数据的共享。在这种实现方式中，有的HA软件要求比较高，如要求磁盘阵列的设备名必须一致。而由于种种原因，这一限制可能很难达到要求。为了解决这个问题，便可以利用ln命令建立链接或符号链接文件。

假定主系统磁盘阵列的设备文件名分别为/dev/dsk/c4t0d0s6和/dev/rdsk/c4t0d0s6，备份系统磁盘阵列的设备文件名分别为/dev/dsk/c2t1d0s6和/dev/rdsk/c2t1d0s6。为了使备份系统与主系统的磁盘阵列设备文件名保持一致，我们可以在备份系统上执行下列命令：

```
# ln -s /dev/dsk/c2t1d0s6 /dev/dsk/c4t0d0s6
# ln -s /dev/rdsk/c2t1d0s6 /dev/rdsk/c4t0d0s6
#
```

注意：如果ln命令行中的源文件（第一个文件）本身即为符号链接文件，则应找出其原始文件，使用原始文件名替换上述命令行中的源文件。例如，假设其原始的文件名分别为/devices/

pci@9,700000/lpfc@4/sd@1,0:g和/devices/pci@9,700000/lpfc@4/sd@1,0:g,raw, 则上述命令应改为:

```
# ln -s /devices/pci@9,700000/lpfc@4/sd@1,0:g /dev/dsk/c4t0d0s6
# ln -s /devices/pci@9,700000/lpfc@4/sd@1,0:g,raw /dev/rdsk/c4t0d0s6
#
```

3.3.6 管道文件

UNIX系统存在两种管道: 即普通管道(简称管道)和管道文件。普通管道是一个可用文件描述符标识和存取的数据缓冲区, 管道内的数据按先进先出的方式处理。普通管道通常是在程序中利用pipe(2)系统调用建立的。当程序执行结束后, 管道也就自动消失了。

管道文件与普通管道的功能基本相同, 只是其创建的方式不同。管道文件是通过下列mknod命令或mknod()系统调用创建的。而且作为一个特殊文件, 存在于文件系统中, 故管道文件也称命名的管道(named pipe):

```
# mknod pipefile p
```

普通管道是一种进程之间的通信机制, 而管道文件则是一种特殊文件。管道文件用于缓存接收到的数据, 同时供从管道文件中读取数据的进程按照先进先出(FIFO)的方式读取数据。尽管管道文件具有文件名和信息节点, 但这个特殊文件并不拥有任何数据。

在下面的例子中, FIFO就是一个管道文件(第一列文件属性中的首字符p即为管道文件的标志)。

```
$ ls -l /etc/cron.d
total 6
prw----- 1 root    root          0 Jun 15 18:52 FIFO
-rw-r--r-- 1 root    sys           40 Aug 25  2008 at.deny
-rw-r--r-- 1 root    sys           40 Aug 25  2008 cron.deny
-rw-r--r-- 1 root    sys           17 Jan 22  2005 queuedefs
$
```

3.4 文件的安全保护机制

UNIX是一个多用户、多任务的操作系统, 为了保证系统和信息的安全, 防止用户间未经许可擅自访问他人的文件等, UNIX系统从信息的基本组织单位——文件出发, 把用户分为三类: 文件属主、同组用户和其他用户(也即所有用户)。

对于任何一个文件, 可以针对三类用户分别赋予一定的访问权限。这些访问权限分为读、写和执行。表3-2描述了文件的三种基本访问权限及其意义。

表3-2 文件的三种基本访问权限

访问权限	简单说明
r (读)	如果文件具有读许可, 相应的用户可以读文件, 如显示文件内容, 复制文件等, 但不能修改文件。如果允许用户进入某个目录(cd命令), 列举其中的文件, 至少应赋予用户“读”目录的访问权限

（续表）

访问权限	简单说明
w（写）	如果文件具有写许可，相应的用户可以读、写文件，包括显示文件内容，复制、修改、移动和删除文件等。对于目录而言，如果允许用户在其中创建新文件和删除文件，必须赋予用户“写”目录的访问权限
x（执行）	如果文件具有执行许可，相应的用户可以运行文件（如程序文件）。对于目录而言，如果允许用户访问其中的任何子目录，必须赋予用户“执行”目录的访问权限

通常，任何用户都可能会允许其他用户读自己的文件，但大多不会允许别人对文件做任何修改。对于普通的程序文件而言，也可能会允许任何人都能执行。UNIX系统的文件目录安全机制使用户能够控制文件和目录的访问权限。

3.4.1 显示文件的访问权限

为了了解文件的访问权限及其属性，可以使用带有“-l”选项的ls命令，按文件名的字符顺序以长格式显示每一个文件的各种属性信息，如图3-7所示。

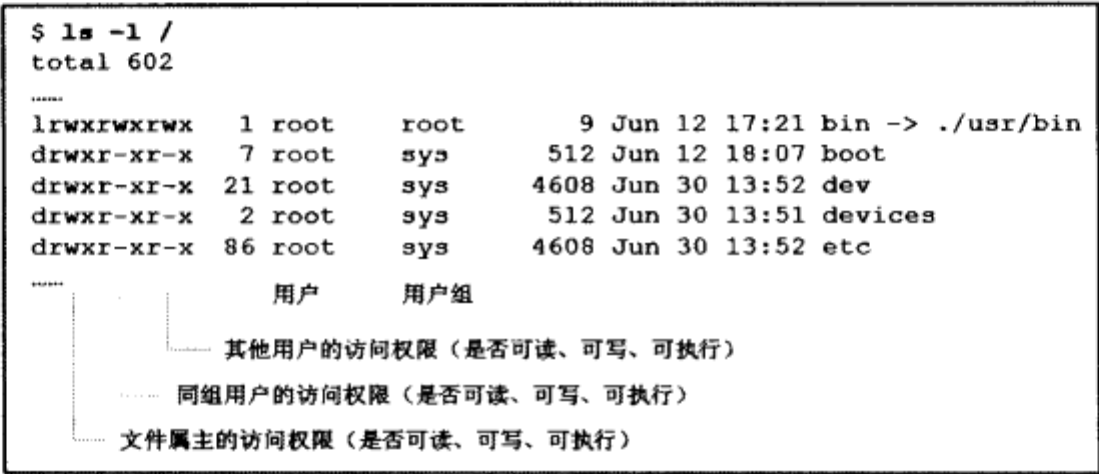


图3-7 显示文件目录的访问权限

ls命令输出第一列中的第一个字符表示文件的类型。后面9个字符表示文件或目录的访问权限。这些访问权限可分为三组，分别对应三组用户：文件属主、同组用户和其他用户。前三个字符为一组，表示文件属主对此文件的访问权限；中间三个字符为一组，表示与文件属主同组的用户对此文件的访问权限；最后三个字符为一组，表示其他所有用户对此文件的访问权限。每一组的三个字符依次为r、w和x，分别表示读、写和执行。

如果r、w或x存在，表示相应的用户分别具有读、写或执行文件的访问权限。下面，我们以/etc目录和/etc/profile文件为例予以说明：

```
$ ls -ld /etc
drwxr-xr-x 86 root  sys    4608 Jun 15 18:57 /etc
$ ls -l /etc/profile
-rw-r--r--  1 root  sys    712 Aug 25  2008 /etc/profile
$
```

在上述例子中，/etc目录的访问权限是“rwxr-xr-x”，表示每个人都能够读和执行这个目录，但只有目录的属主root才能写这个目录。/etc/profile文件的访问权限是“rw-r-r-”，表示文件属主root能够读写这个文件，同组用户和其他用户只能读此文件，但任何人（包括文件属主）

都不能执行这个文件。

3.4.2 修改文件的访问权限

为了修改文件或目录的访问权限，可以使用**chmod**命令。修改文件或目录访问权限的前提是，用户必须是文件或目录的属主，具有修改文件访问权限的权力，或者是超级用户。**chmod**命令的语法格式如下：

```
chmod permissions dir-or-file
```

其中，**permissions**表示新的文件访问权限，**dir-or-file**是准备修改其访问权限的文件或目录。

在设置文件或目录的访问权限时，可以对现有的访问权限进行增减性的调整，也可以直接设置。下面简单说明怎样在现有文件访问权限的基础上进行修改。

(1) 使用一个或多个字符表示用户的类型：

- **u**（表示文件属主）；
- **g**（表示同组用户）；
- **o**（表示其他用户）；
- **a**（表示所有用户）。

(2) 使用加号“+”和减号“-”分别表示增加或取消相应的访问权限。

(3) 使用一个或多个字符表示实际的访问权限：

- **r**（表示读）；
- **w**（表示写）；
- **x**（表示执行）。

在下面的**chmod**命令中，“**g+w**”表示增加同组用户写**script**目录的访问权限。

```
$ cd /export/home/gqxing
$ ls -ld script
drwxr-xr-x  2 gqxing  other          512 Jun 29 17:13 script
$ chmod g+w script
$ ls -ld script
drwxrwxr-x  2 gqxing  other          512 Jun 29 17:13 script
$
```

从最后一个**ls**命令的输出结果可以看出，执行“**chmod g+w script**”命令之后，**script**目录相应于同组用户写目录的权限标志已由“-”变为“w”。

为了取消其他用户读和执行同一目录的访问权限，可以使用下列**chmod**命令：

```
$ ls -ld script
drwxr-xr-x  2 gqxing  other          512 Jun 29 17:13 script
$ chmod o-rx script
$ ls -ld script
drwxr-x---  2 gqxing  other          512 Jun 29 17:13 script
$
```

执行上述**chmod**命令之后，表示其他用户读和执行目录的权限标志已由“**r**”和“**x**”全部变为“-”。

通常，普通文件的默认访问权限不包括执行权限。因此，在新建了一个**settcp**脚本文件之后，为了确保文件属主能够执行脚本文件，可以使用下列**chmod**命令：

```
$ ls -l settcp
-rw-r--r-- 1 gqxing other 1664 Jun 30 12:53 settcp
$ chmod u+x settcp
$ ls -l settcp
-rwxr--r-- 1 gqxing other 1664 Jun 30 12:53 settcp
$
```

如果想要增加或取消所有用户的访问权限，可以使用**chmod**命令的“a+”或“a-”选项。例如，为了使所有的用户都能够执行**settcp**文件，可输入下列**chmod**命令：

```
$ ls -l settcp
-rw-r--r-- 1 gqxing other 1664 Jun 30 12:53 settcp
$ chmod a+x settcp
$ ls -l settcp
-rwxr-xr-x 1 gqxing other 1664 Jun 30 12:53 settcp
$
```

在上述输出信息中，访问权限字段（第一列后9个字符）中的三个“x”表示所有用户均可以执行**settcp**文件。

此外，还可以使用星号“*”通配符，同时修改多个文件的访问权限。例如，为了设置当前目录中所有文件的访问权限，使得除了文件属主能够写之外，其他所有用户均不能修改这些文件，可以使用下列**chmod**命令：

```
$ ls -l
total 14
-rwxrwxrwx 1 gqxing other 1620 Jun 30 12:52 setftp
-rwxrwxrwx 1 gqxing other 2760 Jun 30 12:53 setnfs
-rwxrwxrwx 1 gqxing other 1664 Jun 30 12:53 settcp
$ chmod go-w *
$ ls -l
total 14
-rwxr-xr-x 1 gqxing other 1620 Jun 30 12:52 setftp
-rwxr-xr-x 1 gqxing other 2760 Jun 30 12:53 setnfs
-rwxr-xr-x 1 gqxing other 1664 Jun 30 12:53 settcp
$
```

3.4.3 设置文件的访问权限

上一节讨论了怎样使用**chmod**命令，在文件现有访问权限的基础上进行调整的情况。这种文件访问权限设置法称为相对权限设置法，这一节将要讨论的方法可以称做绝对权限设置法——使用**chmod**命令和数字代码，直接设置文件的访问权限。**chmod**命令的语法格式如下：

```
chmod numcode dir-or-file
```

其中，**numcode**是一个数字代码，用于表示文件的访问权限。**dir-or-file**是文件或目录的名字。

表示文件访问权限的数字代码由三位数字组成，分别对应于文件属主、同组用户和其他用户。例如，下列**chmod**命令设置的访问权限表示，**setftp**文件的属主能够读、写和执行**setftp**文件，其他用户只有读和执行文件的权限。


```
$ ls -l setftp
-rw-r--r--  1 gqxing  other          1620 Jun 30 12:52 setftp
$ chmod 755 setftp
$ ls -l setftp
-rwxr-xr-x  1 gqxing  other          1620 Jun 30 12:52 setftp
$
```

表3-3解释了数字代码755表示的访问权限，以及755数字代码的由来。表中后三列每列对应一类用户。

表3-3 setftp文件访问权限的设定

访问权限	文件属主	同组用户	其他用户
读	100 (4)	100 (4)	100 (4)
写	010 (2)	000 (0)	000 (0)
执行	001 (1)	001 (1)	001 (1)
全部访问权限	7	5	5

表示访问权限的字符串“rwx”可以看做三个二进制数据，如果“r”存在，表示相应数据位为1，写成二进制的数就是“100”，也就是十进制的4。同样，“w”对应的二进制数是“010”，即十进制的2。“x”对应的二进制数是“001”，也即十进制数的1。

为了设置读文件的访问权限，可在适当的行列交叉位置加一个4。为了设置写文件的访问权限，可在适当的行列交叉位置加一个2。同样，为了设置执行文件的访问权限，可在适当的行列交叉位置加一个1。把三列的三行数值分别加在一起，然后再把这三个数据组合起来，就是一个表示文件访问权限的完整数字代码。

例如，为了使所有的用户都能够读、写和执行setnfs文件，可以使用下列命令。其中，数字代码777是能够设置的最大文件访问权限。

```
$ ls -l setnfs
-rw-r--r--  1 gqxing  other          2760 Jun 30 12:53 setnfs
$ chmod 777 setnfs
$ ls -l setnfs
-rwxrwxrwx  1 gqxing  other          2760 Jun 30 12:53 setnfs
$
```

表3-4解释了上述访问权限数字代码的意义及由来。

表3-4 设置setnfs文件访问权限的数字代码

访问权限	文件属主	同组用户	其他用户
读	4	4	4
写	2	2	2
执行	1	1	1
全部访问权限	7	7	7

类似于相对权限设置法，在采用绝对权限设置法时也可以使用星号“*”通配符，对选定的文件设置同一访问权限。例如，我们可以使用chmod命令，统一设置当前目录下所有文件的

访问权限。

```
$ ls -l
total 14
-rw-r--r--  1 gqxing  other      1620 Jun 30 12:52 setftp
-rw-r--r--  1 gqxing  other      2760 Jun 30 12:53 setnfs
-rw-r--r--  1 gqxing  other      1664 Jun 30 12:53 settcp
$ chmod 755 *
$ ls -l
total 14
-rwxr-xr-x  1 gqxing  other      1620 Jun 30 12:52 setftp
-rwxr-xr-x  1 gqxing  other      2760 Jun 30 12:53 setnfs
-rwxr-xr-x  1 gqxing  other      1664 Jun 30 12:53 settcp
$
```

上述例子中的两个ls命令解释了使用chmod命令前后文件访问权限的变化。

使用绝对权限设置的最大好处是，无需知道文件当前的访问权限，只需按照用户自己的意愿直接设置即可。

3.4.4 其他访问权限设置

1. 默认访问权限

在创建任何目录或文件时，系统通常都会赋予新建目录或文件一个访问权限，如果没有设置用户掩码（user mask），系统默认的目录访问权限为777（对应于二进制的111 111 111），文件访问权限为666（对应于二进制的110 110 110）。如果设置了用户掩码，目录和文件的实际访问权限是由系统访问权限与用户掩码经过逻辑异或运算后的结果。

用户掩码通常包含三组八进制的数字，其取值范围为0~7：

- 第一组数字用于设置用户自己的访问权限；
- 第二组数字用于设置同组用户的访问权限；
- 第三组数字用于设置其他用户的访问权限。

通常，系统设置的用户掩码为022（对应于二进制的000 010 010）。经过逻辑异或运算之后，可知新建目录的访问权限为755（对应于二进制的111 101 101），新建文件的访问权限为644（对应于二进制的110 100 100）。

例如，当创建一个新文件时，系统自动设置的访问权限如下：

```
-rw-r--r--
```

当创建一个新目录时，系统自动设置的访问权限如下：

```
drwxr-xr-x
```

用户掩码是由umask命令设置的（在系统提供的/etc/profile初始化文件中，umask命令通常是一个不可或缺的设置命令。在用户注册之后，通过执行/etc/profile文件，即可实现用户掩码的设置，从而达到设置目录和文件访问权限的目的）。umask命令的语法格式如下：

```
umask [-S] [nnn]
```

其中，nnn是一个三位的八进制的数值，表示用户掩码。注意，这个用户掩码与chmod命令中的数字代码意义恰好相反。在chmod命令中，八进制的数字代码777意味着任何人都具有读、

写和执行文件的访问权限，而在umask命令中，数字代码777意味着任何人都没有访问权限，包括文件属主本人也是如此。

为了查询系统当前的用户掩码设置，可以直接输入umask命令，示例如下：

```
$ umask
022
$
```

当需要使用umask命令设置文件的默认访问权限时，需要确定究竟应采用什么用户掩码值。具体做法是，从系统默认的文件访问权限666中减去文件的实际访问权限对应的八进制数值，余数即可用于umask命令中的用户掩码。例如，假定我们想把文件的访问权限设置为664（rw-rw-r--），而666与664的差为002，故可把002用做umask命令的参数。如果运行“umask 000”命令，意味着把文件的访问权限设置为666（rw-rw-rw-）。

也可以把二进制的“000 000 000”看做文件访问权限的基础，其中0表示允许，1表示禁止。如果想屏蔽某一访问权限，可把相应的二进制数值置1，最后再转换成八进制的数字代码即可。例如，如果用户期望自己能够读写新建的文件，同组和其他用户只能读文件，则其二进制数字代码应为“000 010 010”，转换为八进制数字代码即为“022”。因此，可以使用“umask 022”命令设置用户掩码，从而设置文件的默认访问权限。

2. 设置特殊的访问权限

前面我们已经讨论了如何设置文件的访问权限，任何用户均可用以保护自己的文件。还有一些特殊的文件访问权限，可由超级用户进行设置。UNIX系统中的许多管理文件，只有超级用户才有权修改。但有时也需要普通用户能够像超级用户一样做必要的修改。例如，管理用户账号和密码的/etc/passwd和/etc/shadow文件，普通用户是不能修改的。但用户经常需要修改自己的密码，而普通用户又不能修改这些文件，总不能每次都经过系统管理员。

为此，UNIX系统采用了实际用户ID（实际用户组ID）和有效用户ID（有效用户组ID）的概念。通过设置有效用户ID，使用户能够临时地以超级用户的身份执行某些特权命令。例如，任何用户在使用passwd命令修改密码期间，其有效用户ID暂时变为超级用户ID，因而可以访问shadow等文件，修改自己的密码。一旦退出passwd命令，用户的有效ID又恢复原状。

为了设置某一文件的有效用户ID或有效用户组ID，超级用户可以分别使用下列命令：

```
# cd /export/home/gqxing/bin
# ls -l autoroot
-rwxr-xr-x  1 root      other      22628 Aug 15  2007 autoroot
# chmod u+s autoroot
# chmod g+s autoroot
# ls -l autoroot
-r-sr-sr-x  1 root      other      22628 Aug 15  2007 autoroot
# autoroot
#
```

在设置了有效用户ID之后，必须先由超级用户执行一次相应的命令，才能使上述设置发挥作用。当普通用户执行相应的命令时，其有效用户ID才能变为超级用户的有效用户ID。注意，有效用户ID仅在执行过程中才能生效，因此，只有对命令或程序文件设置有效用户ID才有实际的意义。

第4章 文件和目录操作

文件和目录操作是每一个系统都必不可少的基本功能。UNIX系统提供大量的命令和工具，用于处理文件与目录。本章主要介绍UNIX系统中的各种文件与目录操作命令，详细说明怎样创建、显示、移动、删除、复制以及维护文件与目录。在本章的后面，我们还将介绍若干有用的实用程序，如find、diff、grep和sort等。

4.1 创建文件

尽管没有明显的专门命令用于创建新文件，但实现创建新文件的方法有许多种。

其中第一种方法是利用touch命令创建一个新的空文件。touch命令的本意是以当前（或指定）时间更新给定文件的访问与修改时间。如果指定的文件不存在，则创建一个新的空文件，例如：

```
$ touch emptyfile
$
```

而第二种方法相对更简单，即借用重定向符号“>”，创建一个新的空文件。

```
$ > emptyfile
$
```

第三种方法是采用echo命令创建一个含有少量数据的新文件。echo命令的本意是显示提示或说明信息。这里借用echo命令，将其标准输出重定向到一个文件中，从而创建一个新文件。

```
$ echo "Only one line in file" > newfile
$
```

第四种方法是用cat命令创建一个具有多行数据的新文件。cat命令主要用于显示文件的内容。下面的例子是借用cat命令读取标准输入的特点，直接在键盘上输入数据，从而创建一个拥有三行数据的新文件（其中，Ctrl-D是UNIX系统中的文件结束符）。

```
$ cat > myfile
The text I am typing will be stored in "myfile".
Press RETURN at the end of each line.
When finished, hold down the Ctrl key and press D.
Ctrl-D
$
```

此外，还可以使用vi等文本编辑器创建和编辑文件，详见第5章“编辑文件”。实际上，创建文件的方法还有很多，熟悉了UNIX系统之后，读者还会找出更多的方法。

4.2 显示文件列表

4.2.1 使用ls命令显示文件列表

了解系统中都有什么文件，了解自己当前目录下都有哪些文件，也许是用户最常见的动作了。为此，可以使用ls命令。ls命令的语法格式简写如下：

```
ls [options] [dir-or-file]
```

ls命令具有很多选项，表4-1给出了部分常用的选项。

表4-1 ls命令的部分常用选项

选项	简单说明
-a	列出指定（或当前）目录下的所有文件，包括以句点“.”作为起始字符的隐藏文件
-b	当文件名包含不可打印的特殊字符时，以八进制数字形式列出文件的名字
-d	在使用ls命令列举文件时，如果指定的参数是一个目录，仅列出目录的名字，而不是列出目录下的文件。“-d”选项经常与“-l”选项一起使用，以便了解目录的属性信息
-h	以KB、MB和GB的形式显示文件的大小（这个选项应与“-l”或“-s”等选项一同使用）
-i	对于每一个文件，在第一列中显示其信息节点号
-l	以每行一个文件的长格式显示文件的类型、访问权限、链接数、用户属主、用户组、文件大小、最后修改时间和文件名等信息。如果是特殊文件，文件大小字段分为两个字段，分别表示设备的主次设备号。如果是一个符号链接文件，则以“filename → 被引用文件的路径名”的形式给出文件名字段
-r	以文件名反向字符排序的顺序显示文件列表
-R	递归地列出指定（或当前）目录及其子目录下的所有文件
-s	显示分配给文件的数据块（1024字节）的数量，也即文件占用的数据块数量，而非文件的实际大小。因此，文件大小相近的文件，“-s”选项给出的结果完全可能是一样的

例如，为了列出当前目录下的文件，最简单的ls命令形式如下：

```
$ ls
emptyfile  myfile      newfile
$
```

通常，ls命令会按照文件名的字符顺序列出当前目录下的所有文件。上述例子中的ls命令未加任何选项，也没有明确地指定目录或文件参数，故默认为当前目录。输出结果说明当前目录下只有三个刚才创建的文件。如果想了解其他某个特定目录下都有什么文件，可在ls命令后面明确地指定目录名，例如：

```
$ ls /
bin      Documents  lib      opt      system
boot     etc        lost+found platform  tmp
Desktop  export     Mail     proc     usr
dev      home      mnt      rmdisk   var
devices  kernel    net      sbin     vol
$
```

上述ls命令按照文件名的字符顺序列出了根目录下的所有文件。读者可能已经发现，使用ls命令而不加任何选项，系统仅仅列出文件的名字，至于这些文件究竟是目录、普通文件、特殊文件、管道文件，还是链接文件，则不得而知。为能了解更多的信息，可使用“-l”等选项，例如：

```
$ ls -l
总数 4
-rw-r--r--  1 gqxing  other          0  6月 15日 15:45 emptyfile
-rw-r--r--  1 gqxing  other        138  6月 15日 15:46 myfile
-rw-r--r--  1 gqxing  other         22  6月 15日 15:45 newfile
$
```

使用“-l”选项，ls命令将会按照一行9列的形式逐行显示每个文件的属性。其中第一列包含10个字符，第一个字符表示文件的类型。常见的文件类型字符概述如下：

- - 表示相应的文件是一个普通文件；
- d 表示相应的文件是一个目录；
- l 表示相应的文件是一个符号链接文件；
- b 表示相应的文件是一个块特殊文件；
- c 表示相应的文件是一个字符特殊文件；
- p 表示相应的文件是一个管道（FIFO）文件；
- s 表示相应的文件是一个套接字文件。

其余9个字符可分为三组，分别表示文件属主、同组用户和其他用户对相应文件的访问权限，详见第3章“文件系统基础知识”的讨论。第二列是一个数字，表示文件的链接数。如果此数字大于1，说明同一文件数据存在多个文件名字。第三列为用户名，表示文件的属主，说明文件归谁拥有。第四列是文件属主所在用户组的名字。第五列是文件以字节为单位的大小。第六、第七和第八3列给出的是文件最后一次修改的日期和时间。最后一列才真正是文件的名字，如图4-1所示。

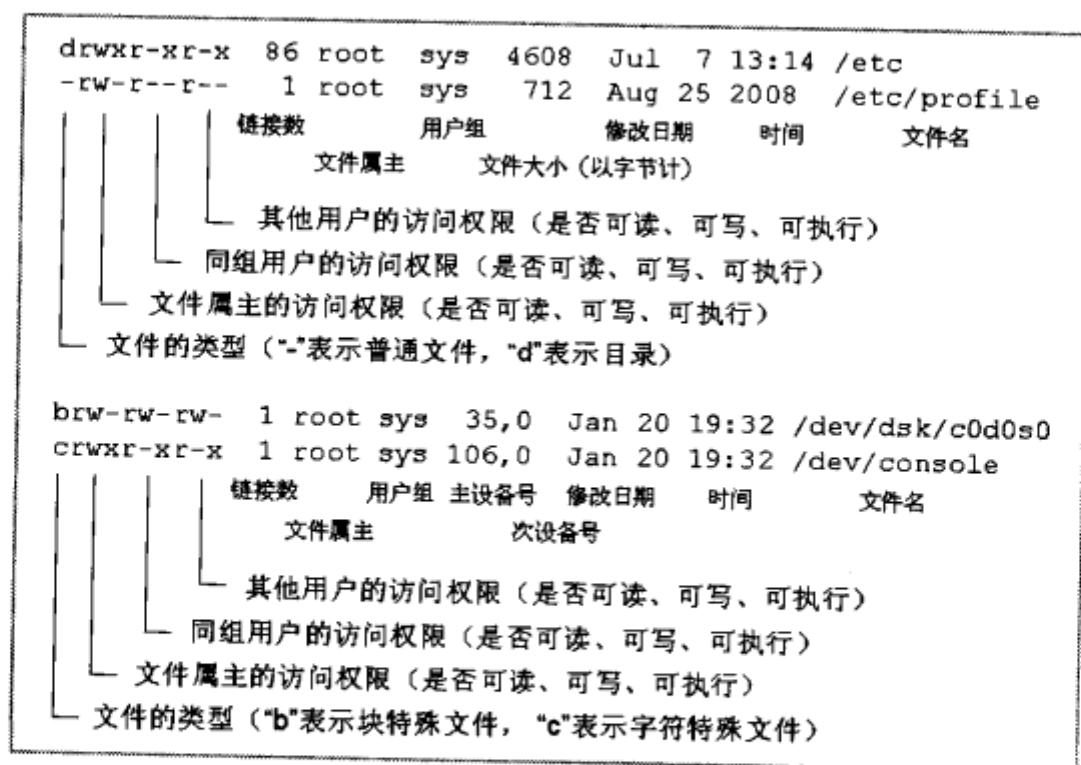


图4-1 文件属性

4.2.2 利用通配符显示文件

在第2章“命令行基础知识”的“元字符与文件名生成”一节中，我们已经介绍了通配符的作用及其与文件名之间的关系。当需要处理一组具有某种共同属性的文件时，可以利用通配符实行模式匹配，生成一个具有同一属性的文件列表。例如，在UNIX系统中，C程序大多以“.c”作为文件名后缀。为了显示当前目录下的所有C程序，可以使用下列命令：

```
$ ls -l *.c
-rw-r--r-- 1 gqxing other 25852 Jun 18 15:31 atmcom.c
-rw-r--r-- 1 gqxing other 26282 Jun 18 15:31 atmmon.c
-rw-r--r-- 1 gqxing other 52125 Jun 18 15:33 handler.c
-rw-r--r-- 1 gqxing other 30625 Jun 18 15:32 listener.c
$
```

在上述命令中，星号“*”就是一个通配符，能够匹配任何字符或字符串。实际上，可以利用的通配符还有问号“?”（用以匹配任何一个字符），以及字符集“[...]”（用以匹配字符集中的任何一个字符）等。

如前所述，如果不提供文件名参数，ls命令将会列出当前目录下的所有文件，而星号“*”通配符又能够匹配任何文件，如果使用下列两个ls命令，其输出结果会有什么不同呢？

```
ls -l
ls -l *
```

上述的第一个ls命令中没有指定任何目录或文件参数，因此，系统将会把当前目录作为默认参数，列出当前目录下的所有文件。第二个ls命令使用星号“*”通配符作为目录或文件参数，而此处的星号“*”通配符则表示当前目录下的所有文件。因此，针对第二个ls命令，Shell将会使用当前目录下的每一个文件作为参数提供给ls命令。如果当前目录下只有普通文件，上述两个命令将会产生同样的输出结果。但是，如果当前目录下含有任何子目录，第一个ls命令只是列出子目录的名字，而第二个ls命令不仅会列出子目录的名字，还将列出子目录下的所有文件。示例如下：

```
$ cd /var/spool/cron
$ ls -l
total 4
drwxr-xr-x 2 root sys 512 Jun 18 15:19 atjobs
drwxr-xr-x 2 root sys 512 Jun 12 17:43 crontabs
$ ls -l *
atjobs:
total 6
-r-Sr--r-- 1 root root 2933 Jun 18 15:19 1245330000.a
crontabs:
total 10
-rw----- 1 root sys 190 Aug 25 2008 adm
-r----- 1 root root 452 Aug 25 2008 lp
-rw----- 1 root sys 482 Jun 12 17:42 root
-rw----- 1 root sys 308 Aug 25 2008 sys
-r----- 1 root sys 404 Jun 12 17:40 uucp
$
```

在使用星号“*”通配符的情况下，为了避免列出子目录中的文件，可以使用“-d”选项。示例如下：

```
$ ls -ld *
drwxr-xr-x  2 root    sys      512 Jun 18 15:19 atjobs
drwxr-xr-x  2 root    sys      512 Jun 12 17:43 crontabs
$
```

4.2.3 显示隐藏文件

在UNIX系统中，如果文件名的第一个字符为句点“.”，如.profile和.sh_history，这样的文件称做隐藏文件。通常，ls命令不会列出隐藏文件，除非在ls命令中使用了“-a”选项。例如，下列ls命令可以列出当前目录中的隐藏文件：

```
$ ls -la
total 24
drwxr-xr-x  7 gqxing  root    512 Jun 29 17:13 .
drwxr-xr-x  6 root    root    512 Jun 22 13:23 ..
-rw-r--r--  1 gqxing  other  189 Jun 23 13:42 .profile
-rw-----  1 gqxing  other  980 Jun 29 17:13 .sh_history
.....
$
```

在UNIX系统中，隐藏文件通常位于文件列表的最前面，而在有的UNIX（或Linux）系统中，隐藏文件将会在不考虑第一个句点“.”的情况下同普通文件一样参与排序。文件名仅有一个句点“.”的文件表示当前目录，文件名为双句点“..”的文件表示父目录。其他隐藏文件通常均用于设置用户的工作环境。

实际上，还有另外一种隐藏文件。当由于某种原因，在命名文件时误输入了不可见的特殊字符时，ls命令可能无法正确地列出文件的名称，甚至根本就列不出文件的名称。在使用文件处理命令操作这样的文件时，可能还会显示“No such file or directory”或“cannot open file”等错误信息。例如，见到下面的信息，用户可能会感到莫明其妙——文件明明存在，怎么会没有名字呢？同一目录下怎么会有两个同名的文件呢？完全不符合UNIX文件系统的规定。两个file3文件的大小也不相同，明显不是同一个文件。而且，文件名的排序也不对。

```
$ ls -l
total 10
-rw-r--r--  1 gqxing  other  720 Jun 29 17:25
-rw-r--r--  1 gqxing  other  810 Jun 29 17:19 file2
-rw-r--r--  1 gqxing  other  290 Jun 29 17:31 file3
-rw-r--r--  1 gqxing  other  945 Jun 29 17:32 file3
-rw-r--r--  1 gqxing  other  716 Jun 29 17:20 file1
$ cat file2
cat: cannot open file2
$ ls -l file3
file3: No such file or directory
$
```

这是因为，当文件名中包含控制字符等不可打印的特殊字符时，UNIX系统通常不会有任何显示。为解决这一问题，可以使用ls命令的“-b”选项，以八进制数字的形式列出文件名中

不可见的特殊字符:

```
$ ls -lb
total 10
-rw-r--r--  1 gqxing  other      720 Jun 29 17:25 \001
-rw-r--r--  1 gqxing  other      810 Jun 29 17:19 \002file2
-rw-r--r--  1 gqxing  other      290 Jun 29 17:31 fi\005le3
-rw-r--r--  1 gqxing  other      945 Jun 29 17:32 fi\024le3
-rw-r--r--  1 gqxing  other      716 Jun 29 17:20 file1
$
```

此时,可以利用mv命令,重新命名文件,但不能采用问号“?”通配符,否则系统会误以为用户想把两个文件移至某个目录下。为此,可以借助于Ctrl-V键,在相应的字符位置上输入控制字符,即可解决文件的改名问题:

```
$ mv ^V^A file
$ mv ^V^Bfile2 file2
$ mv fi^V^Ele3 file3
$ mv fi^V^Tle3 file4
$ ls -l
total 10
-rw-r--r--  1 gqxing  other      720 Jun 29 17:25 file
-rw-r--r--  1 gqxing  other      716 Jun 29 17:20 file1
-rw-r--r--  1 gqxing  other      810 Jun 29 17:19 file2
-rw-r--r--  1 gqxing  other      290 Jun 29 17:31 file3
-rw-r--r--  1 gqxing  other      945 Jun 29 17:32 file4
$
```

注意: Ctrl-V是一个特殊的控制字符,在需要输入各种控制字符或其他特殊字符时,可以先按下Ctrl-V键,接着输入想要输入的任何控制字符,即可达到目的。从某种意义上说,Ctrl-V相当于一个转义符号,可用于引入任何控制字符。

4.2.4 递归显示目录与文件

利用ls命令的“-R”选项,可以递归地逐层显示当前(或指定)目录及其子目录中的所有文件。示例如下:

```
$ ls -lR /var/samba
/var/samba:
total 4
drwxr-xr-x  2 root    bin      512 Jun 12 17:24 /var/samba/locks
drwxr-xr-x  2 root    bin      512 Jun 12 17:24 /var/samba/log
/var/samba/locks:
total 0
/var/samba/log:
total 0
$
```

注意: “ls -l *”与“ls -lR”两个命令的意义和输出结果也不相同。前者仅仅涉及当前目录及其直接子目录两个目录层次中的所有文件。而后者将会遍历当前目录及其所有子目录(包括子目录下的子目录)中的所有文件。

4.3 显示文件内容

4.3.1 使用cat命令显示文件

传统的文件显示命令是cat，其语法格式简写如下：

```
cat [options] [file]
```

cat命令的作用是连续地显示文件的内容。不管终端屏幕的窗口有多大，cat总是从头到尾显示整个文件的所有内容。cat命令的缺点是：如果文件太大、太长，最终只能见到文件最后一部分内容，之前的内容将会快速闪过。例如，为了显示passwd文件，可以输入下列命令：

```
$ cat /etc/passwd
root:x:0:0:Super-User:/:/bin/ksh
daemon:x:1:1:/:
bin:x:2:2:/:usr/bin:
sys:x:3:3:/:
adm:x:4:4:Admin:/var/adm:
.....
$
```

cat命令的另一个常见用法是合并两个或多个文件，最终组成一个大的文件。例如，可以使用下列命令把分章编写的文件合并为一个完整的文件：

```
$ cat chap1 chap2 chap3 chap4 chap5 > UserGuide
$ ls
chap1 chap2 chap3 chap4 chap5 UserGuide
$
```

实际上，实现文件合并的方式有很多种。例如，如果想把两个文件合并到其中的一个文件，可以使用cat命令和I/O重定向(>>)功能，把第二个文件中的内容附加到第一个文件中：

```
cat file2 >> file1
```

其中，第二个文件file2的输出被重定向，并附加到第一个文件file1的后面，最终结果是把第二个文件的内容合并到第一个文件中。

注意：如果采用下列命令形式合并文件，将会清除file1中的数据内容，最终只是把file2的数据内容复制到file1中：

```
cat file1 file2 > file1
```

4.3.2 使用more命令分页显示文件

为了从头到尾一页一页地仔细阅读文件，可以使用more命令，逐页逐屏地显示整个文件的内容。more命令的语法格式简写如下：

```
more [options] [file]
```

例如，为了逐页显示/etc/profile文件，可以输入下列命令（其输出结果如图4-2所示）：

```
$ more /etc/bootrc
```

```

#
# ident "@(#)bootrc 1.7 03/06/04 SMI"
#
# Copyright 1996,2003 Sun Microsystems, Inc. All rights reserved.
# Use is subject to license terms.
#
# initial properties, set during installation
# NOTICE: bootpath and boot-args should be set by this point.
#=====
getprop bootpath bp
getprop boot-args bootargs
if .streq ( "${bootargs}"X , kernel/unixX )
    set bootargs
endif

set ba0 bogus_response
set cmd_err 0

# display current defaults
echo "          <<< Current Boot Parameters >>>"
Boot path: ${bp}
Boot args: ${bootargs}"
--More--(21%)

```

图4-2 more命令的输出结果

more命令在显示一页文件内容之后，屏幕窗口的左下角将会出现“--More--(n%)”信息，其中的“n%”表示已显示的数据内容占整个文件的百分比（如果文件很短，上述信息将不会出现）。在more命令的控制下，可以使用下列字符命令或控制键，前后滚动终端窗口，逐页逐屏地查看文件。

- iSPACE 显示下一页，或显示下“i”行（如果指定行数后，接着按下空格键）。
- iEnter 显示下一行，或显示下“i”行（如果指定行数后，接着按下Enter键）。
- ib (^B) 回显前一页，或回显前数第“i”页（如果指定页数后，接着输入字符“b”或按下Ctrl-B键）。
- id (^D) 显示下半页（初始值为11行），或显示下“i”行（如果指定行数后，接着输入字符“d”或按下Ctrl-D键）。
- if 显示下一页，或第“i+1页”（如果指定页数后，接着输入字符“f”）。
- is 跳过指定的行数之后，接着显示下一页。
- ^L 重新显示终端窗口中原有的数据内容。
- v 调用默认的vi编辑器，编辑当前文件，并把光标定位在当前行。退出vi后再返回more命令的原位置。
- i/pattern 从当前位置开始，检索下一个或下面第“i”个匹配给定模式的字符串。
- !command 调用Shell执行指定的命令。
- :n 显示下一个文件（按照命令行列举的文件名顺序）。
- :p 显示前一个文件（按照命令行列举的文件名顺序）。
- :f 显示当前文件的名称与行号。
- = 显示当前的行号（数）。
- . 重复执行前一个命令。
- h (?) 给出more命令的简要说明。
- q (Q) 退出more命令。

注意：这里所谓的下一页或下一行意指文件结尾方向，前一页或前一行意指文件开始方向。“i”的默认值为1。

4.3.3 使用head命令显示文件前几行内容

有时，只需查看文件的前几行内容，即可对文件有一个清楚的了解。此时可以使用head命令。其语法格式简写如下：

```
head [-number | -n number] [file]
```

其中，number是一个数字，表示需要输出的行数。在默认情况下，head命令将会显示给定文件的前10行（包括空行）的数据内容，例如：

```
$ head /etc/passwd
root:x:0:0:Super-User:/:/bin/ksh
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:
$
```

4.3.4 使用tail命令显示文件最后几行内容

对于比较大的日志文件，有时只想查看文件后面的最新信息。此时需要用到tail命令，其语法格式简写如下：

```
tail [ ±number [-lbcf]] [file]
```

其中，加号“+”表示从文件的起始位置开始计算。减号“-”表示从文件的结束位置开始计算。number是一个数字，表示需要输出的行数。在默认情况下，tail命令将会显示给定文件的最后10行（包括空行）的数据内容，例如：

```
$ tail /var/cron/log
! SIGTERM 6 26 13:56:38 2009
! ***** CRON ABORTED ***** 6 26 13:56:38 2009
! *** cron started *** pid = 226 6 26 15:26:13 2009
! SIGTERM 6 26 15:40:08 2009
! ***** CRON ABORTED ***** 6 26 15:40:08 2009
! *** cron started *** pid = 215 6 26 16:44:07 2009
! SIGTERM 6 26 16:56:59 2009
! ***** CRON ABORTED ***** 6 26 16:56:59 2009
! *** cron started *** pid = 216 6 29 10:56:29 2009
! *** cron started *** pid = 227 6 29 16:59:40 2009
$
```

上述tail命令只能查看静态文件的最后几行内容，对于不断增长的日志文件，有时需要持续地监控文件不断出现的最新信息。此时可以使用带有“-f”选项的tail命令。例如，为了监控

某个日志文件的最新变化，可以使用下列命令：

```
tail -f somelogfile
```

4.4 复制文件

在开发程序时，经常需要复制以前的程序，并以此为基础进行修改，这时就要用到文件的复制命令**cp**。**cp**命令的语法格式简写如下：

```
cp [-ir] source_file target_file
```

其中，“**-i**”选项表示交互复制方式。当指定的目的文件存在时，**cp**命令将会提示用户存在同名的文件。仅当用户输入**y**确认之后，**cp**命令才会继续执行复制；其他任何回答将会终止文件复制的执行。“**-r**”选项表示递归复制方式。当指定的源文件为目录时，**cp**命令将会递归地复制目录及其中的任何文件，包括子目录及其中的文件。

例如，假定我们想利用先前创建的**newfile**文件复制一个**newcopy**文件，可以使用下列命令：

```
$ cp newfile newcopy
$
```

此时，当前目录下将会增加一个新复制的文件，利用**ls**命令进行检验，其结果如下：

```
$ ls -l
total 4
-rw-r--r--  1 gqxing  other          22 Jun 29 17:16 newcopy
-rw-r--r--  1 gqxing  other          22 Jun 29 17:16 newfile
$
```

使用**cp**命令不仅可以复制当前目录下的文件，也可以复制其他目录中的文件。例如，下列命令将会把**/etc**目录下的**profile**文件复制到当前目录，文件名保持不变：

```
$ cp /etc/profile .
$
```

值得注意的是，在使用**cp**命令而不加任何选项时，如果访问权限许可，有可能误删已有的同名文件，将同名的文件覆盖，造成不应有的损失。因此，比较保险的做法是在使用**cp**命令时，有意识地增加“**-i**”选项。使用“**-i**”选项时，如果目的文件不存在，**cp**命令将会正常地执行。如果目的文件存在，**cp**命令将会输出提示信息，请求用户予以确认。示例如下：

```
$ cp -i somefile somefile2
cp: overwrite somefile2 (yes/no)? y
$
```

此外，利用“**-r**”选项，**cp**命令还能够复制整个目录（包括目录中的所有文件及子目录），详情参见4.11节。

4.5 移动文件

使用**mv**命令，可以把文件从一个目录移到另外一个目录中，或重新命名一个文件。**mv**命令的语法格式简写如下：

```
mv [-fi] source_file target_file
```

其中“-f”选项为强制移动或改名，“-i”选项意味着，一旦目标目录或文件存在，在取得用户的认可后才能采取下一步的处理动作。下面是一个利用mv命令把newfile文件移至/tmp目录，而文件名保持不变的例子：

```
$ ls -l
total 4
-rw-r--r--  1 gqxing  other          22 Jun 29 17:16 newcopy
-rw-r--r--  1 gqxing  other          22 Jun 29 17:16 newfile
$ mv newfile /tmp
$ ls -l
total 2
-rw-r--r--  1 gqxing  other          22 Jun 29 17:16 newcopy
$ ls -l /tmp/newfile
-rw-r--r--  1 gqxing  other          22 Jun 29 17:16 /tmp/newfile
$
```

下面的例子则利用mv命令把newcopy文件改名为oldcopy：

```
$ mv newcopy oldcopy
$
```

再次使用ls命令，可以看到原来的文件已不存在，newcopy已经改名为oldcopy：

```
$ ls -l
total 2
-rw-r--r--  1 gqxing  other          22 Jun 29 17:16 oldcopy
$
```

另外，mv命令也存在一个潜在的问题，即当新命名的文件已经存在时，如果访问权限许可，同名的文件将被覆盖。为了防止此类情况出现，可在mv命令中使用“-i”选项，一旦同名文件存在，mv命令将会提示用户，且仅在用户认可的情况下才能覆盖原有的文件，从而避免出现误删文件的问题，例如：

```
$ mv -i file1 file2
mv: overwrite file2 (yes/no)? y
$
```

在文件重命名方面，还有一个比较有用的命令，即basename。basename命令的本意是剔除文件名中的目录部分，即文件路径名中最后一个斜线字符“/”之前的所有目录前缀，使之仅包含文件名本身，或者删除文件名的扩展名后缀。其语法格式简写如下：

```
basename string [suffix]
```

假定有一个C程序/export/home/gqxing/src/atmcom.c，使用下列命令即可取出删除了目录路径名和“.c”后缀的文件名atmcom：

```
$ basename /export/home/gqxing/src/atmcom.c .c
atmcom
$
```

为了编译任何C程序，把编译后的a.out文件改名为相应的程序文件，可以创建下列Shell脚本（注意，命令前后的反向单引号“`”表示用命令的输出结果替换当前位置的内容，参见第

6章“Shell基础知识”的“命令替换”一节)：

```
$ cat comp
#!/bin/sh
cc $1
mv a.out `basename $1 .c`
$
```

其中的\$1是位置参数，可以是提供给comp脚本的任何一个C程序文件，参见第6章“Shell基础知识”。执行下列命令后即可编译atmcom.c源程序，把编译后的a.out程序文件改名为atmcom：

```
$ cd /export/home/gqxing/src
$ comp atmcom.c
$
```

再如，为了把文件的扩展名“.old”改为“.c”，可以使用“*.old”作为参数，运行下列Shell脚本：

```
$ cat rename
#!/bin/bash
for i in $*
do
mv $i `basename $i .old`.c
done
$ ls -l *.old
-rw-r--r--  1 gqxing  other      25852 Jun 18 15:31 atmcom.old
-rw-r--r--  1 gqxing  other      26282 Jun 18 15:31 atmmon.old
-rw-r--r--  1 gqxing  other      52125 Jun 18 15:33 handler.old
-rw-r--r--  1 gqxing  other      30625 Jun 18 15:32 listener.old
$ rename *.old
$ ls -l *.c
-rw-r--r--  1 gqxing  other      25852 Jun 18 15:31 atmcom.c
-rw-r--r--  1 gqxing  other      26282 Jun 18 15:31 atmmon.c
-rw-r--r--  1 gqxing  other      52125 Jun 18 15:33 handler.c
-rw-r--r--  1 gqxing  other      30625 Jun 18 15:32 listener.c
$
```

4.6 删除文件

如果需要删除文件，可使用rm命令。rm命令的语法格式简写如下：

```
rm [-rfi] [file]
```

其中，“-r”选项用于递归地删除目录中的文件及目录本身，参见4.12节。“-i”选项表示以交互方式执行文件删除操作。在删除任何文件之前，rm命令将会请求用户予以确认。“-f”选项表示强制删除文件，即使文件不存在，rm命令也不会输出任何错误信息。

在UNIX系统中，文件一旦删除，很难再恢复。因此，在执行文件删除操作时一定要小心谨慎。在使用rm命令时，建议增加“-i”选项，以便在真正删除文件之前还有一次选择的机会。下面以newcopy文件为例，说明怎样使用rm命令删除文件。

```
$ ls -l
total 4
-rw-r--r--  1 gqxing  other      22 Jun 29 17:16 newcopy
-rw-r--r--  1 gqxing  other      22 Jun 29 17:16 newfile
$ rm newcopy
$ ls -l
total 2
-rw-r--r--  1 gqxing  other      22 Jun 29 17:16 newcopy
$
```

如果使用“-i”选项删除文件，其情况如下：

```
$ rm -i newcopy
rm: remove newcopy (yes/no)? y
$
```

如果想要一次删除多个文件，可以把文件名一一列在rm命令之后，也可以使用通配符，例如：

```
$ rm -i new*
rm: remove newfile (yes/no)? n
rm: remove newcopy (yes/no)? y
$
```

注意：使用星号“*”通配符时要小心谨慎，尤其是下列用法更要慎之又慎：

```
$ rm *
$
```

如果对删除操作确有把握，需要强制删除某些文件时（这在部分系统维护脚本中尤为常见），可以使用“-f”选项，强制删除指定的文件，例如：

```
$ rm -f *.tmp
$
```

4.7 显示当前工作目录

在Bash环境中，命令提示符中通常会包含当前工作目录最低一级的子目录，当需要确定自己当前所处的准确目录位置时，可以使用pwd命令。pwd命令主要用于显示当前的工作目录，以使用户随时了解自己在文件系统目录层次结构中的位置。例如：

```
$ pwd
/usr/include/sys
$
```

实际上，通过设置PS1内部变量，也可以在命令提示符中输出完整的当前工作目录（参见第9章“用户管理”）。

4.8 改换目录

在UNIX系统中，每个用户都有一个属于自己的主目录。在注册之后，系统将会自动地把用户引导至自己的主目录。当需要转入某个目录时，可使用cd命令。cd（改换目录）命令使用

户能够进入文件系统的任何目录层次（除非目录的访问权限不允许）。例如：

```
$ cd /etc/default
$ pwd
/etc/default
$
```

cd命令的默认参数为**\$HOME**，即当前用户的主目录。因此，当输入一个未加任何命令选项与参数的**cd**命令时，相当于执行“**cd \$HOME**”命令，系统将会把当前的工作目录改换到用户的主目录。例如，如果用户的主目录是/export/home/gqxing，输入下列命令即可返回自己的主目录：

```
$ cd
$ pwd
/export/home/gqxing
$
```

在**Bash**、**Korn Shell**以及**TC Shell**等**Shell**中，波浪号“~”可以用做用户主目录的缩写符号。例如，使用下列命令，可以把工作目录改换到当前用户主目录下的**music**子目录：

```
$ cd ~/music
$
```

也可以使用下列缩写形式，指定任何一个用户的主目录。其中，**username**可以是任何合法的注册用户名。

```
cd ~username
```

在不支持波浪号“~”缩写形式的**Shell**（如**Bourne Shell**）中，引用用户主目录的替代方法是引用**\$HOME**变量，这也是每个**Shell**都支持的通用方法。例如，可以使用下列命令，把工作目录改换到当前用户主目录下的**script**子目录。

```
$ cd $HOME/script
$
```

如前所述，句点“.”表示当前目录，双句点“..”表示父目录。为了从子目录返回父目录，可使用“**cd ..**”命令。

```
$ pwd
/export/home/gqxing/script
$ cd ..
$ pwd
/export/home/gqxing
$
```

假定当前的工作目录是/home/user1，现准备转到/home/user2目录下处理其中的文件，可以使用下列命令：

```
$ pwd
/home/user1
$ cd ../user2
$ pwd
/home/user2
$
```

在使用cd命令时，可以指定绝对目录名，也可以使用相对目录名。所谓绝对目录名，是指从文件系统的根目录（/）开始，按层次结构逐级列出每一级子目录，直至最终子目录的路径名。所谓相对目录，是指相对于当前目录的路径名。

在上述例子中，“cd ../user2”引用的就是相对目录。如果改用下列“cd /home/user2”命令，其中引用的就是绝对路径。

```
$ pwd
/home/user1
$ cd /home/user2
$ pwd
/home/user2
$
```

4.9 创建目录

许多用户习惯于创建不同的目录，分类存储各种不同的文件。为了创建新的目录，可以使用mkdir命令，同时在命令后面指定新建目录的名字。例如：

```
$ mkdir src
$ cd src
$ mkdir java
$ mkdir c
$ cd java
$ pwd
/export/home/gqxing/src/java
$ cd ..
$ ls -l
total 4
drwxr-xr-x  2 gqxing  other      512 Jun 29 17:13 c
drwxr-xr-x  2 gqxing  other      512 Jun 18 15:41 java
$
```

4.10 移动目录

同文件操作一样，也可以利用mv命令，把目录从一个位置移动到另一个位置。例如，利用下列mv命令，可以把src目录下的java子目录移至上一级目录，即移至主目录：

```
$ cd /export/home/gqxing/src/java
$ mv java ..
$ cd ..
$ ls -l
total 10
drwxr-xr-x  2 gqxing  other      512 Jun 29 17:13 bin
drwxr-xr-x  2 gqxing  other      512 Jun 18 15:41 conf
drwxr-xr-x  2 gqxing  other      512 Jun 19 12:57 incl
drwxr-xr-x  5 gqxing  other      512 Jun 23 13:28 java
drwxr-xr-x  2 gqxing  other      512 Jun 18 15:34 src
$
```

同样，也可以利用mv命令重新命名一个目录，为目录改换一个名字：

```
$ mv script shell
$ ls -l shell
total 8
-rwxr-xr-x  1 gqxing  other      941 Jun 22 12:41 ftpmget
-rwxr-xr-x  1 gqxing  other      333 Jun 22 12:41 ftpmput
-rwxr-xr-x  1 gqxing  other      604 Jun 30 13:00 prime
-rwxr-xr-x  1 gqxing  other      640 Jun 22 12:43 whatday
$
```

4.11 复制目录

为了把某个目录下的所有文件全部移至另外一个目录，可以使用cp命令的“-r”选项实现。例如：

```
$ cp -r dir1 dir2
$
```

cp命令的“-r”选项意味着递归复制。执行上述命令后，源目录dir1下的所有文件、子目录及其子目录下的所有文件将全部复制到目的目录dir2中。在复制目录时，如果不加“-r”选项，cp命令将会输出一个错误信息。

另外，还可以利用cpio和后面即将介绍的find命令实现目录的复制。通过管道组合使用这两个命令，可以构成一个功能非常丰富的复合命令。cpio命令可用于备份任何目录或文件系统中的文件，把其中的文件归档后复制到任何存储介质中。另外一个重要功能就是能够把位于指定目录下的所有目录文件依其原有的层次结构原封不动地复制到另一个目录中。cpio命令的语法格式简写如下（这里主要使用第三种命令格式）：

```
cpio -i [-bBcdfkmPrsStuvV] [-C bufsize] [-E file] [-H header]
      [-I file] [-M message] [-R id] [< file]
cpio -o [-aABcLPvV] [-C bufsize] [-H header] [-O file] [-M message] [> file]
cpio -p [-adlLmPuvV] [-R id] directory < fname_list
```

其中，“-p”选项表示从标准输入中读取路径文件名，然后按照其他选项的要求，把指定目录中的目录和文件按原有的层次结构复制到目的目录中。表4-2给出了其他有关选项的说明。

表4-2 find命令的其他部分选项及其简单说明

选项	简单说明
-a	完成文件复制后恢复源文件的访问时间属性，使得cpio命令的文件访问不留痕迹
-d	根据源目录的层次结构要求创建目录
-m	使复制的目的文件保持源文件原有的修改时间不变（但对新创建的目录无效）
-R id	利用指定的用户ID，重新设置每一个文件的属主与用户组。注意，给定的ID必须是/etc/passwd文件中的有效用户ID。此外，只有超级用户才能使用这个选项
-u	无条件的复制。通常，时间较早的老文件不能替换复制目的中同名的新文件
-v	显示cpio命令处理的文件列表及其属性信息

例如，为了把dir1目录中的所有子目录及文件原封不动地复制到一个新目录dir2中，可以使用下列组合命令：

```
$ cd dir1
$ find . -print | cpio -padmu dir2
$
```

4.12 删除目录

为了删除一个空目录，可以使用rmdir命令，其语法格式简写如下：

```
rmdir [-ps] directory
```

如果目录中包含文件，使用rmdir命令删除目录时将会出错。此时可以使用带有“-r”选项的rm命令。利用“-r”选项删除目录是一种递归操作，可以把指定目录及其任何子目录中的所有文件全部删除，例如：

```
$ rm -r dir2
$
```

注意：使用“rm -r”命令删除目录及其文件时，一经提交命令，通常是无法挽回的。因此，使用这个命令时必须谨慎行事。

4.13 比较文件之间的差别

4.13.1 使用diff命令比较两个文件

当面对两个类似的文件，想找出其中的细微差别时，可以使用diff命令进行比较，从中找出两个文件的不同之处。diff命令的语法格式简写如下：

```
diff file1 file2
```

diff命令分别读取两个输入文件，逐行分析其中的异同点，从而找出两者之间的差别。当找出不同的行时，diff命令将会尝试确定出现差别的行是否是由于插入、删除或修改文本行等原因造成的，如果确乎如此，还要检查有多少行受到影响。最后，diff命令将会告诉用户，每个文件的哪一行，从哪个字符开始，有几个字符不同，同时给出两个文件中存在差别的文本行。

如果两者之间的差别是由于插入新的文本行造成的，diff命令将会采用下列形式显示新增的行：

```
lline#1[,lline#2] a rline#1[,rline#2]
```

其中，lline#1和选用的lline#2是第一个文件中的行号，rline#1和选用的rline#2是第二个文件中的行号。

如果两者之间的差别是由于删除文本行造成的，diff程序将会采用下列形式显示哪一个文件删除了文本行：

```
lline#1[,lline#2] d rline#1[,rline#2]
```

如果两者之间的差别是由于修改文本行造成的，diff程序将会采用下列形式显示发生变更

的文本行:

```
l1line#1[,l1line#2] c rline#1[,rline#2]
```

在上述的任何情况下，两个文件中的相关文本行将会随行号一并给出。第一个文件中的文本行前面冠以小于号“<”，第二个文件中的文本行前面冠以大于号“>”。

假定有两个文件test1和test2，其中的数据内容分别如下：

```
$ cat test1
You are in a maze of
twisty little passages
which are all alike.
$ cat test2
You are in a maze of
twisty little passages
which are all different.
$
```

利用diff命令进行比较，其输出结果如下：

```
$ diff test1 test2
3c3
< which are all alike.
---
> which are all different.
$
```

其中的“3c3”表示两个文件的第3行数据内容不同，不同的原因是由于部分数据内容的变动造成的。

如果两个文件完全相同，diff命令将不会显示任何信息。有关diff命令的详细说明和其他功能，可以查阅UNIX系统的随机文档。

4.13.2 使用diff3命令比较三个文件

为了比较三个不同版本的文件，可使用diff3命令。其语法格式简写如下：

```
diff3 file1 file2 file3
```

经过对三个不同版本文件的比较，diff3命令将会指出其中是否存在差别。如果三个文件均相同，则不输出任何信息。如果输出：

```
====
```

说明三个文件均不相同。如果输出：

```
====1
```

说明第一个文件不相同。同样，如果输出：

```
====2或====3
```

说明第二或第三个文件不相同。然后通过下列信息说明差别的原因，但不会把存在差别的文本行显示出来：

f:n1a 意味着第“f (f=1,2,3)”个文件的第“n1”行之后插入了文本行。

f:n1,n2c 意味着第“n1”至第“n2”行范围的文本数据已经修改。如果“n1=n2”，则文

本行的范围可以缩写为“n1”。

现在，我们以Solaris UNIX系统的可调参数设置文件/etc/system为例，说明diff3命令的输出结果。在安装数据库软件的时候，通常需要设置系统的可调参数，以提高数据库的运行效率。假定我们修改了其中的部分参数，当前的system文件和system2文件的数据内容如下（省略了其他内容）：

```
shmsys:shminfo_shmmax=1073741824
shmsys:shminfo_shmmin=1
shmsys:shminfo_shmmni=100
shmsys:shminfo_shmseg=10
```

而原始的system.save文件的数据内容如下：

```
shmsys:shminfo_shmmax=2147483648
shmsys:shminfo_shmmin=1
shmsys:shminfo_shmmni=100
shmsys:shminfo_shmseg=10
```

执行diff3命令后，其输出结果如下：

```
$ diff3 system system2 system.save
===3
1:1c
2:1c
shmsys:shminfo_shmmax=1073741824
3:1c
shmsys:shminfo_shmmax=2147483648
$
```

上述输出信息说明，第三个文件有差别（**===3**），第一和第二个文件的第一行与第三个文件的第一行不同，其原因是做过修改（c）。

4.14 从系统中检索文件

当用户想要找出具有某一特征的文件，或了解系统中是否存在某个文件，又不知文件究竟在哪个目录时，可以使用find命令。

find命令将会按照用户指定的检索条件，从指定的目录开始，找出满足匹配准则的所有文件。指定的检索条件可以是文件名（包括通配符）、文件大小，以及文件修改日期等。find命令的语法格式简写如下：

```
find directory [options]
```

其中，**directory**是检索的起始目录，**options**是一种表达式选项，用于指定各种匹配准则或检索条件。每个选项均描述一种文件匹配或检索准则。一个文件必须满足所有的匹配或检索原则才能选中。使用的选项越多，选中的文件越少。表4-3列出了find命令的部分常用选项及其简单说明。

表4-3 find命令的部分常用选项

选项	说明
-name filename	检索匹配指定文件名的所有文件。其中，指定的文件名不必包括目录路径。也就是说，只要文件名本身匹配即可。例如，如果指定的文件名为hosts，则/etc/hosts与/etc/avahi/hosts等均为匹配检索准则的文件。如果指定的文件名中包括通配符“*”、“?”和“[...]”，文件名前后应加单引号或双引号
-user username	检索其文件属主匹配指定用户的所有文件。其中，username可以是任何一个合法的注册用户名，也可以是用户ID号
-group groupname	检索其用户组属性匹配指定用户组的所有文件。其中，groupname可以是任何一个合法的用户组名，也可以是用户组ID号
-nouser	检索其文件属主未在/etc/passwd文件中定义的所有文件，也即检索其文件属主并非本地系统用户的文件
-nogroup	检索其用户组名未在/etc/group文件中定义的所有文件，也即检索其用户组并非本地系统用户组的文件
-atime [±]n	选择在n天之前、之内或恰好n天访问过的文件
-ctime [±]n	选择在n天之前、之内或恰好n天状态信息发生变动的文件
-mtime [±]n	选择在n天之前、之内或恰好n天修改过文件内容的文件
-newer filename	选择其修改日期比给定文件新的文件
-depth	表示逐层深入各级子目录，采用先文件后目录的方式，自底向上依次检索所有的文件和目录
-size [±]n[c]	按照指定的文件大小数值n检索符合条件的文件。其中，n通常表示以512个字节的数据块为单位。如果n后面附加一个字符c，表示指定的文件大小以字节为计数单位。其中： <ul style="list-style-type: none"> • +n 表示大于指定的数量 • n 表示恰好等于指定的数量 • -n 表示小于指定的数量
-inum n	检索匹配指定信息节点号的文件
-type filetype	检索指定类型的文件。其中的文件类型可以是： <ul style="list-style-type: none"> • f 表示普通文件 • d 表示目录 • b 表示块特殊文件 • c 表示字符特殊文件 • p 表示管道（FIFO）文件 • l 表示符号链接文件 • s 表示套接字文件
-perm [±]mode	检索匹配指定访问权限（以八进制数值或符号形式表示）的文件。如果mode字段之前存在一个减号“-”，表示文件的访问权限必须包括mode定义的所有访问权限。如果mode字段之前为加号“+”，表示文件的访问权限至少必须包括mode定义的一种访问权限。如果mode字段之前没有加减号，表示文件的访问权限必须完全匹配mode定义的所有访问权限
-perm [-]onum	用于检索匹配指定访问权限（以八进制数值表示）的文件。如果八进制数值前面有一个减号“-”前缀，表示仅比较八进制数值数据位为1的文件访问权限

(续表)

选项	说明
-links [\pm]n	检索其链接计数大于、等于或小于指定数量n的文件
-exec cmd {} \; +]	把find命令的检索结果作为参数提交给定的命令，由给定的命令再做进一步的加工处理。后面的花括号表示给定命令的参数将由find命令的输出结果予以替换。命令的后面必须以转义的分号“\;”或转义的加号“\+”结束。当命令以加号结束时，意味着把find命令的输出结果汇总为一个参数集合，然后一次性地提交给定的命令。因此，使用加号“+”而非分号“;”能够改善命令的运行性能
-ok cmd ;	其功能类似于“-exec”选项，唯一的差别是在执行给定的命令之前输出请求信息，当且仅当用户输入“y”（或“Y”）确认之后才继续执行
-ls	以“ls -dils”命令的输出格式输出匹配的文件，文件的大小以1K字节的数据块为单位
-print	打印检索结果，即输出符合检索条件的文件名列表

4.14.1 简单检索

1. 按文件名匹配模式检索文件

假定我们想要检索当前工作目录及其子目录下所有以“.c”为后缀的C程序文件，可以输入下列命令：

```
$ find . -name '*.c' -print
./src/atmmon.c
./src/listener.c
.....
$
```

上述find命令中的句点“.”表示当前目录。这意味着从当前目录开始，遍历当前目录及其子目录，检索匹配“-name”选项（“.c”）指定的所有C源程序文件。

注意：当文件名匹配模式中包含通配符时，一定要用单引号或双引号括起来，以便Shell能够正确地解释。

2. 按文件创建日期检索文件

假定我们想要在/export/home/gqxing目录及其子目录中找出在创建了某个标志文件（如memo2）之后修改过的所有文件，可以使用下列命令：

```
$ find /export/home/gqxing -newer memo2
```

4.14.2 使用逻辑运算符

find命令允许用户使用逻辑非“!”、逻辑与“-a”和逻辑或“-o”等逻辑运算符组合各种选项，定义更为严格的检索准则。在使用逻辑表达式时，组合选项前后要加一对转义的圆括号。

如果想要检索不符合某个特定属性选项定义的文件，可以使用逻辑非运算符“!”。例如，假定我们打算找出/etc目录中不属于用户root的所有文件，可以使用下列命令：

```
$ find /etc \( ! -user root \)
.....
$
```

如果想要检索同时具有两种不同属性的文件，可使用逻辑与运算符“-a”定义组合选项。例如，假定我们准备找出系统中属于gqxing用户的所有子目录，可以使用下列命令：

```
$ find / \( -type d -a -user gqxing \)
.....
$
```

如果想要检索具有其中的一种或两种属性的文件，可以使用逻辑或运算符“-o”定义组合选项。例如，假定我们期望检索系统中一个月来从未访问过的，文件扩展名为“.o”或文件名为a.out的所有文件，可以使用下列命令：

```
$ find / \( -name '*.o' -o -name a.out \) -atime +30
.....
$
```

4.14.3 利用find命令本身实现其他处理功能

利用find命令的“-exec”选项，采用下列两种命令形式，还能够以批处理的方式，把检索出来的文件作为参数，交由其他命令再做进一步的加工处理。

```
-exec command {} \;
-exec command {} \+
```

其中，command可为任何文件处理命令，花括号表示其参数取自find命令的输出，即由find命令检索出来的文件名予以替换。注意，“-exec”选项的后面必须附加转义的分号“\;”或转义的加号“\+”，作为命令的终止符。

例如，如果想要删除当前目录及其子目录中扩展名为“.tmp”的所有文件时，可以使用下列命令：

```
$ find . -name '*.tmp' -exec rm {} \;
$
```

假定因为某种原因，使得一个目录（及其各级子目录）中所有文件的文件属主或访问权限发生了变化，导致应用无法正常运行时，可以利用find命令列出所有的文件，由chown或chmod等命令予以修正，例如：

```
$ find . -type f -exec chown gqxing '{}' \;
$
```

4.14.4 利用管道实现其他处理功能

通常，UNIX命令能够接受的参数具有一定的限量，虽然这个限量一般情况下并不妨碍我们执行任何UNIX命令，但当使用上述find命令形式进行批处理时，find命令的输出有时是很可观的，“-exec”选项后面的命令可能无法接受如此多的参数。为了解决这个问题，一种常见的替代做法是利用管道，将find命令的输出提交给xargs命令协助执行。

例如，为了把某个目录下所有普通文件的访问权限改为644，所有子目录的访问权限改为755，可以使用下列两个命令实现：

```
find pathname -type f -print | xargs chmod 644
find pathname -type d -print | xargs chmod 755
```

把find命令的输出通过管道提交给xargs，由xargs命令协助执行，除了上述原因之外，还有性能方面的考虑。当把大量的文件和目录交由“-exec”选项后面的单个命令处理时，对于每个文件或目录的处理都需要创建一个进程，因而需要创建太多的进程。而UNIX系统允许每个用户创建的进程数量是有一定限制的。xargs的做法是把文件和目录名收集在一起，组成多个参数提交给单个UNIX命令执行。这种处理方式只需创建较少的进程，因而能够提高命令的运行速度，改善系统的性能。

但是，当find命令的输出并没有多到无法接受时，如果在find命令的“-exec”选项后面直接采用转义的加号“\+”，在提高运行速度，改善系统性能方面，也能达到同样的目的，两者的效果基本上是一样的。

4.15 检索文件内容

本节主要讨论UNIX系统中具有超强功能的文本检索工具grep。

4.15.1 利用grep检索文件内容

为了检索文件中的特定字符串，可以使用grep命令。grep命令的基本语法格式简写如下：

```
grep [-inv] string file
```

其中，“-i”选项表示在进行比较时忽略字母的大小写。“-n”选项表示在输出的检索结果之前给出文本行在文件中的行号（行号从1开始计算）。“-v”表示检索不包含给定字符串或模式的所有文本行。string是一个检索模式。检索模式可以是一个准备检索的字符串、一个单词或者短语。注意，检索模式可以包含空格、标点符号甚至控制字符，但需要在前后增加引号。file是准备检索的文件。

例如，为了从电话簿文件Phonebooks中检索John Smith的电话分机，可以使用部分或完整的名进行模式匹配。示例如下：

```
$ grep Smith Phonebooks
John Smith          2810
$
```

如果检索模式是一个较长的字符串，由多个字组成，中间也可能包含空格字符，可以在字符串前后加单引号或双引号，例如：

```
$ grep "Louisa May" Phonebooks
Louisa May Alcott   2826
$
```

检索模式越简短，输出的冗余数据就越多。反之，检索模式的限制越多，符合检索条件的输出结果就越少，例如：

```
$ grep Al Phonebooks
Louisa May Alcott   2826
David Allan         2866
Edgar Allan Poe     2812
$ grep Allan Phonebooks
David Allan         2866
```



```
Edgar Allan Poe      2812
```

```
$
```

通常，**grep**命令是严格区分大小写字母的，也就是说，在输入作为检索模式的字符串时必须注意字母的大小写，例如：

```
$ grep allan Phonebooks
$ grep Allan Phonebooks
David Allan          2866
Edgar Allan Poe      2812
$
```

上述第一个**grep**命令之所以检索失败（没有给出检索结果），就是因为字母“a”的大小写拼写错误。

4.15.2 过滤其他命令的输出数据

grep命令的主要用途是从输入文件中获取感兴趣的数据，过滤掉不需要的数据内容。因此，通常把**grep**称做滤通程序或过滤程序。**grep**命令最常见的用法是过滤其他命令的输出结果，从命令输出中抽取含有某种特征的数据。因此，必须采用管道机制，把命令的输出数据提交**grep**命令。

假定当前目录中存在一系列不同时间开发的C程序：

```
$ ls -l *.c
-rw-r--r--  1 gqxing  other      833233 Jun 18 15:31 buttons.c
-rw-r--r--  1 gqxing  other      739245 Jun 20 15:36 changes.c
-rw-r--r--  1 gqxing  other      608368 Jun 18 15:33 clock.c
-rw-r--r--  1 gqxing  other      827114 Jun 20 15:38 commands.c
.....
$
```

如果想要从中找出6月20日开发的程序，可以使用下列命令组合，通过管道把**ls**命令的输出送交**grep**，过滤后的输出结果如下：

```
$ ls -l *.c | grep 20
-rw-r--r--  1 gqxing  other      739245 Jun 20 15:36 changes.c
-rw-r--r--  1 gqxing  other      827114 Jun 20 15:38 commands.c
$
```

4.15.3 使用grep检索多个文件

grep命令可以同时检索多个文件。当找出匹配检索模式的字符串时，**grep**将会在输出信息前冠以文件的名称，然后输出匹配检索模式的文本行。例如：假定存在beijing、shanghai、washington和newyork等文件，其中的内容及其检索结果如下：

```
$ cat beijing
Beijing is the capital of China.
$ cat shanghai
Shanghai is the biggest city in China.
$ cat washington
Washington is the capital of the United States.
```

```
$ cat newyork
New York is the biggest city in the United States.
$ grep capital *
beijing:Beijing is the capital of China.
washington:Washington is the capital of the United States.
$
```

4.15.4 检索不包含特定字符串的文本行

为了输出并不包含特定字符串的所有文本行，可以使用grep命令的“-v”选项。下面的例子说明怎样在当前目录下的所有文件中找出不包含字符串capital的文本行。

```
$ grep -v capital *
newyork:New York is the biggest city in the United States.
shanghai:Shanghai is the biggest city in China.
$
```

4.15.5 在grep中使用正则表达式

在grep命令中，也可以使用正则表达式作为检索模式，检索匹配给定模式的字符串或文本行。正则表达式可以由普通字符、数字以及具有特殊意义的元字符组成。在grep的正则表达式中，可用的元字符包括“^”、“\$”、“.”、“*”和“\”等，如表4-4所示。注意，grep命令仅支持简单的正则表达式。欲实现复杂的模式匹配，可以使用egrep等命令。

表4-4 grep检索模式可用的元字符

元字符	匹配
^	匹配文本行的行首
\$	匹配文本行的行尾
.	匹配任何一个单字符
[...]	匹配字符集或字符范围中的任何一个字符
[^...]	匹配不属于字符集或字符范围中的任何一个字符
*	零个或多个同一字符或正则表达式
+	一个或多个同一字符或正则表达式
\	随后的元字符作为普通字符处理

表4-4中的特殊字符对UNIX系统也有特殊的意义。因此，在grep命令中使用正则表达式时，需要通过转义机制，使系统在解释命令行期间忽略这些元字符的特殊含义。因此，当用户在系统提示符下输入带有正则表达式的grep命令时，应当使用引号括住正则表达式。

此外，如果检索模式中包含元字符，且需要忽略其特殊意义时，可在元字符之前增加转义符号。

现在，我们将以下列五行数据（选自泰戈尔《Stray Birds》的两首诗）作为这一节的测试数据，说明如何在grep命令中使用正则表达式：

```
$ cat stray.birds
I cannot choose the best
The best choose me
```

```
Roots are the branches down in the earth
Branches are the roots in the air
$
```

由于元字符“^”表示行首，故下列命令将会找出输入文件中以字符“T”为行首字符的所有文本行：

```
$ grep '^T' stray.birds
The best choose me
$
```

为了仅仅列出当前目录中的子目录，可以使用下列命令：

```
$ ls -l | grep '^d'
drwxr-xr-x  2 gqxing  other          512 Jun 29 17:13 bin
drwxr-xr-x  2 gqxing  other          512 Jun 18 15:41 conf
drwxr-xr-x  2 gqxing  other          512 Jun 19 12:57 incl
drwxr-xr-x  5 gqxing  other          512 Jun 23 13:28 script
drwxr-xr-x  2 gqxing  other          512 Jun 18 15:34 src
$
```

元字符“\$”表示行尾，故下列命令将会找出输入文件中以字符“t”为行尾字符的所有文本行：

```
$ grep 't$' stray.birds
I cannot choose the best
$
```

下列命令可用于显示文件中只有一个字符“b”的文本行（输入文件中没有这样的文本行，故没有任何输出）：

```
$ grep '^b$' stray.birds
$
```

由于元字符“.”可以匹配任何一个字符，故下列命令能够匹配任何以“an”为头两个字符的三字符字符串，包括“any”、“and”、“cannot”和“plan”（因为空格也计数）等。

```
$ grep 'an.' stray.birds
I cannot choose the best
Roots are the branches down in the earth
Branches are the roots in the air
$
```

为了列出当前目录中其他用户能够读写的文件，可以使用下列命令：

```
$ ls -l | grep '^-.....rw'
-rwxr-xr-x  1 gqxing  other          120 Jul  3 10:39 stray.bird
$
```

为了找出文件中包含字符串“the”的文本行，且忽略大小写字母的区别，在输出数据前冠以行号，可以使用下列命令：

```
$ grep -i -n the stray.birds
1:I cannot choose the best
2:The best choose me
4:Roots are the branches down in the earth
```

```
5:Branches are the roots in the air
$
```

为了找出文件中的所有空行，并给出行号，可以使用下列任何一个命令：

```
$ grep -n ^$ stray.birds
3:
$ grep -n -v . stray.birds
3:
$
```

当检索模式（如字符、字符串或正则表达式）后面附加一个星号“*”字符时，**grep**将会把星号“*”解释为“重复匹配零个或多个模式”。

按照上述定义，检索结果也可以包含零个模式，这有可能使用户对星号的使用及输出结果感到困惑。下列命令或许可以稍作阐释，其中的检索模式“ro*”意味着找出字符串中含有一个字符“r”和零个或多个字符“o”（如“r”、“ro”或“roo”等）的所有文本行。

```
$ grep 'ro*' list
Roots are the branches down in the earth
Branches are the roots in the air
$
```

在上述输出结果中，第一行匹配检索模式“ro*”的字符串是“branches”，第二行匹配检索模式“ro*”的字符串是“roots”。

同样，如果想要找出字符串中至少包含一个字符“d”的所有文本行，可以使用下列检索模式（其中第一个字符“d”仅表示字符文字“d”，第二个字符“d”和“*”是一个正则表达式，表示零个或多个字符“d”）：

```
$ grep 'dd*' stray.birds
Roots are the branches down in the earth
$
```

但如果想要找出字符串中至少包含两个字符“oo”的所有文本行，可以使用下列检索模式（其中前两个字符“o”仅表示字符串“oo”，第三个字符“o”和“*”是一个正则表达式，表示零个或多个字符“o”）：

```
$ grep 'ooo*' stray.bird
I cannot choose the best
The best choose me
Roots are the branches down in the earth
Branches are the roots in the air
$
```

为了检索输入文件中匹配零个或多个任意字符的所有文本行，可以使用下列检索模式：

```
$ grep '.*' stray.bird
I cannot choose the best
The best choose me

Roots are the branches down in the earth
Branches are the roots in the air
$
```

由于句点“.”能够匹配任意一个字符，星号“*”能够重复匹配零个或多个检索模式，故“.*”能够匹配任何文本行，包括空行。

4.15.6 检索元字符本身

为了使用grep命令检索“&”、“!”、“.”、“*”、“?”和“\”等元字符本身，可以在元字符前面加转义符号“\”。转义符号能够使grep命令忽略元字符的特殊含义。例如，下列检索模式可以返回以句点“.”为起始字符的文本行。这在检索经过nroff或troff加工处理的文档时是非常有用的：

```
$ grep ^\. somefile
```

4.15.7 在命令行中使用引号

正如先前所述，如果想把具有多个单词的短语作为一个字解释，可以使用引号把短语括起来。例如，为了从输入文件中检索短语“in the air”，可以使用下列grep命令：

```
$ grep "in the air" stray.bird
Branches are the roots in the air
$
```

此外，还可以使用单引号把多字短语组合为一个检索单位。单引号的另外一个作用是能够确保系统按文字解释一定的元字符，如“\$”符号等。

注意：即使使用引号，命令历史机制中使用的元字符“!”总是作为元字符解释的，除非在前面加转义符号“\”。推而广之，如果想要让grep命令按照普通字符解释“&”、“!”、“\$”、“?”、“.”、“;”和“\”等元字符，必须在每个元字符前面加转义符号“\”。

例如，如果输入下列命令，将会显示stray.bird文件中的所有文本行。

```
$ grep $ stray.bird
I cannot choose the best
The best choose me

Roots are the branches down in the earth
Branches are the roots in the air
$
```

然而，如果输入下列命令，则只会显示包含“\$”字符的文本行。

```
$ grep '\$' stray.bird
$
```

有关grep命令的更多内容，可参考系统提供的随机文档。

4.16 排序

排序是一种经常需要用到的工具。sort命令可对输入数据或文件内容进行排序，按照一定的顺序逐行显示。sort命令的语法格式简写如下：

```
sort [-bdfimnru] -k key -t sepchar -o output [file]
```

其中，“-n”选项表示按照字符串的数值而不是文字进行排序。“-r”选项表示按照从大

到小或反向字符顺序排序。“-k”选项表示关键字的字段位置，或者关键字字段的起始字符位置或范围。“-o”选项用于指定存储排序结果的输出文件（默认值为标准输出）。“-t”选项用于指定除空白字符之外的其他字段分隔符。“-b”选项表示忽略前置的空白字符。“-d”选项表示仅考虑字母数字和空格字符，按字典顺序排序。

例如，“ls -l”命令的输出通常是按文件名的字符顺序输出文件列表的，为了按照文件的大小，从大到小排序，可以观察“ls -l”命令的输出，确定文件大小字段的位置（第五列），然后利用“-k”、“-r”（表示从大到小排序）和“-n”（表示按数值的大小排序）选项进行排序，示例如下：

```
$ cd /var/log
$ ls -l | grep "^-"
-rw-r--r-- 1 root root 25456 Jun 30 16:25 Xorg.0.log
-rw-r--r-- 1 root root 25541 Jun 30 14:11 Xorg.0.log.old
-rw----- 1 root sys 0 Aug 25 2008 authlog
-rw-r--r-- 1 root other 27 Jun 12 17:26 brlog
-rw-r--r-- 1 root root 2184 Jun 12 18:14 postrun.log
-rw-r--r-- 1 root other 689 Jun 12 18:16 sysidconfig.log
-rw-r--r-- 1 root sys 22626 Jun 30 16:26 syslog
$ ls -l | grep "^-" | sort -k 5 -rn
-rw-r--r-- 1 root root 25456 Jun 30 12:23 Xorg.0.log
-rw-r--r-- 1 root root 25456 Jun 29 17:00 Xorg.0.log.old
-rw-r--r-- 1 root sys 21176 Jun 30 12:23 syslog
-rw-r--r-- 1 root root 2184 Jun 12 18:14 postrun.log
-rw-r--r-- 1 root other 689 Jun 12 18:16 sysidconfig.log
-rw-r--r-- 1 root other 27 Jun 12 17:26 brlog
-rw----- 1 root sys 0 Aug 25 2008 authlog
$
```

第5章 编辑文件

vi是UNIX系统提供的一个标准文本编辑器，也是一个著名的可视化文本编辑器。vi具有强有力的文本编辑功能，是一个非常好的开发工具。利用vi，可以编辑文件，开发应用程序。vi提供大量的命令，供用户编辑文件。本章将按照操作类型，介绍最基本的vi命令。

vi可以处于命令和输入两种工作模式。使用vi编辑器时，用户需要知道编辑器当前所处的工作模式。即便如此，只要经过一定的实践，很快就会掌握vi的规律，从而能够得心应手地处理文本文件。

当启动vi编辑器、打开或创建一个文件时，vi即处于命令模式。通过发布vi命令，可使vi处于输入模式。此时即可输入数据，编写自己的应用程序。vi的两种工作模式之间可以相互转换。

5.1 启动vi编辑器

5.1.1 创建文件

在UNIX系统的命令提示符下输入下列命令，即可启动vi编辑器：

```
$ vi myfile
```

如果名为myfile的文件存在，上述命令将会打开指定的文件，同时在编辑窗口中显示文件第一页的数据内容。如果指定的文件不存在，vi将会打开一个新文件，出现如图5-1所示的编辑窗口。

此时，光标将处于编辑窗口左上角的位置，屏幕左边的波浪符“~”表示空行，说明这是一个空文件。注意，启动vi时可以同时指定多个文件名参数，意味着同时编辑多个文件，也可以不指定文件名，等到完成文件编辑之后再使用“:w filename”命令写入一个新文件，然后使用“:q”命令退出vi。

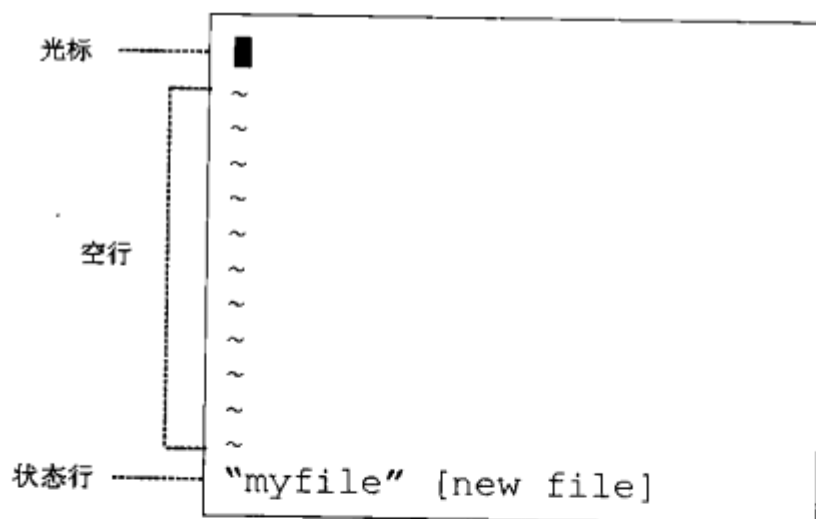


图5-1 vi编辑器启动界面

5.1.2 状态行

编辑窗口的最后一行是vi的状态行，用于显示编辑器的状态、编辑过程中出现的错误信息、光标所在的行列位置、删除或复制的行数等。初始启动时，状态行将会显示文件的名称、文件的行数，以及文件的字节计数。在图5-1中，状态行表示打开的是一个新（空）文件。

5.2 vi编辑器的两种工作模式

vi分为命令和输入两种工作模式。任何时刻，vi编辑器总是处于命令和输入两种工作模式之一。当启动vi编辑器、打开或创建一个文件时，vi即处于命令模式。通过发布vi命令，可使vi处于输入模式。在输入模式下，可以输入数据，编写自己的应用程序。而在命令模式下，可以输入vi命令，执行特定的vi编辑功能。命令模式是vi的默认工作模式。通过使用vi命令和Esc键，可以在两种工作模式之间相互转换。

在某些情况下，vi并不提示编辑器当前所处的工作模式，对于vi的新用户来讲，区分命令模式与输入模式也许是最感困惑的问题。但只要记住一点，就可以做到心中有数。即无论何时，只要按下Esc键，不管vi当前处于何种工作模式，总会进入命令模式。因此，多敲几次Esc键是vi用户的常见动作。

第一次使用vi打开文件时，vi总是处于命令模式。在能够输入任何文本之前，首先必须输入vi的数据输入命令。例如，输入“i”（“插入”）字符命令，即可在当前光标所处字符位置之前插入数据。输入“a”（“附加”）字符命令，即可在当前光标所处字符位置之后附加数据。本章的后面将详细介绍各种数据输入命令。

无论何时需要返回命令模式，只需按下Esc键即可。如果不能确定vi当前处于何种工作模式，只要按下Esc键，即可确保vi总是处于命令模式，然后再决定下一步怎么做。如果在vi处于命令模式时按下Esc键，或按下其他不合法的键时，终端将会发出鸣叫，或发生屏幕闪烁现象，但不会影响正在编辑的文件。

5.2.1 输入模式

为了在前面打开的myfile示例文件中输入数据，可输入vi的“插入”命令“i”。这一命令将使vi编辑器从命令模式转入输入模式。

现在，即可尝试输入几行数据，在每行数据输入结束后按下Enter键。在输入数据的过程中，可以利用退格键（Backspace）做简单的校正（在按下Enter键之前）。在整个输入过程结束之后，按下Esc键，即可返回命令模式。此时，光标将处于刚才输入的最后一个字符位置。然后，还可以利用各种vi命令，对输入的数据进行校正。

5.2.2 命令模式

如上所述，当利用vi打开一个文件时，vi将处于命令模式。在命令模式下，可以输入各种vi命令，以便完成各种编辑功能。vi命令几乎都是由一个字符、两个字符或一个选用的数字加字符组成的。通常，大小写不同的同一字母，意义相同，但作用则完全不同。例如，字符命令“a”意味着在当前光标所处字符位置之后附加数据，而“A”则意味着在当前光标所在行的行尾附加数据。

大多数vi命令不需要按Enter键即可立即执行。但是，以冒号“:”开始的命令需要在输入命令之后再按Enter键。在命令模式下输入冒号“:”时，冒号“:”将会出现在编辑窗口的左下角，然后即可接着输入编辑命令。

以冒号开始的命令实际上是ex命令，ex与vi命令是同一编辑程序的两个不同的用户界面，

vi提供面向屏幕的用户界面，而ex则提供面向命令行的用户界面。所有的ex命令均可在vi中使用。在输入冒号之后，实际上已经切换到面向命令行的ex用户界面。这种切换方式使用户能够在不离开vi的情况下执行许多文件编辑命令，甚至也可以执行其他Shell命令（参见5.5节）。

5.3 保存编辑的文件并退出vi

在使用vi编辑文件期间，用户所做的任何编辑处理并未直接反映到实际的文件中。实际上，整个编辑过程将被保存到vi于内存中临时创建的一个文件副本中。仅当发布“w”（“写”）等命令时，内存缓冲区中的内容才能永久性地保存到磁盘上的文件中。

vi编辑器的这种处理方式既有积极的一面，也有丢失数据之忧。可取之处是在退出文件编辑时，用户可以放弃编辑期间所做的任何修改，而不影响原有的文件。缺点是当系统发生故障时，有可能丢失内存缓冲区中保存的数据。

因此，最好的做法是注意随时保存数据，特别是当编辑重要的文件时。

vi编辑器提供许多命令，用于把内存缓冲区中的数据内容保存到磁盘文件中，然后退出vi编辑器。这些命令允许用户选择“保存并退出”、“强制退出而不保存”等编辑器退出方式，如表5-1所示。

表5-1 vi的保存文件和退出命令

命令	简单说明
:w	保存编辑后的文件内容，但不退出vi编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vi时指定的文件中
:w!	强制写文件，即强制覆盖原有的文件。如果原有文件的访问权限不允许写入文件，例如，原有的文件为只读文件，则可以用这个命令强制写入。但是，这种命令用法仅当用户是文件的属主时才适用，而超级用户则不受此限制
:wq	保存文件内容后退出vi编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vi时指定的文件中，然后退出vi编辑器。另外一种替代的方法是用ZZ命令
:wq!	强制保存文件内容后退出vi编辑器。这个命令的作用是把内存缓冲区中的数据强制写到启动vi时指定的文件中，然后退出vi编辑器
ZZ	使用ZZ命令时，如果文件已经做过编辑处理，则把内存缓冲区中的数据写到启动vi时指定的文件中，然后退出vi编辑器。否则只是退出vi而已。注意，ZZ命令前面无需加冒号“:”，也无需按Enter键
:q	在未做任何编辑处理而准备退出vi时，可以使用此命令。如果已做过编辑处理，vi不会允许用户使用“:q”命令退出，同时还会输出下列警告信息： No write since last change (:quit! overrides)
:q!	强制退出vi编辑器，放弃编辑处理的结果。如果确实不需要保存修改后的文件内容，可输入“:q!”命令，强行退出vi编辑器
:w filename	把编辑处理后的结果写到指定的文件中保存
:w! filename	把编辑处理后的结果强制保存到指定的文件中，如果文件已存在，则覆盖现有的文件
:wq! filename	把编辑处理后的结果强制保存到指定的文件中，如果文件已存在，则覆盖现有的文件，并退出vi编辑器

5.4 vi编辑器的基本命令

本节将分类介绍vi提供的各种编辑命令，其中包括移动光标位置，输入文本数据，修改和替换文本，撤销先前执行的文本编辑命令，删除文本，以及重复执行先前的命令等。

注意：vi命令是严格区分大小写字母的。即使命令形式完全相同，如果不注意区分大小写字母，也会产生完全不同的效果。

5.4.1 移动光标位置

在启动vi编辑器之后，光标将处于编辑窗口左上角的位置，并处于命令模式。在命令模式下，可以使用如表5-2所示的字符命令移动光标位置。

表5-2 光标移动命令

命令	简单说明
← ↑ ↓ →	方向箭头键。可在编辑窗口上将光标左移、上移、下移和右移一个字符（行）位置
h k j l	其功能与箭头键完全相同。在无法使用箭头键（如远程访问）时，可以使用这四个键分别替代相应的箭头键
Enter键	把光标移至下一行的第一个起始字符位置（第一个非空白字符位置）
+	其功能同Enter键
-	把光标移至上一行的第一个起始字符位置（第一个非空白字符位置）
退格键	光标左移一个字符位置
空格键	光标右移一个字符位置
Ctrl-F	往下（文件结尾方向）滚动一屏。在命令方式下按Ctrl-F键，编辑窗口中将会显示文件下一页的内容，光标也同时移至下一页的左上角位置
Ctrl-B	往上（文件开始方向）滚动一屏。在命令方式下按Ctrl-B键，编辑窗口中将会显示文件前一页的内容，光标也同时移至前一页的左下角位置
Ctrl-U	往下滚动半屏
Ctrl-D	往上滚动半屏
Ctrl-E	编辑窗口中的文件内容整体上移一行
Ctrl-Y	编辑窗口中的文件内容整体下移一行
H	把光标移至编辑窗口顶部第一行的起始字符位置（第一个非空白字符位置）
M	把光标移至编辑窗口中间一行的起始字符位置（第一个非空白字符位置）
L	把光标移至编辑窗口底部最后一行的起始字符位置（第一个非空白字符位置）
w	光标右移一个字。如果相邻的两个字之间有标点符号，光标将移至标点符号位置
W	光标右移一个字。即使相邻的两个字之间有标点符号，也忽略之
b	光标左移一个字。如果相邻的两个字之间有标点符号，光标将移至标点符号位置
B	光标左移一个字。即使相邻的两个字之间有标点符号，也忽略之
e	把光标移至当前字（或下一个字）的最后一个字符位置

(续表)

命令	简单说明
E	同上，只是以空格字符作为字的分隔符
^	把光标移至当前行的起始位置，即当前行的第一个非空白字符位置
0 (零)	同上
\$	把光标移至当前行的行尾，即当前行的最后一个字符位置
[n]G	将光标移至指定行的行首位置。其中n表示行号，默认情况下转至文件的最后一行。因此，为了转到当前文件的最后一行，只需输入“G”命令即可。为了移至文件的第一行，可输入“1G”命令。实际上，利用“nG”命令，可以转至当前文件的任何一行。例如，为了修改文件的第60行，只需输入“60G”命令，即可立即跳转到第60行的行首位置
(把光标移至一个完整句子的句首位置
)	把光标移至一个完整句子的句尾位置
{	把光标移至一个完整段落的段首位置
}	把光标移至一个完整段落的段尾位置

5.4.2 输入文本

vi提供许多命令，使用户能够输入文本数据。表5-3给出了vi编辑器中最常用的几种数据输入命令，这些命令将会使vi进入数据输入方式。同时，如果设置了showmode标志（参见5.8.1一节的介绍），还会在编辑窗口的右下方（即状态行右边）显示“APPEND MODE”、“INSERT MODE”或“OPEN MODE”等状态信息，表示vi当前正处于输入模式。为了使用这些命令，首先必须确保vi处于命令模式（多按几次Esc键总能保证vi处于命令模式）。

表5-3 数据输入命令

命令	简单说明
a	使用“a”命令，可在光标当前所在字符位置之后输入数据，输入的数据数量不限，输入结束后可按Esc键退出输入模式。在发布“a”命令之前，可将光标移至任何文本位置
A	使用“A”命令，可在光标当前所在行的行尾（即最后一个字符之后）输入数据，输入的数据数量不限，直至按下Esc键。在发布“A”命令时，不管光标处于任何位置，都会移至当前文本行的行尾
i	使用“i”命令，可在光标当前所在字符位置之前输入数据，输入的数据数量不限，输入结束后可按Esc键返回命令模式。在发布“i”命令之前，可将光标移至任何文本位置
I	使用“I”命令，可在光标当前所在行的行首（即第一个非空白的起始字符之前）输入数据，输入的数据数量不限，直至按下Esc键。在发布“I”命令时，不管光标处于任何位置，都会移至当前文本行的行首
o	使用“o”命令，可在光标当前所在行之后插入数据，行数不限，直至按下Esc键结束
O	使用“O”命令，可在光标当前所在行之前插入数据，行数不限，直至按下Esc键结束

5.4.3 修改与替换文本

为了修改或替换文本数据，vi提供若干不同的命令（参见表5-4），用于编辑或校正文本数据。用户可以根据具体情况，灵活地予以选用。在表5-4所列的编辑命令中，如果设置了

showmode标志，“大部分命令将会在编辑窗口的右下方显示诸如“INPUT MODE”、“REPLACE MODE”或“CHANGE MODE”等状态信息，表示vi当前正处于输入模式。

表5-4 修改与替换命令

命令	简单说明
C	替换文本行。从光标位置开始直至行尾，替换其间的所有数据，直至按下Esc键结束
cw	替换单个字。为了替换一个整字，可将光标移至单字的字首位置，然后输入“cw”命令，接着输入新的文字。输入的字符数量不限，输入结束后可按Esc键返回命令模式。为了修改单字（后面部分）的一部分，可将光标移至该字欲保留部分右边的字符位置，输入“cw”命令后即可校正该字，结束后按Esc键返回命令模式
[n]cc	替换整个文本行。为了替换一整行文本，只需把光标移至目标行的任何字符位置，然后输入“cc”命令。此时，当前文本行将会消失，留下一个空行位置，等待用户输入新的数据，输入的数据数量或行数不限。输入结束后，按Esc键返回命令模式。为了替换多行文本，可把光标移至目标行的第一行，然后输入“ncc”命令，其中n表示需要替换的文本行的数量
[n]s	替换单个字符。为了替换光标位置的单个字符，可输入“s”命令，之后可以输入任何数量或行数的数据。输入结束后，按Esc键返回命令模式。为了替换从光标位置开始的多个字符，可输入“ns”命令，其中n表示需要替换的字符数量
S	替换文本行。为了替换光标当前所在的文本行，可输入“S”命令，之后可以输入任何数量或行数的数据。输入结束后，按Esc键返回命令模式
r	替换单个字符。使用“r”命令，可用随后输入的字符替换光标位置的单个字符。与“s”命令不同，“r”命令只能替换一个字符，替换后，vi编辑器自动返回命令模式（不需要再按Esc键）
R	替换多个字符。使用“R”命令，可以从光标位置开始替换多个字符，数量不限，直至按下Esc键结束
[n]~	转换光标所在位置字母的大小写。如果在输入“~”之前输入一个数字，可以一次转换多个字母的大小写。此外，如果一直按住波浪号“~”键，能够连续转换多个字母的大小写

5.4.4 撤销先前的修改

在编辑文本期间，以及准备把加工后的数据保存到文件之前，有时可能需要放弃某些编辑处理的结果。vi的“u”和“U”命令能够撤销先前执行的编辑命令，使vi回退至处理前的状态。然后，可以从先前的处理位置开始继续进行编辑加工。撤消命令及其说明参见表5-5。

表5-5 撤销命令

命令	简单说明
u	用于撤销先前执行的编辑命令。如果在编辑过程中出现失误，或编辑后又改变了决定，可以使用vi的“u”命令，撤销先前执行的编辑命令。注意，输入“u”命令后不需要按Esc键。如果连续输入多个“u”命令，将会反复执行恢复与撤销动作
U	撤销或恢复对当前文本行所做的全部编辑处理。使用vi的“U”命令可以撤销或恢复最近一次编辑处理的文本行。也就是说，“U”命令仅适用于最近一次修改的文本行。同样，输入“U”命令后也不需要按Esc键

5.4.5 删除文本

利用表5-6给出的vi命令，可以删除指定的字符、字或文本行数据。

表5-6 删除命令

命令	简单说明
[n]x	删除字符。为了删除单个字符，可将光标移至准备删除的字符位置，然后输入“x”命令即可。 “x”命令除删除指定的字符之外，还将删除字符占用的空间位置——如果从单字中间删除一个字符，余下的字符将会合并为一个新的字，中间不会留下任何间隙。利用“x”命令，也可以删除文本行中的空格字符。为了删除多个字符，可将光标移至准备删除的字符串起始位置，然后输入“nx”命令，其中n表示字符的数量
[n]X	删除字符。为了删除光标位置的前一个字符，可以使用大写的“X”命令。为了删除多个字符，可将光标移至准备删除的字符串的右边，然后输入“nX”命令，其中n表示字符的数量
dw	删除单个字或部分字。为了删除一个整字，可以把光标移至该字的起始字符位置，然后输入“dw”命令。相应的字及其占用的空间位置将一并删除。为了删除单字的右边部分，可将光标移至该字欲保留部分的后面，输入“dw”命令，即可删除单字的右边部分
[n]dd	删除文本行。为了删除整个文本行，可以把光标移至文本行的任何位置，然后输入“dd”命令，整个文本行及其占用的空间将会一并删除。为了同时删除多个文本行，可以将光标移至准备删除的第一个文本行，然后输入“n dd ”命令，其中n表示准备删除的行数（包括当前行在内）
D	删除文本行的行尾部分。为了删除文本行行尾的部分文字，可将光标移至文本行欲保留文字部分的下一个字符，输入“D”命令，即可从光标位置开始删除文本行后面的所有字符

5.4.6 复制、删除与粘贴文本

许多字处理软件都提供“复制-粘贴”与“剪切-粘贴”的文本行处理方式。vi编辑器也提供这样的功能。在vi编辑器中，与“复制-粘贴”等价的处理过程是先用“yy”命令复制文本行，接着再用“p（或P）”命令实现文本行的实际复制；与“剪切-粘贴”等价的处理过程是先用“dd”命令删除文本行，接着再用“p（或P）”命令实现文本行的移动。在上述两种组合方式的基础上，如果在“yy”或“dd”命令之前再输入适当的数字，还可以实现多个文本行的复制与移动处理（如表5-7所示）。

表5-7 复制与粘贴命令

命令	简单说明
[n]yy	复制文本行。实际上，“yy”命令只是把文本行的数据内容保存到粘贴板中。一个复制动作需要同时使用两个命令才能完成：即“yy”与“p（或P）”命令。 因此，为了复制一个文本行，可按下列步骤执行： (1) 把光标移至准备复制的文本行的任何位置； (2) 输入“yy”命令； (3) 再把光标移至目标文本行的任何位置； (4) 输入“p”命令，将粘贴板中的数据内容复制到光标所在行的下面； (5) 或输入“P”命令，将粘贴板中的数据内容复制到光标所在行的上面。 如果在输入“yy”命令之前输入数字，则可以同时复制多个文本行。例如，如果想要复制10行数据，可输入“10yy”命令，这将把从光标当前所在行开始的10行数据复制到粘贴板中。此时，vi编辑器将会在编辑窗口底部显示一条信息：“10 lines yanked.”，表示命令已成功地执行

（续表）

命令	简单说明
[n]Y	其功能等同于“yy”命令
[n]dd	删除文本行。为了把一个或若干文本行移至某个位置，需要先删除文本行，然后再粘贴到适当的位置，因此也需要同时使用两个命令才能完成：即“dd”与“p（或P）”命令。 例如，为了移动5行数据，可以把光标移至欲删除文本行的任何位置，发布“5dd”命令，然后把光标移至插入位置，接着输入“p（或P）”命令，即可把文本行移至当前文本行的下方（或上方）
p（小写）	把粘贴板中的文本数据复制到光标所在文本行的下面
P（大写）	把粘贴板中的文本数据复制到光标所在文本行的上面

5.4.7 重复执行指定次数的命令

许多vi命令前面都可以加一个计数值，表明相应的命令需要重复执行的次数。在前面的讨论过程中，我们实际上已经用到了这样的命令。例如，“3dd”命令意味着删除文本行的动作需要执行三次，最终结果是删除三行数据。“2dw”命令意味着删除两个字，“4x”命令意味着删除4个字符（包括空格）。

另外，也可以使用计数命令方式移动光标。例如，“3w”命令意味右移三个字，2Ctrl-F意味往前滚动两屏。

使用句点“.”命令也可以重复执行先前的文本编辑命令。例如，如果用户刚刚使用“dd”命令删除了一个或多个文本行，此时可以把光标移至准备删除的文本行中，仅仅输入一个句点“.”命令即可重复执行删除文本行的处理动作。对于复杂的编辑命令，使用句点命令尤其方便。

5.5 使用ex命令

事实上，在需要处理大块文本行的情况下，与上述的“复制-粘贴”与“剪切-粘贴”处理方式相比，利用ex命令还可以实现更精确、更方便的编辑处理。使用ex命令时，无需在编辑窗口中计数文本行的数量，然后再寻找插入点。用户只需提供一个准备复制或移动的文本行的范围，然后指定插入点的行号即可（当然，删除时无需指定插入点）。

5.5.1 显示行号

使用ex命令时，通常需要知道文本行的编号。为了在编辑的文件中显示行号，可输入下列命令：

```
:set nu
```

按下Enter键之后，新加的行号将会出现在编辑窗口的左边。注意，这些行号实际上并不存在于文件中，只是为方便用户的编辑处理而出现在编辑窗口中，以增加文本数据的可读性（如图5-2所示）。

为了关闭行号显示，可输入下列命令。

```
:set nonu
```

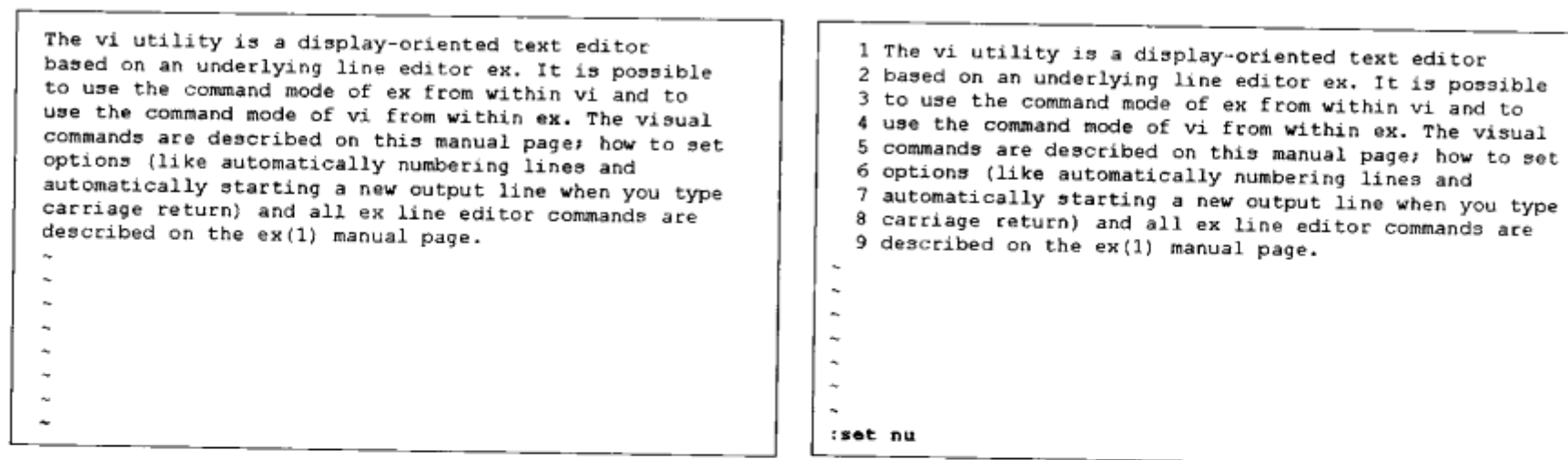


图5-2 显示行号

另外，为了随时了解当前文本行的行号位置，可在命令模式中按下Ctrl-G键。vi编辑器将会在编辑窗口的底部显示当前行的行号位置，以及当前文件的名称等信息。

5.5.2 多行复制

ex复制命令的基本语法格式如下：

```
:line#1,line#2 co line#3
```

其中，前两个数字（即line#1和line#2，中间以逗号分开）用于指定需要复制的文本行的范围，第三个数字（即line#3）表示插入点的行号。例如，为了把myfile文件的第1~第5行复制到当前文件的第12行之后，可以使用下列命令：

```
:1,5 co 12
```

在指定文本行的范围时，可以使用下列缩写形式：

- 句点“.” 表示当前行，意味着从当前行开始。
- 美元符号“\$” 表示文件的结尾，即文件的最后一行。

因此，为了把当前行至后续第5行复制到第12行之后，可以使用下列命令：

```
:. ,5 co 12
```

为了把第6行直至文件最后一行复制到第2行之后，可以使用下列命令：

```
:6,$ co 2
```

5.5.3 移动文本行

ex移动命令的基本语法格式类似于复制命令：

```
:line#1,line#2 m line#3
```

在移动文本行时，可以采用与复制文本行相同的方式指定文本行的范围和插入点，包括使用缩写的句点“.”和美元符号“\$”。两者的差别仅在于“移动”命令将会把指定范围的文本行从一个位置整块地照搬到另一个指定的位置。

例如，为了把第1~第5行移至第12行之后，可以使用下列命令：

```
:1,5 m 12
```


5.5.4 删除文本行

删除文本行时，也可以采用相同的方式指定文本行的范围，包括使用缩写的句点“.”和美元符号“\$”。如果想要删除多个连续的文本行，可以使用下列命令形式实现：

```
:line#1,line#2 d
```

例如，为了删除文件中的第1~第5行，可输入下列命令：

```
:1,5 d
```

5.6 检索与替换

vi提供若干命令，能够以检索指定字符串的方式，直接跳转至期望的文件位置。另外，vi还提供强有力的全局检索与替换功能。

5.6.1 检索字符串

字符串是由一个或多个连续的字符组成的。字符串可以包含字母、数字、标点符号、特殊字符、空格、制表符或回车字符。字符串既可以是一个语法意义上的单词，也可以是单词的一部分。为了检索字符串，可以使用如表5-8所示的命令。

表5-8 检索命令

命令	命令描述
:/str	检索给定的字符串。vi将会从当前光标位置开始检索，当找到指定的字符串时，光标将会移至第一个出现的字符串位置。例如，为了检索meta，可以输入“/meta”命令，然后按Enter键
:?str	从当前光标位置开始，反向检索给定的字符串
n	从当前光标位置开始，继续检索下一个匹配的字符串
N	从当前光标位置开始，反向检索下一个匹配的字符串
/	同“n”命令，从当前光标位置开始，重复执行先前的检索，但在输入“/”字符之后还需要按Enter键
?	同“N”命令，从当前光标位置开始，反向重复执行先前的检索，但在输入“?”字符之后还需要按Enter键
:/pat /+n	将光标移至匹配的字符串“pat”所在文本行之后的第n行
:?pat ?-n	将光标移至匹配的字符串“pat”所在文本行之前的第n行

为了检索字符串，可在“:/”后面附加准备检索的字符串，然后按Enter键。vi将会从当前光标位置开始向下（文件结尾方向）检索。如果发现匹配的字符串，vi将会把光标移至检索方向第一个出现的字符串位置。

例如，为了检索字符串“utility”，可输入“:/utility”命令，然后按Enter键。如果接着输入“n”命令，可以继续检索下一个匹配的字符串。如果输入大写的“N”命令，则可以逆向检索前一个匹配的字符串。

为了在当前编辑的文件中向前（文件开始方向）逆向检索，可以使用“:?”代替“:/”命令。在任何情况下，“n”和“N”的检索方向仍然保持不变。但从逻辑上讲，则恰好与“:?”

的检索方向相反。

通常，vi的字符串检索是严格区分大小写字母的。例如，在检索“china”时，vi不会发现“China”。如果想在检索期间忽略大小写的差异，可以输入“:set ic”命令。完成检索之后，为了返回默认的匹配方式（区分字母大小写），可输入“:set noic”命令。

如果发现检索的字符串，vi将会把光标移至（并停留在）目标字符串的第一个字符位置。如果未发现检索的字符串，vi将会在编辑窗口的底部（状态行中）输出“Pattern not found”信息，说明检索失败。

在检索过程中，某些特殊字符（“/”、“&”、“!”、“.”、“^”、“*”、“\$”、“\”和“?”）具有特定的意义，如果检索字符串中本身也包含这样的字符，必须在其前面增加转义符号“\”，使vi能够按普通字符处理。例如，为了检索字符串“anything?”，可输入“:/anything\?”命令。为了检索一个本身就包含转义符号“\”的字符串，可在转义符号之前再加一个转义符号，即输入两个反斜线“\\”。

5.6.2 模式检索

利用vi提供的模式检索命令（如表5-9所示），还可以使字符串的检索更精确、更有效：

- 仅检索出现在行首位置的字符串；
- 仅检索出现在行尾位置的字符串；
- 仅检索出现在字首位置的字符串；
- 仅检索出现在字尾位置的字符串；
- 使用通配符检索字符串。

表5-9 模式检索命令

命令	命令描述
:/^search	为了检索仅仅出现在行首位置的字符串，可以在字符串前面冠以一个上箭头“^”字符，使vi仅仅匹配行首位置，而忽略出现在其他位置的字符串。例如，为了从当前行开始，检索以“From”为起始字符串的文本行，可以输入“/^From”命令。如果找到匹配的字符串，光标将会移至目的文本行的行首位置。如果想直接定位匹配的字符串，可接着输入“n”命令。如果找不到匹配的字符串，vi将会在编辑窗口底部显示“Pattern not found”信息，而光标则继续停留在原来的位置
:/search\$	为了检索仅仅出现在行尾位置的字符串。可以在字符串后面附加一个“\$”字符，使vi仅仅匹配行尾位置，而忽略出现在其他位置的字符串。例如，为了从当前行开始，检索以“end”字符串结尾的文本行，可以输入“/end\$”命令。同样，如果找到匹配的字符串，光标将会移至目的文本行的行首位置。如果想直接定位匹配的字符串，可接着输入“n”命令。如果找不到匹配的字符串，vi将会在编辑窗口底部显示“Pattern not found”信息，而光标则继续停留在原来的位置
:/^<search\>	为了匹配某个单字起始部分的字符串，可在检索字符串的前面冠以“^<”。为了匹配一个单字的结尾部分，可在检索字符串的后面附加“\>”。因此，为了准确地匹配一个完整的字，而非部分字符串，可在检索字符串前后加上“^<”和“\>”。例如，如果想在整个文件中检索“search”这一英文单词，可以输入“:/^<search\>”命令
使用“.”、“*”和“[...]”等通配符	为了匹配任何一个字符，可以在检索字符串的相应位置中使用句点通配符“.”。例如，为了检索“fun”或“gun”，可以输入“:/un”命令。为了指定一个匹配范围，也可以使用范围通配符进行检索。例如，可以使用“:/[a-z]string”命令，使第一个字符仅限于小写字母。

(续表)

命令	命令描述
	另外，也可以使用列举的方式匹配限定的几个字符。例如，“ <code>:[dm]string</code> ”命令表示仅检索以“ <code>m</code> ”或“ <code>d</code> ”为起始字符的字符串“ <code>string</code> ”，即仅检索“ <code>dstring</code> ”或“ <code>mstring</code> ”。为了检索以某个字符范围开头，中间含有某种模式的字符串，可以组合使用多个通配符进行检索。例如，为了检索以小写字母开头，中间含有“ <code>string</code> ”的字符串，可以使用“ <code>:[a-z]*string*</code> ”命令

5.6.3 替换字符串

替换字符串是在前述字符串检索的基础上实现的。因此，在检索与替换的过程中也可以使用任何通配符。

替换字符串命令的基本语法格式如下：

```
:[g]/search-string/s//replace-string/[g][c]
```

其中，第一个字符命令“`g`”表示全文检索，“`s`”表示替换，第二个字符命令“`g`”表示替换所有匹配的字符串，“`c`”表示在替换之前须经用户确认。因此，为了把文件中的所有字符串“`BankA`”替换为“`BankB`”，可输入下列命令：

```
:g/BankA/s//BankB/g
```

执行上述命令后，`vi`将会把自当前行开始，直至文件结尾范围内的所有“`BankA`”全部替换为“`BankB`”。为了能够在处理之前先经用户确认，然后再替换，可以增加“`c`”命令，使`vi`在执行每个替换之前提请用户予以确认。例如，采用下列命令，可以使`vi`在用“`BankB`”替换“`BankA`”之前，先请用户予以确认。输入“`y`”表示同意替换，输入“`n`”表示不同意替换。

```
:g/BankA/s//BankB/gc
```

注意：利用`Ctrl-C`按键，可以在中途停止这种“确认-替换”的交互式字符串替换方式。

5.7 编辑多个文件

5.7.1 编辑多个文件

`vi`允许用户同时编辑多个文件。例如，为了在编辑`file1`文件的同时，也编辑`file2`文件，可以使用下列命令打开两个文件：

```
vi file1 file2
```

此时，`vi`首先显示第一个文件`file1`，因此可以先行编辑`file1`。编辑结束后输入“`:w`”命令，保存`file1`文件。为了编辑`file2`，可以输入“`:n`”或“`:n file2`”命令。编辑结束后输入“`:w`”命令，保存`file2`文件。编辑结束后，可以输入“`:q`”命令，退出`vi`编辑器，或使用“`:q!`”命令，中途强制退出`vi`编辑器。

在编辑多个文件期间，可以随时使用“`:e filename`”或“`:n filename`”命令，直接转到指定的文件。也可以使用“`:n`”命令转到下一个文件，或使用“`:n #`”命令交替编辑最近处理过的两个文件。如果对文件做过任何编辑加工，在跳转之前，`vi`要求用户使用“`:w`”命令保存当前文件，除非设置了`vi`的`autowrite`选项，`vi`会在跳转之前自动保存修改后的文件。如果想放弃当

前的修改, 可以使用“:e! filename”命令, 强行转至指定的文件, 或使用“:q!”命令, 强制退出vi编辑器。

5.7.2 合并文件与合并文本行

在编辑过程中, 使用vi的“r”命令能够很方便地把指定的文件读入(插入)当前文件的光标位置。这个命令的一般语法格式如下:

```
: [line#] r filename
```

如果未指定行号, vi将会在光标当前所在文本行的后面插入文件。例如, 如果想在文件的第10行之后插入文件chap2, 可以输入下列命令:

```
:10 r chap2
```

另外, 也可以先把光标移至第10行的位置, 然后输入下列命令(这有助于确认插入位置的正确与否):

```
:r chap2
```

为了把相邻的两个文本行合并为一行, 可以把光标移至第一行, 然后输入J命令。

5.8 定制vi编辑器的运行环境

5.8.1 临时设定vi的运行环境

通常, vi编辑器采用一系列默认的选项定义作为自己的运行环境。这些预置的默认选项基本上能够满足大多数用户的日常编辑处理工作。但有时出于某些特定的需要, 或为了提高编辑效率, 需要改变部分选项的默认值。为了查询vi编辑器各种选项的具体设置, 可以使用下列命令:

```
:set all
```

“set all”将会输出vi编辑器当前支持的所有选项及其默认设置, 如图5-3所示。

```
~
~
~
:set all
noautoindent      nomodelines      noshowmode
autoprint          nonumber         noslowopen
noautowrite       nonovice         tabstop=8
nobeautify        nooptimize       taglength=0
directory=/var/tmp paragraphs=IPLPPPQPP Liplpipnptags=tags /usr/lib/tags
noedcompatible    prompt           tagstack
noerrorbells      noreadonly      term=ansi
noexrc            redraw          noterse
flash             remap           timeout
hardtabs=8        report=5        ttytype=ansi
noignorecase      scroll=12        warn
nolisp            sections=NHSHH HUuhsh+c window=25
nolist            shell=/usr/bin/ksh wrapscan
magic             shiftwidth=8    wrapmargin=0
msg              noshowmatch     nowriteany
[Hit return to continue]
```

图5-3 vi编辑器当前使用的选项及其默认设置

为了改变某个选项的设置，可以使用下列set命令：

```
:set option
```

其中，option是vi编辑器支持的选项名称。为了取消某个编辑器选项的设置，可以在选项前增加no字样：

```
:set nooption
```

表5-10给出了vi编辑器支持的部分重要选项及其说明。

表5-10 vi编辑器支持的部分选项

选项	缩写	默认值	简单说明
all	无	无	在编辑窗口中列出编辑器支持的所有选项
autoindent	ai	noai	这个选项应与shiftwidth选项一起发挥作用，使新输入的文本行与上一行的起始位置自动对齐。当vi处于输入模式时，按下制表键Tab将会使光标移至下一个制表位置，按下Enter键将会使光标移至下一行，并与上一行的第一个字符位置对齐，按下Ctrl-T键将会使当前行移至下一个制表位置，而Ctrl-D键将会使当前行反向移至前一个制表位置。这一选项比较适用于程序员
autowrite	aw	noaw	在编辑多个文件的情况下，如果设置了autowrite选项，当使用“:e”或“:n”命令在文件之间切换时，vi将会在切换之前自动保存对当前文件所做的编辑处理。否则，vi将会提示用户把缓冲区中的内容写到磁盘文件中（如果当前编辑的文件中发生了任何变动）
exrc	ex	noexrc	如果在\$HOME/.exrc文件中设置了exrc选项，则允许vi使用当前目录中的.exrc文件（如果存在）
flash	无	flash	当用户输入有误时，vi将以闪动屏幕代替鸣叫
ignorecase	ic	noic	在vi编辑器中，字符或字符串的匹配检索通常是严格区分大小写字母的。如果设置了这一选项，可以使vi在匹配检索过程中忽略大小写字母的差别
list	无	nolist	通常，vi将会按照正常的方式显示文本文件，如按照要求把制表符扩展为适当数量的空白字符，不显示回车字符等。如果设置了这个选项，vi将会把制表符显示为“^I”，在每一个文本行的后部附加一个美元符号“\$”等
magic	无	magic	<div>在检索字符串时，下列字符通常具有特殊的意义：</div> <div><ul style="list-style-type: none">“.”表示匹配任何一个字符“[...]”表示匹配指定字符集合或字符范围中的任何一个字符“*”表示匹配任何字符或字符串</div> <div>在设置了nomagic选项之后，上述字符将会失去其特殊的意义。使用magic选项可以恢复其原有的特殊意义。注意，“^”（表示行首位置）和“\$”（表示行尾位置）总是具有特殊的意义，不受此选项设置的影响</div>
mesg	无	mesg	如果使用“set nomesg”命令关闭此消息选项，则当用户使用vi编辑器时，外来信息不会扰乱当前的编辑窗口
number	nu	nonu	在编辑文本文件时，vi编辑器通常不会显示文本行的行号。为了在每个文本行前面增加一个行号，可以设置这个选项。注意，vi编辑器显示的行号并非文件中的一部分，也不会存储到文件中，只是为了编辑的方便，提供一种临时的标记而已
readonly	无	noreadonly	对正在编辑的文件启用写保护机制，当编辑文件时，vi将会向用户发出警告提示，以避免修改或损害重要的文件

(续表)

选项	缩写	默认值	简单说明
report	无	report=5	这个选项使vi能够确定在删除或复制多少文本行时需要在状态行中显示影响的文本行信息, 如“8 lines deleted”。当删除或复制较少的文本行(少于等于指定值)时, vi不会显示任何信息
scroll	scr	scr=12	这个选项用于控制按下Ctrl-D或Ctrl-U键时, 前滚或后滚多少文本行。这个选项的默认值约为编辑器窗口高度(窗口行数)的一半。为了修改Ctrl-D或Ctrl-U键滚动的行数, 通常有两种实现方式: 一是在按下Ctrl-D或Ctrl-U键之前首先输入一个数字; 二是使用这个选项事先设定一个数值, 如“:set scroll=10”
shell	sh	sh=path	在使用vi编辑器时, 用户可以临时或长久地调用一个Shell, 运行单个UNIX命令, 或交互地访问Shell, 运行多个UNIX命令。这个选项用于确定vi调用哪一个Shell。通常, vi把这个选项设置为用户的注册Shell。为了选用不同的Shell, 可以采用“:set sh=path”形式定义Shell, 其中path为Shell的绝对路径名
shiftwidth	sw	sw=8	这个选项用于设定制表符的跳转位置, 以便在输入模式下, 当按下制表键Tab、Ctrl-T或Ctrl-D键时光标能够自动地跳转到下一个或前一个制表符位置
showmatch	sm	nosm	输入右圆括号、花括号或方括号时提示相应的左圆括号、花括号或方括号。此选项对输入数学表达式或编写程序用到圆括号或花括号时比较有用
showmode	smd	nosmd	根据vi编辑器当前所处的工作模式, 在编辑窗口右下方显示插入模式“INSERT MODE”、附加模式“APPEND MODE”、替换模式“REPLACE MODE”、修改模式“CHANGE MODE”以及输入模式“OPEN MODE”等提示信息
tabstop=8	ts	ts=8	设置制表键Tab的右移距离。默认值为8个空格
wrapmargin	wm	wm=0	指定编辑窗口的右边距。假定终端窗口为80列, 如果使用“:set wm=8”命令设置右边距, 则文本行的逻辑长度不能超出第72列的位置。当输入的文本行到达指定的边界, vi编辑器将会在最接近边界的字(以空格字符为分界符)右边插入一个换行字符。这个选项可以帮助用户摆脱在输入每一行之后再按Enter键。数值0(默认值)表示关闭这一功能, 其他非0数值表示启用并设定编辑窗口的右边距
wrapscan	ws	wrapscan	这个选项影响字符串的检索方式。如果设置了ws选项, 当检索到达文件的结尾仍未发现匹配的字符串时, vi将会从头开始继续检索。否则, 在到达文件的结尾时, 即使仍未发现匹配的字符串, vi也会停止检索

另外, 为了加快数据输入的速度, 还可以使用下列命令形式, 定义用户自己常用的缩写形式:

```
ab abbr string
```

其中, abbr为string的缩写形式, string为实际上应出现在编辑器中的数据。例如, 如果使用下列命令:

```
ab iscas 中国科学院软件研究所
```

则每当输入一个iscas单词时, vi编辑器将会自动代以“中国科学院软件研究所”字符串。

5.8.2 永久性地定制vi的运行环境

上面介绍的方法只能临时地设置vi编辑器的当前运行环境, 一旦退出vi, 这种临时设置也

随之作废。为了永久性地设置各种选项，以免每次进入vi时都要重新设置，可以设置EXINIT变量。设置EXINIT变量的语法格式如下：

```
export EXINIT='set para1 para2 ...'
```

例如，为了适当地显示状态行信息，且在匹配检索过程中忽略大小写字母的差别，我们可以在.profile中增加下列EXINIT变量设置：

```
EXINIT='set showmode ignorecase'; export EXINIT
```

另外，也可以把常用的vi选项及其定义（也即set命令可以设置的选项和定义）加到用户主目录下的vi初始化文件.exrc中。如此，则每当调用vi编辑器时，vi都会自动读取此文件，使定制的选项成为vi的永久性运行环境。下面是一个.exrc文件示例（注意，不要在命令前加冒号）：

```
set showmode
ab abc 中国农业银行
ab boc 中国银行
ab cbc 中国建设银行
ab icbc 中国工商银行
.....
```

实际上，还可以在每个目录中创建一个自己的.exrc文件，以适应不同的需要。例如，可以在C源程序所在的目录中创建一个适应于程序开发的.exrc文件，可以在编写文档的目录中创建一个适应于文档编写的.exrc文件。但应注意，仅当用户主目录下的.exrc文件中包含“set exrc”命令时，其他工作目录中的.exrc文件才能发挥作用。

5.9 其他特殊说明

5.9.1 删除或替换特殊字符

在编辑文件时，有时会遇到文件中含有不可打印的特殊字符。例如，由于Windows与UNIX系统使用的行终止符不同，对于UNIX系统而言，取自DOS或Windows系统中的文本文件会包含多余的回车字符，如图5-4所示。

为了替换或删除这种特殊字符，可以在vi的删除或替换命令中，利用“Ctrl-V”键输入特殊字符的ASCII编码，以删除或替换特殊字符。例如，回车字符的ASCII编码为13，对应于Ctrl-M键。因此，可以采用下列替换命令，删除多余的“^M”字符。

```
1,$ s/^M//
```

注意：上述命令中的“^M”字符不能直接输入，必须先按Ctrl-V键，接着再按Ctrl-M键，才能输入“^M”字符，如图5-5所示。

5.9.2 在编辑期间运行UNIX命令

有时，我们可能需要在编辑过程中运行UNIX命令，或需要把某个命令的输出作为文件的一部分，可以利用表5-11列出的命令，临时或长时间地运行其他UNIX命令，或把UNIX命令的运行结果引入正在编辑的文件中。

(续表)

命令	命令描述
Ctrl-B	往上（文件开始方向）滚动一屏
Ctrl-D	往下滚动半屏
Ctrl-U	往上滚动半屏
Ctrl-E	编辑窗口中的文件内容整体上移一行
Ctrl-Y	编辑窗口中的文件内容整体下移一行
w	将光标右移一个字。光标停留在下一个字的字首位置
W	将光标右移一个字。光标停留在下一个字的字首位置（即使两个字之间存在标点符号）
b	将光标左移一个字。光标停留在下一个字的字首位置
B	将光标左移一个字。光标停留在下一个字的字首位置（即使两个字之间存在标点符号）
e	把光标移至当前字（或下一个字）的最后一个字符位置
E	同上，只是以空格字符作为字的分隔符
^	把光标移至当前文本行的起始位置，也即当前文本行的第一个非空白字符位置
0（零）	同上
\$	把光标移至当前文本行的行尾，也即当前文本行的最后一个字符位置
H	把光标移至编辑窗口顶部第一行的行首位置
M	把光标移至编辑窗口中间一行的行首位置
L	把光标移至编辑窗口底部最后一行的行首位置
插入文本数据:	
a	在光标当前字符位置的后面输入文本数据
A	在光标当前所在文本行的行尾（也即最后一个字符位置）后面输入文本数据
i	在光标当前字符位置的前面输入文本数据
I	在光标当前所在文本行的行首（也即在第一个非空白的起始字符）前面输入文本数据
o	在光标当前所在文本行下面的行首位置输入文本数据
O	在光标当前所在文本行上面的行首位置输入文本数据
修改文本:	
C	替换当前文本行光标所在字符位置之后的所有数据，以Esc键结束
cw	替换光标当前所在字符位置及之后的整个字或部分字，以Esc键结束
[n]cc	替换当前行，或从当前行开始的n行文本，以Esc键结束
[n]s	替换光标当前所在位置的单个字符，或从光标当前位置开始的n个字符，以Esc键结束
S	替换当前行，以Esc键结束
r	替换光标当前位置的单个字符

(续表)

命令	命令描述
r Enter	断行。也可使用“a”或“i”命令加Enter及Esc键实现
R	从光标当前所在的字符位置开始, 替换随后的所有字符, 直至按下Esc键
xp	交换字符位置。交换光标当前所在位置开始的两个字符位置
~	转换光标当前所在位置字符的大小写
u	撤销最近一次执行的编辑命令, 或依次撤销先前执行的编辑命令
:u	同上 (ex编辑命令)
U	撤销对当前文本行的编辑处理
删除文本:	
[n]x	删除光标当前所在位置的字符, 或删除从光标当前位置开始的n个字符
[n]X	删除光标当前所在位置的前一个字符, 或删除光标当前所在位置之前的n个字符
dw	删除光标当前所在位置的一个整字或部分字符。如果光标在字首, 则删除整字。如果光标在字的中间任何位置, 则删除光标位置及之后的字符
[n]dd	删除光标当前所在的文本行, 或删除从当前行开始的n个文本行
D	删除当前文本行从光标位置开始之后的所有字符
dG	删除从当前行开始直至文件最后一行的所有文本行
d[n]G	删除从文件的第n行开始直至当前行的所有文本行
:line#1,line#2 d	删除从指定行号line#1到line#2之间的所有文本行
复制与移动文本:	
[n]yy	复制光标当前所在的文本行, 或从当前行开始的n个文本行
[n]Y	同上
p (小写)	把复制或删除 (“dd”命令) 的文本行粘贴到光标所在文本行的下面
P (大写)	把复制或删除 (“dd”命令) 的文本行粘贴到光标所在文本行的上面
:line#1,line#2 co line#3	把第line#1~line#2行复制到第line#3行之后
:line#1,line#2 m line#3	把第line#1~line#2行移至第line#3行之后
设置行号显示:	
:set nu	在编辑期间增加临时行号
:set nonu	撤销行号显示 (默认情况)
Ctrl-G	显示当前文件的名称和当前文本行的行号
设置大小写字母检索准则:	
:set ic	检索字符串时忽略字母的大小写
:set noic	检索字符串时严格区分字母的大小写 (默认情况)
定位文本行:	
[n]G	将光标移至文件的最后一行或第n行
检索与替换:	
:/string	向前 (文件结尾方向) 检索指定的字符串

(续表)

命令	命令描述
:?string	向后（文件开头方向）检索指定的字符串
n	按检索方向找出下一个匹配的字符串
N	逆检索方向找出前一个匹配的字符串
:[g]/search/s//replace/[g][c]	检索并替换字符串
清除屏幕:	
Ctrl-L	清除因其他进程的输出信息而干扰的编辑窗口
Ctrl-R	Ctrl-L的替代形式。清除因其他进程的输出信息而干扰的屏幕窗口
合并文件与合并行:	
:r filename	在光标所在文本行之后插入指定文件的内容
:[line#] r filename	在第line#1行之后插入指定文件的内容
J	把相邻的两个文本行合并为一行（把下一行合并到光标当前所在文本行的后面）
保存编辑结果与退出vi编辑器:	
:w	保存编辑处理后的最终结果（把内存缓冲区中的数据写到文件中）
:w!	强制保存编辑处理后的结果
:wq	保存编辑处理后的结果，然后退出vi编辑器
:wq!	强制保存编辑处理后的结果，然后退出vi编辑器
ZZ	保存编辑处理后的结果，然后退出vi编辑器
:q	在未做任何编辑处理时，可以使用此命令退出vi编辑器
:q!	强制退出vi编辑器，放弃编辑处理后的结果
:w filename	把编辑处理后的结果写到指定的文件中保存
:w! filename	把编辑处理后的结果强制写到指定的文件中保存，即使文件已经存在
:wq! filename	把编辑处理后的结果强制写到指定的文件中保存，即使文件已经存在，然后退出vi编辑器
其他:	
:f或Ctrl-G	显示文件的名称、编辑状态、文件总的行数、光标当前所在的行号和列号，以及当前行之前的行数占整个文件总行数的百分比
Ctrl-V	用于输入其他控制字符

第6章 Shell基础知识

本章主要介绍Shell编程的基础知识。我们将从什么是Shell脚本开始，介绍Shell变量与变量替换、命令与命令替换，以及各种test语句。讨论怎样编写和运行Shell脚本，以及怎样利用Shell的编程机制和UNIX系统提供的丰富命令，创建自己的Shell脚本。

在本章及随后的第7章中，我们以Bourne Shell、Korn Shell和Bash为基础，结合UNIX系统的日常管理和例行维护，介绍Shell编程的基本概念和技巧。Shell编程涉及的内容非常广泛，而且，如果想把Shell编程的功能发挥得淋漓尽致，还需要对UNIX系统有比较深入的了解。随着对Shell编程和UNIX系统的逐渐熟悉，在日常的系统管理和维护过程中，有关的内容自然而然也就掌握了，没有必要从一开始就求大求全。因此，我们采取的原则是介绍Shell编程的原理，重在实用。

6.1 shell与Shell脚本

在UNIX系统的层次组织结构中，Shell是一个重要的组成部分。Shell是用户与UNIX系统内核进行联系的桥梁。UNIX通过Shell界面，接受用户的请求，利用系统的资源，为用户提供服务。根据Shell的调用方式，UNIX系统中的Shell主要分为交互式注册Shell、交互式非注册Shell，以及非交互式Shell三种。

如果调用方式不同，Shell的初始化过程也不相同。在输入用户名与密码，成功地注册之后，UNIX系统将会调用交互式注册Shell。交互式注册Shell利用/etc/profile文件，以及用户主目录中的初始化文件（或称启动文件），如.profile等，设置用户的运行环境。

在Shell命令提示符下输入sh、ksh或bash等命令，将会进入交互式非注册Shell（可以看做当前Shell的子Shell）。如果进入的是Bash，Shell还会读取并执行/etc/bash.bashrc和~/.bashrc等初始化文件，同时也会继承注册Shell设置的各种环境变量。

非交互式Shell主要用于运行Shell脚本。当利用命令行界面提交一个Shell脚本时，从开始执行到结束，其整个运行环境即为非交互式Shell。非交互式Shell通常不执行任何用户初始化文件，但会继承注册Shell设置的运行环境，从实际效果来讲，相当于间接使用或继承了用户初始化文件的运行结果。但在ksh或bash的情况下，非交互式Shell将会在开始运行之初分别检查ENV或BASH_ENV变量。如果变量已经设置，非交互式Shell将会以ENV或BASH_ENV变量的值作为初始化文件予以执行。

6.1.1 为什么需要Shell编程

UNIX系统提供了丰富的系统命令和各种各样的实用程序，只要经过适当的组合，基本上都可以满足绝大部分应用需求，而不必重新编写新的程序，这是Shell脚本最大的功用所在。实际上，UNIX系统的很多系统配置、启动和管理任务都是通过Shell脚本实现的。

UNIX操作系统中的Shell既是一个命令解释程序，也是一种强有力的编程语言。作为操作系统内核与用户之间的一个交互界面，Shell可以解释执行用户输入的UNIX命令，可以实行I/O

重定向，提供管道、元字符匹配以及文件名生成等功能。

作为一种编程语言，Shell也可以执行简单的批处理，利用UNIX系统的丰富命令、实用程序和各种工具，完成特定的功能需求。如果这样仍然不能满足需要，还可以利用Shell的编程机制，如测试语句、条件转移和循环控制结构，执行复杂的操作，增强Shell脚本的功能和灵活性。

实际上，借助于UNIX系统本身的强大功能，利用Shell的编程机制，很容易实现各种系统管理和日常维护任务，而不需要采用其他高级编程语言，为每一项任务重新开发额外的工具。

在本章和随后的第7章中，我们将利用大量的例子，介绍Shell的各种功能特性。其中的大部分例子都是从实践中挑选出来的，且经过了验证（注意，使用前应首先利用chmod命令，增加Shell脚本的执行权限，然后再运行，参见后面的说明）。

从1979年Steve Bourne开发出第一个Shell，且与UNIX系统第7版一同推出开始，Bourne Shell即成为UNIX系统必备的用户界面。随着UNIX系统的不断发展，David Korn在Bourne Shell和C Shell的基础上，开发出Korn Shell之后，进一步增强了Shell的功能。UNIX系统中提供的Bash与Bourne Shell及Korn Shell一脉相承，在功能上又有更进一步的提高。

如果想要成为专业水平的UNIX系统管理员，熟练地掌握Shell编程技巧是一项必不可少的基本要求，即使不一定非要切实编写Shell脚本。例如，在系统的启动过程中，UNIX执行的就是位于/etc/init.d目录中的一系列Shell脚本，用以完成系统的各种配置、设置，以及启动各种系统服务等。理解和灵活地运用这些Shell脚本，对于分析系统的行为是非常重要的一项技能。

Shell编程是一种简单直观的，能够快速实现复杂功能需求的解决方法。Shell的语法规则简单、直观、易懂，学习和编写Shell脚本并不困难。实际上，只要熟悉UNIX命令，了解Shell编程的基本规则和结构语句，利用Shell的编程机制，把若干UNIX命令、实用程序或应用程序组合在一起，即可写出自己的Shell脚本。

Shell编程吸取了UNIX系统把复杂任务分解成简单子任务的优良传统，使应用开发人员能够把各种系统工具和实用程序灵活地组合在一起，完成特定的功能需求。这是一种非常好的解决问题的手段，避免了针对各种各样的用户需求，都要重新开发一套复杂工具的烦恼。事实上，UNIX系统已经提供了丰富的、功能强大的系统工具和实用程序，只要灵活地运用这些现成的工具，就可以满足我们的绝大部分需求。

6.1.2 什么是Shell脚本

利用文本编辑器，事先把一系列UNIX命令或可执行程序放到文件中，然后修改文件的访问权限，使之能够像系统命令或实用程序一样执行，这样的文本文件就是Shell脚本，或称Shell程序。简单地讲，Shell脚本就是一种包含若干UNIX命令或可执行程序的文本文件。

Shell脚本可由任何UNIX命令组成。当执行Shell脚本时，文件中的所有命令将从头到尾，一个一个地顺序执行（除非Shell脚本中含有控制结构语句）。就像用户在终端前面，以命令行方式，每次输入一个命令，让系统依次执行一样。

同其他面向过程的编程语言一样，除了普通UNIX命令或可执行程序之外，Shell脚本还可以包含控制结构和变量，也可以带有参数，使之完成一定的功能需求。

最简单的Shell脚本可以是仅仅包含一个或一组UNIX系统命令的文件。利用Shell脚本不仅能够容易地实现批量处理，避免一遍又一遍地重复输入一系列常用的命令，而且还可以对Shell脚本进行修改或定制，从而满足不同的应用需求或功能需求。

如果再使用变量、控制结构和参数传递等一整套编程机制，灵活地运用UNIX系统本身提供的丰富命令和工具，Shell编程无疑会如虎添翼，提供更加强大的功能。

6.1.3 运行Shell脚本

Shell脚本有两种执行方式。第一种方式是利用sh、ksh或bash等命令，把Shell脚本文件名作为Shell的参数，由Shell直接读取并解释执行Shell脚本文件中的每一个命令，这些命令就像直接从键盘上输入一样。这种Shell脚本执行方式只要求Shell脚本文件具有“可读”的访问权限。

假定Shell脚本文件whogrep.sh包含下列一行命令：

```
who | grep root
```

采用下列方式运行Shell脚本

```
$ sh whogrep.sh
```

的效果则相当于执行“who | grep root”命令。

第二种执行方式是首先利用chmod命令设置Shell脚本文件，使Shell脚本具有“可执行”的访问权限。然后在命令提示符下直接输入Shell脚本文件名，作为一个普通的命令，交由UNIX系统执行。例如：

```
$ chmod 755 whogrep.sh
$ whogrep
root      console      Jul  7 13:15      (:0)
root      pts/3           Jul  7 13:15      (:0.0)
root      pts/4           Jul  7 13:16      (:0.0)
$
```

因此，一旦写好一个Shell脚本（假定为scriptname），可以采取上述两种调用方式之一。但是，一般不建议使用第一种调用方式——“sh scriptname”，尤其不建议采用“sh <scriptname”调用方式，因为这将禁止Shell脚本读取标准输入。注意，以“sh scriptname”方式调用一个Shell脚本可能会禁止Shell的部分扩充功能，因而引起Shell脚本无法正确地执行。

在采用第二种调用方式直接运行可执行的Shell脚本之前，首先应当使用下列chmod命令之一，把Shell脚本文件设置为可执行文件：

- chmod 755 scriptname（文件属主可以读、写和执行，其他用户只能读与执行）
- chmod a+rx scriptname（增加任何用户读与执行的权限）
- chmod u+rx scriptname（仅增加文件属主读与执行的权限）

按照上述要求设置Shell脚本文件的访问权限之后，可以采用下列方式，直接运行Shell脚本：

- scriptname（如果命令检索路径包含当前目录）
- path/scriptname或./scriptname（如果命令检索路径不包含当前目录）

在测试和调试最终完成之后，如果愿意，也可以作为一个工具，把Shell脚本文件移至某个公用的用户命令目录，如/usr/local/bin目录中，使所有的用户均可运行新增的Shell脚本。

6.1.4 退出与出口状态

当一个命令或进程终止运行时，将会自动地向父进程或Shell返回一个出口状态。如果命令或进程成功执行完毕，将会返回一个数值为零的出口状态。如果命令或进程在执行过程中出现

异常而未正常结束，将会返回一个非零值的出错代码。

同样，一个Shell脚本结束运行时也会返回一个出口状态。在Shell脚本中，最后执行的一条命令将决定整个Shell脚本的出口状态。此外，Shell的内部命令`exit`也可用于随时终止Shell脚本的执行，并返回Shell脚本的出口状态。

因此，在Shell脚本中，开发人员可以利用“`exit [n]`”命令，在终止执行Shell脚本的同时，向调用Shell脚本的父进程（也即Shell）返回一个数值为`n`的出口状态。其中，`n`必须是一个位于0~255范围内的整数值。按照惯例，一个命令或Shell脚本正常终止时应返回0，运行有误时可返回一个1~255范围内的整数值。

当Shell脚本结束运行前执行的最后一个命令是不带任何数值参数的`exit`语句时，Shell脚本的最终出口状态是Shell脚本执行`exit`语句之前执行的最后一条命令的出口状态。依据返回的出口状态，Shell可以判断整个Shell脚本的运行是否正常结束。

例如，假定存在下述Shell脚本，`command_last`命令的返回值就是整个Shell脚本的出口状态：

```
#!/bin/bash
command_1
command_2
.....
command_last
exit
```

在此情况下，其效果等价于最后执行了一个“`exit $?`”语句：

```
#!/bin/bash
command_1
command_2
.....
command_last
exit $?
```

在此情况下，也可以干脆去掉`exit`语句，直接使用最后一条命令的出口状态作为整个Shell脚本的返回值：

```
#!/bin/sh
command_1
command_2
.....
command_last
```

对于顺序执行的Shell脚本，这种做法没有任何问题。但对于带有控制转移和结构语句的Shell脚本而言，则需要在Shell脚本可能终止的任何位置，尽量使用能够返回不同出口状态的“`exit [n]`”语句，以便了解Shell脚本的实际运行结果。

在任何时候或任何位置，都可以使用Shell的内部变量`$?`给出之前执行的最后一条命令的出口状态。Shell函数也是如此，当从函数返回时，`$?`变量的值表示函数中实际执行的最后一条命令的出口状态，这也是Shell给出函数返回值的实现方法。

Shell脚本运行终止之后，也可以在Shell命令提示符下，使用`$?`内部变量返回Shell脚本的出口状态，也即脚本中执行的最后一条实际命令的出口状态。在UNIX系统中，为了测试一个命

令或Shell脚本的执行结果，\$?变量是非常有用的。

下面的例子说明了如何获取命令以及Shell脚本的出口状态：

```
$ cat testexit
#!/bin/sh
LANG=C          # 在英文语言环境中，其说明信息有时更准确一些（本书部分例子中的输出数据
echo hello      # 也是在事先设置了英文语言环境后的结果）
echo $?         # 输出脚本中间单个命令的出口状态。这里的返回值为0，因为echo命令总是
                # 能够成功地执行
lsdf            # 不存在的命令
echo $?         # 输出脚本中间单个命令的出口状态。这里的返回值为非0，因为lskdf并不是
                # 一个合法的命令，因而无法执行

echo
exit 100        # 脚本执行到此结束，同时以100作为整个Shell脚本的返回值
$ testexit
hello
0
testexit: lsdf: not found
127

$ echo $?       # 脚本终止时再执行此命令，将会返回整个Shell脚本的出口状态（100）
100
$
```

“!”是逻辑非运算符，表示命令返回值或测试结果的否定值，因而也能够影响其出口状态。下面的例子就说明了这个问题：

```
$ true          # true是一个内部命令
$ echo "exit status of \"true\" = $?" # 返回值为0
exit status of "true" = 0
$ ! true
$ echo "exit status of \"! true\" = $?" # 返回值为1
exit status of "! true" = 1
$
```

注意：“!”与true之间必须留有一个空格，否则将会导致系统发出“命令不存在”的出错信息。

6.1.5 调用适当的Shell解释程序

通常，Shell脚本的第一行均包含一个以“#!”为起始标志的文本行，这个特殊的起始标志告诉系统：当前文件包含一组命令，需要提交给指定的Shell解释执行。“#!”特殊标志实际上是一个两字节的文件类型“标识码（magic code）”，表示当前的文件是一个可执行的Shell脚本文件。紧随“#!”标志的是一个路径文件名，指向用于执行当前Shell脚本文件的命令解释程序。

在发现“#!”之后，系统将会采用指定的命令解释程序替代当前的Shell，从第二行开始解释执行脚本中的每一个有效的命令（注释行除外）。指定的命令解释程序可以是任何一个Shell、一个普通的UNIX命令，甚至也可以是一个应用程序。

下面列出的特殊标志行都是合法的，分别表示调用不同的命令解释程序或普通程序，解释执行当前的Shell脚本。其中，“#!/bin/sh”表示调用UNIX系统默认的Shell，即Bourne Shell。

Bourne Shell是UNIX系统环境最通用的Shell。“#!/bin/ksh（或#!/usr/bin/ksh）”表示调用Korn Shell解释执行当前的Shell脚本，如此等等。

```
#!/bin/sh
#!/bin/ksh
#!/bin/bash
#!/bin/tcsh
#!/bin/more
.....
```

依据Shell脚本中的特殊标志行，当指定的Shell开始解释执行脚本时，将会把标志行作为注释处理。注意，特殊标志后面的路径文件名必须是正确的。否则，系统将会给出命令解释程序有误的错误信息，并立即终止执行当前的Shell脚本。

按照上述说明，如果Shell脚本中存在特殊标志行，则调用适当的Shell解释执行当前的Shell脚本。如果Shell脚本中包含多个特殊标志行，只有第一个标志行起作用。如果Shell脚本中只用到一些常用的系统命令，而没用使用特殊的Shell内部命令，特殊标志行可以省略。

下面的rmfile是一个比较特殊的Shell脚本，其中的标志行表示需要调用/bin/rm命令，执行当前的脚本文件。注意，执行这个Shell脚本时不会产生任何输出，尽管其中存在若干echo语句，也不会执行到exit语句，唯一的运行结果（或者说唯一的作用）就是删除这个脚本文件本身。假定当前目录包含两个文件，其中之一就是rmfile脚本文件，运行后的结果如下：

```
$ ls -l
total 4
-rw-r--r--  1 gqxing  other          712 Jun 23 12:26 file2
-rwxr-xr-x  1 gqxing  other          112 Jun 23 12:25 rmfile
$ cat rmfile
#!/bin/rm
echo "This line will never print on screen."
echo "And the current file will be deleted also."
exit 1
$ rmfile
$ ls -l
total 2
-rw-r--r--  1 gqxing  other          712 Jun 23 12:26 file2
$
```

依照上述脚本，我们可以尝试用“#!/bin/more”作为特殊标志行，创建一个Shell脚本文件，使之显示紧随其后的文本信息。把文件的访问权限修改为可执行文件，然后再执行这个Shell脚本时，将会显示脚本文件中除了标志行之外的所有内容：

```
$ cat readme.sh
#!/bin/more
This is a script to display this file self.
Contents of this file will be displayed
when you excute this script file.
$ readme.sh
#!/bin/more
This is a script to display this file self.
Contents of this file will be displayed
when you excute this script file.
$
```

6.1.6 位置参数

从命令行上传递给Shell脚本的参数、传递给函数的参数，或通过set命令得到的参数通常称做位置参数。位置参数可以依其出现的位置按顺序号引用（\$0、\$1、\$2、.....），故称做位置参数。按照惯例，\$0是Shell脚本文件的名称，由调用Shell脚本的进程（即Shell）负责设置\$0参数。\$1是第一个实际参数，\$2是第二个实际参数，.....如此类推。注意，从第10个位置参数开始，必须加用花括号，如\${10}、\${11}和\${12}等。特殊变量\$*和\$@表示所有的位置参数，\$#表示位置参数的总数。

因此，利用特殊的内部变量\$#、花括号和感叹号“!”，可以很容易地引用传递给Shell脚本的最后一个位置参数。

```
args=$#           # 传递的参数数量
lastarg=${!args}  # 注意，不能使用“lastarg=${!$#}”直接引用
```

为了改变位置参数的内容，可以使用shift命令。顾名思义，shift命令的功能就是重新分配传递给Shell脚本或函数的位置参数：把参数的位置从右到左依次左移一位，整个位置参数的总数也随之减1。例如，假定位置参数的总数为N，执行shift命令之后，将会发生\$2→\$1、\$3→\$2、.....\$N→\$N-1等变化。原来的\$1和\$N不复存在，但\$0（即Shell脚本的文件名）保持不变。

利用shift命令，可以依次处理每一个位置参数。因此，如果采用适当的循环结构与test语句，即可处理任意数量的位置参数。尽管使用花括号也可以做到这一点，但shift命令更灵活、更方便。

下面是一个利用shift命令和until循环显示所有位置参数的例子。当使用命令行参数“The arguments submitted to the script”调用这个Shell脚本时，通过shift语句，until语句只需检查第一个位置参数是否为空即可遍历每一个位置参数。在依次处理（显示）每一个位置参数之后，整个脚本运行结束：

```
$ cat echoarglist
#!/bin/sh
until [ -z "$1" ] # 表示直至所有的参数均已得到处理再结束。
do
    echo "$1 \c"
    shift
done
echo
exit 0
$ echoarglist The arguments submitted to the script
The arguments submitted to the script
$
```

每执行一次shift命令，\$@和\$*也会依次舍弃第一个位置参数，保留其余的位置参数。同时，\$#的数量也相应地减1。示例如下：

```
$ cat echoarglist2
#!/bin/sh
echo "$# ----- $@"
shift
echo "$# ----- $@"
```

```

shift
echo "$# ----- @$"
exit 0
$ echoarglist2 1 2 3 4 5
5 ----- 1 2 3 4 5
4 ----- 2 3 4 5
3 ----- 3 4 5
$

```

下面是另一个利用\$*和\$@列出所有位置参数的例子。当使用命令行参数“one two three”调用echoarglist3脚本时，由于\$*与\$@之间的细微差别（参见6.2.3节），且\$*的引用方式不同，在终端窗口上显示的位置参数结果也不完全相同：

```

$ cat echoarglist3
#!/bin/sh
if [ -z "$1" ]
then
    echo "Usage: `basename $0` argument1 argument2 ....."
    exit 1
fi
index=1
echo "Listing arguments with \"\$*\": "
for arg in "$*"          # $*加双引号意味着把所有的参数作为一个单词处理
do
    echo "Argument #$index = $arg"
    let "index+=1"
done
echo "Entire argument list is seen as single word."
echo
index=1
echo "Listing arguments with \"\$@\": "
for arg in "$@"          # $@加双引号表示把参数看做多个单词
do
    echo "Argument #$index = $arg"
    let "index+=1"
done
echo "Argument list is seen as separate words."
echo
index=1
echo "Listing arguments with \$* (unquoted): "
for arg in $*            # 如果$*未加引号，表示把参数看做多个单词
do
    echo "Argument #$index = $arg"
    let "index+=1"
done
echo "Argument list is seen as separate words."
exit 0
$ echoarglist3 one two three
Listing arguments with "$*":
Argument #1 = one two three
Entire argument list is seen as single word.

```

```

Listing arguments with "$@":
Argument #1 = one
Argument #2 = two
Argument #3 = three
Argument list is seen as separate words.

Listing arguments with $* (unquoted):
Argument #1 = one
Argument #2 = two
Argument #3 = three
Argument list is seen as separate words.
$

```

在结束这一节的介绍之前，最后再说明一点。在编写Shell脚本时，应注意使用模块化的方法组织Shell脚本。在学习Shell编程的过程中，还应随时注意多积累，多收集一些“模型”代码片断，这对将来编写Shell脚本是非常有用的，甚至可以考虑建立一个代码片断储备库。例如，下列代码片断就是一个比较好的通用开场白，其目的是测试调用Shell脚本时是否提供了正确数量的参数。

```

if [ $# -ne $Number_of_expected_args ]
then
    echo "Usage: `basename $0` script_arguments"
    exit 1
fi

```

6.2 变量与变量替换

变量是每一种编程语言和脚本语言中都不可缺少的重要组成部分之一。在Shell中，变量可用于字符串操作、算术运算、参数传递以及其他运算。实际上，变量无非是系统分配的一个或一组内存位置，用以存储各种数据。

除了特殊字符，Shell变量名可以由任何字母、数字和下画线等字符组成，但第一个字符必须是字母或下画线。变量名的长度没有限制。

与其他编程语言不同的是，Shell并不区分变量的类型。实际上，Shell仅支持一种类型的变量，即字符串变量。也就是说，Shell中的所有变量都是字符串类型的。但对字符串变量的处理并不仅限于字符串操作。

在Shell的处理过程中，取决于上下文及采用的运算符和命令工具，Shell允许对变量执行诸如整数算术运算等多种操作。能够影响变量类型的另外一个因素是变量值的首字符。如果首字符为数字，表示还可以对变量执行整数算术运算，否则只能执行字符串操作。

6.2.1 变量分类

从用途上考虑，变量可以划分为内部变量、本地变量、环境变量、参数变量和用户定义的变量。内部变量是为便于Shell编程而由Shell设定的变量，如表示错误类型的ERRNO变量等，详见6.2.3“内部变量”一节。在代码块或函数中定义的变量，且仅在定义的范围内有效的变量称做本地变量。环境变量是为系统内核、系统命令和应用程序提供运行环境而设定的变量，常见的有PATH、HOME和LANG等变量。参数变量指调用Shell脚本或函数时传递的变量（因此，

在Shell或Shell脚本中，变量替换也称参数替换）。用户定义的变量是为运行用户程序或为完成某种特定的任务而设定的普通变量或临时变量。

广义地讲，每个进程都有一个运行“环境”，这个运行环境将影响进程的执行行为。Shell也存在一组可以引用的环境变量，其中的每个变量都赋予一定的信息。从这个意义上讲，Shell同其他任何进程是一样的。

每当开始运行时，Shell都会创建一组自己的环境变量。更新或增加新的环境变量将会引起Shell更新自己的运行环境。如果在Shell脚本中设置了环境变量，通常需要使用export命令予以公布，以便子进程能够继承。

注意：环境变量的空间分配是有一定限制的。建立太多的环境变量将会占用额外的空间，因而可能会引起问题。

6.2.2 变量赋值

变量的赋值可以采用赋值运算符“=”实现，其语法格式如下：

```
variable=value
```

在Shell中，等号“=”是一种通用的赋值运算符，可用于数字和字符串赋值。变量赋值也可用于声明变量，对变量进行初始化，或改变变量的值。另外，为使其他Shell脚本或程序能够使用定义的变量，需要利用“export variable”命令，把变量导至Shell运行环境，注意，赋值运算符“=”前后不能有空格。例如，为了定义一个EDITOR环境变量，可以使用下列命令：

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

在变量的赋值过程中，如果赋的值是由多个单词组成的字符串，应在字符串前后增加一对单（或双）引号，以便Shell能够把字符串作为一个值处理。例如：

```
$ TITLE="Name      Phone      Email-Address"; export TITLE
$
```

未初始化的变量（即未赋值的变量）的值为“null”（注意，使用未赋值的变量有时会出现问题）。利用下列变量赋值形式，即可声明一个未初始化的变量：

```
variable=
```

另外一种赋值方法是利用read语句在执行过程中为变量赋值，参见第6.3.2节有关read语句的介绍；此外，还可以在for循环结构语句中采用“for variable in word-list”的形式为变量赋值，详见第7章“Shell高级编程”。

在Shell脚本中，适当地使用变量能够提高脚本的可读性和灵活性，确保数据的一致性，而且便于维护。

6.2.3 内部变量

Shell提供一组丰富的内部变量，能够为用户的Shell编程提供支持。熟练地掌握Shell内部变量，对以后的Shell编程有极大的帮助。为了能够对Shell内部变量的概貌有一个全面的了解，也为了阅读本书的方便，以及在实际应用中作为参考，表6-1给出了部分常用的主要内部变量。

表6-1 Shell自动设置的内部变量

变量	简单说明
#	\$#变量是命令行参数或位置参数的数量
-	\$_变量是传递给Shell脚本的执行标志（参见set内部命令）
?	\$?变量是最近一次执行的命令或Shell脚本的出口状态
\$	\$\$变量是Shell脚本的进程ID。Shell脚本经常使用\$\$变量组织临时文件名，确保文件名的唯一性
_	\$_是一个特殊的变量，在Shell开始运行时，变量的初始值为Shell或其执行的Shell脚本的绝对路径名，之后就是最近执行的命令的最后一个选项或参数等。例如： \$ du -s /usr/bin /usr/sbin 103682 /usr/bin 37654 /usr/sbin \$ echo \$_ /usr/sbin \$
!	#!变量的值是最近运行的一个后台进程的PID
*	\$*变量表示所有的位置参数，其值是所有位置参数的值，每个参数均可看做一个单独的字。引用时，\$*相当于\$1、\$2、\$3 ...，表示多个参数。而“\$*”则相当于“\$1 \$2 \$3 ...”，表示一个参数
@	\$@变量类似于\$*，表示所有的位置参数，其值也是所有位置参数的值，每个参数均可看做一个单独的字。引用时，\$@相当于\$1、\$2、\$3 ...，表示多个参数。但“\$@”则相当于“\$1”、“\$2”、“\$3” ...，每个参数均为前后加引号的字符串。也就是说，参数是原封不动进行传递的，未作解释或扩展
LINENO	Shell脚本中当前执行的命令的行号。仅当调试Shell脚本时，这个变量才有意义
OLDPWD	利用cd命令改换到新目录之前所在的工作目录
OPTARG	getopts命令已经处理的前一个选项参数
OPTIND	getopts命令已经处理的前一个选项参数的索引
PPID	\$PPID是当前进程的父进程的PID
PWD	表示当前工作目录，其变量值等同于pwd命令的输出
RANDOM	每次引用这个变量时，即可生成一个均匀分布于0~32 767范围内的随机整数。如果赋予RANDOM变量一个数值，可以达到初始化随机数序列的目的
REPLY	当使用read命令读取输入数据时，如果没有指定变量参数，可以把REPLY变量用做read命令的默认变量，从而把read命令读取的输入数据赋予REPLY变量
SECONDS	\$SECONDS变量是脚本已经运行的时间（秒数）

表6-2给出了Shell使用的部分主要环境变量。

表6-2 Shell使用的环境变量

变量	说明
CDPATH	定义cd命令的检索路径。如果cd命令中指定的目的目录为相对路径名，cd命令首先会在当前目录“.”中搜寻目的目录。如果未发现目的目录，则开始检索CDPATH变量中列举的每一路径名，直至找到目的目录，并成功地改换工作目录。如果最终仍未发现目的目录，则保持当

(续表)

变量	说明
	前工作目录不变。例如，假定把CDPATH变量设置为/home/gqxing，其中存在两个子目录bin和src。如果用户当前位于/home/gqxing/bin目录中，输入“cd src”命令后，即使没有指定全路径名，仍可把工作目录改换到/home/gqxing/src中
COLUMNS	用于定义终端窗口的列宽。select内置命令使用此变量值确定数据显示的宽度，以便输出菜单选项列表。此变量值也用于确定Shell编辑窗口的列数
EDITOR	用于确定命令行编辑使用的编辑程序，通常可以设为vi或emacs等用户熟悉的编辑器
ENV	当调用非交互式Shell运行Shell脚本时，如果这个变量已经设置，Shell将会使用这个变量值指定的绝对路径文件名，作为可在当前环境中执行的初始化文件。典型情况下，.profile等初始化文件是在会话启动阶段执行的，而ENV变量指定的文件是在Shell运行的初始阶段执行的。对于ENV变量指定的文件，Shell采取与点“.”命令类似的方法解释执行，即在当前环境中执行其中的命令。另外，ENV变量指定的文件只要具有可读的访问权限即可，不必是可执行文件。但与使用点“.”命令执行脚本不同的是，执行ENV变量指定的文件时不会使用PATH变量检索命令文件，这主要是出于安全考虑。在设置ENV变量值时，其中可以包含变量替换和命令替换等
FCEDIT	用于设定fc内置命令使用的默认编辑器，以使用户能够使用熟悉的编辑器编辑命令行
HISTFILE	用于指定存储命令历史记录的文件，默认的文件为~/.sh_history
HISTSIZE	用于设定命令历史缓冲区保存的最大命令记录数量，默认值为128（条命令）
HOME	用户主目录的路径名（也是cd命令的默认参数）。普通用户的主目录通常为/home/username，超级用户的主目录为根目录（/）。注意，在Solaris系统中，普通用户的主目录为/export/home/username
IFS	字段分隔符（默认值为空格、制表符和换行符）。字段分隔符通常用于解析命令行或字符串的基本构成元素。简言之，字段分隔符决定了Shell在解析命令行或字符串时怎样确定字段或单字等基本元素的边界。IFS变量值中的第一个字符用于解析\$*变量中的位置参数
LANG	用于设置语言环境，参见第9章“用户管理”
LC_ALL	用于统一设置LC_*系列变量的值
LC_CTYPE	用于确定系统怎样处理各种语言环境的字符集，包括字符的分类，字符的大小写转换规则，以及其他字符属性
LC_MESSAGES	用于确定采用何种语言输出系统提示信息
LC_NUMERIC	用于确定本地化千分数值的显示格式
LINES	用于定义终端窗口的行宽。select内置命令使用此变量值确定数据显示的高度，以便输出菜单选项列表。变量值可用于确定Shell编辑窗口的行数
MAIL	用于定义用户邮箱的路径文件名
MAILCHECK	指定Shell检测邮件的频度（以秒为单位），默认值为60秒。如果未设置这个变量，或把变量设置为小于或等于0的数值，Shell将停止检测邮件
MAILPATH	定义系统检查是否有新邮件到来的文件名
PATH	指定命令的检索路径及顺序（普通用户的命令检索路径通常包括/bin和/usr/bin等目录，超级用户的命令检索路径通常包括/sbin、/bin、/usr/sbin和/usr/bin等目录）。当用户输入命令时，Shell将会根据PATH变量列举的目录及顺序，从中检索并执行匹配的可执行程序。

(续表)

变量	说明
	如果提交的命令其目录不在检索路径中, 必须输入命令的完整路径名才能执行。另外, 命令检索路径的顺序是非常重要的。当检索路径中的不同目录存在同名的命令时, Shell 将会使用第一个发现的命令。例如, 假定PATH变量定义为PATH=/bin:/usr/bin:/usr/sbin:\$HOME/bin, 且存在两个sample命令文件, 分别位于/usr/bin和/home/gqxing/bin目录时, 如果输入sample命令时未指定完整的路径名, Shell将会调用/usr/bin目录下的sample命令。定义PATH环境变量时, 目录之间需加冒号“:”分隔符。通常, PATH环境变量是由/etc/profile及\$HOME/.profile等初始化文件设定的。在Shell脚本中, 也可以把某个目录临时加到命令检索路径中。一旦终止脚本的运行, 立即恢复到之前的PATH变量设置(作为一个子进程, Shell脚本不能改变父进程Shell的运行环境)。注意, 从安全的角度考虑, \$PATH通常不应包含当前工作目录
PPID	表示父进程的PID
PS1	第一级Shell命令提示符, 或称主提示符。通常, 普通用户的命令提示符为“\$”, 超级用户的命令提示符为“#”。详见第9章“用户管理”
PS2	第二级Shell命令提示符, 其默认值为大于号“>”。如果输入的命令不完整, 或需要用户提供附加的数据时, 系统将会按此变量的设置提示用户继续输入
PS3	第三级命令提示符, 其默认值为“#? ”。这个变量主要用于设置select循环控制结构使用的菜单选择提示符。详见第7章“Shell高级编程”
PS4	第四级命令提示符, 其默认值为加号“+ ”。这个变量主要用做Shell脚本的调试标志符, 在跟踪脚本执行的过程中, Shell将会在显示其执行的每一个命令之前, 首先输出这个变量定义的数值。详见第7章“Shell高级编程”
SHELL	Shell命令文件的完整路径名。vi等工具使用这个变量值作为默认的Shell
TMOUT	用于定义用户与系统会话过程的超时值。在一个交互式的Shell中, TMOUT变量值可以解释为自输出命令提示符之后等待用户输入的时间(以秒为单位)。Shell输出命令提示符之后, 如果在TMOUT规定的时间之内未输入任何命令, Shell将会因超时而终止执行, 导致系统关闭用户的终端窗口, 或者断开用户的终端连接。对于read内置命令而言, 如果TMOUT变量值大于0, 这个数值可以用做read内置命令默认的超时值。对于select内置命令而言, 如果在TMOUT规定的时间之内一直没有收到输入数据, select命令将会终止执行

6.2.4 变量引用与替换

使用变量的主要目的是存储或引用其中的数据值。在大型应用程序、Shell或Shell脚本中, 经常使用变量, 引用变量中的数值。引用变量中的数值称做变量替换。当变量用做参数时, 其中的变量替换也称参数替换。

假定variable是一个变量, 在变量名字前面加上一个美元符号“\$”前缀即可引用变量的值, 也即使用变量中存储的数值替换变量名字本身。在Shell中, 引用变量的值(或者说变量替换)主要有以下三种形式:

- \$variable
- \${variable}
- "\$variable"或"\${variable}"

但在下列情况下, 引用变量时无需在变量名字前面加美元符号“\$”前缀:

- 声明语句;
- 赋值语句;
- unset、export命令;
- 引用系统定义的信号。

请注意区分变量的名字及其引用的数值。通常，如果variable是变量的名字，那么\$variable、\${variable}、"\$variable"或"\${variable}"就表示引用变量的值。例如，在执行下列变量赋值语句之后，echo命令中变量引用\$variable意味着显示其对应的数值100:

```
$ variable=100
$ echo $variable
100
$
```

在下列变量设置中，第一个PATH变量定义了命令的检索路径，第二个PATH变量定义采用变量替换的形式，重新定义了PATH变量，把PATH变量的值以及用户主目录下的子目录bin和当前目录赋值到PATH变量中:

```
$ PATH=/bin:/usr/bin:/sbin:/usr/sbin
$ PATH=$PATH:$HOME/bin:..
$
```

执行变量替换后，新的命令检索路径如下:

```
$ echo "Now the PATH is $PATH"
Now the PATH is /bin:/usr/bin:/sbin:/usr/sbin:/export/home/gqxing/bin:..
$
```

为了避免引起歧义，可以采用第二种变量替换形式引用变量。实际上，第一种变量替换形式只是第二种变量替换形式的简化。例如，我们可以把上述PATH变量的定义改写为:

```
PATH=${PATH}:${HOME}/bin:..
```

第三种变量引用方式是在第一或第二种变量引用方式的基础上加上双引号。这也是一种最保险的变量引用方式。注意，在某些情况下，尤其是传递参数时，第一或第二种变量引用形式并不可靠，有时还会产生意想不到的错误。例如，假定存在一个简单的Shell脚本，其功能是显示并处理接收的位置参数:

```
$ cat disparg
#!/bin/sh
echo $1
exit 0
$
```

同时又声明了一个var变量，其中含有一个由5个英文单词组成的字符串:

```
$ var="The variable contains five words"
$
```

如果分别采用上述三种变量引用形式显示同一变量的值，其输出结果如下:

```
$ disparg $var
The
$ disp ${var}
```



```
The
$ disp "$var"
The variable contains five words
$
```

双引号括住的参数可以看做一个单词，即使其中包含空格分隔符。因此，如果希望把包含多个单词的字符串看做一个参数，应注意使用双引号，以防止单词的分割。

由此可以得出一个结论：当一个变量的值是由一个不包含任何字段分隔符的整体字符串或数值组成时，上述三种变量引用方式的最终结果是完全一样的。但是，如果变量的值包含空格、制表符、换行符，以及由内部变量IFS定义的字段分隔符时，只有第三种变量引用方式才是最保险的。在以后的Shell脚本编程过程中，这一点应切实记住。

此外还要注意，使用双引号引用变量时可以进行替换，如果使用单引号，则意味着按文字引用其中的字符串，即使存在变量引用也不能进行替换。单引号表示把特殊字符\$和变量名看做文字，禁止引用其中的变量值，因而不会出现变量替换。也就是说，单引号中的每个字符，包括特殊字符均作文字常量解释。

6.2.5 变量的间接引用

假定一个变量的值是另一个变量的名字，利用第一个变量是否能够引用第二个变量的值呢？例如，如果a=b，而b=c，仅仅引用变量a是否能够返回变量b的值（c）呢？回答是肯定的。实际上，采用“eval var1=\\$\$var2”命令即可达到上述目的。这种情况称做变量的间接引用。

下面是一个间接引用变量的例子：

```
$ Message=Hello
$ Hello="Good Morning"
$ echo "Message = $Message"           # 直接引用
Message = Hello
$ eval Message=\$$Message             # 间接引用
$ echo "Now message = $Message"
Now message = Good Morning
$
```

如果把上述的最后两个命令合并到一起，其效果如下：

```
$ Message=Hello
$ Hello="Good Morning"
$ echo Message = $Message
Message = Hello
$ eval echo Now message = \$$Message
Now message = Good Morning
$
```

在Bash中，还可以采用下列方式实现变量的间接引用：

```
$ Message=Hello
$ Hello="Good Morning"
$ echo ${Message}
Hello
$ echo ${!Message}
Good Morning
```

```
$ Hello="Hello again"
$ echo ${!Message}
Hello again
$
```

6.2.6 特殊的变量替换

在Shell中，为了保证Shell、Shell脚本或应用程序能够正常地运行，Shell提供一种特殊的变量替换机制。特殊变量替换的主要功能如下：

- 对于未设置的变量，采用特殊替换表达式赋予默认值；
- 采用特殊替换表达式替换或设置默认值；
- 采用特殊替换表达式给出错误提示信息。

表6-3给出了Shell支持的部分常用特殊变量替换形式。

表6-3 特殊的变量替换

特殊变量替换	简单说明
<code>\${var}</code>	其作用同 <code>\$var</code> 一样，表示变量 <code>var</code> 的值。在某些情况下，只有采用 <code>\${var}</code> 变量引用形式，其意义才是比较明确的。例如，在定义PATH环境变量时，如果需要连接字符串变量， <code>PATH=\${PATH}:/opt/bin</code> 比 <code>PATH=\$PATH:/opt/bin</code> 更易于阅读和理解
<code>\${var:-value}</code>	如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，使用 <code>vaule</code> 作为变量 <code>var</code> 的值进行变量替换。否则，使用变量 <code>var</code> 的值进行变量替换，但变量 <code>var</code> 的值保持不变
<code>\${var:=value}</code>	如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，把 <code>value</code> 赋予变量 <code>var</code> ，同时执行变量替换
<code>\${var:+value}</code>	如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，使用 <code>null</code> 进行变量替换。如果变量 <code>var</code> 已经设置，则使用 <code>value</code> 进行变量替换，变量 <code>var</code> 的值保持不变
<code>\${var:?value}</code>	如果变量 <code>var</code> 已经设置，使用变量 <code>var</code> 的值进行变量替换。如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，使用 <code>value</code> 作为错误提示信息。如果省略了 <code>value</code> ，则输出默认的错误信息，表示变量 <code>var</code> 未设置。然后，终止Shell脚本的执行，并返回一个非0的出口状态（交互式的Shell会话例外）。变量 <code>var</code> 的值保持不变

在上述表达式中，冒号“:”意味着需要测试给定的变量`var`是否已经设置。如果变量已经设置（声明），再检查其值是否为`null`。如果省略了冒号“:”，意味着只需检查`var`是否尚未设置。表6-4总结了各种变量替换形式中有无冒号“:”的细微差别之处。

表6-4 各种变量替换形式的比较

比较	var已设置且其值为非null	var已设置但值为null	var未设置
<code>\${var:-value}</code>	使用 <code>var</code> 替换	使用 <code>value</code> 替换	使用 <code>value</code> 替换
<code>\${var-value}</code>	使用 <code>var</code> 替换	使用 <code>null</code> 替换	使用 <code>value</code> 替换
<code>\${var:=value}</code>	使用 <code>var</code> 替换	使用 <code>value</code> 赋值并替换	使用 <code>value</code> 赋值并替换
<code>\${var=value}</code>	使用 <code>var</code> 替换	使用 <code>var</code> 替换	使用 <code>value</code> 赋值
<code>\${var:?value}</code>	使用 <code>var</code> 替换	错误，退出，返回1	错误，退出，返回1
<code>\${var?value}</code>	使用 <code>var</code> 替换	使用 <code>null</code> 替换	错误，退出，返回1
<code>\${var:+value}</code>	使用 <code>value</code> 替换	使用 <code>null</code> 替换	使用 <code>null</code> 替换
<code>\${var+value}</code>	使用 <code>value</code> 替换	使用 <code>value</code> 替换	使用 <code>null</code> 替换

例如，假定变量`var`已经设置，但其值为`null`，则`${var:-undefined}`变量替换后的结果是`undefined`，示例如下：

```
$ var=
$ echo ${var}

$ echo var is ${var:-undefined},
var is undefined
$
```

在调用Shell脚本时，如果提供的命令行参数不足，采用上述默认的变量或参数替换方法很有用。例如，如果命令行中未给定文件参数，下列Shell脚本中的`command`命令（任何合法的文件操作命令）将会对`run.data`文件进行处理。

```
$ cat substitute
#!/bin/sh
def_fname=run.data
fname=${1:-$def_fname}
command $fname
exit 0
$
```

在下面的例子中，变量`var`的值为`null`。因此，`${var:=abc}`变量替换后的结果是：首先把`abc`赋予变量`var`，然后使用变量`var`的值（`abc`）进行变量替换。

```
$ unset var
$ echo ${var:=abc}
abc
$
```

在下面的例子中，变量`var`的值为`null`，且“?”后面为空。因此，`${var:=abc}`变量替换的结果是输出一条默认的出错信息：“parameter null or not set”。

```
$ unset var
$ echo ${var:?}
ksh: var: parameter null or not set
$
```

`set`命令可用于设置参数。在下面的例子中，`set`命令设置的位置参数`$1`、`$2`和`$3`，其值分别为`a`、`b`和`c`。如果使用`${3:+posix}`进行变量替换，其结果是`posix`。

```
$ set a b c
$ echo $3
c
$ echo ${3:+posix}
posix
$
```

在上述的特殊变量替换形式中，变量名后面的参数不仅可以是任何字符串，还可以是变量替换、命令替换和算术表达式的计算结果。

假定之前并未设置`username`变量，下列变量替换的结果是“who am i”命令输出的第一个字段：

```
$ echo ${username-`who am i | cut -d' ' -f1`}
root
$
```

在下面的例子中，只要变量username已经声明，不管其值设置与否，变量替换的最终结果总是“uname -n”命令的输出：

```
$ hostname=
$ echo ${hostname+`uname -n`}
iscas
$
```

在下面的例子中，仅当变量var未声明，或即使已经声明但变量值为null，才使用pwd命令的输出（关于\$(pwd)形式的命令替换，参见6.3节“命令与命令替换”）替换变量var的值。

```
$ echo ${var:-$(pwd)}
/export/home/gqxing
$
```

表6-5给出了其他有用的特殊变量替换形式。

表6-5 其他特殊变量替换形式

其他特殊变量替换	简单说明
<code>\${#var}</code>	字符串的长度（即字符串变量var中的字符数量）。对于数组来讲， <code>\${#array}</code> 是数组中第一个元素的长度。注意下列例外情况： <code>\${#*}</code> 和 <code>\${#@}</code> 给出的是位置参数的个数，而针对数组的 <code>\${#array[*]}</code> 和 <code>\${#array[@]}</code> 给出的是数组元素的个数
<code>\${var#pattern}</code> , <code>\${var##pattern}</code>	从\$var变量值的前部删除与给定模式匹配的最短或最长部分子串。如果应用于文件路径名， <code>\${var##pattern}</code> 的效果相当于basename命令
<code>\${var%pattern}</code> , <code>\${var%%pattern}</code>	从\$var变量值的后部删除与给定模式匹配的最短或最长部分子串。其中， <code>\${var%pattern}</code> 可用于抽取路径名的目录部分

在下面的例子中，变量DOCS已经声明，且设置为/docs/posix，变量替换的结果是给出\$DOCS变量值的字符串长度：

```
$ DOCS=/docs/posix
$ echo ${#DOCS}
11
$
```

在下面的例子中，变量var已设置为file.c，变量替换的结果是从整个文件名中抽取除“.c”后缀之外的文件名部分：

```
$ var=file.c
$ echo ${var%.c}.o
file.o
$
```

在下面的例子中，变量var已经声明并设置为posix/src/std，变量替换的结果是从路径名后部删除首次出现并包含斜线字符“/”的最长字符串。

```
$ var=posix/src/std
$ echo ${var%%/*}
posix
$
```

在下面的例子中，变量var已设置为\$HOME/src/cmd，变量替换的结果是抽取整个路径名除\$HOME变量值之外的目录名部分。

```
$ var=$HOME/src/cmd
$ echo ${var#$HOME}
/src/cmd
$ echo ${var#$HOME/}
src/cmd
$
```

在下面的例子中，变量var设置为/one/two/three，变量替换的结果是抽取路径名的文件名部分。

```
$ var=/one/two/three
$ echo ${var##*/}
three
$
```

在下面的例子中，变量var设置为/one/two/three，变量替换的结果是抽取路径名的目录部分。

```
$ var=/one/two/three
$ echo ${var%/*}
/one/two
$
```

6.2.7 变量声明与类型定义

前面说过，Shell并不严格区分变量的类型，一切取决于变量存储的内容。在第一次引用时，Shell将根据变量的内容确定变量的类型。为了加快Shell的运算速度，提高Shell编程的严谨性，在Korn Shell或Bash中，可以使用typeset命令定义变量的类型，并在定义时对变量进行初始化。

变量无类型之分既是好事，也是坏事。这虽然增加了脚本编程的灵活性，简化了代码的编写。但也容易产生潜在的错误，容易给自己套上绳索，造成之后阅读的困难，养成不好的编程习惯。

注意：记住脚本变量的类型是Shell编程人员的责任，不要指望Shell做类型检查。

typeset内部命令用于设定变量的属性。利用typeset命令的“-i”选项，可以把变量声明为整数变量。在声明变量的同时，也可以为变量赋值，进行初始化。事先声明为整数的变量可以直接执行算术运算，而不需要使用expr和let命令，例如：

```
$ typeset -i number          # 把变量number声明为整数变量
$ number=3                  # 之后可以使用整数为变量number赋值
$ echo "Number = $number"
Number = 3
$ number=three              # 但不能使用字符串为变量number赋值
ksh: three: bad number      # bash中不会出现此错误信息
$ echo "Number = $number"
```



```
Number = 3                # 在bash中，number变量的值为0
$
```

如果事先并未把变量声明为整数变量，Shell将会把赋予变量的值看做字符串，例如：

```
$ n=6/3
$ echo "n = $n"
n = 6/3
$ typeset -i n
$ n=6/3
$ echo "n = $n"
n = 2
$
```

利用“-r”选项还可以把变量声明为只读变量。此后，任何试图修改只读变量的操作将会失败，并产生错误信息。如“typeset -r var”将会把变量var声明为只读变量。

6.3 命令与命令替换

6.3.1 Shell内部命令

UNIX系统提供大量的命令供用户使用。但出于性能方面的考虑，Shell也提供若干具有同样功能的内部命令。内部命令的执行速度比外部命令要快得多，这是因为在解释执行内部命令时，Shell不需要创建子进程。而执行外部命令时需要创建单独的新进程，从而加大了系统的开销。

某些Shell内部命令与系统命令具有相同的名字，但其实现方式不同。例如，Shell内部命令echo与系统命令/bin/echo的名字完全一样，功能也相同，但其执行效率并不相同。

例如，下面两条命令的作用都是在终端窗口上显示一条信息：

```
$ echo "This line uses the \"echo\" builtin to output."
This line uses the "echo" builtin to output.
$ /bin/echo "This line uses the /bin/echo system command to output."
This line uses the /bin/echo system command to output.
$
```

表6-6给出了Shell提供的部分主要内部命令及说明。

表6-6 Shell内部命令

内部命令	简单解释
.	用于读取指定的Shell脚本文件，并在当前的Shell环境中执行，运行结束后返回最后执行的一条命令的出口状态。如果指定的文件名参数不是绝对路径，使用PATH变量指定的目录检索脚本文件。如果PATH变量指定的目录中不存在指定的文件，将会在当前目录中检索给定的脚本文件。在Shell脚本中使用点命令时，可以把指定的源文件读入当前脚本中，并从当前位置开始执行。当多个Shell脚本共用同一个数据文件或函数时，点命令是非常有用的。点命令最重要的用途也许就是执行环境变量设置脚本。在此情况下，即使退出环境变量设置脚本，环境变量的设置仍然保持有效。如果直接运行环境变量设置脚本，在脚本终止运行之后，脚本中设置的环境变量也将随之消失而不复存在

(续表)

内部命令	简单解释
:	null语句。本身不做任何处理动作，总是返回一个值为0的出口状态。其功能一是用做无限循环语句的测试条件；二是作为一种特殊的命令形式，令Shell处理紧随其后（与“:”位于同一行）的变量替换或命令替换
alias	alias命令用于设置或显示系统及用户定义的命令别名
bg	把指定的作业置入后台运行模式。如果未给定作业号，则把当前作业置入后台运行模式
break	用于退出for、while和until等循环语句
cd	改变目录。用于转换到指定的目录
continue	用于结束for、while和until等循环语句中的当前循环，并立即开始执行下一轮循环
echo	用于输出字符串、变量值或表达式的计算结果等参数。通常，echo命令将会在输出信息之后附加一个换行字符
eval	用于读取并组合随后的命令参数，然后提交Shell执行
exec	如果exec命令带有参数，则使用命令参数代替当前的Shell进程，而不是像通常那样创建一个新进程，执行给定的命令。如果在Shell脚本中使用exec命令，当由exec命令引入的命令终止运行时，将会强制Shell脚本随之终止执行。因此，如果Shell脚本中有exec命令，通常应作为最后一条命令。如果未带参数，exec命令的作用只是按照I/O重定向的要求，修改文件描述符。例如，“exec < fname”命令意味着使用给定的文件fname代替标准输入
exit	用于无条件地终止Shell脚本的执行。exit命令可以带一个整数参数，作为Shell脚本的出口状态返回Shell。如果使用单独的exit命令（省略整数参数）终止脚本的执行，整个Shell脚本的出口状态则是在exit之前执行的最后一条命令的出口状态。一个好的编程习惯是在脚本执行结束处增加一条“exit N”命令，表示运行成功或出现的不同错误情况。此外，文件结束标志也会引起Shell脚本终止执行
export	export命令用于公布或导出设置的变量，使变量的设置能够用于正在运行的脚本或Shell的所有子进程。export命令的一个重要用途是在profile文件中初始化环境变量，使随后的所有进程都能够访问
fc	fc命令用于列出、编辑或重复执行命令历史缓冲区或文件中记录的命令
fg	把指定的作业置入前台运行模式。如果未给定作业号，则把当前作业置入前台运行模式
getopts	用于解析传递给Shell脚本的命令行参数。getopts使用两个变量：其中的OPTIND是指向下一个命令行选项的指针；OPTARG是与选项有关的参数。getopts命令通常主要用于while循环语句，通过循环方式处理所有的选项和参数
hash	用于记录给定命令的路径名，以便Shell或Shell脚本随后调用hash命令时不必再按PATH变量指定的目录检索命令。当调用hash命令而未给定参数时，系统仅会显示当前Shell散列表中记录的命令
jobs	显示指定或全部活动的作业。其中的“-l”选项表示显示附加的进程ID信息，“-n”选项表示仅显示已经停止或终止运行的作业
kill	用于向指定的进程或作业发送信号，从而终止相应的进程或作业。信号可以是一个数字代码或信号名字
let	let命令用于实现算术运算
print	按照给定的控制格式输出参数的值。输出格式是一个字符串，其中包括三种字符对象：一为普通字符，可直接输出到标准输出；二为转义字符序列，用于输出特殊字符；三为格式规范，用于定义参数值的输出形式。参见标准的printf()格式定义

(续表)

内部命令	简单解释
pwd	显示当前的工作目录。其作用等同于读取PWD变量的值。换言之，`pwd`命令替换的结果等同于PWD变量的值
read	用于从标准输入中读取数据，并把数据赋值到给定的变量中。也就是说，read命令能够以交互方式读取来自键盘的输入数据。“-r”选项意味着忽略转义符号“\”，使之仅作为普通字符处理
readonly	把指定的变量设置为只读变量，因而不允许在随后的操作中对相应的变量进行赋值。如果试图修改相应的变量，将会产生一个错误信息
return	引起Shell函数返回主程序的调用点。在非Shell函数的情况下，return命令相当于exit命令
set	<p>set命令用于改变内部变量或脚本变量的值。set命令的一个重要用途是重置位置参数（例如，set `command`）。如果不带任何参数，set命令将会输出所有环境变量的当前设置，这是set命令的另一个重要用途。set命令的部分选项说明如下：</p> <ul style="list-style-type: none"> -a 使定义的所有变量和函数设置能够自动导入Shell运行环境。 -e 如果其中任何一条命令结束后返回非零值的出口状态，立即终止Shell脚本的执行。 -f 禁用文件名生成机制。 -m 监控模式（启用做业控制功能）。在交互式Shell中，这个选项的默认设置为on -n 读取Shell脚本中的命令，检查是否存在语法错误，但不执行。 -o 使用下列参数设置各种标志： <ul style="list-style-type: none"> • emacs: 使用emacs作为命令行编辑器。 • noclobber: 防止标准输出(>)重定向覆盖现有的同名文件。一旦打开此标志，仅当使用“> ”时才能清除现有的文件。 • nolog: 不允许把函数定义写入命令历史文件中。 • verbose: 同“-v”选项。 • vi: 使用vi作为命令行编辑器。在vi命令行编辑环境中，一开始总是处于命令模式。 • 如果不加任何参数，set命令将会输出当前的各种标志设置。 -t 读入并在执行任何一条命令之后立即终止Shell脚本的运行。 -u 执行变量替换时，事先未设置的变量按错误处理。 -v 显示Shell读入的命令语句。 -x 显示执行的命令及其参数。 - 关闭“-x”和“-v”标志。 -- 重新设置位置参数，或清除位置参数的设置
shift	把位置参数\$2, \$3,, \$N依次重新命名为\$1, \$2,, \$N-1, 原来的\$1和\$N不复存在，位置参数总数相应地减1，故新的\$#变量值也比原来的\$#变量值少1
stop	终止执行指定的后台作业或进程
test	计算条件表达式。参见6.4节“test语句”
times	显示Shell或进程累计占用的用户和系统时间
trap	trap命令主要用于Shell脚本的信号捕捉与错误处理。当收到指定的信号时，Shell将会读入并执行指定的命令。如果指定的命令为null(“”), Shell将会忽略收到的每一个信号。在trap语句定义的处理动作结束之后，其返回值将是调用trap之前执行的最后一个命令的出口状态。如果不希望在执行过程中被Ctrl-C等按键打断，可以在Shell脚本中加入trap命令。详见第7章“Shell高级编程”
type	<p>type命令用于说明Shell如何解释指定的命令，显示命令的完整路径名。例如：</p> <pre>\$ type find find is /usr/bin/find</pre>

(续表)

内部命令	简单解释
	<pre>\$ type echo echo is a shell builtin \$ type history history is an exported alias for fc -l \$</pre>
typeset	用于声明或设置Shell变量，定义变量的属性。其支持的选项如下： -f 用于声明一个函数或显示函数定义 -i 把指定的变量定义为整数变量。当赋予变量值时，按算术扩展的要求执行算术运算 -r 把指定的变量定义为只读变量。在随后的处理过程，不能为只读变量赋值 -x 公布指定的变量，以便其他命令或Shell脚本能够继续使用
ulimit	用于设置或显示用户的资源限制。资源限制的值可以是一个数字，也可以是文字值unlimited。 “-H”和“-S”选项分别用于指定硬性限制和软性限制。一旦设置，硬性限制值不能增加，但软性限制值可以增加，直至达到硬性限制值。如果“-H”或“-S”选项均未指定，则同时适用于软硬性限制。其他选项的意义说明如下（其中“-f”为默认的选项）： -a 显示当前所有的资源限制 -c 显示内存映像转储文件（core）的最大容量限制（以1KB的数据块为单位） -d 显示进程可用数据区（段）的最大容量限制（以KB为单位） -e 显示可用的nice优先级调整值的最大值 -f 显示可以创建的最大文件的容量限制（以1KB的数据块为单位） -n 显示可用的文件描述符的最大数量限制 -s 显示可用的最大栈区的容量限制（以KB为单位） -t 显示每个进程可用的最大CPU时间量限制（以秒为单位） -u 显示单个用户可用的最大进程数量限制 -v 显示可用的虚拟内存的最大数量限制（以KB为单位）
umask	指定创建文件时应设定的默认访问权限。参见第4章“文件和目录操作”
unalias	撤销已经设定的命令别名。“-a”选项意味着撤销当前Shell运行环境中已经定义的所有命令别名
unset	用于清除Shell变量，把变量的值设为null。注意，这个命令并不影响位置参数
wait	等待指定的作业结束，显示作业结束时的出口状态。如果未指定作业号，意味着等待当前作业及其所有子进程的结束。例如，“wait %1”意味着等待作业号为1的后台进程执行结束。 “wait 6618”意味着等待进程ID为6618的后台作业完成

6.3.2 部分命令介绍

1. “.”、source与exec命令

当Shell运行一个Shell脚本时，通常需要创建一个新的Shell进程，新的Shell进程将会继承其父Shell进程的环境变量（包括全局变量和已公布的变量），但不会继承未公布的本地变量。在Shell脚本中设置的变量，不管公布与否，只要退出Shell脚本，其中设置的变量将随之消失。为了利用Shell脚本设置应用程序可用的变量，通常需要使用“.”或source（Bash）内部命令，在当前的Shell环境中解释执行Shell脚本。

与“.”命令类似，exec也是在当前进程的进程空间中执行指定命令的。但与“.”命令不同的是，“.”命令只能运行Shell脚本，而exec既可以运行Shell脚本，也可以运行编译的程序或

命令。而且，在“.”命令调用的Shell脚本运行结束之后，将会返回调用点，但exec则不返回。此外，“.”命令运行的Shell脚本能够访问当前进程空间中的本地变量。

exec内部命令主要有两个用途：运行指定的命令时无需创建新的进程；重定向Shell脚本中的文件描述符。由于exec在运行指定的命令时并不创建新的进程，因而命令的执行速度更快。但是，由于exec命令并不返回到调用位置，因此只能用做Shell脚本中最后执行的一个命令。例如，在下列Shell脚本中，最后一个echo命令是不会执行的。

```
$ cat exec_demo
uname -n
exec date
echo "This line is never displayed."
$ exec_demo
iscas
Tue Jun 23 12:46:34 CST 2009
$
```

exec命令的第二个主要用途是重定向Shell脚本中的文件描述符，包括标准输入、标准输出和标准错误输出。如果Shell脚本中的某个位置出现下列命令，在随后读取标准输入时，其数据均取自exec命令指定的文件infile：

```
exec < infile
```

同样，下列exec命令将会把Shell脚本中随后命令的标准输出和标准错误输出分别重定向到指定的文件outfile和errfile：

```
exec > outfile 2> errfile
```

当以上述方式使用exec命令时，不会替代当前进程，exec命令之后仍然能够附加其他命令。

在使用exec命令重定向标准输出和标准错误输出之后，为了使用户能够看到Shell脚本中输出的任何提示信息，可以使用/dev/tty设备文件克服这一矛盾。/dev/tty是一个伪设备文件，表示用户当前使用的终端窗口（和键盘）。无论何时，都可以使用这个设备文件引用用户的终端窗口，而不必知道当前用户终端的实际设备文件名是什么（如果需要，可以使用tty命令获取用户终端的实际设备文件名）。

把Shell脚本中的标准输出重定向到/dev/tty，可以保证任何提示信息都能送达用户的终端窗口，而无需顾及用户究竟使用哪一个终端注册到UNIX系统。另外，即使整个Shell脚本的标准输出和标准错误输出均重定向到某个文件，其间发送到/dev/tty的输出信息也不会受影响。

例如，下列三个echo命令分别把输出信息送到标准输出、标准错误输出和用户的终端窗口，当重定向整个Shell脚本的标准输出和标准错误输出时，发送到/dev/tty的输出信息仍然能够送达用户的终端窗口，而outfile和errfile则分别存有发送到标准输出和标准错误输出的信息：

```
$ cat exec_demo2
echo "message to standard output"
echo "message to standard error" 1>&2
echo "message to the user" > /dev/tty
$ exec_demo2 >outfile 2> errfile
message to the user
$ cat outfile
message to standard output
```



```
$ cat errfile
message to standard error
$
```

下列**exec**命令直接把Shell脚本的标准输出重定向到用户的终端窗口（实际上，这种重定向有点画蛇添足）：

```
exec > /dev/tty
```

如果把上述**exec**命令加到第一行，创建新的脚本文件**exec_demo3**，由于已经把Shell脚本中的标准输出重定向到**/dev/tty**，此时，即使再把Shell脚本本身重定向到另一个文件，发送到标准输出的信息仍然会输出到用户的终端窗口。但是，发送到标准错误输出的信息则不受影响。

```
$ cat exec_demo3
exec > /dev/tty
echo "message to standard output"
echo "message to standard error" 1>&2
echo "message to the user" > /dev/tty
$ exec_demo3 >outfile 2> errfile
message to standard output
message to the user
$ cat outfile
$ cat errfile
message to standard error
$
```

在Shell脚本中，采用**exec**命令重定向的一大优点是，不必在随后需要重定向的每个命令中一一进行I/O重定向。

除了表示用户的终端窗口，**/dev/tty**也表示终端的键盘输入。如果在Shell脚本中使用下列**exec**命令，意味着读取用户终端的键盘输入（实际上，这种重定向也属画蛇添足）。之后，读取标准输入的所有命令均需接收来自终端键盘的输入。

```
exec < /dev/tty
```

2. “:” 与true命令

实际上，“:”与**true**命令并不执行任何处理动作，其作用只是返回一个出口状态为0的测试条件。尽管两个命令的功能一样，但与“:”不同的是，**true**命令是UNIX系统提供的一个外部命令。这两个命令经常用于实现不定循环，如用做**while**循环结构中的无限循环测试条件。显然，循环语句中还应增加一个**break**语句，以便当某个条件满足时能够退出循环。当然，也可以使用**Ctrl-C**键等方式强行中断循环语句的运行。

例如，当我们需要连续地观察一个日志文件或备份文件的大小是否按预期的那样不断增长时，可以使用下列命令，每5秒钟显示一次：

```
$ while true
> do
> ls -l backupfile
> sleep 5
> done
-rw-r--r--  1 gqxing  other      11111 Jun 23 12:51 backupfile
```

```
-rw-r--r--  1 gqxing  other      22222 Jun 23 12:51 backupfile
-rw-r--r--  1 gqxing  other      33333 Jun 23 12:51 backupfile
^C
$
```

3. echo命令

echo命令也许是UNIX系统中应用最广泛的命令之一。**echo**命令主要用于显示各种信息，也可用于显示文件列表。作为显示信息的字符串，可以直接写在**echo**语句后面，也可以在字符串前后加引号，例如：

```
$ echo Please enter your choice:
Please enter your choice:
$ echo *.c
atmcom.c atmmon.c handler.c listerner.c
$
```

为了组织信息的显示格式，**echo**语句还支持少量的特殊字符，兹列举如下：

- **\n** 表示在相应的位置插入一个换行符号；
- **\t** 表示在相应的位置插入一个制表符；
- **\b** 表示在相应的位置插入一个退格符号（Backspace）；
- **\c** 在输出所有的字符串参数之后，**echo**语句通常还会输出一个换行符，使用“**\c**”意味着取消输出换行符；
- **\0N** 其中N可以是一个1、2或3位的八进制数值，表示一个ASCII字符。

4. read命令

read命令的主要功能是读取来自标准输入的数据，然后存储到指定的变量参数中，实现交互式的变量赋值。**read**命令的语法格式简写如下：

```
read [-r] varnames
```

在读取数据时，**read**命令通常会把“****”字符作为转义符号处理，“**-r**”选项表示按普通字符处理。

在实际应用中，为了在运行时直接从用户终端中读取数据，并为变量赋值，可在Shell脚本中增加下列**read**命令：

```
$ cat install
.....
echo "Enter a Serial number, then press <ENTER>."
read SNO
.....
$
```

如果**read**命令后面列有多个变量参数，输入的数据将按空格（IFS变量定义的字段分隔符，默认情况下为空格）分隔的单词顺序依次为每个变量赋值。如果输入的数据多于变量参数，多余的数据将会全部赋予最后一个变量参数：

```
$ cat readdata
echo "Please enter some data: \c"
read word1 word2 word3
```

```

echo "Word 1 is: $word1"
echo "Word 2 is: $word2"
echo "Word 3 is: $word3"
$ readdata
Please enter data: This is something
Word 1 is: This
Word 2 is: is
Word 3 is: something
$ readdata
Please enter data: This is something else
Word 1 is: This
Word 2 is: is
Word 3 is: something else
$

```

read命令的标准输入也可以重定向到一个文件。如果文件中含有多行数据，只有第一行数据会赋值到变量中。但是，如果利用循环语句，可以依次读取文件的每一行数据。如果成功地读取了任何输入数据，**read**命令将会返回一个值为0的出口状态。如果遇到文件结束标志，即读取了EOF（End Of File）标志字符，**read**命令将会返回一个非0值的出口状态。因此，可以使用**read**命令的这一特性作为循环结束的判断条件。下面是一个利用管道实现文件I/O重定向，从中读取数据且为变量赋值的例子：

```

$ cat install2
#!/bin/sh
while read line
do
    set $line
    if [ "$4" = "swap" ]
    then
        .....
    fi
done < /etc/vfstab
$

```

在Bash中，**read**命令还支持其他选项，用于显示提示信息，禁止回显键盘输入数据，控制输入超时，读取指定数量的字符（而不必非按Enter键不可）等。其语法格式简写如下：

```
read [-d delim] [-n nchars] [-p prompt] [-s] [-t timeout] varnames
```

其中，“-d”选项表示使用指定的分隔符（而不是默认的换行符）作为输入数据的终止符。

“-n”选项表示仅接受指定数量的输入字符。如果输入的字符少于指定数量，需要按下Enter键，否则Shell将会一直处于等待状态。“-p”选项表示首先在标准错误输出中显示一个提示信息（无换行，光标停留在提示字符串最后一个字符之后），然后等待用户输入。这个选项仅适用于从键盘中读取用户输入的数据。“-s”选项表示禁止显示任何输入字符。“-t”选项用于控制输入超时。如果在指定的时间（秒）内无法读取一个完整的输入行，将会引起**read**命令超时，并返回一个错误信息。如果不是从一个用户终端或管道读取输入数据，这个选项不起任何作用。

例如，使用“-p”选项，可以省略echo语句，在**read**语句中直接增加提示信息。因此，上述代码的前两行可以合并为一个**read**语句：

```
read -p "Please enter some data: " word1 word2 word3
```

下列read命令用于读取单个字符：

```
$ read -s -n1 -p "Hit a key " keypress
$ echo "Keypressed was \"${keypress}\"."
```

5. set与unset命令

set命令的一个重要用途是修改或重新设置位置参数的值。按照Shell的规定，用户不能直接为位置参数赋值。但是，利用set命令及其参数变量，则可以修改或重新设置位置参数的值。例如：

```
$ cat reset
echo $1 $2 $3 $4
set value1 value2
echo $1 $2 $3 $4
$ reset This is an argument
This is an argument
value1 value2
$ cat retain
echo $1 $2 $3 $4
set $1 value1 value2 $4
echo $1 $2 $3 $4
$ retain This is an argument
This is an argument
This value1 value2 argument
$
```

使用“set --”命令，也可以把参数变量的值赋予位置参数。如果“set --”命令后面未给定参数变量，意味着清除所有的位置参数。下面是一个重新分配位置参数的例子。

```
$ var="one two three four five"
$ set -- $var
$ echo $1 $2 $3 $4 $5
one two three four five
$ echo "$@"
one two three four five
$ set -- # 清除当前的所有位置参数
$ echo "$1"
$ echo "$@"
$
```

实际上，当使用给定的参数变量设置位置参数时，双减号“--”并非必需的，因而可以省略。下面是一个根据“uname -a”命令的输出，利用set命令设置位置参数的例子：

```
$ uname -a
SunOS iscas 5.10 Generic_137138-09 i86pc i386 i86pc
$ set `uname -a`
$ echo "Positional parameters after set `uname -a` :"
Positional parameters after set `uname -a` :
$ echo "Field 1 = $1"
Field 1 = SunOS
$ echo "Field 2 = $2"
```

```
Field 2 = iscas
$ echo "Field 3 = $3"
Field 3 = 5.10
$
```

如果不带任何选项或参数，**set**命令将会列出所有的环境变量，已经声明或设置的其他变量，以及函数定义等。这是**set**命令的另一个重要用途，例如：

```
$ set
.....
HOME=/export/home/gqxing
HZ=100
IFS='
'
LANG=zh
LINENO=1
LOGNAME=gqxing
.....
$
```

unset命令用于清除Shell变量，把变量的值设置为null。注意，这个命令并不影响位置参数，例如：

```
$ variable=hello
$ echo "$variable"
hello
$ unset variable
$ echo "$variable"
$
```

Shell既是一个命令解释程序，提供用户与UNIX系统的命令行界面，又是一个普通的系统命令，如不特别指定，Shell将会按照默认的环境设置运行。在Shell的运行过程中，如果需要，也可以使用**set**或**shopt**（Bash）命令修改Shell的默认处理行为，定制自己的运行环境。为了启用Shell的某种功能特性，可以使用**set**命令的“-o”选项，而**set**命令的“+o”选项则用于关闭相应的功能特性。如果不加任何选项和参数，“set -o”命令将会显示**set**命令能够控制的每一个特性参数及其开关状态。例如，为了启用**noclobber**特性，防止重定向操作无意中覆盖同名的文件，可以使用下列命令：

```
$ set -o noclobber
$
```

为了关闭这一功能特性（默认），可以使用下列命令：

```
$ set +o noclobber
$
```

6. expr命令

expr命令用于计算表达式的值，然后把计算结果送到标准输出。其中的表达式可以是字符串比较表达式、整数算术运算表达式或模式匹配表达式。其语法格式简写如下：

```
expr expression
```


表6-7给出了expr命令支持的各种字符串比较表达式，以及计算结果的说明。expr命令基本上已被test命令或“[[...]]”结构所取代，且在“[[...]]”结构中，“>”和“<”符号之前也无需加转义符号，故现在很少使用expr命令做字符串比较（但是，test命令和“[[...]]”结构不支持“>=”和“<=”比较）。

表6-7 expr命令支持的字符串比较表达式

表达式	计算结果
str1 = str2	如果字符串str1等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
str1 \> str2	如果字符串str1大于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
str1 \< str2	如果字符串str1小于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
str1 \>= str2	如果字符串str1大于等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
str1 \<= str2	如果字符串str1小于等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
str1 != str2	如果字符串str1不等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1

下面的例子说明了怎样利用expr命令测试字符串表达式：

```
$ TIMEZONE=CST
$ expr "$TIMEZONE" = "CST"
1
$ echo $?
0
$ expr "$TIMEZONE" = "GMT"
0
$ echo $?
1
$
```

expr命令也可用于计算整数表达式的值，计算结果发送到标准输出。因此，如果想把计算结果值保存到一个变量中，需要采用命令替换的方式实现。

表6-8给出了expr命令支持的各种整数算术运算表达式，以及有关计算结果的说明。expr命令仅支持整数表达式的计算，如果给定的参数不是整数，expr命令将会输出一个出错信息。

表6-8 expr命令支持的整数算术运算表达式

表达式	简单说明
exp1 + exp2	计算整数表达式exp1与exp2的和
exp1 - exp2	计算整数表达式exp1与exp2的差
exp1 * exp2	计算整数表达式exp1与exp2的乘积。注意，星号前应加转义符号“\”
exp1 / exp2	计算整数表达式exp1与exp2的商
exp1 % exp2	计算整数表达式exp1与exp2的余数

下面的例子说明了怎样利用expr命令计算整数表达式，以及怎样利用命令替换的方式保存expr命令的计算结果。

```
$ n=3
$ expr $n + 7
10
$ n=`expr $n + 1`
$ echo $n
4
$
```

7. let命令与“((...))”结构

let命令和双括号“((...))”结构用于计算和测试整数算术表达式，执行整数算术运算。实际上，let命令和“((...))”结构是对expr命令的简化，取代并扩展了expr命令的整数算术运算功能。

除了上述5种算术运算之外，let命令和“((...))”结构还支持+=、-=、*=、/=和%=等运算符。此外，在let语句中，表示乘法运算的符号“*”之前无需增加转义符号。表6-9给出了let命令和“((...))”结构支持的各种整数算术运算，及其简单说明。

表6-9 let命令和“((...))”结构支持的算术运算

运算符	简单说明
exp1 + exp2	计算整数表达式exp1与exp2的和
exp1 - exp2	计算整数表达式exp1与exp2的差
exp1 * exp2	计算整数表达式exp1与exp2的乘积
exp1 / exp2	计算整数表达式exp1与exp2的商
exp1 % exp2	计算整数表达式exp1与exp2的余数。可用于生成位于某个范围内的数字
var += exp	组合加运算符。把表达式的值加到变量中。例如，执行let "var += 5"之后，变量var的值将增加5
var -= exp	组合减运算符。从变量中减去表达式的值。例如，执行let "var -= 3"之后，变量var的值将减少3
var *= exp	组合乘运算符。计算变量与表达式的乘积，再把结果赋值到变量中。例如，执行let "var *= 4"之后，变量var的值将扩大4倍
var /= exp	组合除运算符。把变量除以表达式后的商再赋值到变量中。例如，执行let "var /= 2"之后，变量var的值将缩小2倍
var %= exp	组合模运算符。把变量除以表达式后的余数再赋值到变量中

下面以不同的运算符说明怎样利用let命令计算整数表达式。

```
$ n=1
$ let "n = $n + 1"
$ echo "$n"
2
$ let "n = n + 1"
$ echo "$n"
3
$ let n=$n+1
```

```

$ echo "$n"
4
$ let n=n+1
$ echo "$n"
5
$ (( n = n + 1 ))
$ echo "$n"
6
$ let "n += 1"
$ echo "$n"
7
$ let n+=1
$ echo "$n"
8
$

```

`let`命令和`((...))`结构也可以返回算术表达式计算结果的出口状态。如果算术表达式的计算结果为0，出口状态为1。如果计算结果为非0值，出口状态为0。这一点与`test`语句、“`[...]`”和“`[[...]]`”结构恰好相反，但比较运算的判断逻辑是一致的。参见下面的例子：

```

$ (( 5-5 ))
$ echo "The exit status is $?"
The exit status is 1
$ (( 5+5 ))
$ echo "The exit status is $?"
The exit status is 0
$ (( 5/5 ))
$ echo "The exit status is $?"
The exit status is 0
$ (( 5 > 4 ))
$ echo "The exit status is $?"
The exit status is 0
$ (( 5 > 9 ))
$ echo "The exit status is $?"
The exit status is 1
$ let "1<2"
$ echo "The exit status is $?"
The exit status is 0
$ let "5<2"
$ echo "The exit status is $?"
The exit status is 1
$

```

注意：在`let`语句中，如果未加双引号，表达式之间不能有空格。此外，为了加快Shell的运算速度，提高Shell编程的严谨性，最好事先利用`typeset`命令定义变量的类型，并对变量进行初始化。例如，下面的例子首先利用`typeset`命令把变量`z`定义为整数变量，同时初始化为0，然后采用`let`命令和“`((...))`”结构进行计算。

```

$ typeset -i z=0
$ let z=z+1
$ let "z += 1"
$ z=$((z+1))

```

```
$ z=$((z+1))
$ echo $z
4
$
```

8. 数值常数

Shell脚本按十进制数值解释字符串中的数字字符，除非数字前有特殊的前缀或记号。如果数字前面有一个数值零（0），表示这是一个八进制的数，0x或0X表示一个十六进制的数。BASE#NUMBER表示以BASE（2-64）为底数，以NUMBER为指数的数值。

```
$ let "dec = 32" # 默认的十进制数值
$ echo "decimal number = $dec"
decimal number = 32
$ let "oct = 032" # 八进制数值
$ echo "octal number = $oct"
octal number = 26
$ let "hex = 0x32" # 十六进制数值
$ echo "hexadecimal number = $hex"
hexadecimal number = 50
$ let "bin = 2#111100111001101" # 二进制数值
$ echo "binary number = $bin"
binary number = 31181
$
```

6.3.3 命令替换

命令替换的目的是获取命令的输出，为变量赋值，或对命令的输出做进一步的处理。命令替换的实现有两种方法：一是使用反向引号“```”引用命令；二是采用“`$(...)`”形式引用命令。其语法格式简写如下：

```
`command`
或
$(command)
```

例如，为了获取date命令的输出，并把输出结果赋予today变量，可以使用下列命令替换形式：

```
$ today=`date`
$ echo $today
Tue Jun 23 12:56:00 CST 2009
$
```

下面的例子是利用第二种命令替换形式获取uname命令输出系统的主机名，并把结果赋予machine变量：

```
$ machine=$(uname -n)
$ echo $machine
iscas
$
```

原则上，命令替换中的命令可以是任何命令，包括外部命令、Shell脚本，甚至是Shell脚本

中定义的函数。严格地讲，命令替换实际上蕴含着两个过程：执行相应的命令，获取命令标准输出中的数据。在获取命令的输出数据之后，可以使用赋值运算符（=），把数据赋予某个变量，或做进一步的处理。

命令的输出也可以用做另一个命令的输入参数，甚至可用于生成for循环中的参数表（参见第7章“Shell高级编程”）。例如，如果文件filename中包含需要删除的文件列表，使用下列命令可以删除文件filename中指定的所有文件：

```
$ rm -rf `cat filename`
$
```

注意：在执行上述命令时，如果出现类似于“arg list too long”的出错信息，最好改用“xargs rm -- < filename”命令。

不管采用哪一种命令替换形式，其最终效果是完全一样的。下面两个例子是分别采用两种命令替换形式的输出结果：

```
$ cfile=`ls *.c`
$ echo $cfile
atmcom.c atmmon.c handler.c listerner.c
$ cfile2=$(ls *.c)
$ echo $cfile2
atmcom.c atmmon.c handler.c listerner.c
$
```

注意：在使用echo命令输出一个未加引号引用的变量，而相应的变量又是采取命令替换的形式赋值时，将会抛弃命令输出数据后面的换行符，从而有可能导致不期望的结果。在下列例子中，我们希望显示一个文件列表，但实际的输出数据却是另外一种结果。

```
$ dir_listing=`ls -l /rmdisk`
$ echo $dir_listing          # 未使用引号
total 34 drwxrwxrwx 1 gqxing other 16384 Jan 1 1970 noname lrwxrwxrwx 1 root
nobody 8 Jun 23 12:06 rmdisk0 -> ./noname
$
```

这显然不是我们预期的输出结果。如果在引用变量时增加一对双引号，其输出内容恰是我们希望看到的结果。

```
$ echo "$dir_listing"        # 使用引号
total 34
drwxrwxrwx  1 gqxing  other      16384 Jan  1  1970 noname
lrwxrwxrwx  1 root    nobody           8 Jun 23 12:06 rmdisk0 -> ./noname
$
```

在命令替换中，使用I/O重定向的方式或cat命令，还可以把文件的全部内容赋予一个变量。但在实际的应用过程中，把一个较大文本文件的内容赋予一个变量并非必要。尤其不应当把一个二进制文件的内容赋予变量。例如，如果确实需要，我们可以使用下列命令把文件/etc/motd的内容赋予变量var1。

```
$ cat /etc/motd
Sun Microsystems Inc.   SunOS 5.10           Generic January 2005
$ var1=`cat /etc/motd`
$ echo $var1
```



```
Sun Microsystems Inc. SunOS 5.10 Generic January 2005
$
```

如果稍做变换，可以把上述命令替换改写如下（实际上，可以把“< fname”看做“cat fname”命令的一个特例）：

```
$ var2=`< /etc/motd`
$ echo $var2
Sun Microsystems Inc. SunOS 5.10 Generic January 2005
$
```

注意：采用命令替换的形式为变量赋值时，变量中可能包含空格，甚至包含控制字符。

6.4 test语句

每一种编程语言都具有条件测试功能，而条件测试的结果将决定程序的控制流向和下一步的处理动作。通过test命令及其简化形式（两种方括号测试结构），Shell编程也支持这一功能。test语句与if/then和case结构语句一起，构成了Shell编程的控制转移结构，与while和until等结构语句一起，构成了Shell编程的循环结构。

通常，Shell执行的test命令是Shell自己提供的内部命令。在执行test语句时，如果不特别指定，Shell不会调用外部的系统命令/usr/bin/test。

test命令（及其简化形式）的主要功能是计算随后的表达式，如检查文件的属性，比较字符串，或比较字符串表示的整数值等，然后以表达式的计算结果作为test命令的出口状态。如果测试条件为真，test命令将会返回0，否则返回一个非0数值。

test命令经常用于控制Shell脚本的执行流向。test语句中的选项与参数或表达式描述了测试的条件。测试条件可以是文件属性检查、字符串比较或整数值比较。在计算test语句中的表达式之前，Shell首先执行变量替换或命令替换，然后再执行条件测试。

在Shell中，test命令的语法格式主要有下列三种形式：

- test expression
- [expression]
- [[expression]]

“[...]”是一种简化的，但效率更高的test命令。而“[[...]]”结构则是一种比“[...]”结构更通用的测试结构，也是扩展的test命令。同test命令一样，这种测试结构用于计算括号内的表达式，测试文件的属性，或比较字符串及其相应的整数值。然后再根据计算和测试的结果，返回相应的出口状态（0或1）。注意，方括号内侧的两边必须各加一个空格。

例如，在删除文件时，为了避免出错，可以先测试文件是否存在。如果文件确实存在，然后再执行文件删除操作，例如：

```
$ fname=/tmp/somefile
$ if test -e $fname
> then
> rm -f $fname
> echo "File $fname has removed."
> fi
File /tmp/somefile has removed.
$
```

采用 “[...]” 测试结构，可把上述代码片段改写如下：

```
$ fname=/tmp/somefile
$ if [ -e $fname ]
> then
> rm -f $fname
> echo "File $fname has removed."
> fi
File /tmp/somefile has removed.
$
```

在Shell脚本中，使用 “[...]” 测试结构，而不用 “[...]” 结构，有时能够避免出现逻辑错误。例如，在 “[...]” 测试结构中，可以使用 “&&”、“||”、“<” 和 “>” 等运算符。但如果在 “[...]” 测试结构中使用上述运算符就会出错。示例如下：

```
$ fname=/tmp/somefile
$ flag=yes
$ if [[ -e $fname && "$flag" = "yes" ]]
> then
> rm -f $fname
> echo "File $fname has removed."
> fi
File /tmp/somefile has removed.
$
```

注意： “[...]” 测试结构不允许执行文件名生成和单字解析操作。

6.4.1 文件测试运算符

文件测试主要指文件的状态和属性测试，如文件是否存在、文件的类型、文件的访问权限以及其他属性等。

文件的访问权限分为三种类型：文件属主、同组用户和其他用户。如果Shell脚本的执行者是文件的属主，则检查相应于文件属主的访问权限；如果执行者不是文件属主，而是与文件属主同组的成员，则检查相应于同组用户的访问权限；如果执行者既非属主，又非同组成员，则检查相应于其他用户的访问权限。

在test语句中，文件名参数必须显式地指定，不能为空。文件名参数可以是实际的文件名，也可以是变量替换、命令替换或文件名生成机制等的最终结果。如果文件名参数是由替换方式生成的，必须使用双引号括起来。

表6-10是test语句中常用的部分文件测试表达式。

表6-10 文件属性测试表达式

表达式	简单说明
-e file	如果给定的文件存在，则条件测试的结果为真
-r file	如果给定的文件存在，且其访问权限是当前用户可读的，则条件测试的结果为真
-w file	如果给定的文件存在，且其访问权限是当前用户可写的，则条件测试的结果为真
-x file	如果给定的文件存在，且其访问权限是当前用户可执行的，则条件测试的结果为真
-s file	如果给定的文件存在，且其大小大于0，则条件测试的结果为真

(续表)

表达式	简单说明
-f file	如果给定的文件存在，且是一个普通文件（非目录或设备文件），则条件测试的结果为真
-d file	如果给定的文件存在，且是一个目录，则条件测试的结果为真
-L file	如果给定的文件存在，且是一个符号连接，则条件测试的结果为真
-c file	如果给定的文件存在，且是字符特殊文件（如终端等），则条件测试的结果为真
-b file	如果给定的文件存在，且是块特殊文件（如磁盘等），则条件测试的结果为真
-p file	如果给定的文件存在，且是命名的管道文件，则条件测试的结果为真
-u file	<p>如果给定的文件存在，且其setuid标志位已经设置，则条件测试的结果为真。如果文件属主为超级用户的可执行文件已经设置了setuid标志，即使是普通用户，在执行期间也具有超级用户的特权。对于某些可执行文件而言，这是非常有用的。例如，passwd命令需要更新系统文件/etc/shadow。如果没有设置setuid标志位，普通用户就无法使用passwd命令修改密码。为了了解一个文件是否设置了setuid标志位，可以使用下列命令显示相应的文件，检查其文件属主访问权限字段是否存在一个字符“s”：</p> <pre>\$ ls -l /usr/bin/passwd -r-sr-sr-x 1 root sys 22628 Aug 15 2007 /usr/bin/passwd \$</pre> <p>注意，设置setuid标志位可能会留下安全隐患。但对Shell脚本来讲，setuid标志位没有影响。</p>
-g file	如果给定的文件存在，且其setgid标志位已经设置，则条件测试结果真。如果某个目录设置了setgid标志位，则在该目录中创建的文件属于同组成员。这对于工作组成员共享同一目录是非常有用的
-k file	<p>如果给定的文件存在，且其粘性标志位已经设置，则条件测试结果真。粘性标志位是一种特殊的文件访问权限。如果某个可执行文件设置了粘性标志位，当调度相应的程序执行时，在整个运行过程中，相应的程序将始终保持在高速缓存中，访问和执行的速度因而更快捷。如果设置的是目录，将限制用户的访问权限。如果一个目录设置了粘性标志位，在使用下列命令显示相应的文件或目录时，其访问权限字段将会出现一个字符“t”：</p> <pre>\$ ls -ld /tmp drwxrwxrwt 12 root root 4986 2009-01-10 17:40 /tmp \$</pre> <p>如果某个共享目录已经设置了粘性标志位，且任何用户均具有写的访问权限，则用户只能删除目录中属于自己的文件。这将防止用户无意中覆盖或删除共享目录下其他用户拥有的文件。注意，在UNIX系统中，粘性标志位仅适用于目录</p>
f1 -nt f2	如果给定的文件f1存在，且其修改日期比文件f2新，则条件测试的结果为真
f1 -ot f2	如果给定的文件f1存在，且其修改日期比文件f2早，则条件测试的结果为真
f1 -ef f2	如果给定的文件f1和f2存在且指向的是同一个物理文件，则条件测试的结果为真
!	逻辑非运算符。当与上述表达式一起使用时，测试结果与其本意恰好相反

6.4.2 字符串测试运算符

表6-11给出了test语句中常用的字符串测试表达式。

表6-11 字符串测试表达式

表达式	简单说明
-z str	如果给定字符串的长度为0，则条件测试的结果为真。在“! -z”情况下，如果字符串未加引号，或在test语句中单独使用未加引号的字符串，结果将是不可靠的
-n str	如果给定字符串的长度大于0，则条件测试的结果为真。“-n”测试要求字符串前后必须加引号
s1 = s2	如果给定字符串s1等同于字符串s2，则条件测试的结果为真
s1 != s2	如果给定字符串s1不等同于字符串s2，则条件测试的结果为真
s1 < s2	基于字符的ASCII编码值，如果给定的字符串s1小于字符串s2，则条件测试的结果为真。其部分用法列举如下： <ul style="list-style-type: none">• test s1 < s2• [s1 \< s2]• [[s1 < s2]] 注意，在单方括号 “[...]” 中，小于号 “<” 前需加转义符号。
s1 > s2	基于字符的ASCII编码值，如果给定的字符串s1大于字符串s2，则条件测试的结果为真。其部分用法列举如下： <ul style="list-style-type: none">• test s1 < s2• [s1 \> s2]• [[s1 > s2]] 注意，在单方括号 “[...]” 中，大于号 “>” 前需加转义符号。

1. 字符串等同性测试

test语句支持两种字符串等同性比较测试：等于(=)测试用于比较两个表达式的相等性；不等(!=)测试用于比较两个表达式的不等性。

下面是两个test语句的例子。第一个例子测试两个字符串的相等性，第二个例子测试两个字符串的不等性。注意，比较运算符两端必须各有至少一个空格。如果漏掉了空格，test语句就会把比较运算符看做赋值运算符或待测试字符串的一部分：

```
$ name="John"
$ test "$name" = John
$ echo $?
0
$ name="John  "
$ [ "$name" = John ]
$ echo $?
1
$
```

此外还要注意，在test语句的字符串比较表达式中，引用的变量或字符串前后一定要加双引号。否则，当变量或字符串表达式的值为null时，Shell将会忽略变量或字符串表达式的存在，导致语法错误。例如，当NOTSET变量未设置时，第一个test命令认为“!=”运算符左边的参数不存在，故测试出错。

```
$ echo ${NOTSET}

$ test ${NOTSET} != "hello"
```

```
ksh: test: argument expected
$ test "${NOTSET}" != "hello"
$ echo $?
0
$
```

在上述例子中，第一个test语句执行有误说明漏掉了双引号。Shell按IFS处理机制从命令行中删除值为null的字符串参数，因此，当执行test语句时，Shell只看到两个参数：即“!=”与“hello”，因而认为不满足字符串比较所需的三个参数。

2. 字符串长度测试

test语句也可用于测试字符串表达式的长度。test语句中的“-z”和“-n”选项分别用于测试字符串表达式的长度是否为0或非0。如果未明显地指定任何选项，“-n”将会成为默认的选项。也就是说，如果test语句中只有一个参数，则当字符串参数包含一个或多个字符时，测试条件的结果为真，因而返回一个0值。例如：

```
$ test -z "string"
$ echo $?
1
$ test -n "string"
$ echo $?
0
$ test "string"
$ echo $?
0
$
```

字符串长度测试经常用于安装软件时检查变量是否已经设置，根据变量或形式参数的设置与否，组织控制结构，决定Shell脚本命令执行的控制流向。

6.4.3 整数值测试运算符

test语句还可用于比较字符串表达式中包含的整数值。整数字符串中不能包含任何非数字字符，但前后可以有空格字符。在test语句中，整数值的比较采用C语言中的atoi()函数，把字符串转换成等价的ASCII整数值。

下面是一些合法的整数字符串表达式的例子（Bourne Shell还允许数字后面包含其他非数字字符）：

"486"	486
"15 "	15
" 123"	123

下列两个例子说明了怎样使用test语句比较整数字符串表达式：

```
$ test "123" -eq "123"
$ echo $?
0
$ test "123" -eq " 123 "
$ echo $?
0
$
```


表6-12给出了test语句中常用的整数测试表达式

表6-12 整数测试表达式

表达式	简单说明
<code>exp1 -eq exp2</code>	如果表达式exp1的整数值等于表达式exp2的整数值，则计算结果为真
<code>exp1 -ne exp2</code>	如果表达式exp1的整数值不等于表达式exp2的整数值，则计算结果为真
<code>exp1 -gt exp2</code>	如果表达式exp1的整数值大于表达式exp2的整数值，则计算结果为真
<code>exp1 -lt exp2</code>	如果表达式exp1的整数值小于表达式exp2的整数值，则计算结果为真
<code>exp1 -ge exp2</code>	如果表达式exp1的整数值大于等于表达式exp2的整数值，则计算结果为真
<code>exp1 -le exp2</code>	如果表达式exp1的整数值小于等于表达式exp2的整数值，则计算结果为真

6.4.4 逻辑运算符

test语句支持表达式的逻辑运算。其中，符号“!”表示逻辑非运算，符号“-a”或“&&”表示逻辑与运算，符号“-o”或“||”表示逻辑或运算，如表6-13所示。

表6-13 逻辑运算表达式

表达式	说明
<code>(exp)</code>	用于计算括号中的组合表达式。如果整个表达式的计算结果为真，则测试结果也为真
<code>! exp</code>	对表达式进行逻辑非运算，即对测试结果求反。如果表达式的计算结果为假，则最终的测试结果为真
<code>exp1 -a exp2</code> <code>exp1 && exp2</code>	对两个表达式进行逻辑与运算。如果两个表达式的计算结果均为真，最终的测试结果才为真。注意，单方括号 ([...]) 结构中不允许使用&&运算符。逻辑与运算符的部分用法列举如下： • <code>test exp1 -a exp2</code> • <code>test \(exp1 -a exp2 \)</code> • <code>[exp1 -a exp2]</code> • <code>[exp1] && [exp2]</code> • <code>[[exp1 && exp2]]</code>
<code>exp1 -o exp2</code> <code>exp1 exp2</code>	对两个表达式进行逻辑或运算。只要两个表达式的计算结果中有一个为真，最终的计算结果就为真。同样，单个方括号 “[...]” 结构中不允许使用 “ ” 运算符。逻辑或运算符的部分用法列举如下： • <code>test exp1 -o exp2</code> • <code>test \(exp1 -o exp2 \)</code> • <code>[exp1 -o exp2]</code> • <code>[exp1] [exp2]</code> • <code>[[exp1 exp2]]</code>

在比较两个文件时，可以使用diff命令。比较之前，为了确保两个文件都存在，可以写出下列Shell代码片段：

```
if [ -f file1 -a -f file2 ]
then
    diff file1 file2
fi
```

下面的例子说明了怎样使用各种逻辑运算符:

```
$ test ! -d /tmp
$ echo $?
1
$ test \( -d /tmp -a -x /tmp \)
$ echo $?
0
$ ! [ -d /tmp -a -x /tmp ]
$ echo $?
1
$ [[ -d /tmp && -x /tmp ]]
$ echo $?
0
$
```

6.5 命令行的解释执行过程

Shell命令行的解释执行是一个复杂的处理过程,了解命令行的解释执行过程有助于用户正确地访问系统,准确地输入命令,避免误用或出现意外,影响命令的最终运行结果。例如,假定我们想用变量的值实现I/O重定向,把命令的输出保存到一个文件中,其结果如下:

```
$ SAVEFILE="> /tmp/savefile"
$ echo something $SAVEFILE
something > /tmp/savefile
$ cat /tmp/savefile
cat: cannot open /tmp/savefile
$
```

从上述命令的运行结果可以看出,Shell并没有把echo命令的输出重定向到文件savefile中,而是把“something”和SAVEFILE变量的值“> /tmp/savefile”一起输出到终端窗口(标准输出)上。实际上,Shell的I/O重定向是在变量替换之前执行的。替换SAVEFILE变量之后,Shell只是把变量的值作为echo命令参数的一部分送到标准输出,因而没有创建预期的文件/tmp/savefile。

因此,有必要在此讨论一下Shell命令行的解释执行过程。Shell命令行的解释执行过程大体可以分为16个步骤。按照顺序列举如下:

- (1) 读取命令行;
- (2) 命令历史替换;
- (3) 命令别名替换;
- (4) 花括号扩展;
- (5) 波浪号替换;
- (6) I/O重定向;
- (7) 变量替换;
- (8) 算术运算结果替换;
- (9) 命令替换;
- (10) 单词解析;

- (11) 文件名生成;
- (12) 引用字符处理;
- (13) 进程替换;
- (14) 环境处理;
- (15) 执行命令;
- (16) 跟踪执行过程。

6.5.1 读取命令行

Shell命令行解释执行过程的第一步是读取命令行。命令行可以来自终端，也可以取自普通的文本文件，如Shell脚本文件。

不管是以交互方式访问Shell，还是运行Shell脚本，Shell都需要读取命令行。如果是交互方式，Shell还需要在用户终端上回显其读取的每一个字符，然后解析读取的命令行。在处理每个命令行之前，Shell需要读取完整的命令行。部分Shell内置命令，如if/then、for和case等结构语句，以及函数定义与长字符串等，通常都会跨越多个物理行。当读取此类多行形式的命令语句时，在进一步解析和处理之前，Shell将会连续地读取整个命令语句。在交互会话期间，Shell会采用辅助命令提示符“>”（PS2变量的默认值）的形式，提示用户输入后续的命令语句，此时用户可以继续输入，直至Shell认为命令行的输入已经结束。例如：

```
$ while true
> do
> ls -l /var/log/somefile
> sleep 5
> done
.....
^C
$ echo "***** NOTICE *****"
> Today is Christmas.
> Every body could go home after 12:00."
***** NOTICE *****
Today is Christmas.
Every body could go home after 12:00.
$
```

在读取命令行时，Shell将会逐个判断读取的每一个字符，直至遇到一个分号“;”、后台进程符号“&”、逻辑与“&&”、逻辑或“||”或换行字符时，整个命令行的读取过程才算结束。如果读取的是一个结构语句，如if/then、for或while等，Shell将会完整地读入整个结构语句。

在读入一个完整的命令或结构语句之后，Shell开始对命令语句进行语法分析，把命令语句分解为一系列单词或关键字等基本元素。通常，Shell假定每个单词或关键字都是由空格或制表符分隔的一系列连续字符组成的（参见IFS变量的说明）。Shell从命令语句行首的第一个字符开始解析，直至行尾结束，依次剥离出一系列单词，包括由“<”、“>”、“|”和“^”等特殊字符组成的单字。不管IFS变量的设置如何，这一解析过程总是如此处理。至于如何处理IFS变量，则是另外一个解析步骤。

6.5.2 命令历史替换

在读取一个完整的命令行之后，Shell首先会检查是否需要使用命令历史缓冲区中记录的命令替换读取的命令，是否需要编辑命令。如果需要，则取出相应的命令，再经过适当地校正之后，使之作为当前需要执行的命令。例如，当输入“r”或“!!”命令时，意味着重新执行先前执行的最后一条命令，Shell于是会从命令历史缓冲区中取出相应的命令，替换“r”或“!!”命令。

通常，交互式Shell总是首先执行命令历史机制的命令替换与编辑过程。如果想要提高Shell命令行解释执行的效率，省略这一处理步骤，可以使用下列命令禁止Shell执行命令历史替换（注意，命令历史替换处理不适用于非交互式Shell，如运行Shell脚本）：

```
$ set +o histexpand
$
```

6.5.3 命令别名替换

命令行解释执行的第二步是命令别名替换。在此步骤中，Shell将会检查读取的命令是否为命令别名。如果确为命令别名，则根据定义，使用原始命令予以替换。通常，交互式Shell总是执行命令别名检查，非交互式Shell则禁用这一功能特性。为了绕过这一处理步骤，可以使用下列命令禁止Shell执行命令别名替换：

```
$ shopt -u expand_aliases
$
```

6.5.4 花括号扩展

花括号提供了一种简单的文件名生成方法。下列例子说明了怎样利用花括号扩展机制指定文件名，在/home/gqxing/scripts目录中一次创建4个子目录：

```
$ mkdir ~/scripts/{old,new,tools,admin}
$
```

在上述例子中，Shell将会尝试把花括号中以逗号“,”分隔的一系列单词与花括号之外的字符串连接在一起，生成若干由空格分隔的字符串，用于指定或匹配当前目录中的文件。

当文件的路径名较长时，花括号扩展机制是非常有用的。例如，为了把/home/gqxing/src目录中的部分C程序（mon.c、list.c、hand.c和comm.c）复制到当前目录，可以使用下列命令：

```
$ cp /home/gqxing/src/{mon,list,hand,comm}.c .
$
```

但是，Shell并非总是使用花括号扩展机制匹配现有的文件名的，实际上，也可以使用花括号扩展机制生成任何字符串。例如，利用下列命令生成的字符串并非实际存在的目录文件名：

```
$ echo chap-{one,two,three,four}
chap-one chap-two chap-three chap-four
$
```

通常，交互式Shell总是启用花括号扩展机制，非交互式Shell则禁用花括号扩展机制。为了

禁止Shell使用花括号扩展机制，可以输入下列命令：

```
$ set +o braceexpand
$
```

6.5.5 波浪号替换

为了指定用户的主目录，除了使用\$HOME变量之外，还可以在用户名前增加一个波浪号“~”，或直接使用波浪号表示当前用户的主目录。当命令行中出现波浪号“~”这一特殊字符时，Shell会继续读取随后的字符串，直至遇到一个斜线字符“/”或空白字符（表示单字的结束）为止，然后验证读取的字符串是否是一个有效的注册用户名。如果读取的字符串为空（即后面只有一个斜线字符“/”，因而只有波浪号“~”本身），Shell将会使用当前用户的HOME变量值替换波浪号“~”。例如：

```
$ echo $HOME
/export/home/gqxing
$ echo ~
/export/home/gqxing
$ ls -l ~/incl
total 94
-rw-r--r--  1 gqxing  other      18644 Jun 19 12:57 atm.h
-rw-r--r--  1 gqxing  other      16582 Jun 19 12:58 event.h
-rw-r--r--  1 gqxing  other      10806 Jun 19 12:59 status.h
$
```

如果波浪号“~”之后是一个有效的注册用户名，Shell将以相应用户主目录的路径名替换波浪号“~”和用户名。如果“~”之后既不为空，也不是一个有效的用户名，Shell不会做任何替换。例如：

```
$ echo $HOME
/export/home/gqxing
$ ls -l ~gqxing
total 10
drwxr-xr-x  2 gqxing  other      512 Jun 18 15:39 bin
drwxr-xr-x  2 gqxing  other      512 Jun 18 15:41 conf
drwxr-xr-x  5 gqxing  other      512 Jun 23 13:28 doc
drwxr-xr-x  2 gqxing  other      512 Jun 19 12:57 incl
drwxr-xr-x  2 gqxing  other      512 Jun 18 15:34 src
$ echo ~xxx
~xxx
$
```

此外，如果波浪号“~”之后为加号“+”或减号“-”，则“~+”表示PWD变量的值，即当前工作目录；“~-”表示OLDPWD变量的值，即先前的工作目录。例如：

```
$ cd /etc
$ cd
$ echo $PWD
/export/home/gqxing
$ echo ~+
/export/home/gqxing
$ echo $OLDPWD
```



```

/etc
$ echo --
/etc
$

```

6.5.6 I/O重定向

如果不了解文件描述符、进程文件表（也称文件描述符表或用户文件表）、系统文件表和信息节点表，很难理解I/O重定向。这些表都是UNIX系统维护的数据结构。进程文件表中包含每个进程已打开文件指向系统文件表的指针，而系统文件表中的文件指针则指向位于内存中的信息节点表，信息节点表包含文件的信息节点，而信息节点含有文件的打开方式（如读或写等）、文件的当前读写位置，以及怎样获取文件的数据内容等信息。文件描述符是文件在系统文件表中的位置索引。每个进程的文件读写操作都是通过文件描述符实现的。操作系统利用文件描述符，从系统文件表和信息节点中找出目标文件，最终完成文件的处理，如图6-1所示。

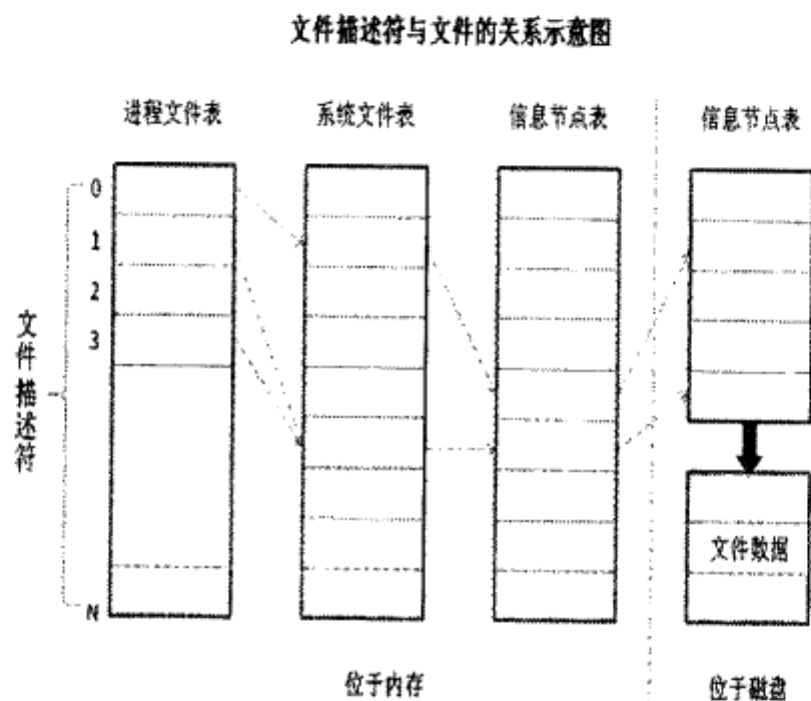


图6-1 文件描述符与文件的关系

文件描述符0是标准输入，通常为终端的键盘输入。文件描述符1为标准输出，文件描述符2为标准错误输出，两者通常为终端显示。当注册到UNIX系统时，这三个文件描述符是已经打开的默认文件描述符。

在通常情况下，Shell通过标准输入读取命令语句，通过标准输出显示命令执行的结果。当执行过程中发现问题时，则通过标准错误输出显示错误信息。

在输入输出重定向的情况下，Shell需要执行若干处理动作：关闭指定的文件描述符，如关闭文件描述符0（如果标准输入被重定向）或关闭文件描述符1（如果标准输出被重定向）等；打开指定的文件；把新的文件指针放入文件指针数据结构表中刚才腾出的位置，以替代刚关闭的相应文件。

下面是一个I/O重定向的例子。

```
command < datafile
```

在这个例子中，标准输入被重定向到datafile中。在实现过程中，Shell首先关闭文件描述符0，然后以输入方式打开datafile文件，最后把打开操作产生的文件指针信息复制到原标准输入

所在文件指针数据结构表中已腾出的位置，如图6-2所示。

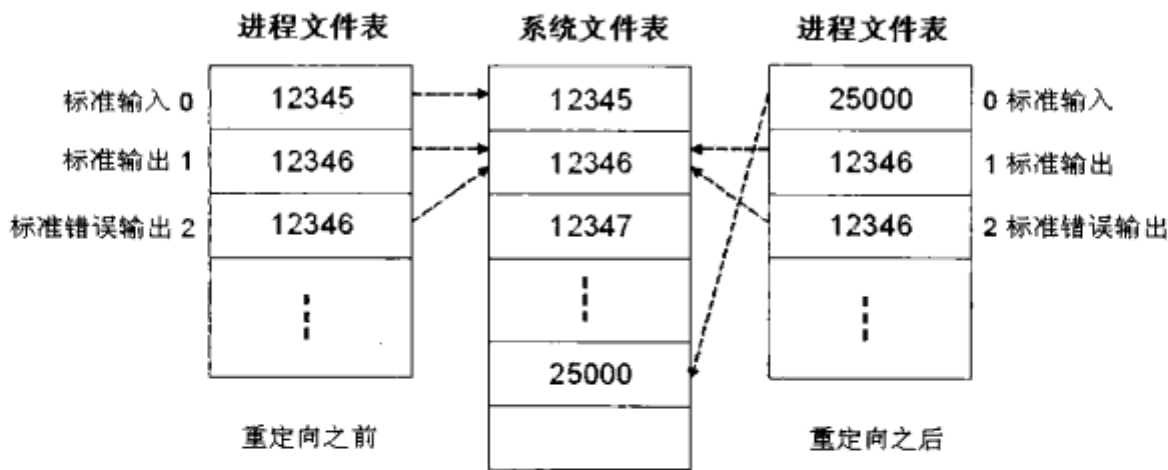


图6-2 标准输入重定向

至此，当命令从标准输入（文件描述符0）读取数据时，将会从新打开的datafile文件，而不是终端的键盘输入中读取数据。如果标准输出或标准错误输出也被重定向，Shell将采取同样的处理动作。

6.5.7 变量替换

变量替换涉及Shell内部变量、用户自定义变量、命令行参数或位置参数等变量替换。

变量替换表达式由美元符号“\$”与随后的变量名构成，如\$var。位置参数替换表达式由美元符号“\$”与随后的数字构成，如\$1、\$2、.....等。变量名或参数名也可以用花括号括起来，如\${var}。变量替换表达式前后还可以加单、双引号，如fname="\${var}"。但在实际替换时，两者存在细微的差别，详见6.2节有关变量替换的介绍。

6.5.8 算术运算结果替换

在Shell中，当命令行中出现算术运算结果替换表达式“\$((expression))”时，Shell首先需要计算双括号中的表达式，然后使用计算结果替换“\$((expression))”。例如：

```
$ echo There are $((60*60*24*365)) seconds in a year
There are 31536000 seconds in a year
$
```

6.5.9 命令替换

命令替换表达式是由反向单引号“`”或\$(...)形式的命令语句组成的，命令的输出就是命令替换表达式的值。命令可以是任何UNIX命令，包括普通命令、顺序执行的命令、组合命令或含有管道符号的并列命令等。

作为命令替换表达式的一部分，命令语句可以使用任何Shell机制，如变量替换和文件名生成机制等。因此，在执行命令之前，Shell需要把整个命令行完全展开，最终形成实际上能够执行的具体命令。

如果在命令的替换过程中出现多余的空格、制表符或换行符等，Shell将会在第二次重读、命令语句解析、处理引用字符时予以删除。因此，如果这些空格、制表符或换行符等应当保留，输入时应使用双引号括住整个命令替换表达式。

在反向单引号之前增加转义符号“\”，可以实现命令替换表达式的嵌套。下面的例子解释了命令表达式的嵌套是怎样实现的：

```
$ cat is.today
for fname in ${*}
do
    appoint=`grep \`date +%m/%d/%y\` ${fname}`
    echo "${appoint}"
done
$ cat calendar
11/08/09          Finish unit 15
11/09/09          Finish unit 16
11/10/09          Finish unit 17
$ date +%m/%d/%y
11/08/09
$ is.today calendar
11/08/09          Finish unit 15
$
```

在上述is.today脚本文件中，最内层的反向单引号各增加了一个转义符号“\”。这个脚本首先确定当天的日期，然后利用这个日期作为模式，使用grep命令检索fname变量指定的文件。最后把检索到的内容赋予变量appoint，并输出最终结果。

6.5.10 单词解析

在完成了变量替换、命令替换以及算术运算结果替换之后，Shell将利用IFS变量设置的字符作为分隔符，对经过各种替换后生成的命令行再次进行解析，以分离出最基本的命令行元素或单词。此时，如果命令行中存在单引号或双引号，其中的内容（包括null参数）将被看做一个单词。否则，Shell会删除额外的空格、制表符和换行符，以及隐含的null参数。

IFS变量的默认值为空格、制表符和换行符，如果需要，用户也可以修改或重新定义IFS变量设置。

在IFS单词解析处理阶段，变量赋值语句是受特殊保护的。除了赋值之外，Shell不会对变量赋值语句做文件名生成等其他任何解析处理。因此，如果希望把一个变量表达式的值赋予等号左边的变量，而变量表达式又可能生成多个单词时，这样的变量表达式前后不必加双引号。

下面的例子解释了IFS变量的设置如何影响命令行的解释执行：

```
$ a=x:y:z
$ cat $a
cat: cannot open x:y:z
$ IFS="$IFS:"
$ cat $a
cat: cannot open x
cat: cannot open y
cat: cannot open z
$
```

6.5.11 文件名生成

命令行解释执行的下一步是扩展元字符，以生成文件名。在这个处理步骤里，Shell将会检

索解析出来的每一个单词，检查其中是否包含任何符合文件名生成规则的元字符，如星号“*”、问号“?”和由方括号“[...]”表示的字符范围等。

对于包含元字符的每一个基本单词，Shell将会尝试匹配当前目录或指定目录中的文件名。如果找到匹配的任何文件名，Shell将会使用这些文件名替换命令语句中的元字符表达式。例如：

```
$ ls file*
file1 file2 file3
$ echo file*
file1 file2 file3
$ rm file?
$ echo file*
file*
$
```

在文件名生成过程中，涉及到下一节将要介绍的单双引号的引用处理。在下面的例子中，为了引用当前目录下的所有文件，我们首先为变量files赋值一个星号“*”：

```
$ files=*
$
```

当使用echo命令显示变量files的值，且在变量的前后增加了双引号时，Shell只做变量替换，但不执行文件名生成：

```
$ echo "${files}"
*
$
```

如果未加双引号，Shell将会扩展元字符“*”，执行文件名的生成过程。例如：

```
$ echo ${files}
file1 file2 file3
$
```

如6.5.10一节所述，变量赋值时并不会执行文件名生成。因此，在执行赋值语句“files=”时，变量files的值只是一个星号“*”，而不是当前目录下的所有文件名。在此步骤中，Shell不会解析双引号括住的变量赋值，但参数和变量替换会照常执行。如果是单引号，Shell将会禁止一切替换。因此，如果想用一个带有元字符的变量值，而又不希望扩展元字符，可用双引号括住变量表达式。

特别需要注意的是，文件名的生成是在变量替换、命令替换和I/O重定向之后进行的。因此，在涉及文件名生成的表达式中要特别小心，尤其是在带有I/O重定向的表达式中，一定要注意。

6.5.12 引用字符处理

引用字符处理的目的是删除引用符号。在此处理步骤中，Shell开始删除转义符号“\”和单双引号等引用字符，展开引号中的表达式，完成命令执行前的所有预备工作。

引用字符处理涉及到文件名的生成及各种替换。如果一个基本单词前后有双引号，Shell会禁止文件名生成，禁止除参数和变量替换之外的所有替换。如果有单引号，Shell将会禁止文件名生成和所有的替换。下面的例子解释了单引号、双引号以及不加任何引号的引用字符处理与

文件名生成:

```
$ ls file*
file1 file2 file3
$ var=file*
$ set | grep var=
var='file*'
$ echo '$var'
$ var
$ echo "$var"
file*
$ echo $var
file1 file2 file3
$
```

6.5.13 进程替换

进程替换是相对于命令替换而言的。命令替换的主要作用是把命令的输出数据赋予某个变量，以便做进一步的处理，或者利用Shell的管道机制，直接接收某个命令的输出数据，或者为其他命令直接提供输入数据等。例如，“var=`command`”或“command | command”。

而进程替换则是利用/dev/fd/<nn>特殊文件或管道文件，把一个进程的输出结果，作为输入数据提交给另一个进程。换言之，进程替换是利用/dev/fd/<nn>特殊文件或管道文件，实现进程之间标准输入与标准输出的交互通信的。

进程替换采用“<(command)”或“>(command)”的形式实现。当圆括号中的进程开始运行时，其标准输入和标准输出将会连接到/dev/fd目录中的两个特殊文件或管道文件，同时把文件名作为参数传递给当前命令的标准输入或标准输出。如果采用的是“>(command)”形式的进程替换，当前命令的标准输出将会写到圆括号中指定进程的标准输入。如果采用的是“<(command)”形式的进程替换，作为参数传递的特殊文件或管道文件可用于读取圆括号中指定进程的标准输出。

注意：在“<”和“>”与圆括号之间没有空格，否则将会出现错误（信息）。

进程替换的一个重要功能特性是可用进程替换命令行中的文件名参数。当一个文件名参数位置出现“<(command)”形式的进程调用时，将会引起Shell执行圆括号中的命令，把进程的标准输出写到一个特殊文件或管道文件中，然后把此文件作为当前命令的标准输入，从指定进程的标准输出中读取输入数据。

同样，当一个文件名参数位置出现“>(command)”形式的进程调用时，也会引起Shell执行给定的命令，但Shell将会把该进程的标准输出写到一个特殊文件或管道文件中，然后以此文件作为圆括号中指定进程的标准输入。

因此，可以使用进程替换比较两个不同的命令，或者同一命令但选项或参数不同时的输出结果。例如，下列例子说明了怎样使用进程替换比较两个目录中的文件有何异同：

```
$ diff <(ls -l doc) <(ls -l doc2)
2c2
< -rw-r--r--  1 gqxing  other          712 Jun 23 12:26 readme
---
> -rw-r--r--  1 gqxing  other          968 Jun 24 15:43 readme
$
```


下列命令将会按照文件的大小，对“ls -l”命令输出的数据进行排序（其中，“-n”选项表示按数值排序，“-k 5”表示按输入数据的第5列排序）：

```
$ sort -n -k 5 <(ls -l)
total 266
-rw-r--r--  1 gqxing  other      25852 Jun 18 15:31 atmcom.c
-rw-r--r--  1 gqxing  other      26282 Jun 18 15:31 atmmon.c
-rw-r--r--  1 gqxing  other      30625 Jun 18 15:32 listener.c
-rw-r--r--  1 gqxing  other      52125 Jun 18 15:33 handler.c
$
```

sort命令的“-o”选项本来用于指定一个输出文件，以便存储排序后的结果。下面的例子以进程替换的形式代替输出文件，借用awk命令，从排序后的结果中抽取文件的名称与大小字段：

```
$ sort -n -k 5 <(ls -l) -o >(awk '{print $9 "\t" $5}')
```

atmcom.c	25852
atmmon.c	26282
listener.c	30625
handler.c	52125

```
$
```

6.5.14 环境处理

命令行解释的第14步是环境处理，其中包括变量赋值，检索PATH变量指定的目录，找出命令文件的存储位置等。

在完成变量赋值之后，Shell将根据PATH环境变量的定义，从左到右依次检索命令文件的存储位置，然后以命令文件的完整路径名替换命令行中的命令名。如果命令名中（最后一个字符位置之前）包含斜线字符“/”，Shell将会绕过PATH变量检索，假定给定的命令是一个规范的路径文件名——相对路径文件名或绝对路径文件名。

6.5.15 执行命令

至此，终于到了真正开始执行命令的时候了。如果命令行要求执行的是一个普通的UNIX命令，Shell将会启动一个单独的子进程执行相应的命令。如果是一个Shell内部命令，将由Shell直接执行。

如果命令是一个经过编译后生成的应用程序，Shell启动的子进程将会使用exec系统调用执行应用程序。如果是一个Shell脚本，Shell将会解释执行脚本文件中的每一行语句。具体的执行过程是，如果访问权限表示命令文件是可执行的，为了确定命令文件是否能够直接运行，子进程将会尝试把命令文件加载到内存中，然后开始执行。如果无法加载到内存，子进程将假定相应的命令文件是一个Shell脚本，因而由Shell采用解释执行的方式运行脚本文件。

如果程序是采用前台方式运行的，Shell将会一直等待，直至子进程执行结束。如果程序采用的是后台运行方式，Shell将不再关心子进程的执行是否终止，而是立即在标准输出上显示一个作业号和进程ID，开始继续处理其他请求。

6.5.16 跟踪执行过程

如果先前使用set命令的“-v”或“-x”等选项设定了跟踪命令的执行过程，Shell将会在此处理步骤中输出当前正在执行的命令语句，同时在每个命令语句之前插入一个加号“+”前缀，然后再输出命令的运行结果。

因为命令替换已在命令执行之前完成，此时输出的是实际执行的命令，而且是按照命令执行的逻辑顺序依次输出的。

如果命令行包含管道形式的并列命令，并列命令的执行顺序并没有严格的定义，命令的调度执行将是随机的。通常，Shell会首先调度执行管道字符之前的第一个命令，然后依次调度执行其他后续命令，但这并不等于每个命令都是严格按照其在命令行中出现的位置顺序执行的。

6.5.17 实例验证

假定当前目录下存在一个文件rundata，其中包含一行数据。在Korn Shell的运行环境中执行下列命令，结果一切正常：

```
$ cat run*
This data is in rundata file.
$ cat < run*
This data is in rundata file.
$
```

而在进入Bourne Shell（如输入sh命令）之后，再次执行上述两个同样的命令，其结果如下：

```
$ sh
$ cat run*
This data is in rundata file.
$ cat < run*
run*: cannot open
$
```

此处，我们看到两个不同的结果，为什么会这样呢？这是因为两个Shell解释命令行的顺序不同，因而造成了不同的执行结果。在Bourne Shell中，I/O重定向是在元字符文件名生成处理之前进行的。因此，在执行第二个命令时，Bourne Shell试图找出一个名字为run*的文件，因为此时Bourne Shell还没有执行文件名生成处理，故文件run*根本不存在。因此，Shell输出一个错误信息，说明run*文件不存在。

而在Korn Shell中执行第二个命令时，因为文件名生成是在I/O重定向之前进行的，故Shell能够找到经过文件名生成处理后的文件rundata，并显示文件的内容。因此，在Korn Shell中没有错误信息出现。

对于第一条命令来讲，因为在执行cat命令之前，两个Shell已经把文件名扩展为rundata，因此都没有问题。

现在，让我们在两个Shell中执行以下命令，看看会有什么结果发生：

```
$ cat < rundata > rundata
cat: input/output files '-' identical
$ ls -l rundata
```

```
-rwxr-xr-x  1 gqxing  other          0 Jun 23 18:30 rundata
$
```

从输出信息中可见，**Shell**清楚地认识到输入和输出文件是相同的。因此，在执行过程中**Shell**会输出一个错误信息。但不幸的是，文件**rundada**中的数据也被删除了。为什么？

根据**Shell**的命令行解释过程可知，**I/O**重定向是在**Shell**执行命令之前完成的，当输出重定向到现有的文件时，文件中的内容也随之清除了。当**cat**命令发现两个文件名相同时，为时已晚，此时**Shell**已经把文件中的内容全部删除了。

第7章 Shell高级编程

除了顺序执行命令之外，Shell还提供丰富的控制结构语句，其中，if-then-else和case等转移控制语句使用户能够通过条件测试与运算机制，控制Shell脚本的执行流向，for、while和until等循环语句能够实现各种循环控制。本章主要介绍Shell的各种控制结构语句，讨论怎样利用控制结构语句，编写功能完善的Shell脚本，介绍怎样利用Shell的编程机制和UNIX系统提供的丰富命令，创建自己的日常系统维护脚本。在此基础上，建议读者仔细阅读利用Shell脚本实现的各种系统维护命令，编写出高水平的Shell脚本。

7.1 if条件语句

在Shell脚本中，if语句是最基本的，也是最常用的重要语句之一。利用if-then-else条件语句，可以实现脚本执行流程的控制转移。最简单的if语句语法格式如下：

```
if command
then
    command-list
fi
```

if语句以给定命令的出口状态作为判断条件，确定转入哪一个控制流向。如果命令执行成功，返回一个0值的出口状态，则执行紧随then之后的命令。如果命令执行失败，返回一个非0值的出口状态，则执行紧随fi之后的命令。fi表示if语句的结束。

if-then之间的命令可以是一个普通命令，也可以是一组命令；可以是一个简单的测试语句，也可以是一组复杂的组合测试语句。当if-then之间存在一组命令时，if-then控制结构测试的是if和then之间最后一条命令的出口状态。

7.1.1 If语句的表现形式

许多if语句均使用test命令作为逻辑判断条件，根据test命令的测试结果控制Shell脚本的流向，或决定下一步应采取的处理动作。例如，许多Shell脚本都会在正式开始运行之前检查命令行的参数个数是否正确，如果给定的参数数量不足，则显示Shell脚本的用法，然后结束运行。基于这一思路，如果Shell脚本要求最少提供两个参数，可以在Shell脚本的开始部分写出下列if语句（其中的“\$#”表示命令行参数的个数）：

```
$ cat chkargs
if test $# -lt 2
then
    echo "Usage: chkargs arg1 arg2 ..."
    exit 1
fi
echo "Go on to the next."
$
```

“if test ...”结构等价于“if [...]”和“if [...]”结构。因此，上述if语句可以改写如下：

```

if [ $# -lt 2 ]
then
    echo "Usage: chkargs arg1 arg2 ..."
    exit 1
fi

```

if语句也可以使用**else**子句处理**test**语句返回非0出口状态的情况，使之执行**else**子句后面的语句。完整的**if**语句语法格式如下：

```

if command
then
    command-list
else
    command-list
fi

```

例如，当运行某个进程（假定为**atmmon**）时，如果要把进程的运行结果记录到系统日志文件中，可根据进程的出口状态是否为0确定其运行是否正常结束，然后利用**logger**命令把结果写入系统日志文件。据此，可以写出下列代码：

```

atmmon                                     # 自己开发的ATM监控程序
if [ test $? -eq 0 ]
then
    logger "atmmon has run successfully."
else
    logger "atmmon terminated abnormally."
fi

```

if语句后面既可以没有**test**语句，也可以没有“[...]”或“[[...]]”等测试结构，而是利用其他命令的执行结果作为测试条件。按照定义，这种形式也是符合**if**语句的语法规定的。

假定仅当某个目录存在，系统才会支持某一功能，因而才能执行其中的命令时，可以首先使用**cd**命令尝试进入指定的目录。如果**cd**命令执行成功，**Shell**脚本才能正常运行。否则，退出**Shell**脚本。根据这个思路，可以写出下列**Shell**脚本：

```

dir=/opt/atm
if cd "$dir" 2>/dev/null
then
    echo "Now we are in $dir."
    .....
else
    echo "Can't change to $dir."
    exit 1
fi

```

在“**if command**”结构语句中，给定命令的出口状态用于控制**Shell**脚本的转移流向。下面的例子说明了怎样利用**diff**命令返回的出口状态判断文件比较的结果。

```

if diff fname1 fname2 > /dev/null 2>&1
then
    echo "Files fname1 and fname2 are identical."
else
    echo "Files fname1 and fname2 differ."
fi

```


下面是一个比较有用的“if-grep”结构。其中，grep命令的“-q”选项用于抑制命令的输出：

```
if grep -q UNIX fname
then
    echo "File contains at least one occurrence of UNIX"
fi
```

当if、then以及条件测试命令位于同一行上时，then之前必须加一个分号。尽管if和then是一个整体，if和then是其中的两个关键字，但它们均表示一个子句的开始。加分号的目的是表示if子句的终止。这同多个命令语句位于同一命令行时中间必须加分号“;”分隔符是完全一样的，例如：

```
if [ condition ]; then
    command-list
else
    command-list
fi
```

7.1.2 嵌套的if-then 条件测试

if-then结构的条件测试还可以嵌套。第一种if-then嵌套结构的语法格式如下：

```
if [ condition1 ]
then
    if [ condition2 ]
    then
        command-list
    fi
fi
```

上述情况相当于使用&&逻辑运算符的组合测试：

```
if [ condition1 ] && [ condition2 ]
then
    command-list
fi
```

第二种if-then嵌套结构采用elif（else if的缩写）子句，其语法格式如下：

```
if [ condition1 ]
then
    command-list
elif [ condition2 ]
then
    command-list
else
    default-command
fi
```

通过使用elif子句，if语句可以实现多重测试结构。同样，如果elif子句后面的测试条件的出口状态为真，则执行紧随elif子句中then后面的语句。

例如，为了在用户注册时能够根据系统的当前时间显示不同的问候语，可以在.profile初始化文件中增加下列语句：

```
$ cat greeting
now=`date +%H`
if [ ${now} -lt 12 ]
then
    greeting="Good morning."
elif [ ${now} -lt 18 ]
then
    greeting="Good afternoon."
else
    greeting="Good evening."
fi
echo "$greeting"
exit 0
$ greeting
Good morning.
$ date
2009年 10月 18日 星期日 10:18:20 CST
$
```

每个嵌套的if语句必须以fi结束。作为一个组合语句，整个if语句的出口状态是then或else子句中执行的最后一条命令的出口状态。如果没有执行任何命令，其出口状态为0。

在下面的例子中，由于给定的文件不存在（文件名有误），ls命令将会返回一个非0的出口状态。因此，Shell脚本不会执行最后一个echo语句。

```
$ cat ifthen
fname=/etc/password                # 正确的文件名应为passwd
if echo "This statement always return true."
    ls -l $fname
then
    echo "Come here."
fi
$ ifthen
This statement always return true.
ls: cannot access /etc/password: No such file or directory
$
```

现在让我们交换if-then之间的两个语句。尽管ls命令将会返回一个非0的出口状态，但由于echo命令总是能够成功地执行，故Shell脚本将会执行到最后一个echo语句，然后才结束Shell脚本的执行。

```
$ cat ifthen2
fname=/etc/password                # 正确的路径文件名应为/etc/passwd
if ls -l $fname
    echo "This statement always return true."
then
    echo "Come here."
fi
$ ifthen2
/etc/passwd: No such file or directory
This statement always return true.
Come here.
$
```

7.2 case分支语句

case分支语句能够提供多路分支转移控制功能。case语句利用一个变量作为测试条件，变量可以具有多个值，不同的数值能够引起不同的程序走向。case语句的语法格式如下：

```
case "$variable" in
    pattern1)
        command-list ;;
    pattern2)
        command-list ;;
    .....
    patternN)
        command-list ;;
esac
```

Shell使用给定变量（variable）的值与每一个模式（pattern）依次进行比较。一旦发现匹配的模式，Shell将会立即执行紧随模式之后的所有命令，直至遇到双分号“;”结束。在双分号前的最后一个命令执行结束之后，Shell将跳转至紧随esac语句之后的第一个语句，开始继续执行。如果用做测试条件的变量值与任何模式都不匹配，则直接跳转到紧随esac语句之后的第一个语句处开始继续执行，而不执行case语句中的任何命令。

每个模式之后必须加一个右括号“)”，以便与随后的一组相关命令分隔。case语句中的每一组命令或代码块之后必须以双分号结束（最后一组命令或代码块除外），这相当于告诉Shell已执行到相关命令或代码块的边界。

模式中可以使用元字符，也可以使用运算符“|”，表示多个模式的逻辑或关系。由于元字符“*”可以匹配任何数值、字符或字符串，因此可以用做默认的匹配模式。这个“万能的模式”必须作为case语句的最后一个模式，因为一旦检查到匹配的模式，Shell将会停止执行case语句的模式匹配检查，致使其他模式永远得不到匹配的机会。

下面是一些与case语句语法格式有关的说明：

- 因为case语句中用做测试条件的变量值通常都是一个单独的数值或单词，不会出现包含分隔符的字符串，故测试变量前后可以不加双引号；
- 每个用于匹配检查的模式之后必须附加一个右圆括号后缀；
- 除了最后一组命令或代码块，其他模式的相关命令或代码块之后均需附加一个双分号；
- 整个case语句以esac（case的反序拼写）结束。

Shell中的case控制结构相当于C/C++中的switch语句，允许整个控制结构根据条件测试变量的值执行相应的一组命令或代码块。因此，case控制结构是一种非常适合建立多路分支转移程序流向的工具，也能够容易地创建选择菜单。实际上，case控制结构是if-then-else嵌套结构的一种简化形式。

例如，利用case控制结构，我们可以设计一个简单的系统信息查询脚本queryinfo，根据用户的选择，提供基本的系统信息。示例代码如下：

```
# cat queryinfo
#!/bin/sh
echo "\t----- System Information -----"
```

```

        1. Cuurent Date and time
        2. Host Name
        3. IP Address
        4. Memory Configuration
\tPlease enter your choice: \c"
read choice
case "$choice" in
    1 )  date ;;
    2 )  uname -n ;;
    3 )  cat /etc/hosts | grep `uname -n` | cut -f1 ;;
    4 )  prtconf | grep Memory ;;
    * )  echo "Invaild choice!"
esac
exit 0
# queryinfo
----- System Information -----
        1. Cuurent Date and time
        2. Host Name
        3. IP Address
        4. Memory Configuration
Please enter your choice: 1
Memory size: 1024 Megabytes
#

```

在UNIX系统的各种日常管理与维护Shell脚本中，按照使用的频繁程度，**case**语句也许是一种仅次于**if-then-else**语句的控制结构。尤其是/etc/init.d或/etc/rcN.d目录中的各种系统服务启动脚本，**case**语句是其中最常见的固定控制结构：利用同一个Shell脚本，在系统的启动、关机或重启过程中，分别以**start**、**stop**或**restart**等作为参数，启动、关闭或重新启动各种系统服务的守护进程。

下面是取自ufs.quota启动脚本文件的部分内容，其中就采用了典型的**case**控制结构：

```

$ cat /etc/init.d/ufs.quota
#!/sbin/sh
.....
case "$1" in
    'start')
        /usr/sbin/quotacheck -a
        /usr/sbin/quotaon -a
        ;;
    'stop')
        /usr/sbin/quotaoff -a
        ;;
    *)
        echo "Usage: $0 { start | stop }"
        exit 1
        ;;
esac
exit 0
$

```

在case命令语句中，还可以使用命令替换作为测试变量，以命令的输出作为测试变量的值。如使用arch命令输出的系统类型作为case语句中的测试变量值，以便在不同的系统中做特定的设置和处理。示例如下：

```
#!/bin/sh
# Using command substitution to generate a "case" variable.

case $(arch) in
    i386 )
        i386-specific-command ;;
    sun4 )
        sun4-specific-command ;;
    *)
        echo "unkown architecture."
        exit 1
        ;;
esac
exit 0
```

7.3 for循环语句

for语句是Shell中的一种最基本的循环控制结构。针对for语句中的每个参数，可以重复执行一组命令或代码块。for语句的语法格式如下：

```
for var [ in word-list ]
do
    command-list
done
```

其中，word-list是一系列参数，或称参数表，中间以空格为分隔符。对于参数表中的每一个参数值，重复执行一次do与done之间的命令。do与done之间的循环体command-list是一个代码块，可以是单个命令、一组命令，也可以是各种控制结构语句，包括for循环结构语句。

利用for循环结构，只要控制条件为真，即可重复执行一组命令或代码块。在每次循环过程中，Shell将会从给定的参数表中，依次取出一个参数值，并把参数值赋予给定的变量var，然后重复执行for循环体中的代码块。

下面以一个简单的例子说明for循环结构的执行过程：

```
for var in arg1 arg2 ..... argN
do
    echo $var
done
```

在上述例子中，第一次循环时把arg1赋予var变量，相当于执行“var=arg1”变量赋值语句，然后运行“echo \$var”命令；第二次循环时执行“var=arg2”变量赋值语句，然后再次运行“echo \$var”命令；依次类推，第N次循环时执行“var=argN”变量赋值语句，然后运行“echo \$var”命令。经过N次循环之后，整个for循环语句执行结束。

下面的例子进一步说明了for循环结构的执行过程。当提交一个命令，为了弄清执行的命令究竟位于哪个目录，尤其当存在不同版本的同名命令，而这些命令又处于不同的命令检索路径

时，可以使用PATH变量中设置的命令检索路径，找出第一个发现的命令位于哪一个目录，即可得到答案。PATH变量中包含一系列由冒号分隔的目录，Shell正是利用PATH变量，检索用户输入的命令的：

```
$ cat where
IFS="${IFS}:"
flag=0
for dirname in `echo ${PATH}`
do
    if test -x "${dirname}/${1}"
    then
        echo "${dirname}/${1}"
        flag=1
    fi
done
if [ $flag -eq 0 ]
then
    echo "The $1 is not exist."
fi
$
```

在执行上述Shell脚本时，其输出结果如下：

```
$ where echo
/usr/bin/echo
$ where findf
The findf is not exist.
$
```

注意：如果do与for位于同一行上，则do之前应加一个分号“;”分隔符。

```
for var in [word-list] ; do
```

下面的for循环利用diff命令，对当前目录下的C文件alpha.c、beta.c和gamma.c与/export/home/gqxing/src目录中的同名文件进行比较：

```
for fname in alpha beta gamma ; do
    diff ${fname}.c /export/home/gqxing/src/${fname}.c
done
```

参数表中的参数也可以包含元字符，故也可以利用Shell的“*”、“?”或“[...]”等通配符建立参数表。例如，我们可以利用元字符重写先前的for循环语句，比较当前目录与/export/home/gqxing/src目录中的每一个C文件：

```
for fname in *.c
do
    diff $fname /export/home/gqxing/src/$fname
done
```

参数表中的每个元素还可以包含多个参数。当需要按组处理参数时，这种形式是非常有用的。在此情况下，可以使用set命令，强制解析参数表中的每一个元素，把其中的每个参数分配到一系列位置参数中。

例如，在下面的for循环语句中，参数表中的每个元素本身包含两个参数：

```

for planet in "Mercury 36" "Venus 67" "Earth 93" "Mars 142" "Jupiter 483"
do
    set $planet      # 解析变量planet, 设置位置参数
    echo "$1 is $2,000,000 miles away from the sun."
done

```

此外, 也可以使用命令替换生成for循环中的参数表。例如, 假定因故需要修改当前目录及其子目录中所有文件的属主与同组属性, 我们可以利用“ls -R”命令递归地列出各级目录中所有文件的特点, 生成文件名参数, 然后再使用chown和chgrp命令修改文件的属性, 例如:

```

for fname in `ls -R`
do
    if [ -f $fname ]
    then
        chown gqxing "${fname}"
        chgrp gqxing "${fname}"
    fi
done

```

下面是另外一个使用命令替换生成for循环语句参数表的例子。这个Shell脚本的功能是从指定(或当前)目录中检索文件的大小超过1MB的文件(注意, 由于find命令“-size”选项的参数值后面未加任何后缀, 故默认的数据单位是512B的数据块, 而ls命令“-s”选项的数据单位是1KB)。

```

$ cat bigfile
#!/bin/sh
dir=${1-`pwd`}
echo "Big file(in 1KB-block) under directory \"$dir\""
cd $dir
for i in `find . -size +2048 -print`
do
    if [ -f "${i}" ]
    then
        ls -s "${i}"
    fi
done
exit 0
$ bigfile /var/sadm
Big file(in 1KB-blocks) under directory "/var/sadm"
35504 ./install/contents
2768 ./pkg/SUNWcslr/save/pspool/SUNWcslr/reloc/lib/libc.so.1
24880 ./wbem/logr/store
2272 ./smc/properties/classlist.txt
$

```

在for循环语句中, “in word-list”部分可以省略, 在此情况下, for循环语句需要使用运行Shell脚本时提供的命令行参数作为参数表。这种for循环结构尤其适用于编写一种通用的Shell脚本, 针对给定的任何参数, 执行同样一组处理命令。其简化的语法格式如下:

```

for variable
do
    command-list
done

```

假定公司有一个员工的电话号码簿，其数据内容如下：

```
$ cat phonebook
Cathy      617-495-1585      Boston
Jim        650-723-2300      Stanford
Ruth       516-367-8800      New York
Tom        410-516-8000      Baltimore
.....
$
```

我们可以写出下列Shell脚本，用于查阅任何给定员工的信息：

```
$ cat findname.sh
#!/bin/sh
for name
do
    if grep $name phonebook
    then
        :
    else
        echo "$name is not in the phonebook."
    fi
done
$
```

例如，为了查阅Cathy、Jim和XXX的电话号码，我们可以按照下列方式运行Shell脚本：

```
$ findname.sh Cathy Jim XXX
Cathy      617-495-1585      Boston
Jim        650-723-2300      Stanford
XXX is not in the phonebook.
$
```

整个for循环语句的输出可以重定向或通过管道输出到一个文件、命令或一组命令。根据这个特点，我们可以利用管道和sort排序工具，对上述的bigfile脚本做少许改进，把整个for循环的输出作为sort命令的输入，使得Shell脚本能够按文件的大小从大到小排序，然后顺序列出指定目录（或当前目录）中大于1MB的文件。示例代码如下：

```
$ cat bigfile2
#!/bin/sh
dir=${1-`pwd`}                                # 使用指定的或当前目录
cd $dir
echo "Big files(in 1KB-block) under directory \"$dir\""
find . -size +2048 2>/dev/null |               # 找出容量大于1MB的文件
while read file
do
    if [ -f "${file}" ]
    then
        ls -s "${file}"
    fi
done | sort -rn                                # 按文件容量从大到小排序
exit 0
$ bigfile2 /var/sadm
```

```
Big files(in 1KB-block) under directory "/var/sadm"
35504 ./install/contents
24880 ./wbem/logr/store
2768 ./pkg/SUNWcslr/save/pspool/SUNWcslr/reloc/lib/libc.so.1
2272 ./smc/properties/classlist.txt
$
```

7.4 while循环语句

while语句的功能是根据一定的条件，循环执行一组特定的命令。**while**循环结构在循环体的前部测试执行条件。只要控制条件的测试结果为真（返回值为0），便继续执行循环体中的命令，否则立即结束整个**while**循环。与**for**循环相比，**while**循环更适合循环次数事先不确定的情况。其语法格式如下：

```
while [ condition-is-true ]
do
    command-list
done
```

下面是一个简单的**while**循环结构，其目的是求出1~100之间所有整数的总和。当\$number变量的值大于100时，Shell将会跳到done后面的echo语句开始执行，输出\$sum变量的值，也即1~100之间所有整数的总和，其代码如下：

```
$ cat sum100
#!/bin/sh
number=1
sum=0
max=100

while [ "$number" -le "$max" ]
do
    sum=`expr $sum + $number`
    number=`expr $number + 1`
done
echo $sum
$
```

同**for**循环一样，如果do与while位于同一行上，do与测试条件之间也应加一个分号“;”分隔符：

```
while [ condition-is-true ] ; do
```

在**while**循环结构中，测试条件可以是一个简单的test语句，也可以是由多个条件表达式与逻辑运算符组成的组合测试语句；可以是一个普通的UNIX命令，也可以是一组命令。循环体中可以包含任何命令语句，当然也包括任何结构语句。

在**while**循环结构中，当控制条件由多个命令语句组成时，实际上只有最后一个命令语句的出口状态才能决定是否继续执行循环体。也就是说，**while**语句仅仅检查do之前最后一条命令的返回值，如果返回值为0，则继续执行do与done之间的代码体；如果返回值大于0，Shell将会终止执行整个**while**循环语句，然后立即跳转并执行紧随done之后的命令语句。

在下面的例子中，用做while循环判断条件的第一个语句，即echo命令总是能够正常地运行。而read命令的特性恰好可以用做while循环结构的判断条件，当读取一个Ctrl-D键时，read命令将会返回一个非0的出口状态，因而可用于结束while循环。

```
$ cat lookup
while
    echo "\nEnter name to searched (Press Ctrl-D to end): \c"
    read name
do
    if test "${name}" = ""
    then
        continue
    else
        lookup.sql "${name}"      # lookup.sql是一个MySQL脚本，用于检索数据库
    fi
done
exit 0
$
```

如果在关键字done后面增加一个重定向符号“<”，while循环结构的标准输入可以重定向到一个文件。另外，while循环结构的标准输入也可以由一个管道提供。

下面是一个利用管道为while循环结构提供数据，由while循环体逐行读取cat命令输出的数据，并对数据进行加工处理的例子（这里的处理仅为显示读取的数据）。

```
$ cat check
#!/bin/sh
line_num=1
cat $1 |
while read text
do
    echo "${line_num}: ${text}"
    line_num=`expr ${line_num} + 1`
done
exit 0
$
```

作为测试数据的文件内容如下（选自泰戈尔《Stray Birds》）：

```
$ cat stray.birds
"What language is thine, O Sea?"
"The language of eternal question."
"What language is thy, Osky?"
"The language of eternal silence."
$
```

运行上述脚本后，其输出结果如下：

```
$ check stray.birds
1: "What language is thine, O Sea?"
2: "The language of eternal question."
3: "What language is thy, Osky?"
4: "The language of eternal silence."
$
```


7.5 until循环语句

until语句的功能是在一定的条件下，循环执行一组特定的命令。**until**循环结构也是在循环体的前部测试循环控制条件。但与**while**循环结构不同的是，**until**循环结构测试的是终止条件。如果控制条件的测试结果为真（返回值为0），则立即结束整个**until**循环。如果控制条件的测试结果为假（返回值大于0），便继续执行循环体中的命令，直至控制条件的测试结果为真，这一点恰与**while**循环相反。**until**语句的语法格式如下：

```
until [ condition-is-true ]
do
    command-list
done
```

同**for**和**while**循环一样，如果**do**与条件测试语句位于同一行上，中间需要加分号“;”分隔符。

```
until [ condition-is-true ] ; do
```

下面是一个采用Euclid算法，利用**until**循环求取最大公约数的例子。首先选取第一个参数作为被除数并赋予**dividend**变量，第二个参数作为除数赋予**divisor**变量。在**until**结构语句的每一次循环过程中，再把当前的除数赋予**dividend**变量，余数赋予**divisor**变量，作为下一次运算的因子。重复上述运算，直至余数为零，最后一个**dividend**即为最大公约数。整个循环运算过程结束。以下是该例子的代码：

```
$ cat gcd
#!/bin/ksh
if [ $# -ne 2 ]
then
    echo "Usage: `basename $0` Number#1 Number#2"
    exit 1
fi
dividend=$1
divisor=$2
remainder=1
until [ "$remainder" -eq 0 ]
do
    let "remainder = $dividend % $divisor"
    dividend=$divisor
    divisor=$remainder
done
echo "The greatest common divisor of $1 and $2 = $dividend"
exit 0
$ gcd 7856 9980
The greatest common divisor of 7856 and 9980 = 4
$
```

7.6 select循环语句

select循环结构源于Korn Shell，主要用于建立Select菜单。**select**菜单的最大特点是，编程人员只需提供作为菜单选择项的参数表，菜单的组织与表现形式由**select**语句负责实现。**select**语句的语法格式如下：

```
select variable [in list]
do
    command-list
break
done
```

执行**select**语句时，Shell将提示用户根据参数表中的选择项，通过输入相应的数字做出选择。通常，**select**语句使用PS3环境变量的值（#?）作为命令提示符。如果需要，编程人员可在**select**语句之前，赋予PS3环境变量一个新的值，以定制命令提示符。

下面便是一个使用**select**语句建立菜单的例子：

```
$ cat menu
#!/bin/bash
PS3='Choose your favorite book: '      # 设置select语句使用的命令提示符
echo
select book in "The Da Vinci Code" "Pride and Prejudice" "Wuthering Heights"
"The Old Man and the Sea"
do
    echo
    echo "Your favorite book is $book."
    break                               # 使用break语句退出select循环
done
exit 0
$
```

运行上述Shell脚本时，其处理结果如下：

```
$ menu
1) The Da Vinci Code           3) Wuthering Heights
2) Pride and Prejudice         4) The Old Man and the Sea
Choose your favorite book: 1

Your favorite book is The Da Vinci Code.
$
```

在**select**语句中，“in list”部分可以省略，如果省略了参数表，**select**语句需要使用运行Shell脚本时提供的命令行参数作为参数表。这种处理方式使编写的Shell脚本更通用，更灵活。

```
$ cat menu2
#!/bin/bash
PS3='Choose your favorite book: '
echo
select book
do
    echo
```

```

        echo "Your favorite book is $book."
        break
    done
    exit 0
$ menu2 "The Da Vinci Code" "Pride and Prejudice" "Wuthering Heights" "The Old
Man and the Sea"

1) The Da Vinci Code          3) Wuthering Heights
2) Pride and Prejudice        4) The Old Man and the Sea
Choose your favorite book: 2

Your favorite book is Pride and Prejudice.
$

```

下面是一个在函数中利用select语句实现菜单的例子。其效果与用法同第一个例子完全一样。

```

$ cat menu3
#!/bin/bash
PS3='Choose your favorite book: '
choice()
{
    echo
    select book
    do
        echo
        echo "Your favorite book is $book."
        break
    done
}
choice "The Da Vinci Code" "Pride and Prejudice" "Wuthering Heights" "The Old
Man and the Sea"
exit 0
$

```

7.7 嵌套的循环

嵌套的循环意味着循环体中包含内嵌的循环语句。外层循环体每执行一次循环，内层循环体就要执行一个完整的循环，直至外层循环结束。当然，也可以利用循环体中的break语句中断相应循环体的继续处理。

例如，下面的例子就是利用嵌套的循环，列出指定目录（或当前目录）的目录层次结构树：

```

$ cat tree.sh
#!/bin/sh
# Written by Rick Boivie. Revised and simplified
# by Jordi Sanfeliu (patched by Ian Kjos).
# Last revised by GQ Xing.
search () {
    for dir in `echo *`
    do
        if [ -d "$dir" ] ; then
            # `echo *`用于列出当前目录下的所有文件
            # 如果$dir是一个目录

```

```

dirlevel=0                                     # 临时变量, 用于记录目录层次
while [ $dirlevel != $1 ]                       # 记录内部嵌套的循环
do
    echo "| \c"                                  # 表示目录层次, 中间不换行
    dirlevel=`expr $dirlevel + 1`
done
if [ -L "$dir" ] ; then                         # 如果目录是一个符号链接
    echo "|--$dir" `ls -l $dir | sed 's/^.*'$dir' //'`
    # tt=`ls -l $dir`
    # echo "|--`$tt | cut -f9-11 -d' '`"
else                                             # 显示横向连接符号并列出原目录及其指
                                                # 向的目录名
    echo "|--$dir"                               # 显示横向连接符号和目录名
    numdirs=`expr $numdirs + 1`                 # 增加目录计数
    if [ -x "$dir" ] ; then                     # 检查目录执行权限
        cd "$dir"
        search `expr $1 + 1`                   # 递归地调用自己
        cd ..
    else
        echo "Changed to above directory is denied"
    fi
fi
fi
done
}

cd ${1:-`pwd`}                                # 移动到指定目录或当前目录
echo "Initial directory = `pwd`"
numdirs=0
search 0
echo "Total directories = $numdirs"
exit 0

$ tree /etc/svc
Initial directory = /etc/svc
|--volatile
| |--svc:
| | |--application
| | | |--font
| | | | |--stfsloader:default
| | | | |--print
| | | | |--rfc1179:default
| | | | |--x11
| | | | |--xfs:default
| | | | |--xvnc-inetd:default
| | |--network
| | | |--cde-spc:default
| | | |--chargen:dgram
| | | |--chargen:stream
.....
$

```

7.8 循环控制与辅助编程命令

这一节主要介绍影响循环行为的控制命令与辅助编程命令语句。

7.8.1 break和continue命令

break和**continue**循环控制命令基本上等同于C或其他编程语言中**break**和**continue**语句的功能。**break**命令用于跳出相应的循环体，终止循环体的继续执行。而**continue**命令则用于越过循环体中尚未执行的命令语句，结束本次循环，直接跳转到下一次循环迭代。

break命令的主要用途是立即停止执行当前的循环体，然后跳转到循环体外（即关键字**done**后面）的命令语句处开始继续执行。**break**命令的语法格式如下：

```
break [n]
```

break命令后面可以附加一个数字参数。一个简单的**break**命令只是终止执行最内层的循环体。但是，如果**break**命令后面有一个数字**n**，则“**break n**”命令能够跳出**n**层循环体，将控制转移到外部第**n**个关键字**done**后面的命令语句处开始继续执行。

continue命令的主要功能是越过本次循环中余下的所有命令语句，从循环体的第一条命令语句开始继续执行循环体。其语法格式如下：

```
continue [n]
```

类似于**break**，**continue**命令也可以附带一个数字参数。一个简单的**continue**命令只是终止执行当前循环中尚未执行的命令语句，继续执行下一次循环。如果**continue**命令后面有一个数字**n**，则“**continue n**”命令能够终止执行当前循环，从嵌套的外部第**n**层循环体的起始命令语句开始继续执行下一次循环。

注意：**break**和**continue**命令只能用于**for**、**while**和**until**等循环语句的循环体中，位于关键字**do**与**done**之间。

下面的Shell脚本说明了**break**和**continue**命令在**while**循环中的作用。其中的**break**和**continue**命令用于控制用户的输入过程。这个Shell脚本的功能是提示用户连续地输入由空格分隔的字符串数据，并把接收的数据存储到一个文件中。直至用户输入**quit**时，退出**while**循环。接着对文件中的数据进行排序，最后把已排序的数据再存回原文件中。

```
$ cat whiledo
#!/bin/sh
> datafile
while :
do
    echo "Please enter data (enter 'quit' to end): \c"
    read response
    case "$response" in
        quit) break ;;                # 停止输入，结束Shell脚本的运行
        *)   continue ;;             # 继续读取下一个数据
    esac
    echo "$response" >> datafile
done
```



```

sort -o datafile datafile          # 对文件中的数据进行排序，排序后仍保存到原文件中
exit 0
$ whiledo
Please enter data (enter 'quit' to end): 888888
Please enter data (enter 'quit' to end): 222222
Please enter data (enter 'quit' to end): 999999
Please enter data (enter 'quit' to end): 666666
Please enter data (enter 'quit' to end): 111111
Please enter data (enter 'quit' to end): quit
$ cat datafile
111111
222222
666666
888888
999999
$

```

下面是一个使用**break**和**continue**命令直接跳出多层循环，或从外层循环体继续执行下一次循环的例子：

```

$ cat levels
#!/bin/sh
while :
do
    echo "Entering level 1"
    while :
    do
        echo "\tEntering level 2"
        while :
        do
            echo "\t\tEntering level 3"
            echo "\t\t\tInput c to continue \c"
            echo "or b to break: \c"
            read cmd
            echo "\t\t\tHow many levels? \c"
            read level
            case ${cmd} in
                [bB]*) break ${level} ;;
                [cC]*) continue ${level} ;;
            esac
            echo "\t\tLeaving level 3"
        done
        echo "\tLeaving level 2"
    done
    echo "Leaving level 1"
done
exit 0
$ levels
Entering level 1
    Entering level 2
        Entering level 3
            Input c to continue or b to break: C

```

```

        How many levels? 2
    Entering level 2
        Entering level 3
            Input c to continue or b to break: B
            How many levels? 2
Leaving level 1
Entering level 1
    Entering level 2
        Entering level 3
            Input c to continue or b to break: C
            How many levels? 3
Entering level 1
    Entering level 2
        Entering level 3
            Input c to continue or b to break: b
            How many levels? 3
$

```

注意：“continue n”命令的组织、应用和理解都比较困难，应尽量避免使用。

7.8.2 true命令

true命令除了用于返回一个表示测试成功的返回值0之外，不执行其他任何动作。因此，true命令经常用于无限循环的循环结构。

事实上，true与冒号“:”命令可以互换。但“:”命令是一个Shell内置命令，其执行速度要快于UNIX系统提供的外部命令true。

7.8.3 sleep命令

sleep命令使Shell能够暂时休眠一定的时间，然后再执行下一个命令。因此，为了在执行两个命令之间暂停一定的时间，然后再执行，可以在Shell脚本中使用sleep命令。sleep命令的语法格式如下：

```
sleep [n]
```

其中，选用的参数n是暂停继续执行的时间间隔（以秒为计量单位）。sleep命令经常用于保持屏幕输出内容的暂时稳定，例如，为了检查当前正在复制的文件大小的变化情况，或查看某个进程是否继续活动，以便得出某种判断时，可以使用下列命令：

```

$ while true
> do
> ls -l fname
> sleep 5
> done
$

```

或

```

$ while :
> do
> ps -ef | grep proc_name
> sleep 5

```

```
> done
$
```

7.8.4 shift命令

shift命令用于修改位置参数的值。按照命令参数指定的数量，所有的位置参数逐次向左平移。最左边的位置参数逐一移走，最右边的位置参数逐一左移后清除。同时，位置参数的总数也相应地减少。**\$0**是脚本文件的名称，在左移的过程中不受影响。**shift**命令语句的语法格式如下：

```
shift [n]
```

其中，**n**是左移的数量，默认值为1。每执行一次“**shift n**”命令语句，整个位置参数都会左移**n**个位置，表示位置参数总数的**\$#**变量值相应地减少**n**，表示全部位置参数的**\$***或**\$@**变量也相应地删除最左边的**n**个参数。

下面的例子说明了**shift**命令的典型用法。其中，**while**语句总是对**\$1**变量进行测试。由于使用了**shift**语句，每次循环之后，**\$1**的变量值都会发生变化。**case**语句依次解析每个命令行参数，并输出相应的信息。这个脚本假定“-a”选项之后必须指定文件名参数。脚本利用**while**、**case**和**shift**语句连续地检测位置参数，直至匹配“-a something”或“-b”，然后执行相应的代码。

```
$ cat options
#!/bin/sh
while test "${1}" != ""
do
    case "${1}" in
        -a* ) fname=`echo ${1} | sed 's/-a//'\`
                echo "Option a selected"
                echo "File name is ${fname}"
                option_a="yes"
                ;;
        -b )   echo "Option b selected"
                option_b="yes"
                ;;
    esac
    shift
done
$ options -aindex
Option a selected
File name is index
$ options -b
Option b selected
$ options -aindex -b
Option a selected
File name is index
Option b selected
$
```

7.8.5 getopt命令

getopt命令主要用于解析命令行选项，检查选项的合法性，为Shell编程提供方便。**getopt**命

令的用法如下:

```
set -- 'getopt optstring $*'`
```

其中, **optstring**是一个包含所有合法命令选项的字符串。如果选项字母之后附有一个冒号“:”, 意味着相应的选项需要提供一个选项参数(选项字母与选项参数之间既可以存在空格分隔符, 也可以直接连接在一起)。“-”特殊选项用于界定命令选项的结束。不管存在与否, **Shell**都会把所有的普通选项之后放置一个“-”选项。

下面是一个取自网上的应用实例, 其主要功能是根据用户提供的参数, 利用**ftp**实现文件的自动下载。其中说明了**ftpget**脚本是怎样利用**getopt**命令处理其合法选项“-h”、“-d”、“-c”、“-f”和“-m”, 解析和验证命令行参数的。

```
$ cat ftpget
#!/bin/bash
CMDFILE=/tmp/ftp.$$
TMPFILE=/tmp/ftp2.$$
usage="Usage: $0 [-h rhost] [-d rdir] [-c ldir] [-f rfile:lfile] [-m
fpattern]"
ftpopts="-i -n -v"
set -f
set -- `getopt h:d:c:f:m: $*`
# echo $*
if [ $# -lt 2 -o "$?" -ne 0 ]; then
    echo $usage
    exit 1
fi
trap 'rm -f ${CMDFILE} ${TMPFILE}; exit' 0 1 2 3 15
echo "user gqxing 123456" > ${CMDFILE}
for i in $*
do
    case $i in
        -h)    remhost=$2; shift 2;;
        -d)    echo cd $2 >> ${CMDFILE};
                echo pwd >> ${CMDFILE};
                shift 2;;
        -c)    echo lcd $2 >> ${CMDFILE}; shift 2;;
        -m)    echo mget "$2" >> ${TMPFILE}; shift 2;;
        -f)    f1=`echo "$2" | cut -d: -f1`; f2=`echo "$2" | cut -d: -f2`;
                echo get ${f1} ${f2} >> ${TMPFILE}; shift 2;;
        --)    shift; break;;
    esac
done
if [ $# -ne 0 ]; then
    echo $usage
    exit 2
fi
echo quit >> ${TMPFILE}
cat ${TMPFILE} >> ${CMDFILE}
ftp ${ftpopts} ${remhost:-169.254.78.100} < ${CMDFILE}
cat ${CMDFILE} >> ftplog
```

```
rm -f ${CMDFILE} ${TMPFILE}
exit 0
$ ftpget -d docs -c /tmp -f design
Connected to 169.254.78.100.
220 (vsFTPd 2.0.6)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
250 Directory successfully changed.
257 "/export/home/gqxing/docs"
Local directory now /tmp
local: design remote: design
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for design (308522 bytes).
226 File send OK.
308522 bytes received in 0.01 secs (53802.0 kB/s)
221 Goodbye.
$
```

7.8.6 getopts命令

getopts命令是对**getopt**命令的更新和替代品。最初推出**getopts**命令的本意是取代**getopt**命令，但实际上这两个命令一直共存于UNIX及Linux系统中。**getopts**命令用于获取以减号“-”或加号“+”为起始字符的命令选项、选项参数和命令参数。其语法格式如下：

```
getopts opts string [arg...]
```

在编写Shell脚本时，最好能够养成使用**getopts**或**getopt**命令获取命令行选项、选项参数和命令参数的习惯。利用**getopts**命令，用户可在运行Shell脚本时以任何顺序输入命令选项。不带选项参数的命令选项既可以单独给出，也可以组合给出，但本身带有选项参数的命令选项需单独给出。

getopts命令语句采用一个合法的选项字符串确定Shell脚本能够接受的命令选项。例如，下述**getopts**命令定义的合法选项字符串为**ior**，表示Shell脚本能够接受的命令选项为“-i”、“-o”和“-r”：

```
getopts ior string
```

每次调用**getopts**命令语句时，命令行中的选项就会依次存入给定的变量**string**中。因此，Shell脚本可以对变量**string**的值做适当的处理。

在运行Shell脚本时，最后一个命令选项与命令参数之间还可以插入一个位置参数“_”。这是一个标志，表示命令选项的结束和命令参数的开始。

如果在运行脚本时提供了非法参数，**getopts**命令将会输出下列形式的错误信息，同时把一个问号“?”字符赋予**string**变量：

```
script_name: illegal option -- option_letter
```

getopts命令的返回值取决于是否能够成功地获取命令选项。如果读取了命令选项（包括非法选项），**getopts**将返回数值0；否则，当遇到第一个位置参数“_”，第一个非选项参数（即

命令参数)，或命令行结束标志时，`getopts`命令语句将返回一个非0数值。

下面的例子说明了怎样使用`getopts`命令语句处理简单的命令行选项：

```
$ cat getoptions
#!/bin/sh
while getopts abc var
do
    case ${var} in
        a) echo Option -a found ;;
        b) echo Option -b found ;;
        c) echo Option -c found ;;
    esac
done
exit 0
$ getoptions -b -a
Option -b found
Option -a found
$ getoptions -adc
Option -a found
getoptions: illegal option -- d
Option -c found
$
```

如果选项带有参数，则需要在相应的选项字母后面附加一个冒号“:”字符。在读取冒号之后，`getopts`将会把选项后的选项参数字符串赋予`OPTARG`变量。注意，选项与选项参数之间可以有空格分隔符，也可以连接在一起。

如果一个选项要求提供选项参数而未提供，`getopts`语句将会输出下列形式的错误信息：

```
script_name: option requires an argument - options_letter
```

`getopts`语句执行结束时，`OPTIND`变量的值将会指向第一个非选项参数的位置参数，也即第一个命令参数。如果位置参数的值为“-”，则忽略之。为了便于脚本按惯例处理命令参数，应将\$1指向第一个命令参数，\$2指向第二个命令参数，如此等等。因此，我们可以利用`shift`命令，使之左移适当的位置。按照上面的说明，我们只需左移“\$OPTIND-1”个位置，即可使\$1指向第一个命令参数。

在下面的例子中，Shell脚本可接收4个选项，其中“-o”和“-r”选项要求提供相应的参数。在运行Shell脚本时，任何一个选项既可以提供，也可以不提供。如果提供了“-o”或“-r”选项，则必须提供相应的参数。

```
$ cat getoptions2
#!/bin/sh
while getopts o:r:nt var
do
    case ${var} in
        o) output_file="${OPTARG}" ;;
        r) report_file="${OPTARG}" ;;
        n) number_option="yes" ;;
        t) title="no" ;;
        \?) exit 2 ;;
    esac
done
```

```

done
echo "Output file = ${output_file}"
echo "Report file = ${report_file}"
echo "Numbering option = ${number_option:-no}"
echo "Title option = ${title:-yes}"
echo "Arguments before shift: ${*}"
shift `expr ${OPTIND} - 1`
echo "Arguments after shift: ${*}"
exit 0

$ getoptions2 -tn -o ofile -r rfile unit.12
Output file = ofile
Report file = rfile
Numbering option = yes
Title option = no
Arguments before shift: -tn -o ofile -r rfile unit.12
Arguments after shift: unit.12

$ getoptions2 -tn *
Output file =
Report file =
Numbering option = yes
Title option = no
Arguments before shift: -tn unit.01 unit.02 unit.03
Arguments after shift: unit.01 unit.02 unit.03

$ getoptions2 unit.12
Output file =
Report file =
Numbering option = no
Title option = yes
Arguments before shift: unit.12
Arguments after shift: unit.12

$ getoptions2 -tn -o ofile -r
getoptions2: option requires an argument -- r
$

```

7.9 循环语句的I/O重定向

作为一个整体，**while**、**until**与**for**循环结构语句中的循环体（包括**while**或**until**与**do**之间的测试条件语句）也可以实现I/O重定向。对于循环结构语句来讲，所谓的I/O重定向主要是重定向循环体中的标准输入。当然，也可以利用I/O重定向和管道的方法，把循环体中的输出结果保存到文件中，以备将来查阅。

另外，函数也可以采用同样的处理方式，实现I/O重定向。为了实现标准输入的I/O重定向，可在循环结构语句的结束标志（如关键字**done**）后面附加重定向符号“<”和适当的输入文件。

这里有一个问题需要说明，在Bourne Shell中，**while**、**for**和**until**等循环结构是在一个独立的子Shell环境中运行的。在同一个Shell脚本中，即使变量的名字完全相同，循环结构内外的同名变量也是完全不同的，无法直接传递数据。因此，在采用管道方式实现循环结构I/O重定向的Shell脚本中，如果存在利用变量传递数据的情况，应避免在Bourne Shell中运行，但可在Korn Shell和Bash等Shell中正确地运行。为了保险起见，在编写复杂的Shell脚本时，可在脚本第一行

中明确引用Korn Shell或Bash等Shell。

7.9.1 while循环的I/O重定向

对于while循环结构语句而言，下列语法格式表示循环体中需要的所有输入数据均取自指定的输入文件：

```
while [ condition-is-true ]
do
    command-list
done < datafile
```

下面的例子是取自一个安装脚本的部分代码段。由于while循环结构语句的标准输入已重定向到/etc/vfstab文件，故循环体中read语句读取的数据均来自/etc/vfstab文件。在while循环中，每执行一次read语句，即从/etc/vfstab文件中读取一行数据，并把有效数据分别赋值到bdev、rdev、mntp、fstype、fsckpass、automnt和mntopts等7个变量中。

这个脚本段的目的是检查系统中是否存在独立的/var文件系统。如果存在/var文件系统，则利用fsck命令检测并修复/var文件系统。

```
# cat checkfs
#!/usr/bin/ksh
while read bdev rdev mntp fstype fsckpass automnt mntopts
do
    if [ "${mntp}" = "/var" ]    # 如果发现/var文件系统，检测并安装
    then
        fsck -F ${fstype} -m ${rdev} > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
            echo "The $mntp file system (${rdev}) is being checked."
            fsck -F ${fstype} -y ${rdev}
        fi
        mount -F ${fstype} ${bdev} ${mntp} > /dev/null 2>&1
        break
    fi
done < /etc/vfstab                # 把read命令的标准输入重定向到/etc/vfstab文件
exit 0                            # 为了验证，这个文件也作了修改

# checkfs
The /var file system (/dev/rdisk/c0d0s4) is being checked.

** /dev/rdisk/c0d0s4
** Currently Mounted on /var
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
94 files, 942 used, 10066886 free (22 frags, 1258358 blocks, 0.0% fragmenta-
tion)

#
```

7.9.2 until循环的I/O重定向

until循环结构的I/O重定向与while循环结构几乎完全雷同，下面的脚本就是根据上述例子改写的。除了采用until循环结构，其功能完全一样。

```
# cat checkfs2
#!/usr/bin/ksh
# 直至遇到/etc/vfstab文件结束标志才终止until循环
until ! read bdev rdev mntp fstype fsckpass automnt mntopts
do
    if [ "${mntp}" = "/var" ]          # 如果发现/var文件系统，执行fsck命令，
    then                               # 然后退出until循环
        fsck -F ${fstype} -m ${rdev} > /dev/null 2>&1
        if [ $? -ne 0 ]
        then
            echo "The $mntp file system (${rdev}) is being checked."
            fsck -F ${fstype} -y ${rdev}
        fi
        mount -F ${fstype} ${bdev} ${mntp} > /dev/null 2>&1
        break
    fi
done < /etc/vfstab                    # 把read命令的标准输入重定向到/etc/vfstab的文件
exit 0
$
```

7.9.3 for循环的I/O重定向

for循环结构也可以实现I/O重定向，使循环体中的输入输出命令能够直接读取或写入指定的文件。许多日常维护任务就是利用I/O重定向，在不改写Shell脚本的情况下，只需修改输入文件，即可实现特定的系统维护目的。

在下面的例子中，我们的目的是找出指定目录或默认目录中两个月以来未曾访问过的文件。为了保存查询的结果，我们稍做变化，利用tee命令实现for循环体标准输出的I/O重定向，在显示检索结果的同时，把输出数据保存到/tmp/filelist文件中。示例代码如下：

```
$ cat oldfile
#!/bin/sh
dir=${1-`pwd`}          # 使用指定的目录或当前目录
cd ${dir}
echo "Old files under directory \"${dir}\""
for file in `find . -type f -atime +60 -print 2>/dev/null`
do
    if [ -f "${file}" ]
    then
        ls -l "${file}"
    fi
done | tee /tmp/filelist  # 显示并保存查询结果
exit 0
$ oldfile /sbin
Old files under directory "/sbin"
-r-xr-xr-x  1 root    bin          10080 Jan  9  2007 ./biosdev
```

```

-r-xr-xr-x 1 root bin 97768 Nov 17 2007 ./dhcpageant
-r-xr-xr-x 1 root bin 9964 Aug 15 2007 ./dhcpcinfo
-r-xr-xr-x 1 root bin 10076 Jan 23 2005 ./hostconfig
-r-xr-xr-x 1 root bin 11548 Jan 23 2005 ./ifparse
-r-xr-xr-x 1 root sys 18380 Oct 4 2008 ./installgrub
-r-xr-xr-x 1 root bin 10032 Jan 23 2005 ./metadevadm
-r-xr-xr-x 1 root bin 14216 Jan 23 2005 ./metainit
-r-xr-xr-x 1 root bin 9976 Jan 23 2005 ./metarecover
-r-xr-xr-x 1 root bin 22728 Jan 23 2005 ./metastat
$

```

此外，**if-then**条件转移语句中的代码块也可以实现I/O重定向，只是实际意义并不大，除非把**if-then**代码块放到一个循环结构中。

总之，在Shell脚本中，灵活地利用I/O重定向，能够生成必要的运行报告和日志文件，完整地记录脚本的处理过程，对脚本的调试和以后的档案留存都有重要的意义。

7.10 here文档

here文档是一种具有特殊用途的代码块，是I/O重定向的一种特例。**here**文档采用I/O重定向的方法，把一系列需要从键盘上输入的命令，模拟人工输入方式，逐行提交给交互式应用程序或命令，如**ftp**和MySQL数据库的**mysql**等。**here**文档的语法格式如下：

```

program <<LimitString
command1
command2
.....
commandN
LimitString

```

其中，特殊的I/O重定向符号“<<”与“**LimitString**”表示**here**文档的开始，单独另起一行的第二个“**LimitString**”表示**here**文档的结束。“**LimitString**”是启动指定程序后中间一系列交互命令的分界符。I/O重定向的效果是把**here**文档列出的命令提交给交互式应用程序或命令的标准输入。**here**文档的作用相当于执行下列命令：

```
interactive-program < command-file
```

其中，**command-file**包含一系列需要人工输入的命令或数据：

```

command1
command2
.....
commandN

```

下面是一个利用**here**文档维护数据库的例子。当MySQL数据库从一个服务器迁移到另一个服务器时，在安装MySQL数据库软件之后，首先需要赋予用户建立数据库的权限，然后还需要把先前利用**mysqldump**命令备份的数据库、数据库表及其数据加载到新的MySQL数据库中。为了简化人工操作步骤，可以使用下列Shell脚本实现数据库的重建：

```

$ cat makedb
#!/bin/sh

```

```
mysql -u root -psqladmin <<EOF
USE mysql;
GRANT ALL PRIVILEGES ON books.* TO gqxing@"localhost" IDENTIFIED BY 'alb2c3';
FLUSH PRIVILEGES;
quit
EOF
mysql -u gqxing -palb2c3 books < /tmp/bookdump.sql
exit 0
$
```

在上述here文档中，GRANT语句表示赋予用户gqxing在当前的系统中创建数据库的权限。here文档之后的mysql命令用于导入先前备份的数据库、数据库表及其数据。

注意：选择字符串分界符“LimitString”时应确保其在Shell脚本中是唯一的，至少不应在其界定的一系列命令中间出现，以免产生混淆。

下面的例子说明怎样利用here文档运行vi编辑器，模拟vi编辑器的交互过程，输入“i”命令和Esc键，插入两行数据，最后把编辑的数据内容写入指定的文件。

```
$ cat emuvi
#!/bin/sh
if [ -z "$1" ]
then
    echo "Usage: `basename $0` filename"
    exit 1
fi
vi $1 <<EOF
i
I cannot choose the best.
The best choose me.
^[
ZZ
EOF
exit 0
$
```

使用下列命令运行Shell脚本时，即可得到一个自动编辑的文本文件：

```
$ emuvi fname
$ cat fname

I cannot choose the best.
The best choose me.

$
```

注意：在上述Shell脚本中，“^”是一个字符，表示Esc键。为了在vi编辑器中输入Esc键，可以先按Ctrl-V键，接着再按Esc键，即可得到一个单字符的“^”字符。此外，上述编辑命令形式将会在实际数据内容前后各加一个空行。为了避免这种情况出现，可以对上述的编辑形式稍加修改，最终得到一个只含两行数据内容的文件。修改的结果如下：

```
vi $1 <<EOF
iI cannot choose the best.
The best choose me.^[
```



```
ZZ
EOF
```

对于非交互式的实用程序或命令，有效地利用**here**文档，有时也会取得非常好的效果。下面是一个利用**here**文档和**cat**命令输出多行信息的例子：

```
cat <<End-of-message
=====
The system ${NODENAME} will be shut down in ${time}.
Please logoff as soon as possible.
=====
End-of-message
```

事实上，如果单纯为了显示多行数据，大可不必使用**here**文档，只需使用简单的**echo**命令，即可实现多行信息的输出。**here**文档的主要用途在于模拟执行各种交互式程序或命令。例如：

```
echo "
=====
The system ${NODENAME} will be shut down in ${time}.
Please logoff as soon as possible.
====="
```

here文档要求其中的输入数据，尤其是作为结束标志的字符串分界符“**LimitString**”必须位于单独另起一行的起始位置。为了使脚本的可读性更强，编程人员喜欢在数据行之前增加制表符，使得脚本错落有致。为了使**here**文档能够正确地读取数据，可在第一个字符串分界符之前增加减号“-”标志（如**<<-LimitString**），使之在读取数据时能够删除数据行前的制表符（但“-”标志并不影响空格，也不影响文本行中间的制表符）。例如：

```
$ cat heredoc
#!/bin/sh
cat <<-ENDOFMESSAGE
    I live in you, you live in me;
    We are two gardens haunted by each other.
    Sometimes I cannot find you there,
    There is only the swing creaking, that you have just left,
    Or your favourite book beside the sundial.
ENDOFMESSAGE
exit 0
$ heredoc
I live in you, you live in me;
We are two gardens haunted by each other.
Sometimes I cannot find you there,
There is only the swing creaking, that you have just left,
Or your favourite book beside the sundial.
$
```

here文档也支持变量和命令替换，因而能够把不同的参数传递到**here**文档的代码体中，相应地改变**here**文档的输出。下面是一个在**here**文档中使用变量替换的例子。

```
$ cat upload.sh
#!/bin/bash
if [ -z "$1" ]
then
```

```

        echo "Usage: `basename $0` Filename-to-upload"
        exit 1
    fi
    if [ ! -f "$1" ]
    then
        echo "Specified file is not exist."
        exit 2
    fi
    fname=`basename $1`                # 删除文件名的目录部分
    host="169.254.78.100"
    dir="/export/home/gqxing/docs"

    ftp -inv $host <<End-Of-Session    # "-n" 选项禁止执行自动注册过程
    user gqxing 123456
    put $fname $dir/$fname
    bye
    End-Of-Session

    exit 0
$ upload.sh design
Connected to 169.254.78.100.
220 (vsFTPd 2.0.6)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
local: design remote: /export/home/gqxing/docs/design
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 File receive OK.
308522 bytes sent in 0.01 secs (57641.3 kB/s)
221 Goodbye.
$

```

为了禁止在here文档中使用变量替换或命令替换，可以在第一个字符串分界符前后增加单引号或双引号，或者在第一个字符串分隔符前增加转义字符。

下面就是一个禁止here文档执行变量替换的例子。

```

$ cat greeting2
#!/bin/sh
name="${1:-Zhang}"
from="the author of this script"
cat <<'Endofmessage'          # 使用 "cat <<"Endofmessage"或使用cat <<\Endofmessage" ,
                                # 三者效果一样

Hello, $name.
Greetings to you, $name, from $from.
Endofmessage
exit 0
$ greeting2 wang

Hello, $name.
Greetings to you, $name, from $from.
$

```

禁止变量替换使here文档能够输出纯文字的文本，生成Shell脚本或其他程序代码（如C程序等）。例如，下面的代码用于生成另一个Shell脚本：

```
$ cat gen.sh
#!/bin/sh
OUTFILE=generated.sh          # 生成的Shell脚本的文件名
(
cat <<EOF
#!/bin/sh
echo "This is a generated Shell script."
echo "and will be named: $OUTFILE"
exit 0
EOF
) > $OUTFILE

if [ -f "$OUTFILE" ]
then
    chmod 755 $OUTFILE          # 使生成的Shell脚本能够执行
else
    echo "Problem in creating file: \"$OUTFILE\""
fi
exit 0
$ gen.sh
$ cat generated.sh
#!/bin/sh
echo "This is a generated Shell script."
echo "and will be named: generated.sh"
exit 0
$
```

利用here文档和“:”命令，还可以注解掉Shell脚本中的代码，或增加自包含的文档数据。这种代码注解技术可用于调试Shell脚本。编程人员只需增加两行标志代码，即可临时注解掉大块的Shell代码，从而避免了在每行代码前增加一个注释字符“#”，调试结束后再逐行地一一删除。其代码如下：

```
$ cat commentblock.sh
#!/bin/bash
: << COMMENTBLOCK
.....          # Shell代码或文档
COMMENTBLOCK

: << DEBUGXXX
for file in *
do
    cat "$file"
done
DEBUGXXX
exit 0
$
```

注意：here文档中的第二个字符串分界符必须占用单独一行，并位于行首，前后不能包括任何空白字符（包括空格和制表符等）。否则将会妨碍Shell的识别，导致无法预料的

后果。

7.11 Shell函数

同其他编程语言一样，Korn Shell和Bash等Shell也支持函数，尽管实现的功能有一定的限度。一个函数是一个子程序，其中利用一组代码块，实现特定的处理功能。与Shell脚本相比，函数经常直接存储在内存中，因而执行速度要快于Shell脚本。

无论何处，只要存在一组需要重复执行的处理动作，或针对不同的参数，能够获取相应的返回值，都可以考虑使用函数。函数的语法格式如下：

```
function_name( ) {  
    command-list  
}
```

或

```
function function_name {  
    command-list  
}
```

第一种函数形式是C程序员比较熟悉的。如同C语言一样，函数的左括号也可以出现在第二行上。注意，如同下列函数定义形式所示，如果整个函数定义与其中的命令位于同一行上，左花括号之后与右花括号之前必须各存在一个空格。同时，每个命令之后除附加分号之外，至少还必须保有一个空格。

```
function_name( ) { cmd1; cmd2; .....; cmdn }
```

在定义函数时，函数名不能与现有的变量同名。如果函数与变量同名，Shell将会给出一个错误信息。此外，函数名与左圆括号之间必须连在一起，中间不能有空格。这个圆括号相当于告诉Shell，这是一个函数定义。

例如，为了在一个Shell脚本中重复执行两个变量值的交换，我们可以定义下列函数：

```
exchange()                # 交换两个变量的值  
{  
    temp=${color1}  
    color1=${color2}  
    color2=${temp}  
    return  
}
```

注意：在定义函数时，不能在函数体内使用exit命令，因为函数的执行与当前的Shell脚本同属一个进程。否则的话，当函数执行到exit命令时，将会终止整个Shell脚本的继续执行。

在函数定义中，表示整个函数执行结束的替代方法是利用Shell的内部命令return实现的。也就是说，函数中return语句的功能同Shell脚本中的exit命令一样，其本意是终止函数的执行。在Shell脚本中，只有在函数定义中才能使用return命令，但也并非必需。实际上，仅当期望函数返回一个数值时，才真正需要使用return语句。如果函数中未用到return命令，则函数运行结束时的返回值就是函数体中执行的最后一条命令的出口状态。

同exit语句一样，return语句也可以带一个整数参数，以便在函数执行结束时返回给定的数

值，同时将这个返回值赋予“\$?”变量。编程人员可以使用`return`语句显式地指定返回值。否则，函数的返回值就是函数中执行的最后一条命令的出口状态（如果命令执行成功则返回0，否则返回一个非0值的错误代码）。`return`语句的语法格式如下：

```
return [n]
```

在Shell脚本中，可以通过“\$?”变量引用函数的返回值。这一点与C等语言中的函数类似，但与C等语言不同的是，在调用Shell函数时，只需直接写出函数的名字，然后给出必要的参数，而无须使用圆括号。当遇到一个函数名时，Shell首先会检查给定的名字是否为一个函数，如果不是，再利用PATH环境变量按命令检索。

如同常规的函数调用或Shell脚本的运行方式一样，调用Shell函数时也可以提供参数。而函数可以处理传递给自己的参数，最终把处理结果返回Shell脚本，以便在脚本中做进一步的处理。例如：

```
function_name $arg1 $arg2 ..... $argN
```

下面是一个求取指定整数范围内所有素数的例子。对于任何给定的整数，我们首先排除偶数，仅对奇数进行处理。通过一个while循环，主程序把3、5，直至小于或等于给定整数的奇数依次传递给prime函数，prime函数则以给定的整数参数作为被除数，依次除以3、5，直至整数参数的平方根（取整）为止。如果中间出现余数为零的情况则返回0（说明给定的整数参数不是素数），否则返回1（说明给定的整数参数为素数）。

```
$ cat prime
#!/bin/ksh
prime()
{
    typeset -i num lim rem
    num=3
    lim=`echo $1 | awk '{print sqrt($1)}'`
    while [ $num -le $lim ]
    do
        rem=`expr $1 % $num`
        if [ $rem -eq 0 ]
        then
            return 0
        fi
        num=`expr $num + 2`
    done
    return 1
}
typeset -i number
while true
do
    echo "\nEnter a number: \c"
    read number
    if [ $number -lt 2 ]
    then
        exit 1
    fi
    echo "The prime number list within $number"
```

```

    echo "2 \c"
    if [ $number -eq 2 ]
    then
        echo
        exit 2
    fi

    i=3
    while [ $i -le $number ]
    do
        prime $i
        ret=$?
        if [ $ret -eq 1 ]
        then
            echo "$i \c"
        fi
        i=`expr $i + 2`
    done
done
exit 0
$ prime

Enter a number: 200
The prime number list within 200
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163
167 173 179 181 191 193 197 199
Enter a number: <Enter>
$

```

return 语句的返回值仅限于0~255的整数范围。为了返回较大的数值，可以使用**echo**等命令语句，然后采用命令替换的形式获取函数的返回值。例如，为了求取两个变量中的最大值，我们可以定义下列函数：

```

$ cat max.sh
#!/bin/sh
if [ $# -ne 2 -a $# -lt 2 ]
then
    echo "Usage: `basename $0` number#1 number#2"
    exit 1
fi
max() {
    if [ $1 -gt $2 ]
    then
        echo $1
    else
        echo $2
    fi
}
echo "The greatest number of $1 and $2 = `max $1 $2`"
$ max.sh 7856 9980
The greatest number of 7856 and 9980 = 9980
$

```


不管在Shell脚本中，还是在交互式Shell环境中，在调用函数之前，首先必须对函数进行定义。定义之后，函数将位于用户的Shell运行环境中。使用不带参数的set命令，可以看到函数的定义。使用unset命令，或终止当前运行的Shell，即可取消当前的函数定义，因而也就不能再执行了，示例代码如下：

```
$ exchange() {
> temp=${color1}
> color1=${color2}
> color2=${temp}
> return
> }
$ set
.....
exchange ()
{
    temp=${color1};
    color1=${color2};
    color2=${temp};
    return
}
.....
$
```

函数也可以在Shell脚本中定义，在脚本中执行，最后随脚本的终止而取消。如果在Shell脚本中定义函数，函数定义可以出现在允许使用命令语句的任何位置。但是，函数的定义必须位于调用函数的语句之前。对于需要经常使用的函数代码，最好在用户自己的.profile文件中定义，这将保证一旦注册，只要需要，随时都可以直接调用函数。

在下面的例子中，我们定义了一个函数，专门负责处理常用的提示与用户响应信息，以便能够在必要时供Shell脚本随时调用：

```
$ cat report.sh
#!/bin/sh
prompt()
{
    while
        echo "Please answer yes or no: \c"
        read yes_no
    do
        case "${yes_no}" in
            y* | Y*) yes_no="y"
                       break
                       ;;
            n* | N*) yes_no="n"
                       break
                       ;;
            *)
                echo
                continue
                ;;
        esac
    done
```

```

}
echo "Would you like headings?"
prompt
if test "${yes_no}" = "y"
then
    echo "Headings selected."
    echo "Start to process ... done."
fi
echo "Would you like a summary only?"
prompt
if test "${yes_no}" = "y"
then
    echo "Summary only selected."
    echo "Start to process ... done."
fi
exit 0
$ report.sh
Would you like headings?
Please answer yes or no: y
Headings selected.
Start to process ... done.
Would you like a summary only?
Please answer yes or no: n
$

```

无论何时调用函数，都会重新设置当前Shell进程的位置参数，使调用函数时提供的参数成为新的位置参数。也就是说，Shell将会使用调用函数时提供的参数重新为位置参数赋值。如果调用函数时未提供参数，位置参数（中的值）将被清除。

函数按位置引用传递给自己的参数\$1、\$2、……。因此，也可以使用shift命令处理传递给函数的参数。有关位置参数与shift命令方面的内容，可以参考前面介绍的例子，这里不再细叙。

与其他编程语言不同，Shell函数通常只接收数值参数。如果把变量名作为参数传递给函数，函数将会把变量名作为字符串常量处理。也就是说，函数将按字符串文字常量解释变量参数，示例如下：

```

$ cat varref
#!/bin/sh
echo_var ()
{
    echo "$1"
}
Message=Hello
Hello="Good Morning"
echo_var Message
echo_var Hello
echo_var "${Message}"
echo_var "${Hello}"
exit 0
$ varref
message
Hello

```

```
Hello
Good Morning
$
```

事实上，函数也是一个代码块，这意味着也可以重定向函数的标准输入、标准输出和标准错误输出。例如，在下面的函数定义例子中，“< \$file”表示把函数的标准输入重定向到\$file变量指定的文件中。当调用function_name函数时，函数中的read语句不再读取来自键盘的输入数据，而是直接读取自\$file变量定义的文件：

```
#!/bin/sh
function_name()
{
    .....
    read lines
    .....
} < $file      # 重定向函数的标准输入
```

在下面的例子中，“function_name2 < \$file”语句表示仅在调用普通的function_name2函数时，再把标准输入重定向到\$file变量定义的文件中。因此，函数中的read语句读取的数据仍然取自\$file变量定义的文件，而非键盘输入：

```
#!/bin/sh
function_name2()
{
    .....
    read lines
    .....
}
function_name2 < $file
```

下面我们再通过一个冒泡排序的例子说明怎样调用函数，结束“Shell函数”一节的介绍。另外，这个例子涉及到一些数组方面的概念，读者可以参考后续“Shell数组”一节的介绍。

冒泡排序算法的基本思想是：对于n个需要排序的元素，执行n-1次循环。每次循环均从第1个元素开始，依次比较相邻的两个元素。如果第i个元素大于第i+1个元素，则交换两个元素的位置，直至比较到最后第n-j+1个元素（其中j为循环次数）结束。第1次循环结束后，最大的元素将会交换到最后一个数组元素位置。第2次循环结束后，仅次于最大元素的元素将会交换到最后第2个数组元素位置。依次类推，直至循环结束。在每次循环中，较大的元素依次后移，较小的元素依次前移。当最后一次循环结束时，整个数组元素将完成从小到大的排序。

```
$ cat bubble.sh
#!/bin/bash
exchange()                                # 交换两个数组元素
{
    temp=${colors[$1]}
    colors[$1]=${colors[$2]}
    colors[$2]=$temp
    return
}
colors=(red green blue yellow black white brown rose grey)
echo "0: ${colors[*]}"                    # 输出排序前的全部数组元素
```

```

elements=${#colors[@]}                                # 求出数组元素的个数
let "count = $elements - 1"
pass=1
while [ "$count" -gt 0 ]
do
    index=0                                            # 每次循环均从第1个数组元素开始
    while [ "$index" -lt "$count" ]
    do
        if [[ ${colors[$index]} > ${colors[`expr $index + 1`]} ]]
        then
            exchange $index `expr $index + 1`
        fi
        let "index += 1"
    done
    let "count -= 1"                                    # 一次冒泡排序循环结束
    echo "$pass: ${colors[@]}"                          # 显示每次循环后的中间排序结果
    let "pass += 1"
done                                                    # 整个冒泡排序循环结束
exit 0
$ bubble.sh
0: red green blue yellow black white brown rose grey
1: green blue red black white brown rose grey yellow
2: blue green black red brown rose grey white yellow
3: blue black green brown red grey rose white yellow
4: black blue brown green grey red rose white yellow
5: black blue brown green grey red rose white yellow
6: black blue brown green grey red rose white yellow
7: black blue brown green grey red rose white yellow
8: black blue brown green grey red rose white yellow
$

```

7.12 逻辑与和逻辑或并列结构

命令的逻辑与（&&）和逻辑或（||）并列结构提供了一种依次执行一系列命令的手段，能够有效地替代复杂的、嵌套的if-then语句结构。

7.12.1 逻辑与命令并列结构

逻辑与命令并列结构的语法格式如下：

```
command-1 && command-2 && ..... && command-n
```

命令的逻辑与并列结构表示从第一个命令开始，依次执行每一个命令。如果当前命令的返回值（出口状态）为0，则继续执行下一个命令。如果中间某个命令的返回值大于0，则整个并列命令链的执行结束。也就是说，第一个返回非0值的命令即为逻辑与并列结构中最后执行的一个命令。

下面是取自Solaris系统/etc/init.d/syssetup启动脚本中的一行代码。这一行代码表示，如果先前的运行级（即\$_INIT_PREV_LEVEL变量的值）既不等于S也不等于1，则终止syssetup脚本的执行：

```
[ $_INIT_PREV_LEVEL != S -a $_INIT_PREV_LEVEL != 1 ] && exit 0
```

如果把逻辑与命令并列结构改为if-then结构语句，上述命令相当于：

```
if [ $_INIT_PREV_LEVEL != S -a $_INIT_PREV_LEVEL != 1 ]
then
    exit 0
fi
```

下面的例子取自Solaris系统中的一个引导块代码复制脚本。当使用dd命令复制一个系统磁盘时，并不能保证磁盘是可引导的系统盘，此外还必须使用insatllboot脚本把适当的引导块代码也复制到磁盘的第1至第15个磁盘块中。这个脚本使用了大量的逻辑与命令并列结构，用于测试命令行参数提供的引导块代码是否存在，磁盘设备是否是字符设备以及是否可写，使用dd命令复制引导块代码是否成功等：

```
$ cat /usr/sbin/installboot
#!/bin/sh
.....
away() {
    echo $2 1>&2
    exit $1
}

Error="Error: `basename $0` is obsolete. Use installgrub(1M)"
Usage="Usage: `basename $0` --force_realmode pboot bootblk raw-device"

test $# -ne 4 && away 1 "$Error"
test $1 != "--force_realmode" && away 1 "$Error"
shift 1

PBOOT=$1
BOOTBLK=$2
DEVICE=$3

test ! -f $PBOOT && away 1 "$PBOOT: File not found"
test ! -f $BOOTBLK && away 1 "$BOOTBLK: File not found"
test ! -c $DEVICE && away 1 "$DEVICE: Not a character device"
test ! -w $DEVICE && away 1 "$DEVICE: Not writeable"

# pboot at block 0, label at blocks 1 and 2, bootblk from block 3 on
stderr=`dd if=$PBOOT of=$DEVICE bs=1b count=1 conv=sync 2>&1`
err=$? ; test $err -ne 0 && away $err "$stderr"
stderr=`dd if=$BOOTBLK of=$DEVICE bs=1b oseek=3 conv=sync 2>&1`
err=$? ; test $err -ne 0 && away $err "$stderr"
exit 0
$
```

7.12.2 逻辑或命令并列结构

逻辑或命令并列结构的语法格式如下：

```
command-1 || command-2 || ..... || command-n
```

与命令的逻辑与并列结构相反，逻辑或命令并列结构意味着从第一个命令开始，依次执行每一个命令。如果当前命令的返回值（出口状态）大于0，则继续执行下一个命令。如果中间某个命令的返回值为0，则整个并列命令链的执行结束。也就是说，第一个返回0的命令即为逻

辑或并列结构中最后执行的一个命令。

下面是取自Solaris系统/sbin/mountall脚本中的一行代码。这一代码首先检查\$mntlist变量是否为空。如果\$mntlist变量是否为空，则结束本行代码，从下一行开始执行后续代码。如果\$mntlist变量不为空，则执行/sbin/mount -a \$mntlist命令。

```
[ -z "$mntlist" ] || /sbin/mount -a $mntlist
```

如果把逻辑或命令并列结构改为if-then结构语句，上述命令相当于：

```
if [ ! -z "$mntlist" ]
then
    /sbin/mount -a $mntlist
fi
```

注意：在逻辑与和逻辑或命令并列结构中，整个结构语句的出口状态是其中最后执行的一条命令的出口状态。

灵活地使用逻辑与和逻辑或命令并列结构及其组合，可以简化Shell编程，但有时也会增加理解的困难，因而需要做额外的调试。

7.13 Shell数组

Korn Shell和Bash等Shell还支持一维数组，且数组的大小没有限制。数组采用整数索引，第一个数组元素从0开始。在Korn Shell中，为了引入一个数组，可以使用下列set命令，在声明一个数组的同时，对数组进行初始化：

```
$ set -A array_name element1 element2 ..... elementN
```

例如，为了定义一个表示周日的数组，可以使用下列命令：

```
$ set -A weekday Sunday Monday Tuesday Wednesday Thursday Friday Saturday
$
```

在Bash Shell中，为了引入一个数组，可以采用下列语法格式：

```
[declare -a] array_name=( element1 element2 ..... elementN )
```

同样，采用上述语法格式定义数组，也能够同时在声明数组的同时对数组进行初始化。如果在定义数组时再增加一个“declare -a”前缀，则由declare预先声明的数组能够加速数组的处理，提高Shell脚本的运行性能。

因此，可以使用下列语句，在Bash Shell中定义并初始化一个表示周日的数组：

```
$ weekday=( Sunday Monday Tuesday Wednesday Thursday Friday Saturday )
$
```

此外，同普通变量一样，每个数组元素也可以采用下列语法格式单独赋值，逐个定义每一个数组元素，从而声明并初始化一个数组：

```
array_name[index]=value
```

因此，我们也可以采用下列赋值语句，定义并初始化一个表示周日的weekday数组：

```
weekday[0]=Sunday
weekday[1]=Monday
```



```
.....
weekday[6]=Saturday
```

由此可见，采用数组元素单独赋值的方式定义任何一个数组元素，即可构建一个数组。但是，如果仅对部分数组元素进行赋值，只能构建一个部分初始化的数组。例如：

```
$ array=( [0]="first " [1]="second " [3]="fourth" )
$
```

按照上述方式定义并初始化数组之后，其结果如下：

```
$ echo ${array[0]}
first
$ echo ${array[1]}
second
$ echo ${array[2]}
$
$ echo ${array[3]}
fourth
$
```

在具体应用过程中，可以像操作普通变量一样操作数组元素，只是引用形式稍微不同而已。而且，许多标准的字符串操作也适应于数组元素。同样，为了引用数组元素的内容，也可以采用`${array_name[n]}`等形式实现变量替换。

例如，在定义了`weekday`数组之后，即可使用数组变量名及其下标引用任何一个数组元素：

```
$ echo ${weekday[0]}
Sunday
$ echo ${weekday[1]}
Monday
$
```

与`$@`或`$*`变量值表示所有的位置参数类似，`${array_name[@]}`或`${array_name[*]}`变量值表示数组的所有元素。示例如下：

```
$ echo ${weekday[@]}
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
$ echo ${weekday[*]}
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
$
```

同样，与`$#`变量值表示位置参数的数量类似，`${#array_name[@]}`或`${#array_name[*]}`变量值表示数组`array_name`的长度，也即所有数组元素的数量。下列例子说明数组`weekday`具有7个数组元素：

```
$ echo ${#weekday[@]}
7
$ echo ${#weekday[*]}
7
$
```

注意：`${#array_name}`变量值仅表示第一个数组元素`array_name[0]`的长度。示例如下：

```
$ echo ${#weekday}
6
$
```

为了求取其他数组元素的长度（字符个数），可以采用`${#array_name[index]}`的变量替换形式，指定具体的数组元素。示例如下：

```
$ echo ${#weekday[0]}
6
$ echo ${#weekday[6]}
8
$
```

对于数组而言，某些Shell命令具有特殊的意义。例如，使用`unset`命令能够删除任何数组元素，甚至整个数组：

```
$ unset weekday[1]           # 相当于执行 “weekday[1]=” 语句
$ echo ${weekday[1]}
$ unset weekday
$ echo ${weekday[@]}
$
```

下面的例子说明了怎样利用数组和一定的算法，计算任何一天是星期几：

```
$ cat whatday
#!/bin/bash
typeset -i year month day mm=1 sum week yy
typeset -i leap=0
set -A months 0 31 28 31 30 31 30 31 31 30 31 30 31
set -A weekday Sunday Monday Tuesday Wednesday Thursday Friday Saturday
month=${1:-`date +%m`}
day=${2:-`date +%d`}
year=${3:-`date +%Y`}

let yy=year-1
let sum=yy*365+yy/400+yy/4-yy/100

if [ `expr $year % 4` -ne 0 ]      # 如果年份不能被4除尽，则不是闰年
then
    leap=0
elif [ `expr $year % 100` -ne 0 -o `expr $year % 400` -eq 0 ]
then
    leap=1                        # 如果年份能够被4除尽，但不能被100除尽，则是闰年
    # 如果年份能够同时被4、100和400除尽，则是闰年
fi                                # 如果年份既能被除尽4，又能被100除尽，但不能被
                                # 400除尽，则不是闰年

while [ $mm -lt $month ]
do
    sum=`expr $sum + ${months[$mm]} `
    mm=`expr $mm + 1`
done
sum=`expr $sum + $day + $leap`
week=`expr $sum % 7`
echo "$month $day, $year is ${weekday[$week]}"
exit 0

$ whatday
7 4, 2009 is Saturday
$ whatday 12 22 2009
```

12 22, 2009 is Tuesday

\$

灵活地组合使用数组初始化语句“`array=(element1 element2 elementN)`”与命令替换，能够把一个文本文件的内容加载到数组中。因此，定义并初始化数组的第二种方法是命令替换法。采用命令替换法初始化数组，可以构建一个具有多个元素的数组，数组的大小依据命令输出的多少而定。

在采用命令替换形式初始化数组时，Shell使用空格作为数组元素的分隔符，把每个字符串分配到一个数组元素中，最终构建成一个字符串数组。在遇到制表符和换行符时，Shell首先把它们转换成空格，并用做元素分隔符。

例如，假定dict文件包含下列四行数据，采用命令替换形式的数组定义语句“`array=(`cat dict`)`”可以声明并初始化一个具有4个元素的数组：

```
$ cat dict
aa
bb
cc
dd
$ array=( `cat dict` )
$ echo ${array[@]}
aa bb cc dd
$ echo ${#array[*]}
4
$
```

下列Shell脚本能够读取以命令行参数形式提供的文本文件，并以空格、制表符和换行符作为分隔符，把每个连续的字符串看做一个单词，使之构成一个数组元素。

```
$ cat array.sh
#!/bin/bash
array=( $(cat "$1") )          # 把命令行参数文件的内容存储到数组中
total=${#array[@]}
echo $total
element=0
while [ $element -lt $total ]
do
    echo ${array[$element]}
    element=`expr $element + 1`
done
exit 0
$
```

如果使用Shell脚本本身作为文件参数，运行上述Shell脚本的输出结果如下：

```
$ array.sh array.sh
25
#!/usr/bin/bash
array=(
$(cat
"$1")
)
```

```

total=${#array[@]}
echo
$total
element=0
while
[
$element
-lt
$total
]
do
echo
${array[$element]}
element=`expr
$element
+
1`
done
exit
0
$

```

显然，如果用于处理普通数据文件，上述输出结果是可以接受的。但如果用于处理Shell脚本文件，上述的输出内容并非是我们想要的结果。这里的问题是，如果把空格或制表符作为分隔符将会使脚本的语句支离破碎而变形。如果稍加改造，在构成数组元素之前，先把空格和制表符替换成文本文件中不会出现的其他特殊字符，然后在进一步处理之前，再还原成原来的字符。示例如下：

```

$ cat array2.sh
#!/bin/bash
array=( $(cat "$1" | tr ' ' '\001' | tr '\t' '\002') )
total=${#array[@]}
echo $total
element=0
while [ $element -lt $total ]
do
    echo ${array[$element]} | tr '\001' ' ' | tr '\002' '\t'
    element=`expr $element + 1`
done
exit 0
$ array2.sh array2.sh
11
#!/usr/bin/bash
array=( $(cat "$1" | tr ' ' '\001' | tr '\t' '\002') )
total=${#array[@]}
echo $total
element=0
while [ $element -lt $total ]
do
    echo ${array[$element]} | tr '\001' ' ' | tr '\002' '\t'
    element=`expr $element + 1`
done

```

```
done
exit 0
$
```

定义并初始化数组的第三种方法是采用Shell元字符与文件名生成方法。采用元字符与文件名生成形式定义数组，可以构建并初始化一个具有多个文件名元素的数组，数组的大小依据文件的数量而定。同使用命令替换形式初始化数组类似，Shell采用空格作为数组元素的分隔符，把每个文件名分配到一个数组元素中，最终构建成一个文件名数组。

例如，为了对当前目录下的C程序进行操作，我们可以使用下列语句生成一个文件名数组：

```
$ filelist=(*.c)
$ echo ${filelist[@]}
atmcom.c atmmon.c atmstat.c handler.c listener.c
$ echo ${#filelist[@]}
5
$
```

此外，还可以把普通变量看做数组的一个特例，即只有一个元素的特殊数组。因此，即使并未明显地把变量声明为数组，Korn Shell和Bash等Shell也允许以数组或数组元素的处理方式处理普通变量。示例如下：

```
$ string=abcABC123ABCabc
$ echo ${string[@]}
abcABC123ABCabc
$ echo ${string[*]}
abcABC123ABCabc
$ echo ${string[0]}
abcABC123ABCabc
$ echo ${string[1]}      # 一个普通变量只能看做第一个数组元素（下标为0）
$ echo ${#string[@]}     # 数组中只有一个元素，也即字符串本身
1
$
```

数组变量具有自己的语法规则，可以利用赋值语句实现整个数组的复制。也可以在数组的原有基础上增加新的元素。例如，下列两个语句均可实现整个数组的复制（其中的第一个语句实为包含变量替换的数组赋值语句）：

```
$ array2=( "${array1[@]}" )
$ array2="${array1[@]}"
```

而且，还可以使用下列语句，把一个数组和一个单独的元素复制到另外一个数组中：

```
$ array=( "${array[@]}" "new element" )
```

Shell仅支持一维数组，但稍加变通即可仿真多维数组。限于篇幅，这里不再赘述。

7.14 信号的捕捉与处理

信号是一种能够影响进程运行状态的外部事件，是由用户终端（如按下Ctrl-C键）、操作系统内核或其他进程（如kill命令）发送给当前或指定进程的消息，用于通知进程某种事件已经发生。进程可根据预定的方案，采取一定的处理措施或终止进程的执行。如果事先没有捕捉

信号，默认的处理动作是停止进程的执行。

在Shell脚本中，**trap**命令用于指定需要捕捉的信号以及应采取的相应处理动作。当接收到某个指定的信号时，**Shell**将会立即触发相应的处理过程。**trap**命令的语法格式简写如下：

```
trap ["command-list"] [signal ...]
```

或

```
trap ['command-list'] [signal ...]
```

其中，**command-list**是无论何时收到指定的信号时都会立即执行的命令或一组命令。一旦命令执行结束，程序的控制逻辑将会恢复到因收到信号而中断的位置开始继续运行，除非命令中包含**exit**语句。如果在收到指定的信号时应当结束脚本的运行，可以使用**exit**语句作为指定命令的一部分（如果存在，**exit**语句通常应为命令组中的最后一个命令）。在通常情况下，指定的命令并非单个命令，而是一组命令，故前后应加单或双引号，命令中间以分号“;”隔开。

signal表示准备捕捉的信号。信号可以是一个标准的信号名，也可以是一个数字。针对Shell编程而言，比较重要的、能够捕捉的常用信号只有0、1、2、3、15和20等。如果捕捉的信号是0（**EXIT**），则仅当脚本运行结束时才能执行相应的处理动作。这种例行处理方式经常用于清除Shell脚本运行过程中创建的临时文件。另外还有三个特殊的信号**DEBUG**、**ERR**和**RETURN**，主要用于调试和控制Shell脚本的执行，详见下一节的讨论。表7-1给出了这些信号的简单说明（至于系统定义的其他信号，详见第8章“进程管理”）。

表7-1 Shell脚本能够捕捉的信号

信号	信号名	简单说明
0	EXIT	进程结束信号。在Shell脚本的情况下，只要执行 exit 语句，终止Shell脚本的执行时，都可产生此信号。在Shell脚本运行正常结束时，即使没有明显地执行 exit 语句，也可以产生此信号
1	SIGHUP或HUP	挂断。终端通信连接断开，或控制进程终止时产生的信号。通常，一旦捕捉到此信号，Shell脚本会立即停止运行，除非使用 nohup 命令
2	SIGINT或INT	中断。按下中断键（也即Ctrl-C键）时产生的信号
3	SIGQUIT或QUIT	退出。按下Quit键（也即按下Ctrl-\或Ctrl-Shift- 组合键）时产生的信号
15	SIGTERM或TERM	终止信号。这是kill命令产生的默认终止信号
24	SIGTSTP或TSTP	键盘停止信号。在交互方式下，通过按Ctrl-Z键停止当前进程时产生的信号。主要用于作业控制
	DEBUG	在执行Shell脚本中的每个命令之前，包括 for 、 case 和 select 命令语句，以及Shell函数中的第一个命令，都要运行 trap 语句中指定的命令
	ERR	Shell脚本中的任何命令，一旦运行结束时返回非0值的出口状态，都要运行 trap 语句中指定的命令。但 while 、 until 或 if 等结构语句中的测试语句除外
	RETURN	每当调用一个Shell函数，或利用“.”命令执行一个Shell脚本结束之后，都要运行 trap 语句中指定的命令

在UNIX系统中，每个信号都有一个名字和数字。在编写Shell脚本时，建议使用名字指定捕捉的信号，这将使Shell脚本更具有可读性，也具有可移植性。

注意：在Shell脚本中，有4个信号不应捕捉：其中，信号9和23是不能捕捉的，而信号18和19则不应捕捉，这些信号的定义详见第8章“进程管理”。

利用trap命令，程序员可以采取下列三种处理措施之一：

- 捕捉指定的信号，然后执行必要的命令或处理动作；
- 忽略收到的信号；
- 清除先前的信号捕捉设置。

如果trap语句中指定的信号为0或EXIT，则当执行exit命令语句，或Shell脚本的执行正常结束时，立即执行trap语句中指定的处理动作。例如，下面的例子说明了怎样在Shell脚本运行结束时捕捉EXIT信号，从而显示脚本执行过程中设定的变量内容：

```
$ cat test.sh
#!/bin/sh
trap 'echo "Variable setting ---\na = $a\nb = $b"' EXIT
echo "This line will print before the \"trap\" statement."
echo "even though the \"trap\" statement occurs first."
echo
a=""
b=36
exit 0
$ test.sh
This line will print before the "trap" statement.
even though the "trap" statement occurs first.

Variable setting ---
a =
b = 36
$
```

注意：上述Shell脚本的“exit 0”语句的存在与否并无太大的区别，因为在Shell脚本执行到最后一条命令语句时都会立即结束。如果最后一条命令语句的出口状态为0，其效果等同于执行“exit 0”语句。Shell脚本终止运行前执行的最后一条命令语句的出口状态即为整个Shell脚本的出口状态。

如果trap语句中指定的捕捉信号为1或HUP，且Shell脚本在运行过程中断开终端连接时，Shell会立即执行定义的处理动作。如果trap语句中指定的信号为2或INT，当用户在Shell脚本执行过程中按下Ctrl-C键时，Shell将会立即执行定义的处理动作。如果trap语句中指定的信号为3或QUIT，当用户在Shell脚本执行过程中按下Ctrl-\键时，Shell将会立即执行定义的处理动作。

在下列例子中，如果在Shell脚本的运行过程中收到信号1、2或3，Shell将会执行trap命令语句中指定的命令序列，显示“Interrupted routine”，删除临时文件tmp\$\$，终止脚本的执行，最终返回一个数值1作为Shell脚本的出口状态：

```
trap 'echo Interrupted routine; rm -f tmp$$; exit 1' 1 2 3
```

或

```
trap 'echo Interrupted routine; rm -f tmp$$; exit 1' HUP INT QUIT
```

有时，我们不希望任何人中途打断Shell脚本的运行。因此，为了防止任何人使用Ctrl-C键打断Shell脚本的正常运行，可以使用下列形式的trap语句，屏蔽中断信号：

```
trap '' 2
```

或

```
trap 'echo "Ctrl-C disabled."' 2
```

如果**trap**语句的命令部分中没有指定任何处理动作，或明显地指定一个**null**值，则表示忽略指定的信号。

```
trap '' signal
```

或

```
trap ':' signal
```

上述两个**trap**命令存在细微的差别：第一个**trap**语句表示完全忽略指定的信号，第二个**trap**语句则意味着在收到指定的信号时不做任何处理。这两者在同一**Shell**脚本中的差别并不明显，但在父进程与子进程的处理方面则大不相同。

在第一种情况下，如果父进程忽略收到的指定信号，则子进程也将忽略相应的信号。例如，下面的例子表示父进程将会忽略捕捉到的中断信号（按**Ctrl-C**键），同时告诉子进程也忽略这一信号。

```
trap "" 2
```

在第二种情况下，如果父进程捕捉到指定的信号，除了自己只是执行“:”语句而不做其他任何处理动作之外，还将告诉子进程清除其相应信号的设置，包括继承自父进程的信号设置（在收到信号2时执行“:”语句），同时把子进程未做“忽略”处理的所有信号恢复为默认的处理动作。在下列例子中，当捕捉到中断信号（按**Ctrl-C**键）时，父进程仅执行一个简单的“:”语句，但不会改变子进程对相应信号的默认处理动作。

```
trap ":" 2
```

在使用**trap**语句时，如果仅指定了准备捕捉的信号而未指定相应的处理动作，**Shell**将会删除先前为相应信号定义的处理动作。这意味着从此时开始，如果捕捉到这些信号，**Shell**将会按默认的处理动作执行。在下面的**trap**语句例子中，**Shell**将会删除为信号1、2和3定义的处理动作。也就是说，如果收到信号1、2或3，**Shell**将会执行默认的处理动作——终止**Shell**脚本的执行。

```
trap 1 2 3
```

通常，**trap**语句应是**Shell**脚本中的第一个语句。但程序员也可根据具体的编程要求做适当的调整。不管**trap**语句位于何处，在捕捉到指定的信号之前，**Shell**只是把**trap**语句读入内存，实际上并不执行。只有在收到指定的信号时，才会开始执行**trap**语句中定义的命令。

例如，在一个数据库更新的**Shell**脚本中，数据的输入过程并不重要，即使中断脚本的执行也无关紧要。但在数据库更新时，通常需要屏蔽一切外来的干扰，防止用户的中断信号等打断脚本的执行，以免破坏数据的完整性。因此，程序员可在数据库更新的关键代码处增加**trap**语句，使之屏蔽任何可能的外部信号干扰。在关键代码执行结束之后，再使用不带命令参数的**trap**语句清除之前的信号捕捉设置。示例代码如下：

```
$ cat updatedb
#!/bin/sh
while true
do
```

```

echo "Please enter name, phone number, and email address"
echo "separated by blank character (or enter quit to end):"
read name phone email
if test "${name}" = "quit"
then
    break
fi

trap "" 2 3
# Critical code segment to update database
update.sql $name $phone $email
trap 2 3
done
$

```

Shell脚本中的trap语句通常会读取两次。在开始执行Shell脚本时，Shell将会读取并存储整个trap语句，其间如有变量替换或命令替换，也将随之进行替换处理。当收到指定的信号时，Shell会再次读取并执行trap语句。同样，如果trap语句中仍然存在变量替换或命令替换，还会再次进行替换处理。

假定trap语句中存在变量替换表达式，如果变量是在脚本执行期间赋值的，则此情况下可能会存在一个问题：当第一次读取trap语句，并执行变量替换时，Shell将会采用初始的null值。

为了防止此类问题的发生，可在trap语句中使用单引号界定其中的命令，以避免在第一次读取trap语句时执行变量替换，使之仅当捕捉到指定的信号，需要执行相应的命令时再进行变量替换。

因此，下列两种命令引用形式是不同的：前者执行两次变量替换或命令替换，而后者只执行一次替换。

```

trap ["command-list"] [signal ...]
trap ['command-list'] [signal ...]

```

7.15 其他Shell课题

7.15.1 子Shell

当用户在键盘上输入单个命令时，Shell将以交互方式解释用户提交的命令，然后通过创建一个新的进程，执行用户提交的命令。而在运行Shell脚本时，当前Shell将会创建一个新的Shell进程，以批处理的方式处理用户提交的Shell脚本，解释执行脚本文件中的一系列命令。事实上，每个运行的Shell脚本都是当前Shell的子进程。

Shell脚本本身也可以创建子进程，也即子Shell。这些子Shell采用并行处理的方式，能够同时执行多个处理任务。通常，脚本中使用的每个外部命令都会使Shell创建一个相应的子进程，但内部命令则不然。由于这个原因，内部命令的执行速度明显快于等价的外部命令。

位于圆括号中的命令将以子Shell的方式运行：

```
( command1; command2; command3; ... )
```

子Shell中的变量仅在子Shell代码块中有效，在调用子Shell的Shell中是不可见的，也就是说，子Shell中的变量属于本地变量。参见下面的例子：

```

$ cat subshell.sh
#!/bin/sh
outer_var=Outer
(
inner_var=Inner
echo "From subShell, \"inner_var\" = $inner_var"
echo "From subShell, \"outer\" = $outer_var"
)

if [ -z "$inner_var" ]
then
    echo "inner_var undefined in main code body of Shell"
else
    echo "inner_var defined in main code body of Shell"
fi
echo "From main code body of Shell, \"inner_var\" = $inner_var"
# $inner_var will show as uninitialized because
# variables defined in a subShell are "local variables".
exit 0
$ subshell.sh
From subShell, "inner_var" = Inner
From subShell, "outer" = Outer
inner_var undefined in main code body of Shell
From main code body of Shell, "inner_var" =
$

```

在子Shell中使用cd命令所做的目录变动也不影响父Shell。也就是说，在子Shell改变目录时，仅在子Shell中有效。这一点要特别注意。参见下面的例子：

```

$ cat subshell2
#!/bin/sh
echo "Current directory = `pwd`"
(cd /etc; cat nodename)
echo "Current directory = `pwd`"
exit 0
$ subshell2
Current directory = /export/home/gqxing/script
iscas
Current directory = /export/home/gqxing/script
$

```

7.15.2 Shell脚本的调试

Shell不提供Shell脚本的调试程序，甚至也不提供任何有关的调试命令或结构语句。脚本中的语法错误或明显的拼写错误产生的错误信息在调试脚本时经常无助于调试一个有误的Shell脚本。

下面是一个其中包含语法错误的Shell脚本，执行时将会出现下列错误信息：

```

$ cat test2.sh
#!/bin/sh
a=40

```

```

if [$a -gt 30 ]
then
    echo $a
fi
exit 0
$ test2.sh
test2.sh: [40: not found
$

```

前面曾经讲过，方括号与表达式之间必须至少保留一个空格。而在上述脚本中，“[”和“\$a”连在一起，故Shell把“[40”看做一个单词，也即看做一个“[40”命令，而这样的命令显然是不存在的。

下面是一个漏掉关键字的例子。执行Shell脚本后产生的错误信息如下：

```

$ cat test3.sh
#!/bin/sh
for a in 1 2 3
do
    echo "$a"
# done                                # 不知何故注解掉了关键字done。
exit 0
$ test3.sh
test3.sh: syntax error at line 7: `end of file' unexpected
$ cat test4.sh
#!/bin/sh
for a in 1 2 3                        # 遗漏了关键字do
    echo "$a"
done
exit 0
$ test4.sh
test4.sh: syntax error at line 3: `echo' unexpected
$

```

注意：在输出产生语法错误的行号时，Shell是把注解行也计算在内的。另外，出错信息指出的行号也并不一定就是出错语句所在行的位置，但这一位置却是Shell解释程序最终发现脚本有误的地方。在上述第一个例子中，Shell读到最后仍然没有发现应有的关键字done，故给出的是一个根本就不存在的行号。

当遇到下述情况之一时，Shell脚本将会终止运行：

- 控制结构语句（包括循环结构语句和条件转移语句）中出现语法错误；
- 接收到某种信号，包括外部信号（如按Ctrl-C键、终端关闭或线路断开等）和内部信号；
- Shell内置语句（read和test语句除外）执行失败；
- 特殊替换中出现语法错误（如引号不匹配或不配对）；
- 赋值语句出现故障（如对使用readonly命令定义的只读变量赋值）。

但下列情况不会引起Shell脚本停止运行：

- 试图执行一个不存在的命令（如命令名字拼写错误，或命令文件没有执行许可等）；
- I/O重定向故障（如试图使用“>”重定向符号写入一个已存在的文件等），此时，除了I/O受到影响外，并不影响脚本的执行；

- 命令异常中止（即使一个程序产生了内存映像文件core，Shell仍会继续执行下一个命令）；

- 命令终止运行时返回非0值的出口状态（除非使用trap语句跟踪此种情况）。

根据上述说明及实际应用，Shell脚本的错误情况可以归结为如下类型：

- （1）运行时出现“syntax error”信息，说明脚本中存在诸如语句不完整，语法格式有误，遗漏关键字，引号不匹配或不配对等语法错误；

- （2）可以运行，但最终的处理结果并非预期，这主要是由于逻辑错误造成的；

- （3）可以运行，处理结果也无问题，但存在严重的边界效应。

为了调试Shell脚本，可以采用下列方法及工具：

- （1）在脚本的关键位置加入echo语句，跟踪和显示关键变量的值，借以判断脚本是否存在逻辑问题；

- （2）在关键位置使用tee命令检查进程或数据流；

- （3）设置命令执行过程的跟踪标志（增加“-n”、“-v”或“-x”选项）：

- “sh -n scriptname”命令用于检查Shell脚本的语法错误，实际上并不运行脚本。这一命令形式等价于在脚本中插入“set -n”或“set -o noexec”语句。注意，这种检查并不能找出所有的语法错误。

- “sh -v scriptname”命令的作用是在执行之前显示每一个命令，然后显示命令的运行结果。这一命令形式等价于在脚本中插入“set -v”或“set -o verbose”语句。

- “-n”和“-v”选项可以一起工作。例如，“sh -nv scriptname”命令意味着给出详细的语法检查信息。

- “sh -x scriptname”命令的作用是以简化的方式显示每一条命令语句的运行结果，等价于在Shell脚本中插入“set -x”或“set -o xtrace”语句。

- 在脚本中插入并执行“set -u”或“set -o nounset”语句，使Shell在遇到试图使用未声明的变量时给出错误信息。

- （4）使用trap命令捕捉Shell脚本终止执行时的位置。在Shell脚本中，一旦执行到exit命令，将会生成一个EXIT信号（0），并终止Shell脚本的执行。因此，捕捉EXIT信号，输出运行结束时的变量值通常是一种很有用的方法。

- （5）在trap命令中，DEBUG信号使Shell能够在执行脚本中的每个命令之前立即执行指定的处理动作。这种“trap 'commands' DEBUG”设置能够在调试Shell脚本时跟踪变量的赋值情况，从中找出Shell脚本中的逻辑错误。

- （6）在trap命令中，ERR信号使Shell能够在执行脚本中的每个命令时，跟踪运行过程中出现异常的命令。这种“trap 'commands' ERR”设置有助于找出Shell脚本中出错的命令语句，从而找出Shell脚本出错的原因。

- （7）在trap命令中，RETURN信号使Shell能够在调用脚本中的Shell函数，或利用“.”命令执行其他Shell脚本之后，立即执行指定的处理动作。这种“trap 'commands' RETURN”设置能够跟踪Shell函数或其他脚本的执行情况。

信号捕捉机制对调试Shell脚本是非常有用的。下面的例子就是利用“trap 'commands' DEBUG”设置，在执行每个赋值语句之前输出var变量的变化情况：


```

$ cat testing
#!/bin/bash
trap 'echo "TRACE VARIABLE-> \${var} = \"${var}\"' DEBUG
var=10
var=20
exit 0
$ testing
TRACE VARIABLE-> $var = ""
TRACE VARIABLE-> $var = "10"
TRACE VARIABLE-> $var = "20"
$

```

如果**trap**语句中跟踪的信号为**ERR**，则Shell脚本中的任何命令无论何时返回非0值的出口状态，Shell都会执行**trap**语句定义的处理动作。下列例子中的**trap**语句表示，只要某个命令返回非0值的出口状态，立即显示**Filename**变量的值。

```

$ cat testing2
#!/bin/bash
trap 'echo "$LINENO: Filename = \"${Filename}\"' ERR
ls -l somefile
exit 0
$ testing2
somefile: No such file or directory
3: VAR Filename = ""
$

```

下面的例子说明，不管脚本是否正常终止，只要执行了**exit**语句或在Shell脚本执行结束时，立即执行**trap**语句中定义的处理动作：

```

$ cat testing3
#!/bin/sh
# EXIT is the signal name generated upon exit from a script.
trap 'echo Variable Setting List: 1st = $min 2nd = $max' EXIT
echo "This line will print before the \"trap\" statement."
echo "even though the \"trap\" statement occurs first."

min=$1
max=$2

if [ $min -gt $max ]
then
    exit 1
fi
$ testing3 6 8
This line will print before the "trap" statement.
even though the "trap" statement occurs first.
Variable Setting List: 1st = 6, 2nd = 8
$

```

下面是另外一种例子，说明怎样使用“**set -v**”命令，使Shell能够显示其读入的每一条命令（包括注释行），以及执行命令后的输出结果。需要说明的是，在调试较大的Shell脚本时，这种方法并不适用——面对满屏幕的命令与输出信息，第一个感觉恐怕就是头痛，简直无从下手。

```
$ cat trace.sh
#!/bin/sh
set -v
# This is a comment
date
echo ${TZ}
set +v
date
exit 0
$ trace.sh
# This is a comment
date
Fri Jul  3 16:36:16 CST 2009
echo ${TZ}
PRC
set +v
Fri Jul  3 16:36:16 CST 2009
$
```

关闭“-v” Shell执行过程跟踪设置

下面的例子说明怎样使用“set -x”命令显示Shell读入并执行的每一条命令。与“set -v”命令不同的是，Shell将会在输出其读入的每个命令语句之前增加一个标记，如加号“+”字符（除了输出PS4的变量值作为标记之外，Shell还会在之前插入一个与PS4变量值第一个字符相同的字符），以区别于命令的输出数据。同时，Shell还会在输出命令之前，执行必要的变量替换、命令替换以及元字符文件名生成，然后才显示命令与替换后的实际参数。

```
$ cat trace2.sh
#!/bin/sh
set -x
# This is a comment
date
dir=`pwd`
echo $dir
set +x
date
echo $HOME
exit 0
$ trace2.sh
+ date
Fri Jul  3 16:36:56 CST 2009
+ pwd
dir=/export/home/gqxing/script
+ echo /export/home/gqxing/script
/export/home/gqxing/script
+ set +x
Fri Jul  3 16:36:56 CST 2009
/export/home/gqxing
$
```

关闭Shell执行过程跟踪设置

原则上，Shell脚本允许使用未声明的变量。但利用“set -u”命令，可以禁止Shell使用事先未声明的变量。利用这一特性，可以发现变量名的拼写错误。下面的例子说明怎样使用“set -v”和“set -u”命令显示Shell执行的每一条命令，一旦遇到事先未曾声明的变量，将会停止

Shell脚本的运行，以便找出是否存在变量名拼写有误的错误。示例如下：

```
$ cat trace3.sh
#!/bin/sh
set -v
set -u
name=first
cat ${nmae}.file           # 变量名拼写错误
wc -l ${name}.file
set +n
set +v
exit 0
$ trace3.sh
set -u
name=first
cat ${nmae}.file
trace3.sh: nmae: parameter not set
$
```

7.15.3 系统性能考虑

1. PATH变量的组织

PATH变量设置的目录顺序直接决定了命令检索的速度。每当执行一个非Shell的内置命令时，系统都需要按照**PATH**变量指定的目录顺序检索命令。如果**PATH**变量设置的目录顺序不恰当，将会影响命令的检索速度。一种典型的错误设置是把当前目录放到**PATH**变量的标准目录组织顺序的前面。这种设置造成的后果是，每次执行命令时，系统都会首先检索当前目录。这便引出两个问题：当前目录下的程序究竟能够执行多少次？**/bin**和**/usr/bin**等目录中的命令需要执行多少次？因此，在下列两个**PATH**变量的目录顺序设置方式中，一般应取前者而不是后者：

```
PATH=/bin:/usr/bin:.
PATH=./bin:/usr/bin
```

把当前目录放到**PATH**变量的前面，对命令检索的时间影响究竟有多大，还取决于当前目录的大小。如果当前目录较大，将直接影响Shell的运行性能。还有一个实际问题，如果把当前目录放到**PATH**变量的前面，则无法创建与**/bin**或**/usr/bin**等目录中的命令同名的文件。

2. 文件的访问方式

当需要访问同一目录中的大量文件时，最好利用**cd**命令首先进入该目录。在解释较长的目录文件名时，操作系统需要从路径名的第一个目录开始，逐层检索各级子目录，直至找到最终的文件名，因而需要花费较多的时间。如果给定的是相对于当前目录的文件名，操作系统只需直接访问给定的文件即可。

例如，下列两个脚本都是处理**/home/icbc/credit/report/month**目录中的所有文件，但因采取的方法不同，其运行效率也不相同。

```
for fname in /home/icbc/credit/report/month/*
do
    cat $fname
done

cd /home/icbc/credit/report/month
```

```
for fname in *  
do  
    cat $fname  
done
```

上述第一种处理方法要求操作系统解释较长的路径名；而第二种方法虽然增加了一个语句，但操作系统只需在当前目录下直接访问每一个文件。

3. 内部命令与外部命令

作为解释程序的一部分，**Shell**提供若干内置命令语句。内置命令语句的执行速度明显快于外部的系统命令，其原因有三：

(1) 执行内置命令语句不必创建新的进程。对于提高运行效率，这是一个非常重要的关键因素。因为在运行一个命令时，耗费时间最多的是创建新进程，其过程包括分配内存空间，填写进程控制表，从磁盘中把程序调入内存等。

(2) 执行内置命令语句时不需要按照**PATH**变量检索指定的目录。例如，当试图执行一个**test**命令时，**Shell**根本就不会检索与内置命令同名的**/usr/bin/test**命令或用户自己编写的**test**程序，除非明确指定必须执行**/usr/bin**目录中的**test**命令或自己提供的**test**程序。

(3) 当执行一个**Shell**脚本时，系统首先需要打开这个文件，然后再逐行解释执行其中的命令语句。如果命令是一个编译的应用程序，则需要创建一个新进程，等到把程序装入内存后才能再执行，而**Shell**内置语句只需在现有的**Shell**进程中执行。

4. 管道中的命令顺序

在使用管道时，应适当地考虑过滤程序的位置顺序，逐步减少数据处理的信息量，以改善整个管道命令的运行效率。其中的一个原则是要尽可能地把过滤程序安排在最前面，把非过滤的程序安排在后面。尤其要注意把比较耗时的**sort**程序放在过滤程序的后面。**grep**命令就是一个较好的过滤程序的例子，可以滤掉大量不必要的数据，从而减少后续命令的数据处理量，提高整个管道命令的运行速度。因此，应像下面第一个例子那样，尽可能地把**grep**等过滤程序放在前面，而不是后面。

```
who | grep something | sort  
who | sort | grep something
```

第8章 进 程 管 理

进程是UNIX系统中的重要概念。本章首先介绍怎样查询进程的状态信息，监控进程及系统资源，必要时终止进程的运行。然后讨论怎样调整进程的调度类别及优先级，使关键业务处理能够以较高的优先级运行，普通的例行业务能够以较低的优先级运行，从而提高系统的整体性能。最后讨论怎样提交后台作业，定时运行系统任务和用户程序等。

UNIX系统首次引入了进程的概念。所谓的进程是指处于运行状态的程序。严格地说，进程是程序从开始调度运行直至终止执行的整个生命周期的全过程。进程管理是UNIX系统中的一个重要组成部分，负责管理和控制所有的动态过程和资源（文件系统负责管理所有的静态信息和资源）。

通常，UNIX系统中的进程可分为两大类：系统进程和用户进程。系统进程主要负责UNIX系统的生成、管理、维护和控制，其中包括init进程等。用户进程指的是由用户通过Shell命令行界面（或GUI桌面）提交系统运行的命令和应用程序等。除了少数进程外，系统中的所有进程几乎都是由init进程直接或间接启动的。也就是说，init进程几乎是所有进程的直接或间接父进程。

每个进程都有一个系统赋予的进程标识（进程ID或PID），并与启动进程的用户（用户ID）等相关联，所有的进程均由进程子系统负责管理。用户可以查询进程的状态信息，但只能控制自己的进程，例如，向进程发送控制信号，重新启动和终止进程等。只有超级用户才有权力控制所有的进程。

UNIX系统采用多用户、多任务的进程管理机制，保证所有处于竞争状态的进程都能够公平合理地分享系统资源，保证重要的进程能够优先分配到资源，普通用户和超级用户均可不同程度地、实时地调整活动进程的优先级。

8.1 ps命令概述

在UNIX系统中，获取进程状态信息的常用工具是ps命令。ps命令可用于查询系统中活动进程的状态信息，如进程的起始运行时间和资源占用情况等，这些信息对进程和系统性能的管理都是非常有用的。ps命令的语法格式简写如下：

```
ps [-aAcefl] [-g grplist] [-p pidlist] [-t termlist] [-u usrlist]
```

表8-1给出了ps命令的部分常用选项。

表8-1 ps命令的常用选项

选项	简单说明
-a	显示系统中所有活动进程的当前状态信息（与终端无关联的进程除外）
-A	显示系统中当前所有进程的状态信息。其作用等同于“-e”选项
-c	与“-l”选项一起使用时能够显示进程的调度信息，包括进程的调度类别与优先级等
-e	显示系统中当前所有进程的状态信息

(续表)

选项	简单说明
-f	显示进程的重要状态信息，尤其是进程的起始运行时间和进程占用的CPU时间等
-l	显示进程的详细状态信息（进程的起始运行时间除外）
-g <i>grplist</i>	显示与指定的有效用户组ID或用户组名有关的进程状态信息
-p <i>pidlist</i>	显示指定进程ID的进程状态信息
-t <i>termlist</i>	显示与指定的终端设备有关的进程状态信息
-u <i>usrlist</i>	显示与指定的有效用户ID或用户名有关的进程状态信息

通常，一个最简单的ps命令仅仅显示当前用户自己的进程状态信息。例如：

```
$ ps
  PID TTY          TIME CMD
  865 pts/4        0:00 ksh
  869 pts/4        0:00 ps
$
```

如果想要了解进程的更多信息，可在ps命令中增加“-e”或“-A”选项，这意味着显示所有进程的状态信息，增加“-f”或“-l”等选项，还会输出更多的重要状态信息。而且，也可以组合使用ps命令的各种选项，例如：

```
$ ps -ef
  UID  PID  PPID  C   STIME TTY          TIME CMD
  root     0     0   0 18:08:59 ?        0:31 sched
  root     1     0   0 18:09:07 ?        0:00 /sbin/init
  root     2     0   0 18:09:07 ?        0:00 pageout
  root     3     0   0 18:09:07 ?        0:00 fsflush
  root     7     1   0 18:09:12 ?        0:04 /lib/svc/bin/svc.startd
  root     9     1   0 18:09:12 ?        0:10 /lib/svc/bin/svc.configd
  .....
  root   854   852   0 18:11:31 pts/3    0:00 -ksh
  root   870   854   0 18:11:50 pts/3    0:00 vi /rmdisk/noname/cmd.out
  gqxing 871   865   0 18:12:02 pts/4    0:00 ps -ef
$
```

取决于提供的命令选项，ps命令通常能够给出下列信息：

- 进程当前的工作状态（S）；
- 进程标识（PID）；
- 父进程标识（PPID）；
- 用户标识（UID）；
- 进程所处的调度级别（CLS）；
- 进程的优先权（PRI）；
- 进程的地址空间（ADDR）；
- 进程占用的内存（RSS）；
- 进程占用的CPU时间（TIME）。

表8-2描述了ps命令的输出数据分为若干字段，这些字段是否能够全部出现以及何时出现，

取决于提供的命令选项。

表8-2 ps命令输出的字段信息一览

字段	简单说明
F	进程的标志字段（由于历史的原因而保留，当前基本上已无实际意义）： 0 进程内存映像已从内存移至交换分区（swap） 1 总是驻留在内存中的系统内核进程（如sched、pageout和fsflush等）
S	采用下列字符表示进程的当前状态： O 进程当前正在处理器中运行。任何时刻，每个处理器只能执行一个进程，因而只有一个进程能够处于运行状态 S 进程因等待某一事件的完成而处于休眠状态 R 进程正处于运行队列，等待再次获得运行的机会 T 由于父进程的跟踪调试或作业控制信号导致进程中断而处于暂停运行状态 W 进程处于等待状态（等待转入运行队列） Z 僵尸进程（进程已终止执行，但父进程并未等待，处于彻底终止运行之前的中间阶段）
UID	进程属主的有效用户ID
PID	进程的进程ID
PPID	进程的父进程ID
C	进程生命周期的CPU利用率（百分比），也即进程实际使用的CPU时间除以进程整个生命周期耗费的时间总量（包括等待时间）
CLS	进程的调度类别，如标准的分时进程调度类别TS等
PRI	进程的优先级。优先级的数字越大表示进程的优先级越高
NI	进程优先级的nice调整值（其范围为19~-20），用于调整进程的优先级
ADDR	proc进程结构的地址空间
SZ	进程核心映像（包括代码段、数据段以及栈空间）占用的物理内存页面的大小
WCHAN	因等待某一资源或事件的发生而使进程处于等待状态的内核函数的名字（“-”表示进程正在运行）
STIME	进程的起始运行时间。如果起始时间位于24小时之内，以“HH:MM”形式表示；如果超过24小时，则以“mmm dd”形式表示起始运行时间，其中mmm表示月（如英文月名所写的前三个字母），dd表示日
TTY	控制终端。表示进程（或其父进程）是从哪一个终端上启动运行的。如果这个字段是问号“？”，则表示进程与任何控制终端无关
TIME	从调度运行开始，进程迄今累计占用的CPU时间总和。以“[dd-]hh:mm:ss”形式表示
CMD	进程对应的命令或程序的名字。如果命令行过长，超出部分（包括选项或参数）将被截断，使得每个进程仅占用一行显示位置（除非使用“-w”选项）

8.2 查询进程及其状态信息

8.2.1 查询当前活动的进程

利用“ps -a”命令，可以显示当前系统中从终端中运行或调用的所有活动进程及其运行时间等信息。例如：

```
$ ps -a
  PID TTY          TIME CMD
  764 pts/2        0:00 ksh
  865 pts/4        0:00 ksh
  870 pts/3        0:00 vi
  872 pts/4        0:00 ps
$
```

从命令的输出结果可以看出，当前系统中直接从终端中运行或调用的活动进程只有vi和ps，其进程ID分别为870和872，分别是在终端/dev/pts/3和/dev/pts/4中运行的。输出信息中的TIME字段表示进程从调度运行开始直至运行ps命令时占用的CPU时间，运行时间以时、分、秒为单位。

8.2.2 查询系统中所有的进程

ps命令经常用于检查期望运行的进程是否已经启动，或准备终止的进程是否已经停止运行。为了获取系统中当前调度运行的所有进程及其状态信息，可以使用“ps -e”或“ps -A”命令。例如：

```
$ ps -e
  PID TTY          TIME CMD
    0 ?            0:31 sched
    1 ?            0:00 init
    2 ?            0:00 pageout
    3 ?            0:00 fsflush
    7 ?            0:04 svc.star
    9 ?            0:10 svc.conf
.....
$
```

从上述输出结果可以看出，在系统启动过程中，首先运行的进程是进程0，即sched，然后是init、pageout和fsflush等进程。sched进程的主要功能是根据一定的原则，在内存与磁盘交换区之间交换进程，确保进程的公平调度，以及系统具有足够的内存空间，能够满足其他进程的内存空间请求。pageout进程的主要功能是释放进程的内存页面，或把仍然有效的内存页面写入磁盘交换区，在进程的正常调度与确保系统维持一定的空闲内存之间取得平衡。fsflush进程的主要功能是把缓存在内存中的文件数据及时写到磁盘上的文件系统中。

在上述输出信息中，TTY字段的问号“?”表示系统中的许多进程都是由系统直接调度运行的，并非源于某个特定的终端。实际上，除了进程0、2或3之外，系统中的所有进程几乎都是由init进程直接或间接启动的。sched是在系统引导过程中由内核直接创建的。

利用“ps -e”命令能够给出系统中所有进程信息的特点，可以查询某个特定进程是否已经启动，这是ps命令的一个重要的应用。例如，为了查询OpenSSH的守护进程sshd当前是否正在运行，可以使用下列组合命令：

```
$ ps -e | grep sshd
  520 ?            0:00 sshd
$
```

也可以使用改进的pgrep命令，查询指定进程的运行状态。如果指定的进程已经启动且正在运行，下列命令将会给出活动进程的PID：

```
$ pgrep sshd
520
$
```

8.2.3 显示进程的重要状态信息

系统管理员经常需要考察系统中的进程，监控并找出影响系统性能、危及系统安全的进程。使用ps命令的“-f”选项，可以获取进程的重要状态信息。例如：

```
$ ps -f
      UID      PID  PPID    C   STIME TTY          TIME CMD
    gqxing    865    861    0 18:11:41 pts/4        0:00 -ksh
    gqxing    884    865    0 18:14:42 pts/4        0:00 ps -f
$
```

从上面的输出结果可以看出，其中给出了很多重要的进程信息，包括用户标识、进程标识、父进程标识、进程起始运行时间和累计运行时间等。“-f”选项经常与“-e”选项一起使用，用于显示系统中所有进程的重要状态信息。这将是一个长长的进程列表，下面只是截取了其中的一部分：

```
$ ps -ef
      UID      PID  PPID    C   STIME TTY          TIME CMD
      root         0      0    0 18:08:59 ?          0:31 sched
      root         1      0    0 18:09:07 ?          0:00 /sbin/init
      root         2      0    0 18:09:07 ?          0:00 pageout
      root         3      0    0 18:09:07 ?          0:00 fsflush
      root         7      1    0 18:09:12 ?          0:04 /lib/svc/bin/svc.startd
      root         9      1    0 18:09:12 ?          0:10 /lib/svc/bin/svc.configd
      .....
$
```

利用ps命令可以监控系统中是否存在异常的进程。为此，可针对ps命令输出的TIME和STIME字段，重点检查进程迄今为止累计占用的CPU时间。除非是系统进程，如果某个用户进程从启动至今无缘无故地耗费了大量的CPU时间，很有可能进程已处于无限循环状态。

8.2.4 显示进程的详细状态信息

利用ps命令的“-l”选项，可以列出每个活动进程的详细状态信息。例如：

```
$ ps -l
F S      UID      PID  PPID    C PRI  NI     ADDR     SZ  WCHAN    TTY          TIME CMD
0 S      100      865    861    0  62  20          ?   943          ?    pts/4        0:00 ksh
0 O      100      886    865    0  40  20          ?   998          ?    pts/4        0:00 ps
$
```

利用ps命令的“-c”选项，可以列出每个活动进程的调度类别及其优先级。根据反映进程调度类别的CLS字段，检查当前是否存在以较高优先级运行的用户进程。一个实时进程有可能会独占全部的CPU时间。同样，一个具有较高优先级（nice值）的分时进程也不容忽视。

```
$ ps -cl
F S      UID  PID  PPID   CLS   PRI     ADDR     SZ  WCHAN TTY          TIME CMD
0 S      100  865   861    IA     38          ?   943          ? pts/4        0:00 ksh
```

```
0  O   100  887   865   IA   59      ?   998      pts/4      0:00   ps
$
```

如果再组合使用ps命令的“-e”选项，可以列出所有进程的调度类别及其优先级等详细状态信息：

```
$ ps -ecl
F  S   UID   PID PPID   CLS  PRI   ADDR      SZ   WCHAN TTY      TIME CMD
1  T     0     0    0   SYS   96     ?        0           ?      0:31 sched
0  S     0     1    0   TS    59     ?       619         ? ?    0:00 init
1  S     0     2    0   SYS   98     ?        0           ? ?    0:00 pageout
1  S     0     3    0   SYS   60     ?        0           ? ?    0:00 fsflush
0  S     0     7    1   TS    59     ?      3801         ? ?    0:04 svc.star
0  S     0     9    1   TS    59     ?      2799         ? ?    0:10 svc.conf
.....
$
```

8.3 监控进程及系统资源

前面介绍了怎样利用ps命令获取进程的各种状态信息。需要说明的是，ps命令的输出只是系统执行命令那一刻的快照信息。为了能够连续地观察进程的实时状态信息以及其他系统信息，可以使用更受欢迎的top命令，UNIX系统一般都不提供这一软件，如果需要，可以从<http://sourceforge.net/projects/unixtop>网站中下载最新的源代码，也可从UNIX系统厂家的网站上获取编译好的top程序和源代码。例如，Solaris系统的用户可以从<http://www.sunfreeware.com>网站中下载。top命令语法格式简写如下：

```
top [-IScquv] [-d count] [-s time] [-o field] [-U username] [number]
```

其中，“-d”选项表示仅更新显示指定的次数，然后即自行退出top命令。如果未指定“-d”选项，top命令会持续地更新屏幕，显示进程的运行状态，直至用户输入“q”命令。“-s”选项表示数据取样和屏幕刷新的时间间隔，默认的取样时间间隔为5秒。

正如其名字所蕴含的那样，top命令仅仅列出系统中的前15个进程（取决于终端窗口的大小）及其他系统信息，并周期地进行更新。top通常会按照进程的CPU占用率，从高到低依次列出每一个进程。

通常，top每隔5秒钟更新一次数据，以反映系统的当前运行状态。top命令的典型输出形式如图8-1所示。

top命令输出的上半部分（前5行）是系统运行状态的汇总信息，其中第1行包括最近一次分配的进程ID（1000）、系统的三个平均负载值（0.02、0.00和0.00），以及系统自启动以来的累计运行时间（1小时5分59秒）。第2行是系统的当前时间（12:01:40）。第3行是进程的统计信息，其中包括系统中现有进程的总数（61）、处于休眠状态的进程数量（60）以及处于运行状态的进程数量（1）等。第4行是对CPU工作状态的分类统计，其中包括CPU处于空闲状态（93.2%）、用户模式（1.1%）、系统内核模式（5.7%）以及等待I/O（0.0%）等的百分比。第五行是内存使用情况的分类统计，其中包括系统配置的物理内存总量（1023MB）、空闲内存数量（495MB）、系统配置的交换区总量（1028MB）以及交换区空闲的数量（1028MB）等。表8-3给出了top命令上半部分的输出信息及简要说明。

```

last pid: 1000; load avg: 0.02, 0.00, 0.00;          up 0+01:05:59
12:01:40
61 processes: 60 sleeping, 1 on cpu
CPU states: 93.2% idle, 1.1% user, 5.7% kernel, 0.0% iowait, 0.0% swap
Memory: 1023M phys mem, 495M free mem, 1028M total swap, 1028M free swap

  PID USERNAME LWP PRI NICE  SIZE  RES STATE   TIME    CPU COMMAND
  999 root         1  60   0 1644K 1028K sleep 0:01  4.86% find
  578 root         1  59   0  48M   43M sleep 0:11  0.19% Xorg
  824 root         1  59   0 7580K 4092K sleep 0:02  0.07% dtterm
  667 noaccess    19  59   0 169M  106M sleep 0:31  0.06% java
  950 root         1  59   0 2992K 1660K cpu/1 0:01  0.03% top
  823 root         1  49   0 8912K 5760K sleep 0:00  0.01% dtfile
  993 root         1  49   0 7512K 3952K sleep 0:00  0.01% dtterm
  773 root         1  59   0  93M 9092K sleep 0:00  0.01% composite_aux_g
  743 root         1  59   0  63M 5256K sleep 0:00  0.01% palette_aux_gtk
  820 root         5  59   0 9320K 6144K sleep 0:05  0.00% dtwm
    7 root        14  59   0  15M   10M sleep 0:03  0.00% svc.startd
  143 root        31  59   0 9192K 2700K sleep 0:00  0.00% nscd
  716 root         1  59   0 7520K 1740K sleep 0:00  0.00% sendmail
  369 root         1  59   0 2068K  924K sleep 0:00  0.00% sac
  415 root         1  59   0 2432K 1168K sleep 0:00  0.00% ttymon

```

图8-1 top命令主界面

表8-3 top命令上半部分的输出信息及说明

字段	简单说明
last pid	最近一次分配的进程ID
load avg	其中给出了三个平均系统负载值，分别表示在最近1分钟、5分钟和15分钟内系统运行队列的平均长度
up	系统自启动以来迄今为止的运行时间
hh:mm:ss	第2行是系统的当前时间，以小时、分和秒的形式显示，表示top命令给出的是此刻的系统运行状态信息
processes	<p>第3行是进程运行状态信息的汇总统计。其中的三个字段分别表示处于不同运行状态下的进程数量：</p> <ul style="list-style-type: none"> • processes: 当前正在运行，或已处于运行队列，一旦调度即可运行的进程数量 • on cpu: 当前正在运行的进程数量 • sleeping: 因等待I/O或外部事件完成而处于休眠状态的进程数量
CPU states	<p>第4行是CPU工作状态的分类统计信息，分述如下：</p> <ul style="list-style-type: none"> • user: CPU处于用户模式的时间量所占的百分比 • kernel: CPU处于系统模式的时间量所占的百分比 • idle: CPU处于空闲状态的时间量所占的百分比 • iowait: CPU等待I/O完成的时间量所占的百分比
Memory	<p>第5行是系统物理内存与交换区的汇总统计信息。其中每个字段的意义分述如下：</p> <ul style="list-style-type: none"> • phys mem: 系统配置的可用物理内存总数 • free mem: 当前仍然空闲的物理内存计数 • total swap: 系统配置的交换区总数 • free swap: 当前仍然空闲的交换区计数

从图8-1的输出数据中可以看出，截止到12时01分40秒，在最近1、5和15分钟之内，系统的平均负载量分别是0.02、0.00和0.00。总共有61个进程，但除了top进程之外，其他大部分进程均处于休眠状态。CPU有93.2%的时间是空闲的，1.1%的时间直接服务于用户，5.7%的时间用于系统开销。通过这些数据，可以了解系统资源的使用情况。

top输出的第二部分根据每个进程的CPU占用量，按照从高到低的顺序，给出每个进程的状态信息，参见表8-4。

表8-4 top提供的进程状态信息

字段	简单说明
PID	系统赋予进程的进程ID。进程ID通常是一个小于65 536的正整数。一旦进程结束或被强行终止，进程ID还可以重用
USERNAME	启动或拥有进程的用户名
PRI	分配给每个进程的优先级，用于确定进程调度的优先顺序
NICE	进程优先级的nice调整值。nice调整值可以是负数，表示提高优先级。在UNIX系统中，nice调整值的取值范围是-20至20。大多数用户进程采用0作为优先级的nice调整值，表示采用最基本的优先级。但也可以选择使用一个大于零的nice调整值启动进程，使系统能够降低进程的优先级。对于一个需要长时间占用CPU（运行时间长，以CPU处理为主）的处理任务来讲，这是一种常规的做法，可以避免干扰交互进程。注意，只有超级用户才能采用负数提高进程的优先级。UNIX系统中的nice或renice命令可用于设置进程优先级的nice调整值
SIZE	进程的大小，即进程代码段、数据段和栈段的总和（以KB为单位）
RES	进程当前实际占用的物理内存数量，也即进程内存驻留部分的大小（以KB为单位）。一个进程可以申请分配较大的虚拟内存空间，但通常只有一部分驻留在物理内存中
STATE	进程当前运行的状态信息。常见的运行状态如下： <ul style="list-style-type: none">• run: 进程正在运行• sleep: 进程因等待I/O或外部事件的完成而处于休眠状态• idle: 进程处于空闲状态• stop: 进程因跟踪调试，或因收到某个信号（如按Ctrl-Z键）而暂时停止运行• zombie: 进程已终止，但其父进程尚未完成善后处理
TIME	进程累计占用的CPU时间（以秒为单位），其中包括内核与用户CPU时间。表示进程自开始运行以来，迄今为止占用的整个CPU时间。这个时间量也是运行进程时实际耗费的CPU时间
CPU	进程占用CPU时间的百分比。top通常以此列数据基准，按照从大到小的顺序显示进程
COMMAND	进程对应的命令行或程序名

```
Top version 3.6.1, Copyright (c) 1984 through 2006, William LeFebvre

A top users display for Unix

These single-character commands are available:

^L      - redraw screen
<sp>    - update screen
C       - toggle the use of color
M       - sort by memory usage
N       - sort by pid
P       - sort by CPU usage
T       - sort by time
c       - toggle the display of process commands
d       - change number of displays to show
e       - list errors generated by last "kill" or "renice" command
h or ?  - help; show this text
i or I  - toggle the displaying of idle processes
k       - kill processes; send a signal to a list of processes
n or #  - change number of processes to display
o       - specify sort order (cpu, size, res, time)
q       - quit
r       - renice a process
...More...
```

图8-2 top命令交互操作说明

top命令在输出各种统计信息，不断刷新屏幕的同时，也提供一个交互界面。利用top命令支持的各种交互命令，可以影响top命令的输出结果。例如，输入“P”和“T”命令，可以分别按占用CPU时间的百分比以及占用CPU的时间量，显示进程的运行状态。输入“n”或“#”命令，可以限定top能够显示的进程数量。输入“h”或“?”命令，可以显示top命令的使用说明。若想退出top命令，可以直接输入“q”命令，如图8-2所示。

8.4 终止进程的运行

有时，用户需要强行终止某个进程。例如，进程的运行时间过长，或进程因逻辑错误而陷于无限循环状态等。普通用户只能终止自己的进程，只有超级用户才能终止系统中的任何进程（进程标识号较小的内核进程除外，终止此类进程通常会引起系统瘫痪）。

UNIX系统既支持POSIX标准信号，也支持POSIX实时信号。在收到信号之后，进程通常会采取下列默认的处理动作之一：

- 终止进程的运行；
- 忽略收到的信号，进程继续运行；
- 终止进程的运行，并把进程的内存映像转储到一个core文件；
- 暂时停止进程的执行。

为了强行终止一个进程，可使用Shell的内部命令kill，或/usr/bin/kill命令。kill命令的语法格式简写如下：

```
kill [-s signame | -signame | -signum] pid ...
kill [-l] [exit_status]
```

其中，*signame*是一个规范的信号名（如SIGKILL），或缩写的信号名（如KILL）。*signum*是一个信号编码，表示kill命令发送给指定进程的信号（参见表8-5）。*pid*是进程ID。此外，利用“-l”选项还可以列出系统支持的所有信号及其编码值。

表8-5 UNIX系统支持的部分信号

信号	信号名	默认的处理动作	简单说明
1	SIGHUP或HUP	终止进程	挂断。终端通信连接断开或控制进程终止时产生的信号
2	SIGINT或INT	终止进程	中断。按下中断键（也即Ctrl-C键）时产生的信号
3	SIGQUIT或QUIT	终止进程，生成内存映像文件（core）	退出。按下Quit键（也即Ctrl-\或Ctrl-Shift-I键）时产生的信号
4	SIGILL或ILL	终止进程，生成内存映像文件（core）	非法指令。执行非法机器指令时产生的信号。其中包括由非法操作码、非法操作数、非法寻址模式、寄存器或内部栈错误等原因产生的信号
5	SIGTRAP或TRAP	终止进程，生成内存映像文件（core）	硬件故障或断点跟踪等情况产生的信号
6	SIGABRT或ABRT	终止进程，生成内存映像文件（core）	异常终止信号（由abort()函数产生的异常终止信号）
7	SIGEMT或EMT	终止进程，生成内存映像文件（core）	仿真陷阱指令（EMT）故障产生的信号
8	SIGFPE或FPE	终止进程，生成内存映像文件（core）	浮点算术运算异常（如除数为零或浮点运算溢出）时产生的信号

(续表)

信号	信号名	默认的处理动作	简单说明
9	SIGKILL或KILL	终止进程	进程无法捕捉与封锁，也不能忽略的终止信号。可供系统管理员使用kill命令终止任何进程（部分核心进程除外）
10	SIGBUS或BUS	终止进程，生成内存映像文件（core）	总线故障信号。包括由非法地址、不存在的地址或其他硬件错误等原因产生的信号
11	SIGSEGV或SEGV	终止进程，生成内存映像文件（core）	内存地址越界或访问权限不足时产生的信号（当代码地址超出进程的地址空间时产生的信号）
12	SIGSYS或SYS	终止进程，生成内存映像文件（core）	系统调用有误。发现非法系统调用（如参数有误）时产生的信号
13	SIGPIPE或PIPE	终止进程	管道断开信号（当进程写入一个管道或套接字，但读取管道或套接字的另一个进程已经终止时产生的信号）
14	SIGALRM或ALRM	终止进程	由alarm()系统调用产生的时钟超时信号
15	SIGTERM或TERM	终止进程	终止进程信号。这是kill命令产生的默认终止信号
16	SIGUSR1或USR1	终止进程	用户定义的信号1，供应用编程使用
17	SIGUSR2或USR2	终止进程	用户定义的信号2，供应用编程使用
18	SIGCHLD或CHLD	忽略	子进程状态发生变动信号，如子进程结束或停止运行时向父进程发送的信号
19	SIGPWR或PWR	忽略	电源故障信号。当电源出现故障，改由UPS提供系统电源供电时产生的信号。这个信号通常主要由操作系统处理，应用程序不受影响，除非UPS电源供电不充足
20	SIGWINCH或WINCH	忽略	窗口大小发生变动时产生的信号
21	SIGURG或URG	忽略	当从网络连接（套接字）中接收到的数据发生错误，通知进程出现紧急情况时发送的信号
22	SIGPOLL或POLL	终止进程	事件轮询信号
23	SIGSTOP或STOP	停止进程	停止进程运行信号。这是用于停止一个进程的作业控制信号，这个信号主要用于作业控制，既不能捕捉，也不能忽略
24	SIGTSTP或TSTP	停止进程	键盘停止信号。在交互方式下，通过按Ctrl-Z键停止当前进程时产生的信号。主要用于作业控制
25	SIGCONT或CONT	继续（或忽略）	令进程继续运行的信号，主要用于作业控制。如果进程已暂停运行，则默认的处理动作是继续运行。如果进程正在运行，则默认的处理动作是忽略收到的信号
26	SIGTTIN或TTIN	停止进程	后台进程试图从控制终端中读取数据时产生的信号。主要用于作业控制

(续表)

信号	信号名	默认的处理动作	简单说明
27	SIGTTOU或TTOU	停止进程	后台进程试图向控制终端输出数据时产生的信号。主要用于作业控制
28	SIGVTALRM或VTALRM	终止进程	由setitimer()系统调用设置的虚拟间隔时钟超时信号
29	SIGPROF或PROF	终止进程	由setitimer()系统调用设置的内核抽样间隔时钟超时信号
30	SIGXCPU或XCPU	终止进程，生成内存映像文件（core）	当进程超过了其最大的软性CPU时间限制时产生的信号
31	SIGXFSZ或XFSZ	终止进程，生成内存映像文件（core）	当进程创建的文件超过了其能够创建的软性最大文件容量限制时产生的信号

如果运行kill命令时未指定任何信号，则发送默认的信号15（SIGTERM）。在表8-5中给出的信号中，信号9（SIGKILL）是不能捕捉的，因而可用于强行终止任何进程。也就是说，在kill命令中使用信号9，可以确保进程无条件终止。然而，信号9不能轻易使用。建议不要使用信号9终止诸如数据库服务器等进程，以免造成数据的丢失。

任何用户均可以终止自己的进程，如果想终止其他用户的进程，必须拥有超级用户的权限。为了终止进程的继续运行，通常可参照下列步骤。

首先，可以使用ps命令获取进程的PID。例如，下列ps命令用于显示属于daemon，或以daemon用户身份启动的所有进程：

```
$ ps -fu daemon
  UID  PID  PPID  C   STIME TTY          TIME CMD
daemon  136    1    0  13:25:39 ?           0:00 /usr/lib/crypto/kcfd
daemon  297    1    0  13:25:49 ?           0:00 /usr/sbin/rpcbind
```

此外还可以使用pgrep命令直接获取指定进程的PID。实际上，pgrep命令是对ps与grep组合命令的改进，可以直接给出指定进程的PID。例如，下列pgrep命令的输出结果表示当前系统中共有2个远程注册的用户：

```
$ pgrep in.telnetd
767
834
$
```

然后，可以使用kill（或pkill）命令终止进程的继续运行。例如，为了终止Telnet远程注册守护进程in.telnetd的运行，禁止任何远程访问，可以使用下列kill命令：

```
# kill -9 767 834
#
```

注意，在正式利用kill命令终止一个进程时，之前应首先尝试使用不带任何信号参数的kill命令，然后再观察进程是否已经终止。如果不奏效，再使用信号9。最后，可以利用ps或pgrep等命令验证进程是否已经终止。

8.5 调整进程的调度类别及优先级

最初，UNIX系统只是一个分时系统，为了提供实时支持能力，从System V 4.0开始，UNIX增加了实时进程调度级别。当前，UNIX系统通常主要支持下列三种进程调度类别（某些UNIX系统还增加了其他进程调度类别），其中，实时进程调度类别的全局优先级最高，系统进程调度类别次之，分时进程调度类别最低。

- RT实时进程调度类别。主要用于实时进程调度，其优先级范围为：100～159。
- SYS系统进程调度类别。主要用于系统进程调度，其优先级范围为：60～99。
- TS分时进程调度类别。主要用于用户进程调度，其优先级范围为：0～59（Solaris系统为-60～60）。

利用*prctl*命令，可以调整进程的调度类别与优先级。注意，当需要设置或调整实时进程的调度参数时，需要拥有超级用户的访问权限。为此，可以先使用*root*注册或使用*su*命令成为超级用户。*prctl*命令的语法格式简写如下：

```
prctl {-l | -d [-i idtype] [idlist]}
prctl -s [-c class] [class-specific-opts] [-i idtype] [idlist]
prctl -e [-c class] [class-specific-opts] command
```

其中，*command*表示提交运行的命令。*idlist*可以是进程ID、用户ID或用户组ID，用于指定运行的进程。*prctl*命令的主要选项及其说明参见表8-6。

表8-6 prctl命令的主要选项

选项	简单说明
-l	用于显示系统当前配置的调度类别，以及每一个调度类别的相关配置信息
-d	用于显示进程的调度参数
-s	用于设置进程的调度参数，如调度类别与优先级等
-e	表示按指定的调度类别和优先级执行指定的命令
-c class	用于指定进程的调度类别，如RT、SYS或TS等，表示在指定的调度类别中运行给定的命令
-i idtype	表示采用的ID类型，如pid（进程ID）、uid（用户ID）或gid（用户组ID）等。如果忽略了“-i”选项，默认的ID类型为进程ID

8.5.1 显示进程的调度类别与优先级

进程的优先级是由进程调度程序根据调度策略分配的（使用*disadmin*命令可以获取或修改系统的进程调度参数）。利用*prctl*命令，可以把进程分配到一个特定的调度类别，同时调整进程的优先级。

1. 显示进程调度的配置信息
- 为了获取系统配置的进程调度信息，如进程的调度类别与优先级配置范围等信息，可以使用下列*prctl*命令：

```
# priocntl -l
CONFIGURED CLASSES
=====
SYS (System Class)
TS (Time Sharing)
    Configured TS User Priority Range: -60 through 60
RT (Real Time)
    Maximum Configured RT Priority: 59
.....
#
```

2. 显示进程的全局优先级

如果想要显示进程的全局优先级，可以使用“**ps -ecl**”命令，其中的**PRI**字段即进程的全局优先级。在下面的例子中，**PRI**一列的数值**98**说明**pageout**进程具有最高的优先级：

```
$ ps -ecl
F  S   UID   PID  PPID   CLS   PRI  ADDR   SZ   WCHAN TTY   TIME CMD
1  T    0     0    0    SYS    96   ?     0           ?    0:31 sched
0  S    0     1    0    TS    59   ?    619         ? ?  0:00 init
1  S    0     2    0    SYS    98   ?     0           ? ?  0:00 pageout
1  S    0     3    0    SYS    60   ?     0           ? ?  0:00 fsflush
0  S    0     7    1    TS    59   ?   3801         ? ?  0:04 svc.star
0  S    0     9    1    TS    59   ?   2799         ? ?  0:10 svc.conf
.....
$
```

8.5.2 按照指定的调度类别与优先级运行进程

利用**priocntl**命令，可以调整进程的调度类别与优先级等进程调度参数，使之按照指定的调度类别与优先级运行进程。为此，针对不同的调度类别，为了调整进程的调度参数，可以分别使用下列命令：

```
# priocntl -e -c TS -m tsprilim -p tspri command
# priocntl -e -c RT -t rtquantumm -p rtpri command
```

对于分时进程，“**-m**”选项用于设置用户可调优先级的上限，“**-p**”选项用于设置分时进程的优先级。对于实时进程，“**-t**”选项用于设置实时进程的时间片（默认的时间单位为毫秒），“**-p**”选项用于设置实时进程的相对优先级。**command**是进程对应命令或程序的名字。

例如，下列**priocntl**命令表示按照用户能够选用的最高优先级，在分时进程调度类别中运行**find**命令（在Solaris系统中，分时进程的优先级范围为-60~60）：

```
# priocntl -e -c TS -m 60 -p 60 find / -name core > corelist &
[1]      882
#
```

然后，利用下列**ps**命令即可看到，**find**命令确实是在分时进程（**TS**）调度类别中以最高优先级（**60**）运行的：

```
# ps -ecl | grep find
0 S    0   882   812   TS   60 d5b314b8   431 d3832308 pts/2   0:02 find
$
```

下面的例子展示了怎样以20为相对优先级，在实时进程调度类别中运行给定的命令：

```
# prionctl -e -c RT -p 20 find / -size +2000 > bigfile &
[1]      882
# ps -ecl | grep find
0 S      0    888    812    RT 120 d5b314b8    431 d7409bd0 pts/2    0:02 find
$
```

8.5.3 调整进程的调度类别与优先级

利用下列**prionctl**命令，还可以调整现有进程的调度类别，在调整进程调度类别的同时，还可以设定进程的优先级：

```
# prionctl -s -c TS -m tsprilim -p tspri -i idtype idlist
# prionctl -s -c RT -t rtquantumm -p rtpri -i idtype idlist
```

注意，只有超级用户才能分配实时进程。一旦把用户进程改为实时调度类别的实时进程之后，普通用户无法使用“**prionctl -s**”命令修改进程的实时调度参数。

下面的例子说明了怎样利用**prionctl**命令，把用户**gqxing**（其用户ID为100）的所有进程重新分配到实时进程调度类别，以实时进程的方式运行原有的分时进程。

```
# ps -ecl | grep 100
0 S      100 1170  1158    TS 59  d3446030  1010  d698456e pts/5    0:00  vi
0 S      100 1158  1155    TS 59  d679f9f8   402  d679fa64 pts/5    0:00  ksh
# prionctl -s -c RT -p 20 -i uid 100
# ps -ecl | grep 100
0 S      100 1170  1158    RT 120 d3446030  1010  d698456e pts/5    0:00  ksh
0 S      100 1158  1155    RT 120 d679f9f8   402  d679fa64 pts/5    0:00  vi
#
```

由上可见，在输入**prionctl**命令之前，用户**gqxing**的所有进程均在分时进程调度类别中运行，属于分时进程。执行**prionctl**命令之后，用户**gqxing**的分时进程均变为实时进程，在实时进程调度类别上运行，同时也把进程的相对优先级设定为20（其全局优先级为120）。

而且，自发布上述**prionctl**命令变更进程调度类别之后，用户**gqxing**再输入的任何命令都会在新的进程调度类别中运行。

8.5.4 设置实时进程的时间片

prionctl命令还可用于设置实时进程的时间片（time quantum），即分配给实时进程的最大CPU时间。一个实时进程在其时间片内可以优先占用CPU，即使另一个进程已经分配到一个较高的实时优先级。

```
# prionctl -s -c RT -t rtquantumm -p rtpri -i idtype idlist
```

在设置实时进程时间片的同时，还可以设置进程的优先级。例如，下列命令将会把进程988的进程调度类别设置为实时进程调度类别，其时间片为100毫秒，实时进程的相对优先级为20。

```
# prionctl -s -c RT -t 100 -p 20 -i pid 988
#
```


8.6 调整分时进程的优先级

分时进程的优先级分配取决于系统的调度策略。但是，利用**nice**或**renice**命令可以适当地调整分时进程的优先级。

8.6.1 nice命令

nice命令是与早期UNIX系统兼容的唯一进程优先级调整命令，**renice**命令是伯克利版**renice**命令的实现，具有更灵活的进程管理功能。一个进程的优先级是由进程所在调度类别的调度策略确定的，具体地讲是由进程的全局优先级与**nice**调整值共同确定的。实际上，确定进程的优先级涉及许多因素，其中包括进程调度类别，进程已经占用的CPU时间，以及在分时进程情况下的**nice**调整值等。

在开始运行之后，也可以通过**nice**或**renice**命令，适当地调整进程的优先级。从下列**ps**命令输出的PRI与NI字段可以看到每个进程的优先级与**nice**调整值：

```
$ ps -el
F S  UID  PID PPID  C  PRI  NI  ADDR  SZ  WCHAN  TTY  TIME CMD
1 T   0    0    0   0    0  SY   ?    0           ?    0:31 sched
0 S   0    1    0   0   40  20   ?   619         ? ?    0:00 init
1 S   0    2    0   0    0  SY   ?    0           ? ?    0:00 pageout
1 S   0    3    0   0    0  SY   ?    0           ? ?    0:00 fsflush
0 S   0    7    1   0   40  20   ?  3801         ? ?    0:04 svc.star
0 S   0    9    1   0   40  20   ?  2799         ? ?    0:10 svc.conf
.....
0 S   0  854  852   0   50  20   ?   945         ? pts/3 0:00 ksh
0 S   0  870  854   0   50  20   ?  1589         ? pts/3 0:00 vi
$
```

通过增加**nice**调整值，普通用户可以降低一个进程的优先级。但只有超级用户才能通过减少**nice**调整值，达到增加进程优先级的目的。这个限制可防止用户增加自己的进程优先级，因而无止境地独占共享的CPU时间。对于超级用户而言，**nice**调整值的取值范围为-20至19，其中-20是可用的最大**nice**调整值。对于普通用户而言，**nice**调整值的取值范围为0至19，其中19是可用的最小**nice**调整值。

nice命令的语法格式简写如下：

```
nice [-n number] [command [arguments]]
```

其中，“-n”选项用于指定进程的**nice**调整值，如果未指定“-n”选项，默认的**nice**调整值是10。

按照上述说明，普通用户只能降低进程的优先级，而超级用户既可以降低进程的优先级，又能够提高进程的优先级。为了修改进程的优先级，针对普通用户与超级用户，分述如下。

作为一个普通用户，如果确实需要改变进程的优先级，使一个运行时间较长的普通应用程序能够以一个较低的优先级运行，以免影响其他业务的处理，可通过提高**nice**调整值，降低指定命令的优先级。

例如，下列**nice**命令采用增加5个**nice**调整值的方式，以一个相对较低的优先级执行给定的命令：

```
$ nice -n 5 find / -name core > corelist 2>&1
$
```

作为一个超级用户，如果确实需要改变进程的优先级，使某个关键业务处理能够以一个较高的优先级运行，而普通的应用能够以较低的优先级运行，可以通过降低或提高**nice**调整值，相应地提高或降低进程的优先级。

例如，下列**nice**命令表示降低10个**nice**调整值，以提高指定命令的优先级：

```
# nice -n -10 find / -name core > corelist
#
```

注意：如果赋予进程较高的**nice**调整值，有可能会过多地挤占其他进程的CPU时间，从而影响系统的整体性能。此外，进程优先级的调整不能超出进程调度类别允许的范围。

8.6.2 renice命令

UNIX系统提供的另外一个进程优先级调整命令是**renice**，可用于调整正在运行的分时进程的优先级。**renice**命令的语法格式简写如下：

```
renice priority [[-p] pids] [[-u] users]
```

其中，**priority**是一个整数数值，表示赋予进程的**nice**调整值，正数表示降低进程优先级的**nice**调整值，负数表示增加进程优先级的**nice**调整值，0表示保持进程的基本优先级不变。“-p”选项用于指定进程的进程ID。“-u”选项用于指定特定用户的所有进程。

同样，在调整进程的优先级时，普通用户只能利用**renice**命令增加**nice**调整值（其范围为0~19），从而降低进程的优先级。超级用户可以利用**renice**命令，调整任何进程的优先级，增加或减少**nice**调整值，其范围为-20~19。如果把**nice**调整值设定为最小值19，则仅当系统中没有其他可以调度运行的进程时，才会调度执行**renice**命令指定的进程。采用负数增加进程的优先级时，进程能够得到更多的调度机会，从而运行得更快。**nice**调整值-20是超级用户可用的最大优先级。

例如，使用下列命令可以降低进程ID号为770和771，以及用户**daemon**所有进程的优先级：

```
# renice +2 -p 770 771 -u daemon
#
```

8.6.3 调整进程优先级的作用

下面，我们将利用**time**或**timex**命令，分别采用提高或降低进程优先级**nice**调整值的方式运行同一命令，观察修改**nice**调整值前后的运行效果。

在此之前，首先简单介绍一下**time**和**timex**命令。这两个命令可用于考察一个命令的执行效率。当利用**time**命令提交的程序运行结束之后，**time**命令将会从三个方面，给出在运行指定的程序时占用的时间。

- **real**：表示运行给定程序时耗费的全部时间（以秒为单位）——这个时间包括从调用程序开始，直至程序执行结束期间的全部时间。

- **user**：程序的实际运行时间（以秒为单位）。这个时间仅包含系统执行用户代码时占用

的时间，而不包括操作系统执行系统调用等时间开销。

- **sys**: 表示在系统内核地址空间中执行代码期间占用的时间（以秒为单位）。这个时间包括内存分配，系统调用，以及数据I/O等开销。

现在让我们回过头来考察在提高或降低进程**nice**调整值前后，执行同一个命令时有何变化。下面的例子是在提高10个**nice**调整值时，运行**dd**命令的结果。

```
# time nice -n -10 dd if=/dev/rdisk/c0d0s9 of=/dev/null
32130+0 records in
32130+0 records out

real    0m8.95s
user    0m0.13s
sys     0m2.67s
#
```

下面的例子是在降低10个**nice**调整值时运行**dd**命令的结果：

```
# time nice -n 10 dd if=/dev/rdisk/c0d0s9 of=/dev/null
32130+0 records in
32130+0 records out

real    0m44.63s
user    0m0.14s
sys     0m2.72s
#
```

从上方两个以不同进程优先级运行**dd**命令的输出结果可以看出两点：一是在程序的实际执行时间方面并无太大的差别。这是可以理解的，因为执行的是同一个程序，其代码和处理内容完全一样。但两者在执行程序时耗费的全部时间方面差别却非常之大，这是因为具有不同优先级的进程在系统的进程调度方面得到的运行机会不同而造成的。进程的优先级越低，得到调度运行的机会就越少；反之，得到调度运行的机会就越多。

nice和**renice**命令的主要用途是降低一些运行时间较长，但又并非紧急事务的进程的优先级，使之能够以较低的优先级运行，提高系统对关键业务的响应速度，从而达到提高整体系统性能的目的。

8.7 定时运行系统任务和用户程序

UNIX系统提供两种以后台方式定时运行各种作业的机制，供用户提交定时的、自动执行的任务（包括系统命令、应用程序和Shell脚本等）。例如，在每天晚上的某一时间或每周的周末执行某一系统维护任务，每天按一定的时间间隔收集数据等。这种定时运行后台作业的功能是由**cron**守护进程与**crontab**文件，以及**at**等命令共同实现的。

8.7.1 cron守护进程的调度过程

cron守护进程负责管理和调度以**crontab**文件或**at**命令形式提交的后台作业，按照指定的日期和时间自动执行指定的命令。在系统启动过程中，可通过执行/etc/rc2.d目录中的**cron**脚本等方式，启动**cron**守护进程。然后，**cron**守护进程开始检测/var/spool/cron/crontabs目录中是否存在**crontab**文件，如果存在，**cron**守护进程将会执行下列任务：

- 检测是否存在新的crontab文件；
- 读取文件中指定的执行时间；
- 按指定的时间调度命令开始执行；
- 监听crontab命令更新crontab文件后发送的通知。

此外，cron守护进程还以同样的方式负责调度执行at作业。cron守护进程负责执行以at命令形式提交的作业，这些作业以文件的形式存储在/var/spool/cron/atjobs目录中。

为了提交定时和重复执行的例行任务（命令或Shell脚本），用户可以利用/var/spool/cron/crontabs目录中原有的crontab文件，在选定的文件中增加自己指定的命令或Shell脚本，也可以使用crontab命令，创建或提交自己的crontab文件。对于只需执行一次的命令或Shell脚本，可以使用at命令提交，使之按照指定的时间调度运行。

注意：请勿混淆crontab命令与所谓的crontab文件。实际上，UNIX系统仅提供一个crontab命令，但并不存在名字为crontab的文件。crontab命令用于创建、编辑、显示和删除/var/spool/cron/crontabs目录中的文件。所谓的crontab文件只是一种泛称，借指/var/spool/cron/crontabs目录中的所有文件。

cron守护进程仅在自己的进程初始化阶段，或者在运行crontab及at命令时考察crontab或at作业文件。这种实现方式可以避免cron守护进程不断检测crontab文件的变化，从而减少cron作业调度的系统开销。因此，为了提交定时执行的后台作业，正确的做法是利用crontab命令创建或编辑crontab文件，或利用at命令提交定时执行的作业。

如果直接利用编辑器，手工编辑原有的crontab文件，增加自己的定时执行任务时，除非重新运行cron守护进程或重新启动系统，否则不会执行新增加的任务。在此情况下，为了避免重新启动系统，又能使新增加的任务立即生效，可以利用kill命令，临时终止cron守护进程，然后再重新启动。在重启cron守护进程之前，要注意删除/etc/cron.d目录中的FIFO文件。这个过程可以使用下列Shell脚本实现：

```
# cat restartcron
pid=`ps -ef | grep cron | grep -v grep | awk '{print $2}'`
kill -9 $pid
if [ -f /var/cron.d/FIFO ]
then
    rm -r /etc/cron.d/FIFO
fi
/usr/sbin/cron &
#
```

除了人为因素，一经启动，cron守护进程将会无休止的连续运行。而且，整个系统只能运行一个cron进程——系统使用/etc/cron.d/FIFO文件作为封锁文件，防止用户执行多个cron进程。

cron守护进程通常会捕捉作业的标准输出和标准错误输出中的数据，并通过电子邮件方式，把输出结果回送给用户（如果作业不产生输出数据，则不发送电子邮件）。如果作业是利用at命令提交的，并且指定了“-m”选项，则不管作业是否产生输出，cron守护进程总是发送电子邮件。

为了保留一份cron守护进程执行过程的日志，必须在/etc/default/cron文件中指定CRONLOG=YES。如果CRONLOG=NO，cron守护进程不会记录任何日志。

在随后的两小节中，我们将详细介绍`crontab`文件和`at`命令，讨论如何实现定期、定时、自动执行日常的系統维护任务。这里再强调一次，`crontab`文件主要用于调度重复执行的任务，而`at`命令则仅仅用于提交一次性执行的任务。

8.7.2 调度定时重复执行的任务

利用`crontab`文件，用户可以调度运行各种日常的系統管理与维护任务，并使系統在指定的日期和时间重复执行。

`crontab`调度的日常管理和维护任务可以包括：

- 删除临时目录中的过期文件；
- 调整日志文件，淘汰过时的数据；
- 执行系統记账任务（参见`/var/spool/cron/crontabs/adm`文件）；
- 使用`df`和`ps`命令截取系統某一时刻运行状态的快照信息；
- 执行日常的安全监控等。

`crontab`按周调度的系統管理和维护任务可以包括下列内容：

- 执行系統备份；
- 运行“`fsck -n`”命令，检查磁盘文件系統可能存在的问题等。

`crontab`按月调度的系統管理和维护任务可以包括下列内容：

- 列出指定月内一直未用的用户文件；
- 生成每月的记账报告等。

另外，用户还可以利用`crontab`执行其他日常的系統管理和维护任务，定时备份数据库中的重要数据等，参见本章中的应用实例。

8.7.3 提交一次性定时执行的任务

`at`（或`batch`）命令使用户能够提交在指定的时间开始执行的作业。提交的作业可以是一个或一组命令，也可以是一个`Shell`脚本。类似于`crontab`，`at`命令允许用户调度执行一定的系統维护任务。但与`crontab`文件不同的是，`at`命令提交的任务仅在指定的时间执行一次。执行之后，立即从其作业目录中删除相应的文件。因此，如果需要一次性地定时执行任何命令或`Shell`脚本，`at`命令是非常有用的。注意，在利用`at`命令提交作业时，应把命令或`Shell`脚本的输出信息重定向并存储到一个文件中，以备将来考察。

`at`命令将会将`/var/spool/cron/atjobs`目录下以文件方式保存用户提交的命令或`Shell`脚本，以及用户当前工作环境的副本。

一经启动，`cron`守护进程即开始检测是否存在`at`作业，同时随时监听用户是否利用`at`命令提交了新的作业。在完成`at`作业之后，`cron`守护进程将会删除`atjobs`目录中相应`at`作业的文件。

在8.9节，我们将详细地说明怎样提交定时执行的`at`作业。

8.8 调度重复执行的任务

这一节开始详细介绍怎样创建、编辑、显示和删除`crontab`文件，讨论如何实现定期、定时、自动执行系統的日常管理与例行维护任务，以及怎样实现`crontab`文件的访问控制。

8.8.1 crontab的工作原理

UNIX系统提供的系统crontab文件通常均存储在/var/spool/cron/crontabs目录中，用户也可以在此目录中创建以自己的注册名命名的crontab文件，以便调度运行自己的处理任务。

cron守护进程每分钟考察一次利用各种crontab文件定义的后台作业，并按照每个crontab文件中的指令，定时执行指定的命令或Shell脚本。crontab文件通常由若干指令组成，每个指令包含6个字段，中间以空格作为分隔符，其语法格式如下：

```
minute hour day month week command
```

在crontab文件的指令中，前5个字段分别表示分、小时、日、月和周等时间信息。最后一个命令字段包含需要在指定的时间调度执行的命令或Shell脚本。这些信息告诉cron守护进程应当在什么时候执行指定的命令。根据前5个字段设定的日期和时间，系统能够自动地重复执行指定的命令或Shell脚本，以便完成各种系统维护任务。

在命令字段中，必要时可以同时指定多个命令，命令之间需加分号“;”分隔符。但每个指令必须由一个完整的逻辑文本行组成，即使文本行太长，需要占用多个物理行也是如此，中间不能有回车或换行字符。表8-7给出了前5个字段的说明。

表8-7 crontab文件中的时间字段

时间字段	取值范围
分	0~59
小时	0~23
日	1~31
月	1~12
周	0~7（0或7均表示星期日）

在crontab文件的每个时间字段中，还可以使用下列特殊字符，以提高时间定义的灵活性。

- 可以使用逗号并列多个数值，如“1,3,5,7,9”；
- 可以使用减号“-”指定一个数值范围，如“1-9”；
- 可以使用星号“*”通配符表示所有可能的合法数值；
- 可以在每一行的起始位置采用注释符“#”增加一个注释行或加一个空行。

例如，如果存在一个crontab文件，其中包含下列内容，即可在每个星期五的下午4时，在控制台上显示一条提示信息：

```
0 16 * * 5 echo "\n***** Timesheets Due *****" > /dev/console
```

/var/spool/cron/crontabs/sys是系统提供的一个crontab文件，其目的是定时收集系统性能的统计数据，供系统维护人员使用sar命令分析系统的性能，从中找出影响系统性能的原因。这个文件包含下列有效内容（每一行前面加的注视符“#”表示尚未启用）：

```
$ cat /var/spool/cron/crontabs/sys
.....
# 0 * * * 0-6 /usr/lib/sa/sa1
# 20,40 8-17 * * 1-5 /usr/lib/sa/sa1
# 5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
$
```


第一行表示在每天每小时的整点时间运行一次/usr/lib/sa/sa1脚本，收集系统活动数据，并存储到/var/log/sa/sadd文件中（其中的dd为01~31范围内的一个数字，表示当天的日期）；第二行表示在每周一至周五上午8:00至下午5时的工作时间内，按照指定的时间间隔（20分钟）运行/usr/lib/sa/sa1脚本，收集系统活动数据；第三行表示在每周一至周五的18时5分开始运行/usr/lib/sa/sa2脚本（也即以“-s 8:00 -e 18:01 -i 1200 -A”以及/var/log/sa/sadd文件作为参数，运行sar命令，把运行结果写入/var/log/sa/saradd文件中）。

8.8.2 创建和编辑crontab文件

在实际应用过程中，最简单、最正确的方法是利用“crontab -e”命令创建或编辑crontab文件。这个命令将会调用由EDITOR环境变量设定的文本编辑器。如果事先没有设置这个环境变量，crontab命令将会调用默认的编辑器ed。如果直接使用编辑器，而不用crontab命令创建或编辑crontab文件，除非重新启动UNIX系统，否则cron守护进程无从知道crontab文件是否发生变动，提交的后台作业也不会立即生效。

通常，无需具有超级用户的访问权限，任何用户都可以利用“crontab -e”命令，创建或编辑自己的crontab文件。如果crontab文件存在，则调用编辑器，打开现有的文件。如果crontab文件不存在，在写入并退出编辑器之后，创建一个新的crontab文件。创建的文件自动存储在/var/spool/cron/crontabs目录中，并以用户的注册名命名。为了采用vi编辑器，可以事先定义EDITOR变量。示例如下：

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

如果是超级用户，可以使用下列命令形式，为自己或其他用户创建或编辑一个crontab文件。注意，只有拥有超级用户的特权，才能创建或编辑其他用户的crontab文件。如果未指定用户名，则创建或编辑自己的crontab文件，并以root命名。例如：

```
# crontab -e [-u username]
```

输入上述命令之后，crontab将会调用事先设定的编辑器。然后即可利用编辑命令，按照crontab文件的语法格式，把新增的作业加到crontab文件中。

下面以运行MySQL数据库备份的mysqldump命令为例，说明怎样在/var/spool/cron/crontabs目录中创建一个以gqxing用户身份运行的crontab文件。首先输入下列crontab命令：

```
$ crontab -e
```

然后利用编辑命令，把下列内容（其中第一行为注释）加到打开的临时文件中，使系统能够在每天的午夜23:55，利用mysqldump命令自动备份数据库、数据库表及其业务数据：

```
# Backup MySQL database at 23:55 everyday.
55 23 * * * gqxing mysqldump --add-drop-table -u gqxing -p1b2c3 books > /tmp/
books.`date +%m%d` >> /export/home/gqxing/backup/backup.log 2>&1
```

保存文件，退出编辑器。最终将会在/var/spool/cron/crontabs目录中创建一个名为gqxing的文件。

mysqldump命令后面的“>> /export/home/gqxing/backup/backup.log 2>&1”意味着把其标准输出及标准错误输出重定向到backup.log日志文件中，从而把mysqldump命令的任何输出信息

均记录下来，以备在上班时间随时查阅。

8.8.3 显示crontab文件

为了查询系统中是否存在用户提交的crontab文件，可以利用ls命令，列出/var/spool/cron/crontabs目录中的文件。例如，下列输出信息表明系统存在一个属于用户gqxing的crontab文件（其他文件是系统提供的crontab文件）：

```
$ ls -l /var/spool/cron/crontabs
total 12
-rw----- 1 root sys 190 Aug 25 2008 adm
-rw----- 1 root other 188 Jul 15 17:58 gqxing
-r----- 1 root root 452 Aug 25 2008 lp
-rw----- 1 root sys 482 Jun 12 17:42 root
-rw----- 1 root sys 308 Aug 25 2008 sys
-r----- 1 root sys 404 Jun 12 17:40 uucp
$
```

任何用户均可使用“crontab -l”命令显示或检查属于自己的crontab文件。如同cat等命令能够显示任何文本文件的内容一样，“crontab -l”命令能够专门显示crontab文件的内容。且在输入这个命令时，无需指定crontab文件的路径名。

下面的例子说明了怎样使用“crontab -l”命令显示当前用户gqxing自己的crontab文件：

```
$ crontab -l
# Backup MySQL dadabase at 23:55 everyday.
55 23 * * * mysqldump --add-drop-table -u gqxing -pitsme books > /tmp/
books.`date '+%m%d'` >> /home/gqxing/backup/backup.log 2>&1
$
```

如果是超级用户，还可以利用下列命令形式，显示自己或其他用户的crontab文件。注意，只有具有超级用户的特权，才能查询其他用户的crontab文件。如果未指定用户名，则显示自己的crontab文件内容：

```
# crontab -l [username]
```

下面的例子说明了超级用户怎样显示其他用户的crontab文件：

```
# crontab -l gqxing
# Backup MySQL dadabase at 23:55 everyday.
59 23 * * * mysqldump --add-drop-table -u gqxing -pitsme books > /tmp/
books.`date '+%m%d'` >> /home/cathy/backup/backup.log 2>&1
#
```

8.8.4 删除crontab文件

通常，/var/spool/cron/crontabs目录的访问权限不允许普通用户直接删除crontab文件。为了删除自己的crontab文件，可以采用下列命令：

```
$ crontab -r
```

上述命令只能删除属于自己的crontab文件，且无需指定crontab文件的路径名。如果是超级用户，可以利用下列命令，删除自己或其他用户的crontab文件。注意，只有具有超级用户的特

权，才能删除其他用户的crontab文件。如果未指定用户名，则删除自己的crontab文件：

```
# crontab -r [username]
```

下面的例子说明了怎样使用“crontab -r”命令删除gqxing用户自己的crontab文件，然后使用“crontab -l”命令，验证crontab文件已经删除：

```
$ crontab -r
$ crontab -l
crontab: can't open your crontab file.
$
```

8.8.5 crontab命令的访问控制

UNIX系统通过/etc/cron.deny文件或选用的/etc/cron.allow文件限制用户使用crontab命令。只有这两个文件认可的用户才能够使用crontab命令创建、编辑、显示和删除自己的crontab文件。如果需要，超级用户可以设置cron.deny或cron.allow文件，限制或允许指定的用户使用crontab命令。注意，只有超级用户才能够创建或访问cron.deny以及cron.allow文件。

cron.deny和cron.allow文件可以包含一系列用户名，每个用户名占用一行。两个访问控制文件共同作用的结果如下：

- 如果cron.allow文件存在，只有这个文件中列出的用户才能够创建、编辑、显示和删除crontab文件。
- 如果cron.allow文件不存在，除了cron.deny文件中列举的用户之外，其他用户均可创建、编辑、显示和删除crontab文件。
- 如果cron.allow或cron.deny文件均不存在，任何用户都能够运行crontab命令。

初始安装UNIX系统之后，系统通常仅提供/etc/cron.deny而不提供/etc/cron.allow文件。而且，/etc/cron.deny文件仅包含少量的系统管理用户，按照上述说明，这意味着大部分用户均可使用crontab命令。如果需要，超级用户可以在/etc/cron.deny文件中增加用户名单，禁止其使用crontab命令，因而也就禁止其提交后台作业。

之后，除了cron.deny文件中列出的用户之外，其他任何用户均可运行crontab命令。如果创建了cron.allow文件，只有其中指定的用户才能够运行crontab命令。

因此，为了禁止某些用户运行crontab命令，可以直接编辑cron.deny文件，增加禁止使用crontab命令的用户名单（每个用户名占一行）。例如，下面给出的cron.deny文件表示，除了原有的系统用户之外，也不允许guest用户使用crontab命令：

```
# cat /etc/cron.deny
daemon
bin
nuucp
listen
nobody
noaccess
guest
#
```

为了明确指定能够提交crontab作业的用户，也可以创建一个cron.allow文件，首先把用户名root加到cron.allow文件中（如果不把root加到cron.allow文件中，超级用户访问crontab命令时也

会遭到拒绝)；然后再增加其他用户名，每行一个，加入的用户即可使用**crontab**命令。

为了验证普通用户是否能够访问**crontab**命令，可以注册到相应的用户，然后执行下列**crontab**命令：

```
$ crontab -l
```

如果用户能够访问**crontab**命令，而且已经创建了**crontab**文件，上述命令将会显示出文件中的内容。否则，如果相应的**crontab**文件不存在，系统将会输出下列类似的错误信息：

```
$ crontab -l
crontab: can't open your crontab file.
$
```

上述信息说明，当前的用户或者位于**cron.allow**文件，或者不在**cron.deny**文件列举的名单中，但其**crontab**文件不存在。

如果用户无权运行**crontab**命令，不管相应的**crontab**文件是否存在，将会显示下列出错信息，意味着用户或者位于**cron.deny**文件中，或者不在**cron.allow**文件列举的名单中：

```
$ crontab -l
crontab: you are not authorized to use cron. Sorry.
$
```

8.8.6 应用实例——数据库定时备份

在MySQL数据库应用中，为了能够在每天的指定时间定时备份数据库中的数据，可以利用**cron**机制，创建自己的**crontab**文件（也可以在**/var/spool/cron/crontabs**目录现有的文件中增加一个数据库备份任务）。

例如，假定有一个MySQL数据库应用项目，需要在每天的晚上11:55，利用MySQL数据库的**mysqldump**命令备份一次数据。考虑到数据非常重要，系统中至少应保留7天的备份数据。

因此，我们可以利用**crontab**命令，以用户**gqxing**的名义创建一个**crontab**文件，在文件中增加一个定时执行MySQL数据库备份的Shell脚本，以便在每天晚上11:55执行一次数据备份：

```
55 23 * * * /export/home/gqxing/script/sqldump > /dev/null 2>&1
```

其中，**sqldump**是一个Shell脚本，其功能是以**gqxing**用户的身份，执行**mysqldump**命令，把**gqxing**用户的数据库、数据库表及其业务数据备份到**books.mmdd**文件中。然后，检查备份文件的数量，如果超出7个文件，则删除早期的备份文件。**sqldump**脚本的内容如下：

```
$ cat /home/gqxing/script/sqldump
#!/bin/bash
BAKDIR=/export/home/gqxing/backup
mysqldump --add-drop-table -u gqxing -pitsme books > $BAKDIR/books.`date
'+%m%d'`
amount=`ls -l books* 2>/dev/null | wc -l`
if [ "${amount}" -gt 7 ]
then
    fname=`echo books*`
    set $fname
    mv -r $1
fi
exit 0
$
```

采用上述备份方法之后，将会在/home/gqxing/backup目录中保留7个数据库备份文件，例如：

```
$ cd /export/home/gqxing/backup
$ ls -l
total 42
-rw-r--r--  1 gqxing  other      2760 Jul 15 23:55 books.0715
-rw-r--r--  1 gqxing  other      2760 Jul 16 23:55 books.0716
.....
-rw-r--r--  1 gqxing  other      2760 Jul 21 23:55 books.0721
$
```

8.9 调度一次性执行的作业

这一节开始介绍怎样使用at或batch命令执行下列任务：

- 提交at作业（命令或Shell脚本），使之在某个指定时间开始执行；
- 显示和删除已经提交的at作业；
- 控制用户是否能够使用at命令提交定时作业。

at命令的语法格式简写如下：

```
at [-mld] time [date]
```

其中，“-m”选项表示，一旦作业执行之后立即向用户发送电子邮件。**time**可以是1、2或4位数字，以时分形式（HHMM或HH:MM）指定作业开始运行的时间。如果指定的时间为整点时间（1或2位数字），分可以省略。如果按12小时制指定时间，时间后面应附加am或pm。其他可接受的关键字是midnight、noon和now。**date**是以、月、日年（MMDDYY、MM/DD/YY或MM.DD.YY）、“月名 日（如June 1）”、星期几（如Monday）、关键字today或tomorrow等表示的日期。为了简化输入，也可以使用月、星期几或其他特殊关键字的前三个字符。

通常，任何用户均可以创建、显示和删除自己的at作业文件，但只有超级用户才有权访问其他用户的at作业文件。当利用at或batch命令提交一个at作业之后，系统将会以文件的形式保留提交的at作业，并存储在/var/spool/cron/atjobs目录中，由cron守护进程负责处理以at或batch命令形式提交的作业。例如：

```
# ls -l /var/spool/cron/atjobs
total 8
-r-Sr--r--  1 cathy    other      537 Jul 15 18:24 1247655600.a
-r-Sr--r--  1 root     root       2894 Jul 15 18:29 1247725800.a
#
```

at作业文件采用一串数字加一个“.a”或“.b”扩展名命名，其中数字表示作业在at作业队列中的位置，文件名后缀表示作业的类型，其中“a”表示利用at命令提交的作业，“b”表示利用batch命令提交的作业。

8.9.1 提交at作业

提交at作业包括三个要素：输入at命令；按照at命令的语法要求指定作业执行的时间；输入准备执行的命令或Shell脚本。

为了提交一个at作业，可在系统命令提示符下输入at命令，同时指定作业的执行时间，按下Enter键；在at命令提示符“at >”下，输入准备执行的命令或Shell脚本。在输入命令或Shell脚本之后，按下Ctrl-D键，即可提交at作业。

例如，下列命令提交的at作业将会在当天晚上的6:30删除当前用户gqxing主目录中的core文件：

```
$ at 1830
at> rm -r /export/home/gqxing/core > /dev/null 2>&1
at> <EOT>                                     (即按下Ctrl-D键，下同)
commands will be executed using /bin/ksh
job 1247653800.a at Wed Jul 15 18:30:00 2009
$
```

如果希望同时输入多个命令或Shell脚本，每个命令或Shell脚本应占用一行，以Enter键结束。在输入所有命令或Shell脚本之后，按下Ctrl-D键，结束at命令的输入，从而完成at作业的提交。

例如，下列命令提交的at作业将会在11月30日午夜进入/export/home/gqxing/backup目录，压缩其中的所有数据文件：

```
$ at 11:59 pm nov 30
at> cd /export/home/gqxing/backup
at> bzip2 data*
at> <EOT>
commands will be executed using /bin/ksh
job 1259596740.a at Mon Nov 30 23:59:00 2009
$
```

当需要提交一个作业而不关心其究竟何时执行时，可以使用batch命令。下面仍以上述的文件压缩为例，说明怎样利用batch命令提交一个at作业，使之压缩/export/home/gqxing/backup目录中的数据文件，同时把开始执行时间和压缩完成时间等信息保存到一个日志文件bzip2.log中：

```
$ batch
at> cd /export/home/gqxing/backup
at> date >> bzip2.log
at> bzip2 data*
at> date >> bzip2.log
at> <EOT>
commands will be executed using /bin/ksh
job 1247652640.b at Wed Jul 15 18:10:40 2009
$
```

此外，还可以利用Here文档（参见第7章），提供at或batch命令需要的输入数据。示例如下：

```
$ batch <<|
> cd /export/home/gqxing/backup
> date >> bzip2.log
> bzip2 data*
> date >> bzip2.log
> |
commands will be executed using /bin/ksh
```



```
job 1247652805.b at Wed Jul 15 18:13:25 2009
```

```
$
```

注意：如果执行命令或Shell脚本的输出信息很重要，可以使用I/O重定向的方式把输出信息写到一个文件中，以便将来能够查阅。

8.9.2 显示at作业及作业队列

为了查询已经创建的，目前仍然位于at队列中的作业，可以直接访问/var/spool/cron/atjobs目录，也可以使用atq或“at -l”命令。

在下列例子中，atq命令列出了用户已经提交的，目前尚未执行，仍然位于at队列中的作业，以及at作业的执行时间信息：

```
$ atq
Rank      Execution Date      Owner      Job      Queue      Job Name
1st       Jul 15, 2009 19:00    gqxing     1247655600.a    a          stdin
2nd       Jul 16, 2009 14:30    root       1247725800.a    a          stdin
$
```

8.9.3 删除at作业

在at作业尚未执行之前，无需具有超级用户的特权，任何用户均可以使用下列命令，从队列中删除自己的at作业。其中，“at-job”表示at作业的文件名。

```
$ atrm at-job
```

或

```
$ at -d at-job
```

如果是超级用户，可以利用同样的命令，删除自己或其他用户的at作业。注意，只有具有超级用户的特权，才能删除其他用户的at作业。如果未指定用户名，则删除自己的at作业。

在下列例子中，假定我们准备删除预定在7月16日14:30执行的at作业。首先可以使用atq命令显示at作业队列，然后使用atrm命令从at作业队列中删除指定的作业。最后可以使用atq命令，检验指定的at作业已经删除（删除的at作业不应再出现）。

```
$ atrm 1247655600.a
$ atq
Rank      Execution Date      Owner      Job      Queue      Job Name
1st       Jul 16, 2009 14:30    root       1247725800.a    a          stdin
$
```

8.9.4 at命令的访问控制

UNIX系统通过/etc/at.deny文件或选用的/etc/at.allow文件限制用户使用at系列命令。只有这两个文件认可的用户才能够使用at系列命令提交自己的at作业。如果需要，超级用户可以设置at.deny或at.allow文件，限制或允许指定的用户使用at系列命令。注意，只有超级用户才能够访问at.deny和at.allow文件。

at.deny和at.allow文件与at系列命令，犹如crontab.deny和crontab.allow文件与crontab命令，其组合作用的结果可以限定用户是否能够提交at作业等。

安装UNIX操作系统之后，系统提供的`at.deny`文件包含部分系统管理用户。这意味着超级用户与任何普通用户均可使用`at`系列命令，创建、删除或显示自己的`at`作业及其队列信息。必要时超级用户可以编辑这个文件，增加禁止使用`at`系列命令的用户名单，限制其使用`at`系列命令，禁止其提交`at`作业。

UNIX系统通常不提供`at.allow`文件。因此，在安装UNIX系统之后，除了`at.deny`文件中列出的用户之外，其他任何用户均可以提交`at`作业。如果创建了`at.allow`文件，则只有其中指定的用户才能够提交`at`作业。

`/etc/at.deny`文件由一系列用户名组成，每个用户占用一行。`at.deny`文件中列举的用户不能使用`at`系列命令提交`at`作业。下面是经过修改之后的`at.deny`文件，其中除了原有的系统管理用户，又增加了一个`guest`，表示该用户不能再使用`at`系列命令提交定时执行的作业：

```
# cat /etc/at.deny
daemon
bin
nuucp
listen
nobody
noaccess
guest
#
```

为了验证新加入`/etc/at.deny`文件中的用户是否能够使用`at`系列命令，可以使用`su`命令改变用户身份，然后运行`atq`命令，系统将会输出下列信息：

```
$ su - guest
Password:
$ atq
atq: you are not authorized to use at.  Sorry.
$
```

同样，如果试图提交一个`at`作业，其结果如下，说明`guest`属于限制的用户：

```
$ at 10:30 am Saturday
at: you are not authorized to use at.  Sorry.
$
```

8.9.5 应用实例——系统定时关机

为了在指定的时间关机，以应对用电高峰期间的停电或紧急系统维护，我们可以使用`wall`命令提前向注册的用户发出停机通知，例如：

```
# wall <<!  
> THE SYSTEM iscas WILL BE SHUT DOWN AT 14:30 ! ! !  
> Please log off ASAP or risk your files being damaged.  
> !  
Broadcast Message from root (pts/4) on iscas Tue Jun 16 12:20:57...  
THE SYSTEM dbserver WILL BE SHUT DOWN AT 14:30 ! ! !  
Please log off ASAP or risk your files being damaged.  
#
```

然后利用`at`命令，按照预定的时间关闭系统，例如：

```
# at 1430
at> init 0
at> <EOT>
commands will be executed using /bin/ksh
job 1245157200.a at Tue Jun 16 14:30:00 2009
#
```

输入“init 0”命令，接着按下Enter键和Ctrl-D键之后，将会看到/var/spool/cron/atjobs目录中增加了一个at作业文件：

```
# ls -l /var/spool/cron/atjobs
total 6
-r-Sr--r--  1 root    root      2894 Jun 16 12:21 1245157200.a
#
```

查阅上述文件可以看到，at作业文件主要包括三部分内容。其中前四行说明这是一个at作业，完成作业之后是否需要利用电子邮件通知用户等；从第5行开始的第二部分主要是环境变量设置；第三部分是需要执行的实际命令，包括最后一行的init命令，示例如下：

```
# cat /var/spool/cron/atjobs/1245157200.a
: at job
: jobname: stdin
: notify by mail: no
: project: 1
export _; _='/usr/bin/at'
.....
export LANG; LANG='C'
.....
export PATH; PATH='/usr/sbin:/usr/bin:/usr/dt/bin:/usr/openwin/bin:/usr/ucb'
.....
export EDITOR; EDITOR='vi'
.....
export LOGNAME; LOGNAME='root'
.....
export USER; USER='root'
export DISPLAY; DISPLAY=':0.0'
export SHELL; SHELL='/bin/ksh'
.....
$SHELL << '...the rest of this file is shell input'
#ident "@(#)proto 1.6 00/05/01 SMI" /* SVr4.0 1.2 */
cd /
umask 22
ulimit unlimited
init 0
#
```

第9章 用户管理

本章主要介绍UNIX系统的用户管理，讨论怎样增加、修改以及删除用户与用户组，介绍与用户管理有关的系统文件，说明如何设置用户的运行环境。

用户与用户组是UNIX系统管理的一个重要部分，也是系统安全的基础。在UNIX系统中，所有的文件、程序或正在运行的进程都从属于一个特定的用户。每个文件和程序都具有一定的访问权限，用于限制不同用户的访问行为。作为系统管理员，管理用户和用户组是一项重要的任务，有助于防止用户越权访问与其身份不相符的文件，执行系统任务，对系统造成破坏。总之，如果没有有效的用户管理，就无法保证系统的安全。

9.1 增加与删除用户

在UNIX系统中，每个用户都具有一个唯一的身份标识，称做用户ID（或简称UID），以区别于其他用户。UNIX系统按照一定的原则把用户划分为用户组，以便相关的同组用户之间能够共享文件。

一般地讲，UNIX系统中的用户可以分为三类：超级用户（root）、管理用户和普通用户。但也可以把超级用户和管理用户通称为系统用户。

超级用户是一个特殊的用户（其用户标识号为0），拥有至高无上的访问权限，可以访问任何程序和文件。任何系统都会自动提供一个超级用户账号。

管理用户用于运行一定的系统服务程序，支持和维护相应的系统功能。这些用户的ID号位于1~99的范围之内。例如，lp就是一个管理用户，用于管理打印机，为用户提供打印服务。

除了系统用户之外，其他均为普通用户。在访问UNIX系统之前，每个用户都需要拥有一个用户账号。因此，系统管理员需要为每个普通用户事先分配一个注册账号，把用户名及其他有关信息加到系统中。在利用用户名和密码成功地注册之后，用户才能访问系统提供的资源和服务，执行系统命令，开发和运行应用程序，以及访问数据库等。

对属于自己的文件，用户拥有绝对的权力，可以赋予自己、同组用户或其他用户访问文件的权限。例如，允许同组用户共享自己的文件，其他用户只能阅读或执行，但不能写文件等。

9.1.1 /etc/passwd文件

在安装UNIX系统之后，系统已经事先创建了若干系统用户账号，其中包括超级用户root与管理用户bin、sys以及adm等，用于执行不同类型的系统管理和日常维护任务。

UNIX系统中的用户账号信息是由/etc/passwd和/etc/shadow文件共同维护的。在这两个文件中，每个用户都有一个相应的记录。当利用控制台等终端注册，按照系统的提示输入用户名和密码之后，系统将会根据用户提供的用户名检查/etc/passwd文件，然后根据用户提供的密码，利用同一加密算法加密后再与/etc/shadow文件中的密码字段进行比较，同时检查其他诸如密码有效期等字段。如果通过了验证，按照passwd文件指定的主目录和命令解释程序，用户即可进入自己的主目录，通过命令解释程序等界面访问UNIX系统。

除了用户名和密码之外，每个用户还具有用户ID、用户组ID、主目录和命令解释程序等相关信息。这些信息分别存放在passwd和shadow文件中。

passwd文件中包含UNIX系统中除密码之外的主要用户信息，每个用户信息占用一行，每一行由7个字段组成，字段之间以冒号“:”作为分隔符。passwd文件的格式定义如下：

```
username:password:uid:gid:comment:home_dir:login_shell
```

表9-1给出了passwd文件中的每个字段及其简单说明。

表9-1 /etc/passwd文件

字段	简单说明
username	注册用户名。由2~8个字母（A~Z，a~z）、数字（0~9）、句点“.”、下画线“_”和连字符“-”等字符组成。在UNIX系统中，用户名必须是唯一的，其中至少必须包含1个小写字母，且第一个字符应选用字母
password	这个字段原为用户密码，但密码现已移至/etc/shadow文件。因此，如果用户设有密码，这个字段将会包含一个小写字母“x”，加密后的密码存储在shadow文件中。如果这个字段为星号“*”，表示相应的用户无法正常地注册到系统
uid	用户ID（或称用户标识）。用户ID是系统分配给每个用户的唯一数字标识，是系统识别每个用户的主要手段。当系统需要了解用户信息（如账号字段的内容）时，通常均以用户ID作为索引，检索passwd文件。用户ID是一个32位的无符号整数，最大值为2147483647。其中0~99保留为系统用户使用，自定义的普通用户ID应位于100~60000范围之内。考虑到与其他系统的兼容性，建议使用16位无符号整数的最大值32 767作为用户ID的上限
gid	用户组ID（或称用户组标识）。UNIX系统中的每个用户均应属于某个用户组，每个用户组除有组名之外，也有一个相应的用户组ID。同样，ID号0~99保留为系统用户组使用
comment	注释信息。通常包含用户全名、电话号码以及电子邮件地址等用户信息
home_dir	用户主目录（全路径名）。用户主目录是分配给用户的一个子目录，供用户存储个人文件用，也是用户注册后的起始工作目录。通常，UNIX系统采用/home/username形式的用户主目录结构（环境变量HOME就是按这个字段设置的），其中username为注册的用户名。注意，Solaris系统采用/export/home/username形式的用户主目录结构
login_shell	指定用户注册后调用的Shell，也即命令解释程序。如果这个字段为空，默认的命令解释程序为/bin/sh，即Bourne Shell。用户也可以根据自己的爱好，选用其他命令解释程序，如Korn Shell (/bin/ksh) 等

下面是UNIX系统提供的初始用户信息：

```
$ cat /etc/passwd
root:x:0:0: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:
.....
$
```

9.1.2 /etc/shadow文件

/etc/shadow是一个限制普通用户访问的系统文件，其中存有加密形式的密码以及其他相关

信息。与passwd文件相对应，shadow文件中的每个用户密码信息占用一行，每一行由9个字段组成，中间以冒号“:”作为分隔符。其文件格式定义如下：

```
username:password:lastchanged:mindays:maxdays:warn:inactive:expire:reserve
```

表9-2给出了shadow文件中的每个字段及其简单说明。

表9-2 /etc/shadow文件

字段	简单说明
username	用户注册名。参见/etc/passwd文件
password	加密形式的密码（由crypt(3)函数生成）。通常，用户输入的密码至少应包含6个字符（参见/etc/default/passwd文件中的PASSLENGTH变量设置），但生成的密码较长（至少包含13个字符）。如果这个字段为空或“NP”，意味着相应的用户没有设置密码。如果其前4个字符为“*LK*”，表示用户账号已经封锁，相应的用户不能再访问系统。如果为“NONE”，当再次注册时，系统将会要求用户设置密码
lastchanged	从1970年1月1日开始算起，直至最后一次修改密码之日的天数
mindays	保持密码稳定不变的最小天数，仅当超过此限，才能修改密码。这个字段必须大于等于0，才能启用密码有效期检查
maxdays	保持密码有效的最大天数。超过此限，系统将会强制提请用户更换新的密码
warn	指定在密码有效期到期之前需提前多少天向用户发出警告信息
inactive	指定在密码有效期到期之后一直不访问系统，但仍保证其账号信息有效的最多天数，超过此限将封锁用户账号。/var/log/lastlog文件存有用户最后一次访问系统的时间记录
expire	指定用户账号有效期的截止日期。到期后，账号将会自动失效，用户无法再注册到系统
reserve	保留字段

下面是UNIX系统提供的初始shadow文件内容：

```
# cat /etc/shadow
root:nV4B1C6sdbcfU:6445:::
daemon:NP:6445:::
bin:NP:6445:::
sys:NP:6445:::
adm:NP:6445:::
lp:NP:6445:::
.....
#
```

9.1.3 用户管理实例

在UNIX系统中，为了管理用户和用户组，可以利用系统提供的管理工具，增加、修改和删除用户。也可以使用各种基本命令，实现用户和用户组的管理与维护。下面以命令方式为例，说明怎样增加、修改和删除用户。

1. 增加新用户

在命令行方式中，为了增加用户，可以使用useradd命令，其语法格式简写如下：

```
useradd [-u uid] [-g group] [-d home_dir] [-s shell] [-c comment]
        [-m [-k skel_dir]] [-f inactive] [-e expire] login
```


注意：在增加、修改或删除用户时，如果采用编辑器，手工修改passwd和shadow文件，必须考虑以下因素或处理步骤：

- 用户属于哪个用户组，相应的用户组是否存在。
- 除了修改passwd文件之外，还要在/etc/shadow文件中增加、修改或删除相应的用户项。
- 创建和删除用户主目录。
- 创建或复制用户初始化文件，如.profile等文件（取决于选定的命令解释程序）。

2. 修改用户账户信息

修改用户信息时，可以利用编辑器，以手工方式直接编辑passwd和shadow文件，也可以使用usermod命令。usermod命令的语法格式简写如下（命令选项的使用说明参见表9-3）：

```
usermod [-u uid] [-g group] [-d home_dir] [-s shell] [-c comment] [-f inactive]
        [-e expire] login
```

例如，为了把用户cathy的命令解释程序sh更换为Korn Shell，可以使用下列命令：

```
# usermod -s /usr/bin/ksh cathy
#
```

除非用户名或用户ID与其他用户冲突，一般情况下不要轻易修改用户账号中的用户名和用户ID，因为这将涉及到用户已经创建的所有文件和目录。否则，需要同时修改用户所有文件和目录的文件属主等属性。但可以放心地修改用户账号其他字段的有关信息，例如：

- 注释字段；
- 命令解释程序；
- 密码及密码的各种属性（shadow文件）；
- 用户主目录及主目录的访问权限。

此外还要注意，除了超级用户root的密码之外，不要试图修改和删除系统用户账号的任何信息。

此外，还可以使用passmgmt命令，修改用户帐号的属性信息（也可用于增加或删除用户帐号，但不会创建或删除有关的目录），其语法格式简写如下：

```
passmgmt -a | -m [-c comment] [-e expire] [-f inactive] [-g gid]
        [-h homedir] [-k skel_dir] [-s shell] [-u uid] login
passmgmt -d login
```

其中，“-a”选项表示增加一个新用户，“-m”选项表示修改用户账号的指定属性，“-d”选项表示删除指定的用户账号。其他选项的意义参见表9-3。

3. 删除用户账户

一旦用户不再需要访问UNIX系统，理应从系统中删除其账号信息。删除用户账号时可以使用userdel命令，也可以采用手工编辑的方式。使用userdel命令的好处是，只需一个命令即可删除passwd、shadow和group文件中的相应用户和用户组信息，同时还会把用户主目录中的所有文件和目录一同删除。如果采用手工方式，既需要编辑passwd、shadow或group文件，还要删除用户主目录及其中的文件。userdel命令的语法格式如下：

```
userdel [-r] login
```

其中，“-r”选项意味着同时从系统中删除用户的主目录，包括其中的文件和子目录。

4. 封锁用户账号

有时，也许需要临时或永久禁止或者封锁某个用户账号，禁止其注册。为此，可以使用 `passwd` 命令，或者直接修改 `/etc/shadow` 文件，在用户密码字段的前部增加一个 “*LK*” 字符串，以封锁相应的用户账号，从而防止用户再利用此账号注册到系统中。另外一种方法是利用 `usermod` 或 `passmgmt` 命令，修改 `shadow` 文件中的 `expire` 字段，把其中的日期改成一个过时的日期。

`passwd` 命令的语法格式简写如下：

```
passwd -s [-a] [login]
passwd [-l | -u] [-f] [-n min] [-w warn] [-x max] login
```

其中，`login` 是注册的用户名。表9-4给出了 `passwd` 命令的部分选项及其说明（除非特别说明，绝大部分选项仅限于超级用户使用）。

表9-4 `passwd` 命令的部分选项

选项	简单说明
-a	这个选项只能与 “-s” 一起使用，用于查询所有用户账号的状态信息
-f	使指定用户的密码立即失效，强制用户在下一次注册时立即更换密码
-l	封锁指定用户的密码（在加密形式的密码字段中增加一个 “*LK*” 前缀），也即封锁指定用户的注册账号，拒绝相应用户再次注册。当某用户离开公司，有关文件尚未完全交接时，可以使用此选项先封锁用户账号，待清理完毕之后再删除用户账号信息
-n min	指定密码保持不变的期限——即在更换密码之前必须保持密码不变的最少天数。换言之，即指定多少天之内不能更改密码（相当于设置 <code>shadow</code> 文件中的 <code>mindays</code> 字段）
-s	显示指定用户的密码状态属性
-u	解除被封锁的用户注册账号
-w warn	指定在密码有效期到期之前的多少天向用户发出警告信息（相当于设置 <code>shadow</code> 文件中的 <code>warn</code> 字段）。如果已禁止检查密码有效期（ <code>shadow</code> 文件中的 <code>mindays</code> 字段为0），这个选项没有意义。默认值是有效期截止日的前一天
-x max	指定密码的最大有效期限，以天数计算（相当于设置 <code>shadow</code> 文件中的 <code>maxdays</code> 字段）。如果已禁止检查密码有效期（ <code>shadow</code> 文件中的 <code>mindays</code> 字段为0），这个选项（或字段）没有意义

例如，为了封锁 `guest` 的用户账号，可以使用下列命令：

```
# passwd -l guest
passwd: password information changed for guest
#
```

5. 定期更改密码

普通用户可以使用不带任何选项和参数的 `passwd` 命令修改自己的密码，而超级用户可以利用 `passwd` 命令的大量选项及参数维护系统中的用户账号信息。例如，利用密码的有效期设置，可以强制用户定期修改自己的密码。

例如，为了强制用户 `cathy` 下一次注册时立即更换密码，可以使用下列命令：

```
# passwd -f cathy
passwd: password information changed for cathy
#
```

为了查询用户gqxing的密码属性，可以使用下列命令：

```
# passwd -s gqxing
gqxing      LK
#
```

在上述输出信息中，LK表示gqxing的用户账户仍处于封锁状态。其他密码属性包括NP，表示未设置密码，PS表示已经设置了密码等。

9.2 定制用户的工作环境

除了分配用户名和用户ID，设定用户主目录，供用户创建和存储文件之外，还需要有一个工作环境，使用户能够借助系统提供的各种工具和资源，访问系统，开发和运行自己的应用程序。在注册到系统之后，用户的工作环境是由选定的命令解释程序和相应的用户初始化文件确定的。因此，用户管理的另外一个任务是选择作为用户界面的命令解释程序和用户初始化文件。

9.2.1 选择命令解释程序

UNIX系统配备的标准命令解释程序是sh（即Bourne Shell），但同时也支持C Shell（csh）、Korn Shell（ksh）、Bash（bash）、TC Shell（tcsh）以及Z Shell（zsh）等，用户可以自由选用。每个Shell各具特色，用户可以根据自己的需要，选用不同的命令解释程序。为了了解UNIX系统支持的命令解释程序，可以查询/etc/shells文件，或者直接查询/bin或/usr/bin目录。

下面简单介绍UNIX系统中广泛应用的，且最具代表性的6个Shell，供用户在选用命令解释程序时参考。

1. Bourne Shell

Bourne Shell是贝尔实验室的Steven R. Bourne于1975年开发的，并随UNIX系统第7版一同推出。Bourne Shell是为UNIX系统正式配备的第一个命令解释程序，具有使用简单、方便，编程功能丰富，资源开销小，执行速度快的特点。而Borune Shell的不足之处主要在于人机交互能力方面。Bourne Shell的设计目标是为UNIX系统提供一个用户界面和强有力的编程环境。从诞生之日起，Bourne Shell即成为UNIX的基本配置和重要组成部分之一，成为各种UNIX版本中必备的标准命令解释程序，其他Shell大都是在Bourne Shell的基础上发展起来的。

由于其历史地位，Bourne Shell也是应用最为广泛的命令解释程序。超级用户和管理用户均采用Bourne Shell作为其命令解释程序，如果没有明确指定，普通用户的默认命令解释程序也为Bourne Shell。实际上，UNIX系统的许多日常管理和维护任务也都是利用Bourne Shell脚本实现的。

2. C Shell

C Shell原为BSD版UNIX系统的命令解释程序，由加州大学伯克利分校计算机系的Bill Joy开发。C Shell的语法格式类似于C语言，故称为C Shell。C Shell的设计目标是创建一个友好的用户界面和工作环境，其中比较有名的特性是第一次引进了命令历史、命令别名和作业控制等机制，改善了人机交互能力。但需要注意的是，C Shell与Bourne Shell系列（包括Korn Shell）是不兼容的，两个环境中开发的脚本不能互换使用。

3. Korn Shell

Korn Shell是UNIX系统中继Borne Shell与C Shell之后推出的第三个著名的Shell。Korn Shell以Borne Shell为基础，同时充分吸纳了C Shell的优点，兼具Bourne Shell的功能与C Shell的交互能力，支持命令行编辑、命令历史、命令别名和作业控制等功能，极大地增强和丰富了Bourne Shell的基本功能，且与Borne Shell完全兼容，同时也奠定了现代Shell（如Bash）的基础。

4. Bourne Again Shell

Bash（Bourne Again Shell）是基于POSIX 1003.2标准开发的一个免费版的Shell，与Borne Shell和Korn Shell一脉相承，在功能上又有较大的提高。Bash兼收并蓄，广泛采纳了其他Shell的优点，支持emacs与vi两种命令行编辑功能，支持命令历史、命令别名和作业控制等机制，改善了人机交互能力，具有良好的用户界面和工作环境。作为基本配置，Bash是Linux系统首选的Shell，其地位就像Bourne Shell之于UNIX。

5. TC Shell

TC Shell是在C Shell的基础上开发的，继承并发展了C Shell的全部功能特性，支持emacs命令行编辑、命令历史、命令别名和作业控制等机制，改善了人机交互能力，具有良好的用户界面和工作环境。但需要注意的是，TC Shell与Bourne系列的Shell（如Korn Shell和Bash等）是不兼容的，在两个系列的Shell环境中开发的脚本不能互换使用。

6. Z Shell

Z Shell是一个功能非常强大的命令解释程序，它吸收并集成了Korn Shell、Bash以及TC Shell等的许多功能特性，能够提高用户与UNIX系统的人机交互效率，非常适合用做交互式Shell。可以说，Z Shell是上述Shell的超集，同时也增加了许多先进的功能特性，这些功能特性位于动态库中，可根据具体需要加载或卸载，以节省内存。

任何时刻，每个用户只能使用一个命令解释程序，但可以随时切换到其他Shell环境。为了确定当前究竟处于哪一个Shell环境，可以使用“ps -f”命令，找出ps的父进程，即可确定当前使用的是哪一个Shell，例如：

```
$ ps -f
  UID    PID  PPID    C   STIME TTY          TIME CMD
  root    861   859    0 16:56:36 pts/4      0:00  -ksh
  root    909   894    0 17:09:47 pts/4      0:00  ps -f
  root    894   873    0 17:04:49 pts/4      0:00  ksh
$
```

增加新用户时，每个用户都可以一次性地选定一个命令解释程序，如果未做出选择，系统默认的命令解释程序为/bin/sh（即Bourne Shell）。如果需要，也可以通过usermod等命令修改用户账号信息，或直接修改passwd文件，随时更换命令解释程序。一经注册，即可调用默认或指定的命令解释程序。

Shell是大多数用户访问UNIX系统的主要命令行界面，Shell功能的任何改进，如命令行编辑、命令或文件名补充以及命令提示符定制等，都有助于提高用户访问UNIX系统的工作效率。Shell的功能与特点是用户选择Shell的依据。

9.2.2 设置用户初始化文件

初始化文件（也称启动文件）是一种Shell脚本，其中包含用户注册后必须执行的各种命令，

以便在注册之后，能够设置用户的工作环境。原则上讲，初始化文件与普通Shell脚本一样，均可执行任何处理任务。但是，初始化文件的主要任务是设定用户的工作环境。

初始化文件分为系统初始化文件和用户初始化文件，其中系统初始化文件主要用于设置系统范围的用户工作环境，如设置命令检索路径、用户主目录、命令提示符以及终端类型等环境变量，用户初始化文件主要用于进一步定制自己的工作环境，如设置命令别名、个性化的命令提示符以及命令检索路径等。

每个Shell都提供并支持各自的初始化文件，其中包括位于/etc目录的系统初始化文件，如/etc/profile文件，以及位于用户主目录的用户初始化文件，如\$HOME/.profile文件等。表9-5给出了各种Shell支持的用户初始化文件。

表9-5 Shell使用的用户初始化文件

Shell	初始化文件	目的
Bourne Shell	\$HOME/.profile	用于设置用户自己的工作环境，如设置PATH变量等
C Shell	\$HOME/.login	用于定义每次注册时需要执行的命令，如声明环境变量，设置终端类型等
	\$HOME/.cshrc	用于设置用户自己的工作环境，如设置环境变量，定义命令别名等
Korn Shell	\$HOME/.profile	用于设置用户自己的工作环境，如设置PATH变量，定义命令别名等
Bash	\$HOME/.profile	用于设置用户自己的工作环境，如设置PATH变量，定义命令别名等
	\$HOME/.bash_profile	作用同上
	\$HOME/.bash_login	作用同上
	\$HOME/.bashrc	用于设置交互式非注册Shell的用户工作环境
TC Shell	\$HOME/.tcshrc	用于设置用户自己的工作环境，如设置环境变量，定义命令别名等
	\$HOME/.cshrc	同上
	\$HOME/.login	用于定义每次注册时需要执行的命令，如设置环境变量，定义终端类型等
Z Shell	\$HOME/.zshenv	用户初始化文件，用于设置用户自己的工作环境
	\$HOME/.zprofile	用户初始化文件，用于设置用户自己的工作环境，如设置命令别名和函数等
	\$HOME/.zlogin	用户初始化文件，用于设置用户自己的工作环境，如设置命令别名和函数等
	\$HOME/.zshrc	用户初始化文件，用于设置用户自己的工作环境，如设置命令别名和函数等

如果选用的是Bourne系列的Shell，当用户注册或打开一个新的终端窗口时，系统首先会运行系统初始化文件/etc/profile（Bash还会执行/etc/bash_profile文件）。用户可以查阅系统初始化文件，但不应修改其中的任何内容，以免影响其他用户。之后，系统还会检测并执行相应的用户初始化文件，以定制用户自己的运行环境。

每个UNIX系统通常都提供若干标准的用户初始化文件，其中的默认设置基本上都能够满足用户的常规要求。如有特殊需求，如定制系统提示符，设置命令别名等，只需在此基础上进行适当的修改，增加特定的变量设置或命令即可。

下面是Solaris系统提供的用户初始化模板文件：

- /etc/skel/.profile（用于Bourne Shell与Korn Shell）；
- /etc/skel/local.profile（可用于Bourne Shell与Korn Shell）；
- /etc/skel/local.cshrc（可用于C Shell）；
- /etc/skel/local.login（可用于C Shell）。

在利用useradd命令增加新的用户账号时，如果使用“-m”或“-k”选项指定了/etc/skel目录，useradd将会把/etc/skel目录中的初始化文件复制到用户的主目录。此时，应根据选用的命令解释程序，删除不必要的文件（如果存在）。然后再利用.profile文件作为用户初始化文件的起点，定制自己的Shell工作环境。

注意：在Bourne Shell与Korn Shell运行环境中，需要设置的用户初始化文件通常只有\$HOME/.profile文件。在Bash运行环境中，必要时还可以设置\$HOME/.bash_profile初始化文件。

9.2.3 定制Shell工作环境

1. Shell环境

每个Shell都会维护一组由login程序、系统初始化文件以及用户初始化文件定义的各种变量，其中包括：

- 环境变量：可用于Shell调度执行的所有进程的变量称为环境变量。使用env或set命令可以查阅这些变量的设置情况。
- Shell本地变量：仅对当前Shell有效的变量称为本地变量。

在Bourne系列的Shell中，本地变量和环境变量通常均统一采用大写字母（但并不禁止使用小写字母和其他字符）。如果期望变量的设置能够用于随后调度执行的任何命令或程序，必须使用export命令公布用户自己设置的变量。

在用户初始化文件中，可以修改预定义的变量值，也可以设置附加的变量，定制自己的Shell工作环境。如果想在Bourne系列Shell的用户初始化文件中设置变量，可以使用下列命令：

```
VARIABLE=value; export VARIABLE
```

例如，为了使vi编辑器等能够正常地工作，需要根据自己使用的具体终端类型，采用“TERM=termtype; export TERM”的命令形式定义终端，以便系统能够检索TERMINFO数据库，确定终端的特性。有关变量的更多信息，参见第6章。

2. 设置环境变量

在数据库等应用程序中，需要用到许多环境变量。一些软件厂商通常也都提供一个环境变量设置脚本，供用户设置运行环境。

为使环境变量的设置能够用于当前的Shell运行环境，可以采用两种方法：一是利用“.”命令读取并执行Shell脚本，在当前的Shell中设置运行环境；二是把环境变量的设置语句加到.profile等用户初始化文件中。

“.”命令的主要功能是在当前Shell的控制下，读取指定的Shell脚本文件，由当前的Shell解释执行。“.”命令的调用方法如下：

```
. script
```

其中，script是一个Shell脚本文件，文件中包含必要的环境变量设置语句。

使用“.”命令的主要目的是执行指定脚本文件中的命令，在当前的Shell中设置环境变量。例如，在使用MySQL数据库之前，用户也许需要运行诸如“mysql_env.sh”的环境变量设置脚本，或者在.profile文件中增加下列内容，从而设置自己的运行环境：

```
MYSQL_HOST=iscas
MYSQL_TCP_PORT=3306
TMPDIR=/tmp
export MYSQL_HOST MYSQL_TCP_PORT TMPDIR
```

假定mysql_env.sh是一个环境变量设置脚本，可使用“.”命令读取并运行mysql_env.sh脚本文件中的语句，在当前的Shell中设置环境变量。即使退出脚本，环境变量的设置仍将保持有效。

```
$ . ./mysql_env.sh
$
```

如果使用下列命令，则在退出子Shell或Shell脚本之后，当前的Shell环境不会留存脚本中设置的任何变量。

```
$ sh mysql_env.sh

或

$ ./mysql_env.sh
```

3. 设置命令检索路径

设置环境变量的一个重要内容是设置PATH变量。Shell变量PATH用于设置命令的检索路径，定义命令所在的目录。PATH变量包含一系列目录，目录名之间由冒号“:”分隔。目录的列举顺序确定了命令的检索顺序。为了提高系统的响应速度，常用命令所在的目录应放在靠前的位置。

假定环境变量PATH设置为“/usr/bin:/usr/sbin:”，意味着命令的检索路径依次为/usr/bin、/usr/sbin和当前目录。指定当前目录至少有三种方法：一是使用两个相邻的冒号；二是在原有路径名的前部或后部增加一个冒号“:”；三是增加一个表示当前目录的句点“.”。

当用户采用完整的绝对路径提交命令（输入的命令名以斜杠“/”开始）时，Shell将会省略PATH变量检索步骤，按照给定的路径检索命令，直接加载并执行指定的命令。否则，当用户仅输入了命令名本身时，Shell将会根据PATH变量指定的目录顺序，依次检索与之匹配的命令文件。

不管在哪个目录位置，如果发现了匹配的命令文件，文件的访问权限是可执行的，而且也不是目录文件时，则创建一个新的进程，执行该命令。如果指定的命令既非绝对路径，从PATH变量列举的目录中也找不到匹配的命令文件，系统将会输出下列错误信息：

```
ksh: cmd: commang not found
```

上述错误信息表明：或者命令名的输入有误；或者给定的命令确实不在PATH变量指定的检索目录中。对于第二种情况，校正的办法一是使用绝对路径名，二是修改PATH变量，把命令所在的目录加到PATH变量中，如附加到PATH变量值的后面。

在安装第三方的软件时，通常需要修改PATH变量，以免出现“命令不存在”或使用绝对路径名运行命令的麻烦。

初始化文件通常已经设定了PATH变量，包括命令路径的检索顺序。但是，大多数用户都会修改这个变量，增加自己的命令检索路径，或调整命令路径的检索顺序。例如，为了把当前目录作为PATH变量中的一个命令检索路径，可以使用下列命令：

```
PATH=$PATH:.; export PATH
```

注意：如果PATH变量的命令检索路径设置不当，有可能导致系统响应时间过长，也可能导致系统执行不正确的命令。为了提高PATH变量检索的有效性，下面是设置目录检索路径的若干建议：

- 如果想加入当前工作目录“.”，通常应把当前工作目录作为最后一个检索路径，如果安全并非主要问题，也可以把当前工作目录作为第一个检索路径；
- 应尽可能地缩短命令检索路径，以减少响应时间，提高系统的性能；
- 命令的检索顺序是从左到右，经常使用的命令，其所在目录应位于检索路径的最前面；
- 确保命令检索路径不包含重复的目录，以免执行重复的检索；
- 应尽量避免检索大的目录，如果可能，应把大的目录置于检索路径的最后面；
- 如果存在，应把本地目录置于NFS远程目录前端，以减少出现系统挂起的机会，也减轻不必要的网络负载。

为了定制用户自己的命令检索路径，使之能够首先检索系统目录，同时把自己的特定目录及当前工作目录也包括在检索路径中，可在用户初始化文件中增加下列内容：

```
PATH=$PATH:$HOME/bin:;
export PATH
```

4. 设定语言环境

LANG和LC系列环境变量用于设置特定的语言环境（locale），包括系统提示信息、格式转换、应用惯例和表达方式、字符排序原则，以及日期、时间、货币和数值的输出形式等。

通常，只需设置LANG环境变量，即可确定一个语言环境。实际上，在设置LANG环境变量时，系统也能够连带设置所有的LC系列变量（LC_ALL变量除外），因而能够提供正确的语言环境。同样，设置LC_ALL变量也能同时设置其他LC系列变量，但LANG与LC_ALL两个变量互不影响。此外，也可以单独设置下列LC系列变量，以设置部分语言环境：

- LC_COLLATE（指定字符排序原则）；
- LC_CTYPE（指定语言类型，包括字符类型、字符转换与多字节字符的宽度等）；
- LC_MESSAGES（指定系统提示信息采用的语言）；
- LC_NUMERIC（指定小数点与千分位的表达方式）；
- LC_MONETARY（指定货币符号等）；
- LC_TIME（指定日期与时间的表示方式）；
- LC_ALL（意味着设定上述的所有LC*变量）。

为了利用环境变量设置简体中文语言环境，可以把LANG变量设置为zh、zh_CN.EUC、zh_CN.GBK、zh_CN.GB18030或zh_CN.UTF-8。因此，可在用户初始化文件中增加下列内容：

```
LANG=zh_CN.UTF-8; export LANG
```

为了查询UNIX系统支持的语言环境，可以运行“`locale -a`”命令。欲了解当前设定的语言环境，可以运行下列`locale`命令：

```
$ locale
LANG=zh_CN.UTF-8
LC_CTYPE="zh_CN.UTF-8"
LC_NUMERIC="zh_CN.UTF-8"
LC_TIME="zh_CN.UTF-8"
LC_COLLATE="zh_CN.UTF-8"
LC_MONETARY="zh_CN.UTF-8"
LC_MESSAGES="zh_CN.UTF-8"
LC_ALL=
$
```

如果想要在中英文语言环境之间切换，可把环境变量`LANG`交替地设置成`zh_CN.UTF-8`或`C`。

5. 定制命令提示符

当以命令行方式注册到UNIX系统，或在图形桌面打开一个终端窗口时，窗口的左边将会出现Shell命令提示符，说明调用的Shell已经就绪，用户可以输入命令。如果调用的是Bash，其命令提示符为“`PS1=\s-\v\S '`”。其中，“`\s`”表示当前使用的Shell名，“`\v`”表示Shell的版本号，“`\S`”表示根据用户的身份显示美元符号“`$`”（普通用户）或注释符号“`#`”（超级用户），作为命令提示符的一部分，例如：

```
bash-3.00$
```

UNIX系统的命令提示符共分为四级，依次由Shell的内置变量`PS1`、`PS2`、`PS3`和`PS4`分别定义。当输入的命令不完整，输入的引号未成对出现，或直接在命令行上输入Shell编程语句时才会用到第二级命令提示符。`PS3`和`PS4`分别用于`select`循环结构和Shell脚本调试。

最常见的命令提示符主要是第一级命令提示符。在Bash中，命令提示符能够给出丰富的提示信息，有助于用户输入命令。为了定制自己的命令提示符，可以参照下列方法，设置`PS1`变量：

```
bash-3.00$ PS1='[\u@\h \W]\$ '
[ggxing@iscas ~]$
```

表9-6中给出了设置Bash命令提示符时可用的部分特殊字符，这些特殊字符可用于显示注册的用户名、系统的主机名或当前的工作目录等，用户可根据自己的爱好予以选用。

表9-6 Bash命令提示符中可用的部分特殊字符

字符	简单说明
\S	输出表示用户身份的提示符。超级用户为“#”，普通用户为“\$”
\!	显示当前命令在命令历史记录中的序号
\#	显示当前命令自调用Shell起的序号。从1开始编号
\a	输出提示音
\d	按照“星期 月 日”的格式显示日期，如“Mon May 08”
\h	显示系统的主机名（不包括域名）

(续表)

字符	简单说明
\H	显示系统的规范主机名（包括域名）
\j	显示后台作业号（包括当前正在运行或暂停的作业）
\l	显示终端的设备名
\n	在提示信息中插入一个换行字符
\r	在提示信息中插入一个回车字符（相当于清除回车字符之前的所有提示信息）
\s	显示当前使用的Shell
\@	以12小时“AM/PM”的格式显示时间，如“03:30 PM”或“10:12 上午”
\T	以12小时“HH:MM:SS”的格式显示时间，如上述的“3:30 PM”将会显示为“03:30:00”，没有AM和PM之区分
\t	以24小时“HH:MM:SS”的格式显示时间，如“15:30:00”
\A	以24小时“HH:MM”的格式显示时间，如“15:30”
\u	显示当前的注册用户名
\v, \V	显示Bash的版本信息
\w	显示当前工作目录的完整路径名
\W	显示当前工作目录最后一个子目录的名字
\	在命令提示符中插入一个反斜杠
\[, \]	“\[”标志一个不可打印字符序列的开始，以便在命令提示符中嵌入一个终端控制序列，“\]”表示不可打印字符序列的结束

但在Korn Shell中，命令提示符只是简单的注释符号“#”或美元符号“\$”。为了改变这种单调的命令提示符，也可以简单地修改PS1变量的定义。例如，假定我们希望命令提示符中能够包含命令序号和当前工作目录，可以采用下列设置方法：

```
$ PS1='[! `pwd`]$ '
[160 ~]$
```

当输入了部分命令，或误输入的引号未成对出现而按下Enter键时，Shell将会输出第二级命令提示符（通常为大于号“>”）。出现这一情况时，解决的办法：一是继续完成命令的输入，二是利用中断键Ctrl-C终止误输入的命令，使Shell能够继续处理新的命令。为了把默认的第二级命令提示符改为more，可以使用下列命令：

```
$ PS2='more ? '
$
```

6. 定义命令别名

命令别名的定义仅在当前的Shell中有效，一旦退出Shell，所有的命令别名定义也将随之消失。为了设置永久性的命令别名，可在\$HOME/.profile文件中定义，不管何时注册到UNIX系统，随时都可以直接使用。

正如第4章所述，cp、rm或mv等命令存在误删或覆盖原有同名文件的问题。为了防止此类

情况出现，可将下列命令别名加到自己的.profile文件中（为了彻底解决这个问题，也可以由系统管理员一次性地修改/etc/skel目录中的.profile文件，把下列命令别名定义加到其中）：

```
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

注意：每当启动Shell时，Shell都会读取并执行.profile文件。如果定义的命令别名过多，Shell的启动过程将会减慢。因此，一般不要定义太多的命令别名。

7. 定制用户初始化文件举例

下面以Korn Shell为例，说明怎样以标准的用户初始化文件为基础，定制自己的用户初始化文件。

通常，系统提供的标准.profile文件内容如下：

```
$ cat $MHOME/.profile
#       This is the default standard profile provided to a user.
#       They are expected to edit it to meet their own needs.

MAIL=/usr/mail/${LOGNAME:?}
$
```

为了定制自己的运行环境，修改Shell的检索路径，使之包含用户主目录及当前目录，设置中文语言环境，以vi作为命令行编辑器等，可以利用任何编辑器，修改.profile，增加下列内容：

```
PATH=$PATH:$HOME/bin:.
LANG=zh_CN.UTF8
EDITOR=vi
export PATH LANG EDITOR
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
.....
```

9.3 增加与删除用户组

为使相关的用户能够访问共同的资源与服务，UNIX系统支持用户组的概念。UNIX系统中的每个用户都从属于某个用户组，同组的用户具有相同的用户组访问权限。此外，每个用户还可以临时改换自己的有效用户组，以便与其他用户组相关联。这种灵活性意味着用户除了可以按工作性质或所在部门（如产品部或技术支持部）从属于一个主要用户组，还可以根据工作的需要（如项目组），与其他用户组共享数据与访问权限。

在UNIX系统中，用户组的所有信息通常均存储在/etc/group系统文件中（详细介绍见表9-7）。文件中的每一行定义一个用户组，其语法格式如下：

```
group_name:password:gid:user_list
```

下面是Solaris系统提供的默认用户组信息：

```
# cat /etc/group
root::0:
```



```

other::1:root
bin::2:root,daemon
sys::3:root,bin,adm
adm::4:root,daemon
uucp::5:root
mail::6:root
tty::7:root,adm
lp::8:root,adm
.....
#

```

表9-7 /etc/group文件

字段	简单说明
group_name	用户组的名字。用户组名可由2~8个字符组成
password	通常为“x”。这个字段目前并无实际的意义
gid	用户组ID。同用户ID一样，系统中的用户组ID必须是唯一的。用户组ID是一个32位的无符号整数，最大值为2147483647。其中0~99保留为系统用户组使用，自定义的用户组ID应位于100~60000的范围之内。考虑到与其他系统的兼容性，建议使用16位无符号整数的最大值32 767作为用户ID的上限
user_list	用户组成员列表。其中可以包含属于当前用户组的所有用户名，用户名之间需加逗号“,”分隔符

为了在系统中增加新的用户组，可以手工编辑/etc/group文件，也可以使用groupadd命令。groupadd命令的语法格式简写如下：

```
groupadd [-g gid [-o]] group
```

其中，“-g”选项用于指定新增的用户组ID。如果忽略此选项，默认的用户组ID为已分配的最大ID号加1。例如，如果用户组ID号100、105和200已经分配，默认的用户组ID为201。

如果想要增加一个名为banking，用户组ID为180的用户组，可以使用下列命令：

```
$ sudo groupadd -g 180 banking
$
```

然后，可以使用grep、cat命令或其他工具验证用户组是否已经加到系统中：

```
$ grep banking /etc/group
banking:x:180:
$
```

若想修改或删除用户组，可以分别使用groupmod和groupdel命令。因为用户组的修改和删除比较简单，故此处不再赘述。在管理和维护用户组时，最简单的方法其实就是利用编辑器，手工编辑/etc/group文件。

9.4 监控用户

用户管理的一个重要目的就是保证用户的活动总是处于一个正常的范围之内。如果某个用户正在过度地占用CPU资源，则可以使用nice命令降低用户进程的优先级别。从系统安全的角

度考虑，监控用户的活动可能会发现一些未经授权或非正常的用户行为。一旦发现诸如此类现象，可以采取各种措施，如封锁用户账号等。

9.4.1 利用who命令查询系统中的用户

为了查询当前系统中都有哪些用户，这些用户都来自哪里，可以使用who命令。who命令能够输出当前已注册到系统中的所有用户列表，其语法格式如下：

```
who [-abdHlmprstTu] [file]
```

其中，“-a”选项表示同时选用其他多个选项，从而给出较多的信息。“-H”选项表示在输出数据之上增加一个标题行。“-r”选项用于查询系统当前所处的运行级。如果未指定文件名，表示读取默认的/var/adm/utmpx文件，显示其中的内容，也即当前系统中的用户注册与活动信息。为了查询早期的历史记录，可以使用/var/adm/wtmpx文件。

who命令的输出数据通常包括下列字段：

```
NAME LINE TIME [IDLE] [PID] COMMENT [EXIT]
```

其中，NAME字段表示注册的用户名。LINE字段表示用户使用的终端设备名，TIME字段表示用户的注册时间，IDLE字段表示用户自上一个处理活动以来的空闲时间，PID字段表示用户的进程ID，COMMENT字段通常给出用户所在系统的主机名，EXIT字段表示用户的退出状态。

例如，下列信息表明当前系统中只有三个用户，其中一个来自系统控制台，两外两个均来自同一远程系统，其IP地址为169.254.78.56，使用两个不同用户名访问本地系统，三个用户均在9月17日上午9时左右陆续注册：

```
$ who
root      console   Sep 17 09:46    (:0)
gqxing    pts/2      Sep 17 09:59    (169.254.78.56)
root      pts/4      Sep 17 09:54    (:0.0)
cathy     pts/5      Sep 17 09:53    (169.254.78.56)
$
```

如果使用“-H”选项，将会在输出数据之上增加一个标题行，以便了解每一行输出数据的意义，例如：

```
$ who -H
NAME      LINE      TIME
root      console   Sep 17 09:46    (:0)
gqxing    pts/2     Sep 17 09:59    (169.254.78.56)
root      pts/4     Sep 17 09:54    (:0.0)
cathy     pts/5     Sep 17 09:53    (169.254.78.56)
$
```

使用“-a”选项可以显示更多的系统与用户活动数据，例如：

```
$ who -Ha
NAME      LINE      TIME          IDLE    PID  COMMENTS
.         system boot Sep 17 09:29
.         run-level 3 Sep 17 09:30    3      0    S
LOGIN     console   Sep 17 09:30  0:25    396
```

```

zsmon      .          Sep 17 09:30  2:42    418
root      + console   Sep 17 09:46  0:25    836      (:0)
gqxing    + pts/2     Sep 17 09:59  0:23    780      (169.254.78.56)
root      + pts/4     Sep 17 09:54  0:05   1072      (:0.0)
cathy     + pts/5     Sep 17 09:53  .       1059      (169.254.78.56)
$

```

who命令的另外一个用途就是确定系统当前所处的运行级，例如：

```

$ who -r
.          run-level 3  Sep 17 09:30    3      0  S
$

```

其中，“run-level 3”表示系统当前的运行级为3，即多用户加网络运行模式（参见第15章）。“Sep 17 09:30”表示最近一次改变系统运行级的日期和时间。最后一个字段是进入当前运行级之前系统所处的运行级，“S”表示系统是从单用户引导至当前运行级的（实际上也是加电后直接引导至当前运行级的）。

9.4.2 利用finger命令查询系统中的用户

finger命令用于查询当前注册到系统中的用户（包括本地和远程注册的用户），输出注册用户名及全名、所用终端的设备名、空闲时间、注册时间或远程主机名等。**finger**命令的语法格式简写如下：

```
finger [-bflqsw] [user] [user@host]
```

finger命令的输出数据通常包含下列字段：

```
Login  Name  TTY  Idle  When  Where
```

其中，**Login**字段表示注册的用户名，**Name**字段是用户的全名，也即**passwd**文件中的注释字段，**TTY**字段是用户注册时使用的终端设备名，**Idle**字段表示自上一个处理活动以来的空闲时间，以分（单个数字）或“时:分”为单位。**When**字段表示用户注册的时间，**Where**字段是注册用户的系统名或IP地址。

例如，下列信息表明当前系统中只有三个用户，其中一个来自系统控制台，另外两个均来自同一远程系统，其IP地址为169.254.78.56，使用两个不同用户名访问本地系统，三个用户在下午4时左右陆续注册：

```

$ finger
Login      Name          TTY          Idle       When       Where
root      Super-User    console      Mon 15:54  :0
gqxing    gqxing        pts/5        5 Mon 15:59 169.254.78.56
cathy     ???           pts/6        1 Mon 16:01 169.254.78.56
$

```

9.4.3 利用w命令查询系统中的用户活动

为了查询系统中现有的注册用户，以及每个用户当前正在做什么等信息，还可以使用**w**命令。**w**命令的语法格式如下：

```
w [-hlsuw] [user]
```

其中，“-h”选项表示禁止输出包括标题的前两行信息。如果未加任何选项，w命令的输出信息如下：

```
$ w
10:22am up 2:52, 3 users, load average: 0.02, 0.01, 0.01
User      tty      login@   idle    JCPU PCPU what
root      console  9:46am   36      /usr/dt/bin/sdtperfmeter -f -H -
gqxing    pts/2    7:59am   32      -ksh
root      pts/4    9:54am    1      -sh
cathy     pts/5    9:53am    1      w
$
```

在上方的命令输出信息中，第一行信息分别表示当前的系统时间、系统已经运行了多长时间、当前有多少个注册的用户，以及在过去的1分钟、5分钟和15分钟内的系统平均负载。User字段表示注册的用户名，tty字段表示用户使用的设备名，“login@”字段表示用户注册的起始时间，idle表示空闲时间（从上一次输入命令后至今的持续时间），JCPU字段表示从同一终端设备上运行的所有进程及其子进程所用的整个CPU时间，PCPU表示当前进程使用的CPU时间，what字段表示当前进程的命令名及其参数。

9.4.4 向注册用户发送消息

UNIX系统提供大量的工具供注册的用户之间相互进行实时通信。通常，向注册用户发送日常消息的方法是利用/etc/motd（message of the day）文件。管理人员可以随时修改这个文件，一旦用户注册，即可见到motd文件中的内容。

但上述方法仅在用户注册时有效，对于已经注册的用户，则看不到motd文件修改后的内容，除非再次注册。一旦有紧急情况需要通知用户，如因停电或系统故障需要立即停机时，可利用wall（write to all）命令向注册的用户发送紧急通知。例如：

```
# wall
This system will shut down for emergency maintenance now.
You have 5 minutes to save your work and log out.
^D
Broadcast Message from root (pts/4) on iscas Thu Sep 17 09:47:52...
This system will shut down for emergency maintenance now.
You have 5 minutes to save your work and log out.
#
```

不管用户当前正在做什么，wall命令将会把按下Ctrl-D键之前输入的文字信息立即写到用户的终端窗口中。

9.5 以不同用户的身份访问系统

UNIX系统提供一个su（substitute user）命令，其主要功能是改变用户的身份。在输入超级用户（或指定用户）的密码之后，一个用户可以“合法地变身”为另外一个用户，尤其是能够成为超级用户，从而获得超级用户的访问权限。也就是说，su命令能够随时改变当前用户的有效用户ID，而不必另行注册。其语法格式简写如下：

```
su [-] [username [-c command]]
```

利用su命令，普通用户能够以超级用户或指定用户的身份和权限，在新的Shell运行环境中访问系统，直至输入exit命令或按下Ctrl-D键，返回先前的Shell工作环境。当然，这样做的前提是需要知道超级用户或其他用户的密码。

如果单独运行su命令而不加任何选项与参数，系统将会调用超级用户的注册Shell，进入超级用户的工作环境，改变用户的有效用户ID与用户组ID，但保持先前的工作目录不变。下面的例子说明了用户gqxing使用su命令前后的环境变化情况（其中的id命令用于检查用户当前的有效用户ID）：

```
$ id
uid=100(gqxing) gid=1(other)
$ echo $PATH
/usr/bin:
$ pwd
/export/home/gqxing
$ su
Password:
# id
uid=0(root) gid=0(root)
# echo $PATH
/usr/sbin:/usr/bin
# pwd
/export/home/gqxing
#
```

如果在su命令后面附加一个连字符“-”，还可以直接进入超级用户（或指定用户）的主目录，进入超级用户（或指定用户）的Shell工作环境，这与使用root（或其他用户名）直接注册一样。不仅是用户ID与用户组ID，整个运行环境也完全等同于超级用户（或其他用户）的运行环境。下面的例子说明了用户cathy使用“su -”命令前后的环境变化情况：

```
$ id
uid=101(cathy) gid=1(other)
$ echo $PATH
/usr/bin:
# pwd
/export/home/cathy
$ su -
Password:
# id
uid=0(root) gid=0(root)
# echo $PATH
/usr/sbin:/usr/bin
# pwd
/
#
```

使用“-c”选项运行su命令时，能够以超级用户（或指定用户）的身份及其访问权限执行指定的命令，命令运行一旦结束，立即恢复原来的用户身份及Shell工作环境。下面的例子表示普通用户无法使用kill命令终止一个进程，但当使用su命令获取超级用户的特权之后，即可正常

执行“-c”选项指定的kill命令（注意，“-c”选项后面的命令、选项及其参数前后必须加双引号）：

```
$ kill -15 920
kill: 920: permission denied
$ su - root -c "kill -15 920"
Password:
$
```


第10章 软件包的制作与管理

本章主要介绍怎样利用UNIX系统提供的软件包管理工具，安装、卸载和检查软件包，讨论怎样制作用户自己的应用软件包。

为了便于软件的分发、安装和维护，UNIX系统提供了一整套全新的软件包管理工具，用于软件包的制作、安装、卸载、检查和维护等软件生命周期后期的整个过程。在SVR4.0之前的UNIX系统版本中，软件的打包、安装和卸载等是通过Shell脚本实现的，现在的软件管理主要是通过软件包的信息文件实现的。与其他软件打包工具相比，UNIX系统提供的软件包管理工具有明显的优势：

- 可以统一管理与维护UNIX系统中的系统软件和应用软件；
- 软件的安装、查询、检测以及删除过程简单、统一；
- 在开发系统和目标系统中，软件的存储位置并不要求非集中存储不可，而是可以根据需要分布到不同的位置；
- 软件包的安装可以重新定位（整体或部分）。

10.1 软件包组成简介

软件包是由一组相关的软件（包括可执行的目标文件、数据文件与配置文件等）、信息文件以及安装脚本组成的。

从软件包制作的角度来看，一个软件包通常可由下列三个组成部分构成（如图10-1）：

- 基本组成部分。其中包括软件本身以及两个描述软件包组成部分的信息文件——pkginfo与prototype（或pkgmap）文件；
- 选用的信息文件。其中包括copyright、depend、space以及compver文件；
- 选用的安装脚本文件。如preinstall、postinstall、preremove和postremove等。

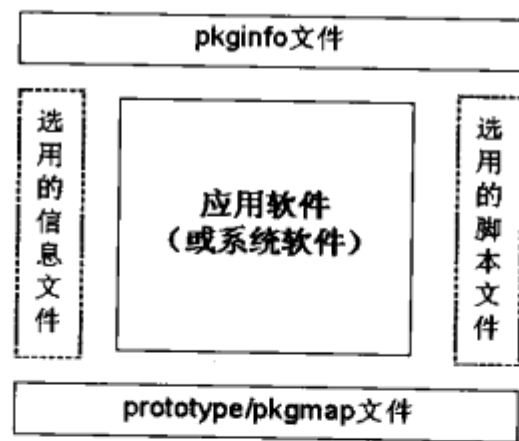


图10-1 软件包的组成

10.1.1 基本组成部分

一个最基本的软件包至少必须包含下列三部分内容。

- 软件：软件是由一组为实现某一功能目标的相关软件和数据组合而成的。其中可以包括可执行目标文件、数据文件以及配置文件等。
- pkginfo文件：pkginfo文件是一个必备的信息文件，其主要用途是定义软件包的描述变量，给出软件包的简要说明信息，如软件包的全名和缩写、软件包的版本、发表日期以及适用的目标系统等。
- prototype/pkgmap文件：其中详细描述了组成软件包的每一个文件，包括文件的路径名、

类型、访问权限、文件属主或文件大小等。需要特别说明的是，在利用pkgproto命令生成prototype文件之后，尚需再做必要的加工。但是，经由软件包制作工具pkgmk命令自动生成的pkgmap文件是不能随意改变的。一旦软件包制作完成，prototype文件也就不需要了，但pkgmap文件将会随软件包一同分发。

10.1.2 选用的信息文件

除了pkginfo、prototype或pkgmap文件之外，如有必要，一个软件包还可以选用下列4个信息文件：

- copyright文件：其中含有软件包的版权信息。在安装软件包时，系统将会根据此文件中的内容显示软件的版权信息。
- depend文件：说明当前的软件包与其他软件包的依赖关系。
- space文件：指定软件包的磁盘空间需求。
- compver文件：说明与当前软件包兼容的软件版本。

10.1.3 选用的Shell脚本文件

尽管UNIX系统并不需要用户提供安装脚本，但为了便于软件开发商执行定制的安装动作，UNIX系统仍然支持三种类型的安装脚本。

- request脚本：这是一个在整个安装过程中唯一需要与安装者进行交互的脚本。在这个脚本中，可以请求安装者提供必要的信息，为变量赋值或重新定义变量，如软件系列号等。
- 针对特定的安装类别提供的处理脚本：针对某一类软件而定义的，在安装或卸载软件包时执行的处理动作；对软件包中划分的每个安装类别都可以提供两个脚本，并分别命名为i.classname和r.classname。其中，前者是在安装软件包时需要执行的脚本，后者是在删除软件包时应执行的脚本。
- 过程脚本：提供安装过程中某个特定阶段需要执行的处理动作，其中包括安装软件包前后的准备与设置（preinstall和postinstall），以及删除软件包前后的准备与善后处理（preremove和postremove）等。每个安装脚本必须是可执行文件，如Shell脚本或程序。

10.2 软件包的相关文件和命令

综上所述，在制作、安装和维护软件包时，需要用到下列信息文件：

- pkginfo文件；
- prototype/pkgmap文件；
- copyright文件；
- depend文件；
- space文件；
- compver文件。

除了pkgmap文件由pkgmk命令自动生成，prototype文件由pkgproto命令生成之后再编辑之外，其他所有文件均需软件包制作者利用编辑工具自行创建。

此外，如果采用非交互式安装方式，还需要用到admin文件，本书不打算介绍非交互式安

装，故也不讨论admin文件。下面开始逐一介绍上述的每个信息文件。

10.2.1 pkginfo文件

pkginfo是一个ASCII文本文件，其中给出了软件包的简单描述，以及若干必要的安装控制信息。如软件包的缩写名和全名、软件包的版本、发表日期以及适用的目标系统等。这个文件应由软件包制作者自行创建，其中的每个逻辑行均由下列形式的语句组成：

PARAMETER=value

在pkginfo文件中，每个文本行的顺序并不重要，但必须包括PKG、NAME、ARCH、VERSION和CATEGORY等5个变量（或称参数），其他变量是选用的。表10-1给出了pkginfo文件中常用变量的说明。

表10-1 pkginfo文件中的常用变量

变量	简单说明
PKG	软件包的缩写名（最多不超过32个字符）。每个软件包都必须赋予一个缩写名，用于唯一地引用相应的软件包。所有的软件包维护工具都是利用这个缩写名引用、管理和维护软件包的
NAME	指定软件包的名字（最多不超过256个字符）。软件开发商可以使用NAME变量描述软件包的功能和用途。如果限于规定而意犹未尽，还可以利用DESC变量做补充说明
ARCH	指定软件适用的目标系统。在制作软件包时，也可以使用pkgmk命令强制修改这个变量的值。目标系统名的长度限于16个字符。如果不知道怎样确定目标系统的ARCH变量值，可以使用uname命令，把“uname -m”命令的输出作为ARCH变量的值
VERSION	指定软件包的版本号（最多不超过256个字符）。在制作软件包时，还可以使用pkgmk命令创建或修改这个变量值
CATEGORY	指定软件包的类型。一个软件包至少必须属于system和application两种默认的类型之一（但也可以是其他类型）。每个类型名的长度仅限于16个字符
DESC	给出软件包的描述（最多不超过256个ASCII字符）。可以使用这个变量作为NAME变量的补充，给出详尽的软件产品描述，以便在执行“pkginfo -l”命令时能够完整地反映软件包的全貌
VENDOR	用于指定软件的版权（最多不超过256个ASCII字符）
HOTLINE	电话或通信地址（最多不超过256个ASCII字符），以使用户能够获取更多的产品信息，以及报告软件故障
EMAIL	电子邮件地址（最多不超过256个ASCII字符），以使用户能够获取更多的产品信息或报告软件故障
VSTOCK	软件厂商的股票编码（最多不超过256个ASCII字符），用于标识软件产品的归属
CLASSES	定义软件包中的安装类别，多个安装类别之间应加空格分隔符。安装类别的列举顺序确定了每一类文件实体的安装顺序（利用request脚本可以修改这个变量）
ISTATES	列出能够执行软件包安装的运行级。例如，“S s 1”表示只能在运行级为S、s或1时安装软件包。可以选用的运行级为s、S、1、2和3
RSTATES	列举能够执行软件包删除的运行级。例如，“S s 1”表示只能在运行级为S、s或1时删除软件包。可以选用的运行级为s、S、1、2和3
BASEDIR	用于指定安装可重定位文件的默认目录。如果这个变量为空，说明安装的软件包不能重定位。在此情况下，如果软件包中存在具有相对路径名的文件，则不予安装

（续表）

变量	简单说明
ULIMIT	如果存在，将会把这个变量的值传递给ulimit命令，以便在安装过程中能够创建较大的文件
ORDER	列举每个安装类别，以便在创建软件包时pkgmk命令能够按此顺序把软件写入介质中。未在这个字段中列出的安装类别将按默认的标准顺序存储到介质中
MAXINST	指定系统只能同时安装的最大软件包实例数量。通常，一个系统只允许安装一个软件包实例。如果准备安装多个软件包实例，必须设置这个变量
PSTAMP	用于标记软件包中pkgmap文件的制作时间。如果PSTAMP变量未定义，则使用默认的时间标记。默认的时间标记由UNIX系统的机器名和紧随其后的字符串“YYYYMMDDHHMMSS（年月日时分秒）”组成
INTONLY	如果赋予此变量任何非空值，表示只能以交互方式安装软件包

下面是一个pkginfo文件的例子：

```
$ cat pkginfo
PKG=LK
NAME=LifeKeeper Core Utilities
ARCH=i386
VERSION=1.02.00
CATEGORY=application
VENDOR=AT&T
HOTLINE=1-800-677-BUGS
CLASSES=none
ISTATE=S 1 2
RSTATE=S 1 2
$
```

其中，软件包的缩写名为LK，软件适用的目标系统为Intel x86计算机的UNIX系统，软件的类型为应用程序等。

10.2.2 prototype文件

prototype是一个需要由软件开发商自行创建的ASCII文本文件，用于描述一个软件包中包含的所有文件。prototype文件中的每个描述行用于定义一个需要打包的文件实体。一个文件实体可以是Shell脚本、可执行文件、数据文件和配置文件等。pkgmk命令就是根据prototype文件构建软件包的。

prototype文件的主要用途概述如下：

- 定义一个软件包都是由哪些文件实体组成的。
- 定义每个文件实体的属性及其在开发系统中的位置。
- 说明文件实体或整个软件包是否可以重新定位。可重定位的软件包能够安装在目标系统的任何位置，从而提高软件安装的灵活性。
- 定义符号链接文件。
- 确定是否需要把整个软件包强制分为多个卷，以便能够把大型软件包分装到多个存储介质上。

• 根据安装或卸载过程的具体需要，确定是否需要从逻辑上把文件实体划分为不同的安装类别。

prototype文件的每个描述行由若干字段组成，字段之间需加空格分隔符，其语法格式如下（每个字段必须按下列顺序出现）：

```
[part] ftype class pathname [major minor] mode owner group
```

表10-2给出了**prototype**文件中的每个字段及其解释。

表10-2 **prototype**文件中的字段及意义

字段	简单说明
part	选用的字段，表示相应文件实体所在的卷号。卷是一组文件的集合，是软件包的最小处理单位。如果需要，开发商可以按一定的准则把文件分为若干卷。如果未指定卷，则假定只有一个卷。之前，当采用软盘形式提供软件包时，如果软件包过大，通常需要把软件包分为多个卷，分装到多个软盘，现在无此必要
ftype	<p>采用单个字符表示的文件类型。文件类型分为可修改与不可修改两类。其中，不可修改的文件类型为：</p> <ul style="list-style-type: none"> • f 普通文件，包括可执行文件、数据文件或其他不容修改的文件 • d 目录 • b 块特殊文件 • c 字符特殊文件 • l 硬链接文件 • p 管道文件 • s 符号链接文件 • i 信息文件（如pkginfo文件）或安装脚本（如preinstall脚本） • x 只有当前软件包能够访问的专属目录 <p>可修改的文件类型包括：</p> <ul style="list-style-type: none"> • e 在安装或卸载过程中可以编辑的文件，如配置文件等 • v 可变长文件（安装后可以覆盖或扩展的文件，如日志文件等） <p>在安装过程中，除了e和v类型的文件，其他所有类型的文件都不能改变。因此，如果文件需要根据实际情况进行修改，应标记为e或v</p>
class	文件实体所属的安装类别。注意，这个字段不适用于软件包的信息文件或安装脚本文件
pathname	<p>文件在目标系统中的路径名。如果为相对路径名，则表示文件可（需要）重新定位。如果指定为“path1=path2”，则有两重目的：定义一个链接文件，其中path1表示链接的目标文件，path2表示链接的源文件；在第二种情况下，path1表示文件安装到目标系统后的路径名，path2表示文件在开发系统上的实际位置，可以是绝对路径名，也可以是相对路径名。另外，路径名中也可以包含变量，以便安装时重新定位。但不要使用下列保留字：</p> <ul style="list-style-type: none"> • PKG_INSTALL_ROOT • BASEDIR • CLIENT_BASEDIR
major	主设备号。仅适用于字符或块设备特殊文件
minor	次设备号。仅适用于字符或块设备特殊文件
mode	八进制数据表示的文件访问权限（如0664）。问号“?”表示原有的访问权限保持不变，这意味着目标系统中已经存在相应的文件。注意，这个字段不适用于链接文件、软件包的信息文件或其他非安装文件

(续表)

字段	简单说明
owner	文件属主（如root）。问号“?”表示原有的文件属主保持不变，这意味着目标系统已经存在相应的文件。注意，这个字段不适用于链接文件、软件包的信息文件或其他非安装文件。但在必要时，开发商也可以指定软件包信息文件的文件属主，表示安装脚本将会以指定的用户身份进行安装
group	文件的用户组。问号“?”表示原有的用户组属性保持不变，这意味着目标系统中已经存在相应的文件。注意，这个字段不适用于链接文件、软件包的信息文件或其他非安装文件。但在必要时，开发商也可以指定软件包信息文件的用户组，表示安装脚本将会以指定的用户组身份进行安装

除了上述的描述行之外，**prototype**文件中还可以包含以感叹号“!”为起始字符的命令行，表示当前行是一个命令，可用于检索指定的目录，合并其他（目录中的）文件内容，以及设置变量的默认值等。可用的命令包括。

- **search**: 用于指定检索目录，以便**pkgmk**命令能够从指定的目录中收集组成软件包的文件实体。当使用**search**命令指定多个目录时，中间必须加空格分隔符。另外还要注意，**search**命令仅在当前的**prototype**文件中有效，对采用**include**命令合并的文件不发生作用。

- **include**: 指定其他**prototype**文件，使之合并到当前文件中。

- **default**: 指定创建软件包时使用的默认属性（如访问权限、文件属主和用户组等）。当**prototype**文件中描述的文件未提供属性时，则使用默认的属性设置。注意，这里设定的属性值不适用于**include**命令中指定的文件。

- **param=value**: 定义制作软件包时需要用到的环境变量，使**pkgmk**命令能够找出分布在各个目录位置中的软件实体，而无需使用实际的路径名。

下面是一个**prototype**文件的例子：

```
$ cat prototype
!PROJDIR=/export/home/gqxing
!search $PROJDIR/src $PROJDIR/incl
i pkginfo=$PROJDIR/pkginfo
i copyright=$PROJDIR/copyright
d none src 0755 gqxing other
!default 0644 gqxing other
f none /export/home/gqxing/src/Makefile 0755 gqxing other
f none /export/home/gqxing/src/recv.c
f none /export/home/gqxing/src/send.c
.....
$
```

上述的**prototype**文件说明，组成软件包的文件实体分别位于/home/gqxing/src和/home/gqxing/incl两个目录中。**pkginfo**和**copyright**两个信息文件均位于/home/gqxing目录中。如果未明确给出，位于/home/mynome/src目录中的文件实体，其默认属性分别为0644（访问权限）、gqxing（文件属主）和other（用户组）。

prototype文件的制作主要有两种方式：一是使用编辑器手工创建**prototype**文件，按照**prototype**文件的要求，一行一行地描述每一个文件实体；二是利用**pkgproto**命令自动创建**prototype**文件。之后，还可以利用编辑器，进一步调整生成的文件。

10.2.3 pkgmap文件

pkgmap是一个ASCII文本文件，其中包含软件包的完整文件列表。注意，pkgmap是由pkgmk命令根据prototype信息文件自动生成的。pkgmap文件中的每个逻辑行描述一个软件包中的文件实体。文件实体可以是Shell脚本、可执行文件、数据文件以及目录等。每个逻辑行由若干字段组成，字段之间需加空格分隔符，其语法格式如下（每个字段必须按下列顺序出现）：

```
part ftype class pathname [major minor] mode owner group [size]
[cksum][modtime]
```

其中，前9个字段的意义完全等同于prototype文件，参见表10-2。表10-3给出了pkgmap文件中其他字段的简单说明。

表10-3 pkgmap文件中的字段及意义

字段	简单说明
size	文件的实际大小，以字节为单位。这个字段不适用于管道文件、特殊文件、链接文件或目录等
cksum	文件内容的校验和。这个字段不适用于管道文件、特殊文件、链接文件或目录等
modtime	最后一次修改时间。这个字段不适用于管道文件、特殊文件、链接文件或目录等

另外，pkgmap文件还会增加一个逻辑行，说明软件包需要分装为多少卷，每个卷的存储容量（以512个字节为单位的数据块数），以及软件包压缩后的大小（以512个字节为单位的数据块数）。其语法格式如下：

```
: number_of_parts maximum_part_size compressed_pkg_size
```

以注释号“#”为起始字符的逻辑行为注释行，处理时忽略不计。下面是一个取自Solaris系统的pkgmap文件实例：

```
$ cat /var/sadm/pkg/HPFC/save/pspool/HPFC/pakmap
: 1 798 208
1 i copyright 93 8259 1200606821
1 i depend 1073 24423 1200606821
1 d none kernel 0755 root sys
1 d none kernel/drv 0755 root sys
1 f none kernel/drv/hpfc 0755 root sys 373708 43587 1105024561
1 f none kernel/drv/hpfc.conf 0644 root sys 754 62003 1105024561
1 e sed kernel/drv/sd.conf 0644 root sys 6068 46250 1105024561
1 i pkginfo 665 50950 1225126178
1 i postinstall 2819 8424 1200606822
1 i postremove 347 27920 1200606817
1 i preinstall 660 55354 1200606817
1 i preremove 226 19258 1200606817
$
```

10.2.4 copyright文件

copyright是一个ASCII文本文件，用于提供软件包的版权信息。copyright文件没有任何格式要求。在安装或删除软件包时，这个文件中的所有内容（包括注释行）将会全部显示在终端屏幕上。下面是一个copyright文件的例子：

```
$ cat copyright
Copyright (c) 2009 My Company
All Rights Reserved.
$
```

10.2.5 depend文件

depend是一个由软件包制作者创建，描述软件包依赖关系的ASCII文本文件。**depend**文件中定义的软件包由若干逻辑行组成，其语法格式如下：

```
type pkg name
(arch)version
(arch)version
.....
```

其中每个字段的意义简述如下。

- **type**: 定义软件包的依赖类型。当前支持的依赖类型包括：
 - (1) **P** 表示安装当前软件包的前提条件，说明在安装软件包前必须先安装哪一个软件包。
 - (2) **I** 表示指定的软件包与当前的软件包是兼容的。
 - (3) **R** 表示软件包的反向依赖关系，说明哪一个软件包依赖于当前的软件包。
- **pkg**: 软件包的缩写名。
- **name**: 软件包的全名。
- **(arch)version**: 软件包适用的目标系统与版本号。这个逻辑行是选用的。如果省略，则表示任何版本。

下面是一个**depend**文件的例子：

```
$ cat depend
P acu      Advanced C Utilities
           (i386/i486)Issue 4 Version 1
P cc       C Programming Language
           (i386/i486)Issue 4 Version 1
$
```

10.2.6 space文件

space文件用于定义除了安装软件包本身之外，目标系统的附加磁盘空间需求。其语法格式如下：

```
pathname blocks inodes
```

其中每个字段的意义简述如下：

- **pathname**: 用于指定一个目录，表示其中需要预留额外的存储空间。指定的目录可以是绝对路径名，也可以是相对路径名。如果为相对路径名，表示可重新定位。
- **blocks**: 用于定义指定的目录中需要预留的数据块（512个字节）数量，以便安装时能够动态地创建文件或目录。
- **inodes**: 定义在指定目录中创建文件或目录时需要使用的信息节点数量。

下面是一个**space**文件的例子：

```
$ cat space
# extra space required by config data which is
# dynamically loaded onto the system
data 500 1
$
```

10.2.7 compver文件

compver是一个ASCII文本文件，用于说明当前软件包与之前（或将来）哪个软件版本是兼容的。**compver**文件中的每个逻辑行用于定义一个与当前软件包版本兼容的版本。定义版本的字符串必须与**pkginfo**文件中定义的版本完全匹配。下面是一个**compver**文件的例子：

```
version 1.3
version 1.0
```

10.2.8 软件包的相关工具

表10-4简单列出了软件包的制作、安装与维护工具。其中前5个工具主要用于软件包的安装、删除、查询和检测，后3个工具主要用于软件包的制作。我们将在随后详细介绍这些工具的功能和用法。

表10-4 软件包的相关工具

软件包工具	功能说明
pkgadd	安装或复制软件包
pkgrm	删除已安装的软件包
pkginfo	查询软件包的各种相关信息
pkgchk	检查已安装软件包的完整性和每个文件实体的准确性
pkgask	提供request脚本需要的响应信息
pkgproto	生成软件包的原型文件，以便使用 pkgmk 命令制作软件包
pkgmk	制作可以分发和安装的软件包
pkgtrans	把软件包转换为不同的存储格式

10.3 制作软件包

利用UNIX提供的软件包管理工具，可以很容易地把应用程序的二进制代码文件、配置文件和相关的文档组织在一起，制作成一个可安装的软件包。

如前所述，除应用程序本身之外，软件包中还应包含必要的信息文件，提供安装脚本等，使软件包的安装、维护和管理变得简单和方便。

本章将通过一个ATM状态监控应用程序，说明软件包的制作、安装和检查过程。这个应用程序的功能是监控ATM的通信和工作状态，监控ATM每个钞箱的余款情况等。制作后的软件包仅包含atmmon、atm.conf、RefManual和UserGuide等4个文件，其在开发系统与目标系统上的目录结构如图10-2所示。

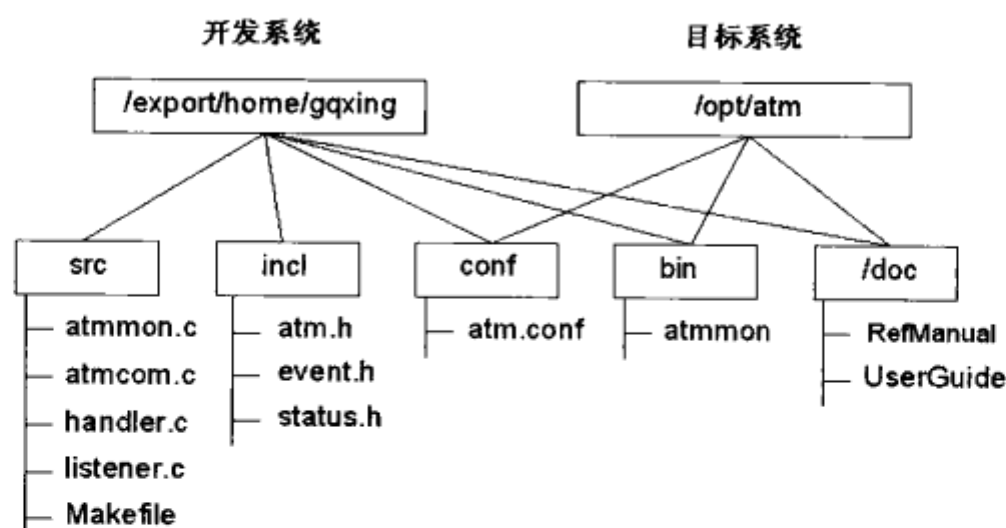


图10-2 ATM监控程序在开发与目标系统上的目录层次结构

10.3.1 制作软件包的步骤

在制作软件包之前，首先要有一个总体的规划，确定软件包的各个组成部分。例如，除了软件本身之外，软件包中尚需包括哪些信息文件，是否需要安装脚本，软件包是否需要重定位，等等。

下面介绍的软件包制作步骤仅供软件包制作者参考。其中，并非所有的步骤都是必需的。软件包的制作者应当根据自己的实际情况做必要的取舍和简化。

1. 定义软件包，如分配软件包缩写名、版本以及适用的目标系统等。
2. 确定是否需要划分安装类别。
3. 确定是否需要在目标系统上重定位。
4. 确定是否需要安装脚本，需要哪一类脚本，以及需要哪一些脚本。
5. 如果存在，定义软件包的依赖关系。
6. 编写版权信息文件。
7. 创建pkginfo文件。
8. 创建prototype文件。

9. 确定是否需要采用多卷方式存储软件包。如果需要采用多卷方式，实现方式有两种。

- 在创建prototype文件时，由软件包制作者根据目标存储介质的容量，自己决定软件包应分装为几个卷，在每个文件实体之前增加一个卷号字段，从而把软件包分装到不同的卷中。这种做法比较麻烦，需要软件包制作者自己计算每个卷中文件实体的大小，确保其总和不超过目标存储介质的容量。

- 利用pkgmk命令的“-l”选项，指定目标存储介质的最大存储容量，由pkgmk命令自动计算，自动划分卷。与第一种方法相比，这种做法相对比较简单。

10. 使用pkgmk命令制作软件包

在开始详细讨论软件包的制作过程之前，下面分别就软件包实例、安装类别以及安装脚本等概念做简单的介绍。

1. 软件包实例

按照软件包管理机制的规定，软件包的缩写名（PKG）、版本（VERSION）和适用的目标系统（ARCH）三个参数可以唯一地标识一个软件包。同一个软件包，如果版本和适用的目

标系统不同，可以看做不同的软件包实例。例如，下列软件包就是两个不同的软件包实例，因为其版本号是不同的：

```
PKG=abc
VERSION=Release 1
ARCH=i386
```

```
PKG=abc
VERSION=Release 2
ARCH=i386
```

2. 安装类别及其特定的处理脚本

在安装和删除软件包的过程中，为了便于对其中的文件实体采取特定的处理，提高软件包安装与删除的灵活性，UNIX系统允许制作者自己定义，并从逻辑上把文件实体分为若干安装类别，以便在安装和删除软件包时能够针对属于不同类别的文件实体采取不同的处理过程。UNIX系统提供了三个预定义的标准类别和一个默认的安装类别（none）供用户使用。同时也允许制作者定义自己的安装类别。这三个预定义的标准类别列举如下。

- **sed**类：用于提供一种方法，以便在软件包的安装和删除过程中能够使用sed命令编辑目标系统上原有的文件。
- **awk**类：用于提供一种方法，以便在软件包的安装和删除过程中能够使用awk命令编辑目标系统上原有的文件。
- **build**类：用于提供一种方法，以便在软件包的安装过程中能够动态地组织和创建必要的文件。

在制作软件包时，如果确实需要采用分类安装的方式安装软件包，制作者需要在pkginfo文件的CLASSES参数中列出附加的安装类别，并在prototype文件内，把所有的文件实体分配到相应的安装类别中。按照要求，不管是否需要分类处理软件包，软件包中的任何文件实体必须归属于一个已定义的安装类别。如果没有明确指定，所有的文件实体通常均属于默认的安装类别none。这意味着在安装和删除软件包时，并不需要对软件包中的文件实体采取特殊的处理。

pkginfo文件中的CLASSES参数定义了软件包中都包括哪些安装类别。如果没有额外定义，CLASSES参数至少应包含安装类别none。如果prototype文件中的文件实体从属于某个安装类别，而又没有在CLASSES参数中列出此安装类别，pkgadd命令不会安装相应的文件实体。每个安装类别究竟应当执行什么处理，可以在安装类别的处理脚本中规定。

例如，为了定义和安装一组从属于安装类别class1的文件实体，可遵循下列步骤。

- 在prototype文件中定义从属于class1的文件实体。例如：

```
f class1 /usr/src/myfile
f class1 /usr/src/myfile2
```

- 确保pkginfo文件中的CLASSES参数定义包括安装类别class1：

```
CLASSES=class1 none
```

- 为class1提供两个安装类别特定的处理脚本。按照规定，class1的安装脚本应命名为i.class1，删除脚本应命名为r.class1。如果定义了某个安装类别，但又没有提供相应的处理脚本，安装程序采取的处理动作只是把其中的文件实体从安装介质复制到目标系统而已。

3. 安装脚本及其处理顺序

在安装软件包时，根据软件包的信息文件，pkgadd命令会自动地执行必要的软件包安装处理动作，通常不需要软件开发商提供任何安装脚本。但是，如果需要定制安装过程，如修改系

统已有的配置文件等，可以使用下列三种安装脚本：

- request脚本；
- 安装类别特定的处理脚本；
- 过程脚本。

在安装软件包时，**pkgadd**命令将按照下列顺序，依次执行三种脚本：

- 首先执行request脚本。
- 执行preinstall脚本。
- 安装软件包中的文件实体。如果定义了额外的安装类别，此时将按照**pkginfo**文件中**CLASSES**参数定义的安装类别顺序，依次执行相应的处理脚本，安装相应的文件实体（注意，利用request脚本，能够修改**CLASSES**变量的值，因而也能够改变安装类别的安装顺序）。

- 执行postinstall脚本。

在删除软件包时，**pkgadd**命令将按照下列顺序，依次执行三种脚本：

- 首先执行premove脚本。
- 按照反向顺序，依次执行安装类别特定的处理脚本，删除相应的文件实体。
- 执行postremove脚本。

4. request脚本

在安装软件包的过程中，用户可以使用request脚本进行交互。只有在这个脚本中，软件包才能与用户直接交互，请求用户输入信息。request脚本的输出必须以变量赋值语句的形式出现，其中的变量可以是**pkginfo**文件中定义的任何变量，包括**CLASSES**变量，当然，也可以引入其他未定义的变量。

为request脚本输出变量赋值的目的是为**pkgadd**命令以及其他安装脚本提供动态的安装信息。下面是一个早先Informix数据库使用的request脚本，其功能是在安装时请求用户提供软件的序列号和用户注册键码，供安装脚本及数据库软件使用：

```
$ cat request
# request script for Infomix RDBMS
#
SERNUM2=`ckstr -Q -l11 -d "RDS-R223176" -p "Enter your 11-character serial
number(e.g. RDS-R999999)\nexactly as it appears on your media:"`
echo
SKEY=`ckstr -Q -l6 -d "R8HEKM" -p "Enter your 6-character serial number KEY
(e.g. XXXXXX)\nexactly as it appears on the customer registration form\nenclosed with
this shipment:"`
echo
cat >$1 <<!
SERNUM='${SERNUM2}'
KEY='${SKEY}'
!
exit 0
$
```

究竟怎样编写request脚本呢？下面给出编写request脚本时的注意事项和使用原则。

- request脚本不能修改任何文件。编写request脚本的主要目的是提供软件包与用户交互的机会，以便能够根据用户的响应，执行必要的变量赋值操作。

- 在调用request脚本时，pkgadd将会提供一个文件参数，使request脚本的输出数据能够写入指定的文件。
- 所有的参数赋值应当加到安装环境中，以便pkgadd和其他安装脚本能够使用（参见上述例子）。
- 除CLASSES变量外，request脚本不能修改其他系统变量和标准的安装变量。也就是说，除了CLASSES变量之外，request脚本只能定义和修改软件包自己的变量。
- 输出数据应采用“PARAMETER=value”形式的赋值语句。例如：

```
CLASSES=none class1
```

- 输出数据应写到第一个参数（\$1）表示的文件中。
- 用户终端（键盘）是request脚本默认的标准输入。
- 在删除软件包时不能执行request脚本。但在安装过程中由request脚本提供的变量赋值将会保留，并能够在删除软件包时使用。

5. 过程脚本

UNIX系统支持四种标准的过程脚本：即preinstall、postinstall、preremove和postremove。软件包的制作作者可以选用任何过程脚本，但只能借用其名字（名字是固定的），内容必须自己编写。顾名思义，这些脚本分别是在安装或删除软件包文件实体的前后执行的。

- preinstall：在安装软件包中的所有文件实体之前执行。
- postinstall：在安装软件包中的所有文件实体之后执行。
- preremove：在删除软件包中的所有文件实体之前执行。
- postremove：在删除软件包中的所有文件实体之后执行。

下面给出编写过程脚本时的注意事项和编写原则：

- 过程脚本是以“uid=root和gid=other”的身份执行的。
- 每个过程脚本都应能多次执行，因为当软件包分为多个卷，安装每个卷时都需要执行一次，因而需要多次执行。这个规定意味着，当以同样的输入数据多次执行时，将产生相同的输出结果。

下面是Informix数据库早先提供的一个preinstall过程脚本，其功能是在安装任何文件实体之前创建informix用户和用户组，用于说明如何编写安装脚本：

```
$ cat preinstall
#!/bin/sh
# preinstall script to add informix user and group

minid=100                                # Minimum user and group ID
groupname=informix
grep "^informix" /etc/group >/dev/null 2>&1
if [ $? -eq 0 ]
then
    gid=`grep "^informix" /etc/group | cut -d: -f3`
else
    allgids=`cat /etc/group | cut -d: -f3 | sort -n | tr '\n' ' '`
    set $allgids
    while [ $1 -le $minid ]
    do
```

```

        shift
    done
    gid=$minid
    sum=$#
    while [ $sum -ge 1 ]
    do
        gid=`expr $gid + 1`
        if [ $gid -eq $1 ]
        then
            shift
            sum=$#
            continue
        else
            break
        fi
    done
    groupadd -g $gid $groupname
fi

username=informix
comment="Informix RDBMS"
homedir=/export/home/$username
grep "^informix" /etc/passwd >/dev/null 2>&1
if [ $? -ne 0 ]
then
    alluids=`cat /etc/passwd | cut -d: -f3 | sort -n | tr '\n' ' '`
    set $alluids
    while [ $1 -le $minid ]
    do
        shift
    done
    uid=$minid
    sum=$#
    while [ $sum -gt 1 ]
    do
        uid=`expr $uid + 1`
        if [ $uid -eq $1 ]
        then
            shift
            sum=$#
            continue
        else
            break
        fi
    done
    useradd -u $uid -g $gid -d $homedir -c $comment -m $username
fi
exit 0
$

```

10.3.2 创建pkginfo文件

制作软件包的前期准备工作之一是创建pkginfo文件（如果需要，也可以提供copyright、depend、space和compver等文件）。按照前面的介绍，pkginfo文件至少应包括PKG、NAME、ARCH、VERSION和CATEGORY等5个参数。

以前面的ATM状态监控应用程序为例，让我们逐一确定每一个参数。

在这5个参数中，NAME、VERSION和CATEGORY（application）实际上可以说是已知的。为了确定PKG，我们将软件包命名为atmmon。ARCH也容易确定，由于这个应用程序主要是在Intel x86系列的UNIX系统中运行的，故其目标系统为i386（可在目标系统中使用“uname -m”命令确定ARCH参数）。

除了上述5个参数之外，软件包制作者还需要考虑的是BSAEDIR变量。BASEDIR用于指定软件包在目标系统上的起始目录位置。如果制作的软件包能够重定位，可以根据运行和管理的方便，磁盘空间的使用情况，采用灵活的安装方式，把软件包安装到目标系统的不同目录位置。在此例子中，我们假定把软件包安装到/var/atm目录中，或在安装时利用pkgadd命令的“-a”选项（或admin文件），临时确定安装位置。

另外还可以考虑CLASSES参数，但由于制作起来比较复杂，且需要熟练掌握sed和awk等命令，故一般很少使用。在此例子中，我们将采用常规的制作方法，令所有的文件实体均归属于默认的安装类别none。

至此，pkginfo文件的创建就算基本上完成了。下面就是atmmon软件包的pkginfo文件：

```
$ cat pkginfo
PKG=atmmon
NAME=ATM Status Monitor
ARCH=i386
VERSION=1.2
CATEGORY=application
DESC=Monitor of ATM Communication and Cash Store Status
VENDOR=My Company
HOTLINE=110
EMAIL=gqxing@gmail.com
CLASSES=none
BASEDIR=/opt/atm/
$
```

作为补充，我们再为atmmon软件包制作一个简单的copyright文件：

```
$ cat copyright
Copyright(c) 2009 My Company
All Rights Reserved.
$
```

10.3.3 利用pkgproto命令创建prototype文件

制作软件包的前期准备工作之二是利用pkgproto命令创建prototype文件。

pkgproto命令的主要作用是生成软件包的原型文件，以便利用pkgmk命令制作软件包。pkgproto命令的语法格式简写如下（表10-5给出了pkgproto命令常用选项与参数的简单说明）：

```
pkgproto [-i] [-c class] [path1]
pkgproto [-i] [-c class] [path1=path2] ...
```

表10-5 pkgproto命令的选项与参数

选项与参数	简单说明
-i	表示忽略符号链接，并把文件实体的类型标记为f（普通文件）而非s（符号链接文件）
-c class	意味着把所有的文件实体都归并到指定的安装类别中
path1	指定软件实体在目标系统上的路径名
path1=path2	表示输出时应以path1替换path2。path1=path2定义了同一软件实体在目标系统与开发系统之间的映射关系，表示开发系统中的软件实体path2在目标系统中的路径名应为path1

通常，pkgproto命令将会根据命令行中指定的目录，检索其中的所有文件和子目录，生成一个prototype原型文件。根据软件包对prototype文件的要求，还可以利用编辑器，适当地调整prototype文件。一旦完成，即可将prototype文件作为pkgmk命令的输入，完成软件包的制作。

在运行pkgproto命令时，如果未指定任何路径名，pkgproto将会等待并读取用户输入的路径名。注意，对于以标准输入形式提供的目录，pkgproto命令仅做单个文件实体处理，并不会对其中的目录做递归检索。

此外，也可以采用下列方式，即以find命令的输出作为pkgproto命令的标准输入，制作prototype文件：

```
$ find path -print | pkgproto > prototype
```

prototype文件实际上描述了软件包在目标系统上的布局，决定了软件包中所有文件实体在目标系统上的安装位置和目录层次结构，软件包的安装位置是固定还是可重新定位，软件包中的部分文件实体是否可重定位等。因此，在制作prototype文件时，我们需要考虑下列三种情况：

- 软件包在目标系统上的安装位置固定不变；
- 整个软件包可以重新定位；
- 软件包中的个别文件实体可以重新定位。

为了制作一个可重定位的软件包，我们只需使用下列pkgproto命令，创建一个prototype文件（如果生成的文件包含当前目录中的其他文件或目录，如src和incl目录及其中的文件，可以使用编辑器直接删除）：

```
$ cd /export/home/gqxing
$ pkgproto . > prototype
$ cat prototype
d none bin 0755 gqxing other
f none bin/atmmon 0555 gqxing other
f none pkginfo 0644 gqxing other
d none conf 0755 gqxing other
f none conf/atm.conf 0644 gqxing other
f none copyright 0644 gqxing other
d none doc 0755 gqxing other
f none doc/RefManual 0644 gqxing other
f none doc/UserGuide 0644 gqxing other
$
```

对于个别文件实体重新定位的情况，由于涉及到许多其他内容，限于篇幅，且为使整个软件包的制作步骤显得简洁，紧凑，本书不准备介绍，有兴趣的读者可以查阅相关的文档。

如果采用固定安装位置，我们需要编辑上述prototype文件，在每个文件实体的相对路径名前面增加一个“/opt/atm”前缀；或按原有的目录层次结构，把构成软件包的所有文件复制到/opt/atm目录，然后再使用下列pkgproto命令，制作prototype文件：

```
$ pkgproto /opt/atm > prototype
$ cat prototype
d none /opt/atm/bin 0755 gqxing other
f none /opt/atm/bin/atmmon 0555 gqxing other
f none /opt/atm/pkginfo 0644 gqxing other
d none /opt/atm/conf 0755 gqxing other
f none /opt/atm/conf/atm.conf 0644 gqxing other
f none /opt/atm/copyright 0644 gqxing other
d none /opt/atm/doc 0755 gqxing other
f none /opt/atm/doc/RefManual 0644 gqxing other
f none /opt/atm/doc/UserGuide 0644 gqxing other
$
```

至此，我们已经制作了两种形式的prototype文件。细心的读者可能已经发现，pkgproto命令把我们制作的两个软件包信息文件pkginfo和copyright，也作为普通文件实体加到prototype文件中了。实际上，信息文件与普通文件实体不仅类型不同，而且也没有文件访问权限、文件属主和用户组等描述字段。因此可使用编辑工具，把pkginfo与copyright两行分别改为（如果存在，其他信息文件也需如此处理）：

```
i pkginfo
i copyright
```

以制作可重定位的软件包为例，prototype文件修改后的最终结果如下：

```
$ cat prototype
d none bin 0755 gqxing other
f none bin/atmmon 0555 gqxing other
d none conf 0755 gqxing other
f none conf/atm.conf 0644 gqxing other
d none doc 0755 gqxing other
f none doc/RefManual 0644 gqxing other
f none doc/UserGuide 0644 gqxing other
i pkginfo
i copyright
$
```

如果信息文件不在当前目录，或处于软件包文件实体之外的其他目录位置，可以使用下述形式定义信息文件（参见prototype文件的语法定义）：

```
i pkginfo=path/pkginfo
i copyright=path/copyright
```

至此，我们已经创建了两种形式的prototype文件，下一步即可开始制作软件包了。

10.3.4 利用pkgmk命令制作软件包

pkgmk命令用于制作可安装的软件包。制作的软件包可以是一个具有一定层次结构的一组目录文件，也可以是一个数据流格式的文件。在运行pkgmk命令之前，软件包的制作者需要提供两个信息文件：即pkginfo和prototype文件。pkginfo定义了软件包的基本概况，如软件包的缩写名、适用的目标系统和版本等信息；prototype文件则提供了构成软件包的完整文件实体列表。

pkgmk命令使用prototype文件作为输入，并以此为基础，收集并计算位于开发系统上的每个软件包文件实体的有关信息，如文件大小、校验和以及最后访问时间等，然后把两者合并到一起，写到一个新创建的pkgmap文件中。最后，pkgmk命令会把prototype文件中列出的所有文件实体复制到指定的目录位置，以固定的目录结构存储，或写到指定的安装介质中。软件包的制作者不需要知道目录结构的细节，pkgmk命令将负责处理软件包的组织和存储格式。

在开发系统上，构成软件包的文件实体可以位于任何目录位置，pkgmk命令将会基于prototype文件提供的路径信息，精确地组织文件实体在目标系统中的目录结构。制作后的软件包可以存储到任何安装介质上，如软盘、光盘、移动磁盘或磁带等，也可以存储在当前系统中，按照目标系统的实际安装位置，构建成一个具有目录文件层次结构的软件包。

pkgmk命令的语法格式简写如下（表10-6给出了pkgmk命令常用选项与参数的简单说明）：

```
pkgmk [-o] [-a arch] [-b base_dir] [-d device] [-f prototype] [-l limit]
[-p pstamp] [-r root_path] [-v version] [variable=value...] [pkginst]
```

表10-6 pkgmk命令的常用选项与参数

选项与参数	简单说明
-a arch	使用指定的名字强制替换pkginfo文件中设定的目标系统名
-b base_dir	把“-b”选项指定的目录加到prototype文件中所有可重定位的相对路径文件名前面，构成一个新的路径文件名，在开发系统中检索prototype文件中列出的所有文件实体。如果“-b”选项给出的并非绝对路径，pkgmk命令还需要在“-r”选项指定的目录或默认的根本目录“/”中检索源文件实体
-d device	把制作的软件包存入指定的设备或目录中。指定的设备（或目录）名可以是磁带等特殊文件名，也可以是一个具有绝对路径的目录名。默认的存储位置是系统中的软件包暂存目录（/var/spool/pkg）
-f prototype	使用指定的文件作为prototype文件。默认的文件为[Pp]rototype
-l limit	指定输出设备或目录所需空间的最大数据块（512字节）数量。如果输出位置是一个目录或可安装的设备，pkgmk命令通常会使用df命令动态地计算输出设备中可用的存储空间容量。当需要创建一个多卷存储方式的软件包时，可与pkgtrans命令一起使用，首先创建数据流格式的软件包，然后分装到多个存储介质上
-o	如果存在相同的软件包实例，则覆盖原有的软件包实例
-p pstamp	使用指定的时间标记信息强制替换pkginfo文件中设定的制作时间
-r root_path	使用指定的目录作为检索源文件的起点，找出prototype文件中列出的所有文件实体在开发系统中的位置。也就是说，把这个前置目录与prototype文件中的相对目录文件名合在一起，即可确定软件包中所有文件实体在开发系统中的完整路径名。如果需要指定多个目录，中间应加逗号分隔符。如果指定了“-r”选项，pkgmk命令将会在指定的目录中按照全路径名检索prototype文件中列出的所有源文件。如果既未指定“-b”选项，也未指定“-r”选项，pkgmk

(续表)

选项与参数	简单说明
	命令将会在当前目录中检索prototype文件中列出的文件（注意，这里只是按照文件名进行检索，忽略文件路径名中的任何目录部分）
-v version	使用指定的版本信息强制替换pkginfo文件中设定的版本
variable=value	指定制作软件包时使用的环境变量
pkginst	指定软件包的名字。这个名字可以是软件包的缩写名，也可以是软件包的实例名（如pkgname.1或pkgname.2）。软件包的所有实例可以采用“pkgname.*”形式表示

在创建软件包时，**pkgmk**命令将会按照下列步骤进行处理。

(1) 首先处理prototype文件中的所有命令行。prototype文件中的**search**命令告诉**pkgmk**命令应从什么目录位置检索软件包中的所有文件实体。**include**命令的存在与否说明是否需要把其他prototype文件中的内容合并到当前的prototype中。**default**命令用于定义软件包文件实体的默认访问权限、文件属主和用户组等属性。变量定义命令能够为**pkgmk**命令提供环境变量。

(2) 根据prototype文件，把软件包中的所有文件实体以适当的结构或存储格式复制到指定的安装介质或目录中。

(3) 如果需要，把软件包分装到多个存储介质中，组成多卷存储形式的软件包。

(4) 创建pkgmap文件，并以此文件替代prototype文件，置于制作后的软件包中。

运行时，**pkgmk**命令将会根据prototype文件中列举的每个文件实体及其描述，按照下列原则进行检索（注意，下列检索原则并不适用于prototype文件中使用**include**命令合并进来的其他prototype文件）。

(1) 如果“-b”或“-r”选项均未指定，**pkgmk**命令将认为prototype文件中列出的每个文件实体（不考虑路径名的目录部分）与prototype文件位于同一目录中。注意，这一点非常重要，否则**pkgmk**命令将会找不到源文件，同时输出一系列“no object for <somefile> found in search path”的错误信息。

(2) 如果指定的“-b”选项为相对路径名（其首字符不是“/”），则把base_dir加到prototype文件中的每个文件实体的相对路径名前面，构成一个新的相对路径名。最后再从“-r”选项中指定的根目录中检索每个源文件。如果未使用“-r”选项中指定根目录，则假定根目录为“/”。

(3) 如果指定的“-b”选项为绝对路径名（其首字符为“/”），则把base_dir加到prototype文件中的每个文件实体的相对路径名前面，构成一个绝对路径名。此时不考虑“-r”选项指定的根目录。

(4) 如果指定了“-r”选项，则使用全路径名（“-r”选项指定的目录 + “-b”选项指定的目录 + prototype文件中的相对路径文件名）在开发系统中检索文件实体。如果未指定“-b”选项，则其默认目录为空。最终将按“-r”选项指定的每个目录检索源文件。

注意：如果prototype文件是利用“pkgproto 相对路径”或“pkgproto 相对路径=安装路径”形式的命令创建的，应当使用“-r”选项指定“相对路径”的位置，以便**pkgmk**命令能够正确地找到源文件。

下面，我们首先采用最先创建的，即采用相对路径形式的prototype文件制作软件包，其结果如下：

```
$ pkgmk -f prototype
## Building pkgmap from package prototype file.
ERROR in prototype:
    no object for <bin/atmmon> found in search path
    no object for <conf/atm.conf> found in search path
    no object for <doc/RefManual> found in search path
    no object for <doc/UserGuide> found in search path
pkgmk: ERROR: unable to build pkgmap from prototype file
## Packaging was not successful.
$
```

上述输出信息表示**pkgmk**命令找不到**prototype**文件中给出的文件实体，即使采用绝对路径形式的**prototype**文件运行**pkgmk**命令，结果也是如此。**prototype**文件中的文件路径名明明是正确的，为什么**pkgmk**命令会说在检索路径中找不到这些文件呢？

回想一下**prototype**文件的规定可以知道，**prototype**文件中的路径名指的是文件实体在目标系统上的路径名，而非开发系统上的源文件路径名，尽管这些文件确实是根据文件实体当前的目录位置，利用**pkgproto**产生的。

事实上，如果不加任何选项，**pkgmk**命令通常只会在**prototype**文件所在的目录中检索其中定义的文件实体，而不管文件实体的路径名如何。也就是说，**pkgmk**命令实际上并不考虑文件路径名中的目录部分，而只考虑文件名本身，故找不到上述文件实体。

针对上述问题，有三种解决办法（注意，当多次运行**pkgmk**命令制作同一软件包时，应有意识地增加“-o”选项）。

1. 在**prototype**文件中增加**search**命令，为**pkgmk**命令指明检索路径。为此，可在**prototype**文件的第一行增加下列3个检索路径：

```
$ cat prototype
!search /export/home/gqxing/bin /export/home/gqxing/conf /export/home/
gqxing/doc
d none bin 0755 gqxing other
f none bin/atmmon 0555 gqxing other
d none conf 0755 gqxing other
f none conf/atm.conf 0644 gqxing other
d none doc 0755 gqxing other
f none doc/RefManual 0644 gqxing other
f none doc/UserGuide 0644 gqxing other
i pkginfo
i copyright
$ pkgmk -o -f prototype
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter <PSTAMP> set to "iscas20090619132702"
## Attempting to volumize 10 entries in pkgmap.
part 1 -- 396 blocks, 43 entries
## Packaging one part.
/var/spool/pkg/atmmon/pkgmap
/var/spool/pkg/atmmon/pkginfo
/var/spool/pkg/atmmon/root/export/home/gqxing/bin/atmmon
/var/spool/pkg/atmmon/root/export/home/gqxing/conf/atm.conf
/var/spool/pkg/atmmon/root/export/home/gqxing/doc/RefManual
```

```

/var/spool/pkg/atmmon/root/export/home/gqxing/doc/UserGuide
/var/spool/pkg/atmmon/install/copyright
## Validating control scripts.
## Packaging complete.
$

```

2. 使用**pkgmk**命令的“-b”选项，提供文件实体的前置目录路径。如果指定的目录为绝对路径，**pkgmk**将会组合指定的目录和**prototype**文件中的路径名，检索文件实体：

```

$ pkgmk -o -b /export/home/gqxing -f prototype
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter <PSTAMP> set to "iscas20090619134354"
## Attempting to volumize 7 entries in pkgmap.
part 1 -- 348 blocks, 16 entries
## Packaging one part.
/var/spool/pkg/atmmon/pkgmap
/var/spool/pkg/atmmon/pkginfo
/var/spool/pkg/atmmon/reloc/bin/atmmon
/var/spool/pkg/atmmon/reloc/conf/atm.conf
/var/spool/pkg/atmmon/install/copyright
/var/spool/pkg/atmmon/reloc/doc/RefManual
/var/spool/pkg/atmmon/reloc/doc/UserGuide
## Validating control scripts.
## Packaging complete.
$

```

如果指定的前置目录为相对路径，**pkgmk**命令还需要使用“-r”选项指定的目录作为根目录。如果未指定“-r”选项，则假定根目录为“/”。因此，也可以使用下列命令，其效果是一样的：

```

$ pkgmk -o -r / -b export/home/gqxing -f prototype

```

3. 使用**pkgmk**命令的“-r”选项，提供文件实体的根目录路径。**pkgmk**命令将以指定的目录为起点，结合“-b”选项指定的前置目录和**prototype**文件中的路径名，检索文件实体。如果未指定“-b”选项，则假定其前置目录为空。例如：

```

$ pkgmk -o -r /export/home/gqxing -f prototype
## Building pkgmap from package prototype file.
## Processing pkginfo file.
WARNING: parameter <PSTAMP> set to "iscas20090619134354"
## Attempting to volumize 7 entries in pkgmap.
part 1 -- 348 blocks, 16 entries
## Packaging one part.
/var/spool/pkg/atmmon/pkgmap
/var/spool/pkg/atmmon/pkginfo
/var/spool/pkg/atmmon/reloc/bin/atmmon
/var/spool/pkg/atmmon/reloc/conf/atm.conf
/var/spool/pkg/atmmon/install/copyright
/var/spool/pkg/atmmon/reloc/doc/RefManual
/var/spool/pkg/atmmon/reloc/doc/UserGuide
## Validating control scripts.

```

```
## Packaging complete.
$
```

不管采用哪一种方式，运行上述`pkgmk`命令之后，都会在`/var/spool/pkg`目录中创建一个以`atmmon`子目录为起始位置的目录文件层次结构的数据包，其中包括一个`install`目录（内含`copyright`和`prototype`两个文件）、两个信息文件（`pkginfo`文件和`pkgmap`文件）以及以`reloc`（表示可重定位）为根目录的软件包源文件副本，如图10-3所示。

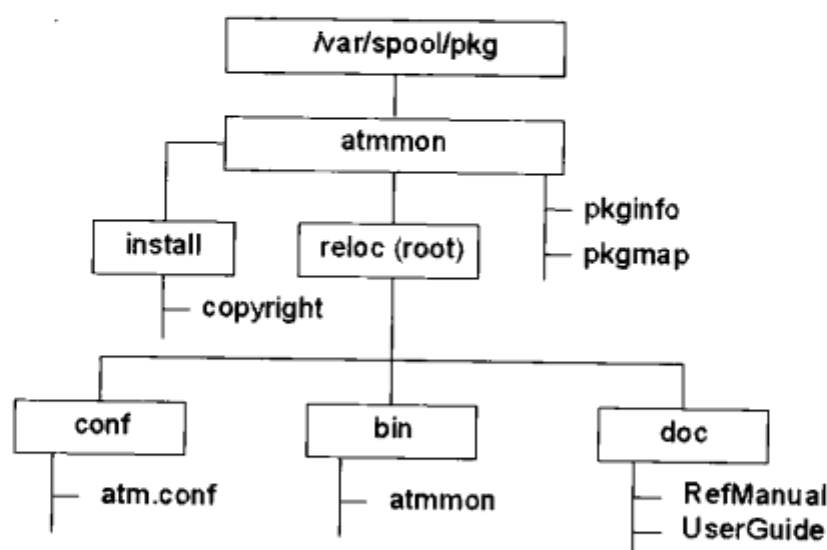


图10-3 新生成的`atmmon`软件包结构

另外，如果软件包采用固定的安装位置，`pkginfo`文件中定义的`BASEDIR`参数将不会发生任何作用。创建的`atmmon`子目录中同样包括一个`install`目录和两个信息文件，但软件包源文件副本的根目录为`root`（表示绝对路径），如上面图10-3所示。

在10.3.5节中，我们将说明怎样利用`pkgtrans`命令，将目录文件层次结构的软件包转换为数据流格式。

现在，让我们具体检验一下`pkgmap`文件。从文件中的内容可以看出，`pkgmap`文件是在`prototype`文件的基础上，由`pkgmk`命令为每个文件实体增加了三个字段。这三个字段（最后三个）分别为文件的大小、校验和以及最后访问时间。

```
$ cat /var/spool/pkg/atmmon/pkgmap
: 1 348
1 d none bin 0755 gqxing other
1 f none bin/atmmon 0555 gqxing other 24888 55994 1245310548
1 d none conf 0755 gqxing other
1 f none conf/atm.conf 0644 gqxing other 12022 49090 1245387609
1 i copyright 51 4257 1245387691
1 d none doc 0755 gqxing other
1 f none doc/RefManual 0644 gqxing other 62725 65458 1245388095
1 f none doc/UserGuide 0644 gqxing other 66080 3682 1245388137
1 i pkginfo 245 19687 1245390234
$
```

10.3.5 pkgtrans命令

`pkgtrans`命令用于转换软件包的存储格式，以便适用不同的需要（软件分发和安装等）：

- 从目录文件层次结构转换成数据流格式；
- 从数据流格式转换成目录文件层次结构；

- 从一个目录文件层次结构转换到另一个目录文件层次结构。

pkgtrans命令的语法格式简写如下（表10-7给出了pkgtrans命令常用选项与参数的简单说明）：

```
pkgtrans [-nos] device1 device2 [pkginst]
```

表10-7 pkgtrans命令的常用选项与参数

选项与参数	简单说明
-n	即使已经存在任何软件包实例，仍旧把新建的软件包存储到目标设备中，直至达到pkginfo文件中MAXINST变量定义的最大数量为止
-o	如果目标设备中已经存在相同的软件包实例，则覆盖原有的软件包实例
-s	表示以数据流格式把软件包写入目标设备中。默认的处理方式是把层次结构形式的软件包写入同时支持两种存储格式的设备中
device1	源设备，表示对其中的软件包进行格式转换
device2	目标设备，表示把转换后的软件包存储到此设备中
pkginst	指定准备转换的软件包。如果给定的参数为all，表示转换所有的软件包。如果未指定软件包，系统将会列出其发现的所有软件包，提示用户选择需要转换的软件包

为了制作数据流格式的软件包，需要采取两个步骤。

1. 首先使用pkgmk命令制作一个目录层次结构的软件包，将制作后的软件包存储在默认的软件包暂存目录（/var/spool/pkg），或指定的目录中。如果期望制作一个多卷存储的数据流格式的软件包，应在运行pkgmk命令时使用“-l”选项，指定目标存储介质的容量。

2. 然后使用pkgtrans命令，把目录层次结构的软件包转换为数据流格式的软件包。

例如，为了把我们新制作的，目录层次结构的atmmon软件包转换为数据流格式的软件包，可以使用下列命令：

```
$ pkgtrans -s /var/spool/pkg atmmon.stream atmmon
Transferring <atmmon> package instance
$
```

现在，我们已成功地在/var/spool/pkg目录中创建了一个数据流格式的软件包。如果想把软件包存储在不同的目录中，需要给出绝对路径名，如/export/home/gqxing/atmmon.stream：

```
$ cd /var/spool/pkg
$ ls -l atmmon.stream
-rw-r--r--  1 gqxing  other      170496 Jun 19 14:48 atmmon.stream
$ file atmmon.stream
atmmon.stream:      package datastream
$
```

为了制作一个多卷形式的数据流软件包，把一个较大的软件包分装到多个1.44MB的软盘上，我们可以采用下面的方法：

```
$ pkgmk -r /somedir -f prototype -l 2880 somepkg
$ pkgtrans -s /dev/rdiskette somepkg
```

其中，“-l”选项表示每个卷不应超过2880个数据块，也即1.44MB的容量。

下面的例子说明怎样转换位于软盘（/dev/rdiskette）上的所有软件包，并把转换后的结果存储到/tmp目录中：

```
$ pkgtrans /dev/rdiskette /tmp all
```

下面的例子说明怎样转换位于/tmp目录中的软件包pkg1和pkg2，并把转换后的数据流格式的软件包输出到磁带上：

```
$ pkgtrans -s /tmp /dev/rmt/0 pkg1 pkg2
```

下面的例子说明了怎样转换位于/tmp目录中的软件包pkg1和pkg2，并把转换后的数据流格式的软件包写到软盘上：

```
$ pkgtrans -s /tmp /dev/rdiskette pkg1 pkg2
```

10.4 安装软件包

pkgadd命令的主要用途是把位于存储介质或某一目录中的软件包安装到系统上。具体执行时，pkgadd命令将会根据pkgmap文件的定义，分以下两种情况进行处理。

- 对于“f”类型的普通文件，pkgadd命令将会按指定的路径名，把文件复制到目标系统中。
- 对于其他文件类型，包括目录、链接文件、管道文件以及特殊文件，如果目标系统中不存在，pkgadd命令将会创建相应的文件实体（之前会提请用户予以确认）。

pkgadd命令的语法格式简写如下（表10-8给出了pkgadd命令常用选项与参数的简单说明）：

```
pkgadd [-d device] [-nv] [-R root_path] [pkginst]
pkgadd -s [-d device | all] [pkginst]
```

表10-8 pkgadd命令的常用选项与参数

选项与参数	功能说明
-d device	从指定的设备中安装或复制软件包。这里所谓的设备既可以是磁带或光盘等存储介质，也可以是采用绝对路径名形式的目录，还可以是由pkgtrans命令创建的数据流格式的软件包文件
-n	以非交互方式安装软件包，同时禁止显示安装的文件列表。默认的安装方式为交互式安装
-R root_path	用于定义一个绝对路径名的目录，作为可重定位软件包的根目录。软件包中的所有文件，包括信息文件，按照其原来的层次目录结构，均重定位于指定的目录下面
-s spool	把软件包写入指定的目录暂存，而不执行实际的安装
pkginst	通常，pkgadd命令将会检索指定的设备或目录，显示一个可安装的软件包列表，供用户选择安装哪一个、哪一范围或全部软件包。也可以在命令行中指定准备安装的软件包名。 all表示指定设备或目录中的所有软件包

在运行pkgadd命令时，如果未指定“-d”选项，pkgadd命令将会从默认的软件包暂存目录（var/spool/pkg）中检索指定的或可用的软件包。如果指定了“-s”选项而未指定“-d”选项（两者不能同时使用），pkgadd命令将会把软件包写到软件包的暂存目录，而不执行实际的软件安装。

为了安装某个特定的软件包，可以直接指定软件包的名字。例如，假定我们现在准备安装 **atmmon**。因为这个软件包是已知的，故可以在命令行中直接指定：

```
# pkgadd atmmon

Processing package instance <atmmon> from </var/spool/pkg>
ATM Status Monitor(i386) 1.2
Copyright (c) 2009 My Company
All Rights Reserved.

The selected base directory </opt/atm> must exist before installation
is attempted.

Do you want this directory created now [y,n,?,q] y
Using </opt/atm> as the package base directory.
## Processing package information.
## Processing system information.
    7 package pathnames are already properly installed.
## Verifying disk space requirements.
## Checking for conflicts with packages already installed.
## Checking for setuid/setgid programs.

Installing ATM Status Monitor as <atmmon>

## Installing part 1 of 1.
[ verifying class <none> ]

Installation of <atmmon> was successful.
#
```

如果软件包位于某个存储介质，如软盘（其设备名为 **/dev/rdiskette**）上，可以使用下列命令进行安装：

```
# pkgadd -d /dev/rdiskette
The following packages are available:
    1  atmmon      ATM Status Monitor
                        (i386) 1.2

Select package(s) you wish to process (or 'all' to press
all packages). (default: all) [?,??,q]: Enter
```

输入软件包的序号、**all**或直接按下**Enter**键，即可开始安装（上述可供选择的软件包只有一个，故选择序号1、**all**或直接按**Enter**键的效果是一样的）。

在安装软件包的过程中，除了把软件包中的所有文件实体复制到指定的目录位置之外，作为其中的一个步骤，**pkgadd**命令还会执行下列处理任务。

- 使用软件包的缩写名，在 **/var/sadm/pkg** 目录中创建一个子目录，同时把软件包中的所有信息文件和安装文件（如果存在）复制到新建的目录中。在软件包的维护过程中，**pkginfo**和**pkgchk**命令就是利用相应目录中的**pkgmap**文件，对软件包进行完整性检测的。在卸载软件包时，**pkgrm**命令将会在适当的阶段，执行相应目录中的卸载脚本，如**preremove**、**postremove**以及安装类别特定的卸载脚本（如果存在），删除软件包。

- 以**pkgmap**文件为基础，通过增加和删除某些字段，适当地编辑其中的每一个文件实体描述项，并删除其中所有信息文件和安装脚本的描述项。最终，把编辑后的文件内容复制到

`/var/sadm/install/contents`文件中的适当位置。

从上述说明中可知，`/var/sadm/install/contents`文件含有已安装的所有软件包的所有文件实体列表。其中，每个文件实体描述项的语法格式如下：

```
path type class perm owner group size checksum instdate pkginst
```

表10-9给出了文件实体描述项每个字段的简单说明。

表10-9 contents文件说明

字段	简单说明
path	文件的路径名
type	文件类型。参见prototype文件的说明
class	文件的安装类别，通常为none
perm	文件的访问权限
owner	文件的属主
group	文件的用户组
size	文件的大小
checksum	文件的校验和
instdate	文件的安装日期（从1970年1月1日开始算起的秒数）
pkginst	文件所属软件包的缩写名

利用contents文件，可以查询已安装的任何软件包及其文件：

```
$ grep atmmon /var/sadm/install/contents
/opt/atm/bin d none 0755 gqxing other atmmon
/opt/atm/bin/atmmon f none 0555 gqxing other 24888 55994 1245310548 atmmon
/opt/atm/conf d none 0755 gqxing other atmmon
/opt/atm/conf/atm.conf f none 0644 gqxing other 12022 49090 1245387609 atmmon
/opt/atm/doc d none 0755 gqxing other atmmon
/opt/atm/doc/RefManual f none 0644 gqxing other 62725 65458 1245388095 atmmon
/opt/atm/doc/UserGuide f none 0644 gqxing other 66080 3682 1245388137 atmmon
# Last modified by pkgadd for atmmon package
$
```

如果想要确定某个文件属于哪一个软件包，也可以检索contents文件。

```
$ grep RefManual /var/sadm/install/contents
/opt/atm/doc/RefManual f none 0644 gqxing other 62725 65458 1245388095 atmmon
$
```

10.5 查询软件包

无论何时，当需要了解系统中究竟安装了哪些软件包，软件介质上存有哪些软件包，以及某个目录中含有哪些软件包时，都可以使用pkginfo命令获取相关的信息。pkginfo命令的语法格式简写如下（表10-10给出了pkginfo命令的常用选项与参数）：

```
pkginfo [-q | -x | -l] [-p | -i] [pkginst]
pkginfo [-d device] [-q | -x | -l] [pkginst]
```

表10-10 pkginfo命令的常用选项与参数

选项与参数	简单说明
-q	不显示任何软件信息，仅供在脚本或程序中检测指定的软件包是否已经安装
-x	列出软件包的简要说明。其中包括软件包的缩写名、全名，适用的目标系统以及软件包的版本等信息
-l	显示软件包的详细说明。参见pkginfo文件
-p	如果存在，仅列出安装不完整的所有软件包
-i	列出安装完整的所有软件包
-d device	指定软件包所在的存储介质（如光盘或磁带等）或目录名
pkginst	软件包的名字

例如，为了确定系统中已经安装的某个软件包的有关信息，可以使用下列命令（如果未指定软件包，将会列出系统已经安装的所有软件包）：

```
$ pkginfo atmmon
application atmmon ATM Status Monitor
$ pkginfo -x atmmon
atmmon  ATM Status Monitor
        (i386) 1.2
$ pkginfo -l atmmon
  PKGINST:  atmmon
    NAME:  ATM Status Monitor
CATEGORY:  application
   ARCH:  i386
  VERSION:  1.2
  BASEDIR:  /opt/atm
  VENDOR:  My Company
    DESC:  Monitor of ATM Communication and Cash Store Status
  PSTAMP:  iscas20090619144621
INSTDATE:  Jun 19 2009 14:52
  HOTLINE:  110
   EMAIL:  gqxing@gmail.com
  STATUS:  completely installed
   FILES:           7 installed pathnames
                  3 directories
                  1 executables
                326 blocks used (approx)
$
```

10.6 检测软件包

pkgchk命令用于检测软件包的完整性（包括目录结构和文件的完整性），检验软件安装的正确性。如果发现问题，pkgchk将会给出详细的解释。pkgchk命令也可用于显示软件包的详细信息（“-l”选项）。其语法格式简写如下：

```
pkgchk [-l | -acfnvx] [-m pkgmap] [pkginst]
pkgchk -d device [-l | -fv] [pkginst]
```

上面所列的第一种命令形式用于检测或显示系统当前已安装的软件包，或pkgmap文件中指定的文件。如果未指定软件包名，pkgchk命令将会检测系统中已安装的所有文件。

第二种命令形式用于检测或显示位于暂存目录，但尚未安装的软件包。注意，对于暂存目录中的软件包，pkgchk命令不会做文件属性检测。

表10-11给出了pkgchk命令常用选项与参数的简单说明。

表10-11 pkgchk命令的常用选项与参数

选项与参数	简单说明
-a	仅审查文件属性，而不检测文件的内容。默认情况下既审查文件属性，又检查文件内容，包括文件和目录是否存在，其属性（访问权限、属主和用户组，以及大小等）是否发生变化
-c	仅审查文件内容，而不检测文件的属性。默认情况下既审查文件属性，又检查文件内容，包括文件和目录是否存在，其属性（访问权限、属主和用户组，以及大小等）是否发生变化
-d device	指定软件包所在的设备文件名（如光盘或磁带等）或目录
-f	如果可能，pkgchk命令将会校正文件的属性。如果同“-x”选项一起使用，pkgchk命令还会删除隐藏的文件。使用这个选项时，如果指定的文件实体不存在，pkgchk命令将会适当地创建目录、管道文件、链接文件和设备文件等。如果同时使用“-d”选项处理未安装的软件包，“-f”选项仅对处于目录文件层次结构的软件包进行处理，对数据流格式的软件包不会产生任何影响。除了setuid、setgid和粘性位之外，pkgchk命令将严格按照pkgmap文件中的定义设置所有文件的属性
-l	显示软件包中的指定文件或所有文件的属性与状态信息。这个选项不能与“-a”、“-c”、“-f”和“-v”等选项一起使用
-m pkgmap	按照指定的pkgmap文件检测软件包
-n	不检测可变或可编辑文件的内容。这种方式适用于安装后期的检测。随着时间的推移，配置文件和数据文件等有可能会发生变化，使用“-n”选项可以忽略这些可变文件的变化。通常，可以组合使用“-n”和“-f”选项，忽略可变文件，而校正其他普通文件和目录
-v	详细显示方式，逐一显示pkgchk命令处理的每一个文件
-x	检索位于专属目录中既不属于软件包，又未在pkgmap文件中描述的文件
pkginst	检测指定的软件包。默认情况是检测或显示已安装的所有软件包的信息

假定在运行过程中需要修改atm.conf文件，而编辑后的atm.conf文件，其大小与访问时间等均发生了变化。那么执行pkgchk命令后，将会输出下列错误信息：

```
# pkgchk atmmon
ERROR: /opt/atm/conf/atm.conf
      modtime <06/19/09 01:00:09 PM> expected <06/19/09 03:05:49 PM> actual
      file size <12022> expected <12048> actual
      file cksum <49090> expected <51742> actual
#
```

这是因为，pkgchk命令通常会对软件包中每个文件的内容和属性进行检测。如果我们使用下列命令，只检测文件的属性是否有变化，将会发现一切正常：

```
# pkgchk -a atmmon
#
```

如果再单独检查文件的内容是否有误，就会发现其中有错误：

```
# pkgchk -c atmmon
ERROR: /opt/atm/conf/atm.conf
      modtime <06/19/09 01:00:09 PM> expected <06/19/09 03:05:49 PM> actual
      file size <12022> expected <12048> actual
      file cksum <49090> expected <51742> actual
#
```

假定我们只是修改了atm.conf文件的属主，分别使用“-a”和“-c”选项运行pkgchk命令，其输出结果如下：

```
# pkgchk -a atmmon
ERROR: /opt/atm/conf/atm.conf
      owner name <gqxing> expected <root> actual
# pkgchk -c atmmon
#
```

10.7 卸载软件包

pkgrm命令用于删除系统中先前安装的软件包，或删除暂存目录中的软件包。其语法格式简写如下（表10-12给出了pkgrm命令常用选项与参数的简单说明）：

```
pkgrm [-nv] [pkginst]
pkgrm -s spool [pkginst]
```

表10-12 pkgrm命令的常用选项与参数

选项与参数	简单说明
-n	采用非交互方式删除软件包。如果执行过程需要安装人员的干预，程序通常会终止执行
-s spool	用于指定软件包暂存目录，表示仅从指定的目录中删除软件包，而不是从系统中删除软件包。默认的软件包暂存目录是/var/spool/pkg
pkginst	指定准备删除的软件包

在删除软件包之前，首先需要确定系统中是否存在依赖于指定软件包的其他软件包。如果存在这种依赖关系，在交互方式下系统会输出提示信息，请用户确认是否需要强行删除软件包。在非交互方式（“-n”选项）下，如何处理取决于admin文件中的定义。

例如，如果想删除刚才安装的atmmon软件包，可以使用下列命令：

```
# pkgrm atmmon

The following package is currently installed:
  atmmon  ATM Status Monitor
          (i386) 1.2

Do you want to remove this package? [y,n,?,q] y
```

如果输入y，则相应的软件包将被删除。

```
## Removing installed package instance <atmmon>
## Verifying package <atmmon> dependencies in global zone
## Processing package information.
## Removing pathnames in class <none>
/opt/atm/doc/UserGuide
/opt/atm/doc/RefManual
/opt/atm/doc
/opt/atm/conf/atm.conf
/opt/atm/conf
/opt/atm/bin/atmmon
/opt/atm/bin
## Updating system information.

Removal of <atmmon> was successful.
#
```

如果无法确定软件包的名字，或者想同时删除多个软件包，也可以采用交互方式运行 **pkgrm** 命令。此时，系统将会从头开始，一次显示10个已安装的软件包，供用户做出选择：

```
# pkgrm

The following packages are available:
  1  BRCMbnx          Broadcom NetXtreme II Gigabit Ethernet Adapter Driver
                        (i386) 11.10.0,REV=2007.06.20.13.12
  2  CADP160          Adaptec Ultra160 SCSI Host Adapter Driver
                        (i386) 1.21,REV=2005.01.17.23.31
.....
 10  SUNW1394          Sun IEEE1394 Framework
                        (i386) 11.10.0,REV=2005.01.21.16.34
... 1253 more menu choices to follow;
<RETURN> for more choices, <CTRL-D> to stop display:
... 3 more menu choices to follow;
<RETURN> for more choices, <CTRL-D> to stop display:

1261  SUNWzsh          Z shell (zsh)
                        (i386) 11.10.0,REV=2005.01.08.01.09
1262  SYMhis1          Symbios 895A, 896 and 1010 SCSI driver
                        (i386) 11.9.0,REV=2005.01.06.07.16
1263  atmmon           ATM Status Monitor
                        (i386) 1.2

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]: 1263
```

此时，可以输入欲删除软件包的序号，给出一个序号范围，或一一列出多个软件包的序号（序号之间用逗号隔开）。假定我们选择 **atmmon**，输入序号1263，即可删除选定的软件包：

```
The following package is currently installed:
  atmmon  ATM Status Monitor
            (i386) 1.2

Do you want to remove this package? [y,n,?,q] y

## Removing installed package instance <atmmon>
## Verifying package <atmmon> dependencies in global zone
```



```
## Processing package information.
## Removing pathnames in class <none>
/opt/atm/doc/UserGuide
/opt/atm/doc/RefManual
/opt/atm/doc
/opt/atm/conf/atm.conf
/opt/atm/conf
/opt/atm/bin/atmmon
/opt/atm/bin
## Updating system information.
Removal of <atmmon> was successful.
#
```

第11章 磁盘空间管理

本章主要介绍磁盘空间管理，说明如何查询磁盘空间的使用与空闲情况，怎样找出超大容量的文件，以及长期闲置不用的文件，定期清除磁盘中的垃圾文件，利用UNIX系统提供的各种工具备份或恢复文件，配置文件系统磁盘空间和信息节点的使用限额等。

11.1 查询磁盘空间信息

磁盘空间管理的主要目的是了解磁盘存储空间的使用情况，包括系统当前已经使用的空间、可用的空闲空间、现有的文件数量、空闲的信息节点等；及时清理垃圾文件（如内存影像转储文件），删除超大容量及长期闲置不用的文件，以及清除临时目录或文件。在此基础上，利用UNIX系统提供的标准工具，复制、备份或恢复文件甚至整个文件系统；设置磁盘空间的限额，确保磁盘空间资源的合理分配与使用等。

当文件系统空间容量的使用接近百分之九十时，需要利用cp等命令，把其中的文件转储到相对空闲的其他磁盘中，或利用cpio、tar以及dd等命令把文件转储到磁带上，或者干脆删除其中无需继续保存的文件。

11.1.1 常用的磁盘空间管理工具

表11-1给出了部分磁盘空间管理的常用工具。

表11-1 磁盘空间管理的常用工具

命令	简单描述
df	查询文件系统中的可用或已用存储空间及文件信息节点数量
du	查询指定（或当前）目录中每个文件或目录占用的磁盘空间
find -size	检索指定目录中指定大小的文件
ls -s	以1KB数据块为单位，显示文件的大小
cpio	用于创建、转储或恢复cpio档案文件，实现文件或文件系统的备份与恢复。也可用于实现整体目录层次结构的复制
tar	用于创建、转储或恢复tar档案文件，实现文件或文件系统的备份与恢复
dd	用于实现原始数据复制。可以复制文件甚至文件系统（也即整个磁盘分区）
quot	用于显示指定文件系统中每个用户当前占用的数据块（1024字节）数量

11.1.2 使用df命令检查存储空间的使用情况

系统管理员经常需要监控磁盘空间的使用情况。即使系统配置的硬盘比较大，如果分区不当，如“/”文件系统过小，仍然会产生磁盘空间紧张的情况。为了监控磁盘空间的使用情况，自然会用到df命令。利用df命令，可以查询每个文件系统磁盘空间的使用与空闲状况。df命令的语法格式简写如下：

```
df [-abehklvtv] [-F fstype] [filesystem | block_device | resource] (Solaris)
df [-i | -t | -v] [-kmgs] [filesystem] (AIX)
```

表11-2给出了df命令的部分常用选项及其简单说明。

表11-2 df命令的部分常用选项

选项与参数	简单说明
-a	显示所有文件系统（包括虚拟文件系统，如/proc）的存储空间及其使用情况
-b	以KB为单位，显示可用磁盘空间的容量
-c	显示可用文件信息节点（文件）的数量
-h (Solaris)	以KB、MB或GB为单位，显示每个已安装文件系统的空间使用情况。输出信息包括文件系统的设备文件名、文件系统总容量、已分配存储空间容量、可用存储空间容量、已用存储空间容量占文件系统总容量的百分比，以及文件系统的安装点
-m、-g (AIX)	分别以MB和GB为单位，显示每个已安装文件系统的空间使用情况。输出信息包括文件系统名、文件系统总容量、已用磁盘空间容量、可用存储空间容量、已用存储空间容量占文件系统总容量的百分比，以及文件系统的安装点
-i	显示文件系统的设备文件名、信息节点总量，空闲信息节点数量、已用信息节点数量、已用信息节点数量占信息节点总量的百分比，以及文件系统的安装点
-k	以KB为单位，显示每个文件系统的空间使用情况。输出信息包括文件系统的设备文件名、文件系统的总容量、已分配存储空间容量、可用存储空间容量、已用存储空间占文件系统总容量的百分比，以及文件系统的安装点
-l	显示已安装的本地文件系统的空间使用情况，包括可用的存储空间容量，以及可用的文件信息节点数量等
-t	显示文件系统的空间总量与可用容量，信息节点（文件）总数与可用信息节点的数量
-v	以512个字节的数据块为单位，报告每个文件系统的空间使用情况。其输出信息包括下列字段： <ul style="list-style-type: none">• 文件系统的安装点• 文件系统的名字• 以数据块为单位的文件系统容量• 已分配给文件的数据块数• 空闲可用的数据块数• 已用数据块数与整个文件系统容量的百分比
filesystem	指定文件系统、磁盘分区的设备文件名或文件系统的安装点。通常，df命令仅显示本地系统已安装的所有文件系统的空间使用信息
block_device	如果文件系统尚未安装，可以指定磁盘分区的设备文件名，如/dev/dsk/c1d0s7，检查相应文件系统的空间使用情况
resource	指定NFS远程资源的名称，检查相应文件系统的空间使用情况

如果不加任何选项和参数，df命令通常会以512个字节的数据块为单位，显示系统中所有已安装文件系统的空间使用情况，包括可用数据块的数量，以及可以创建的文件数量。下列df命令的输出数据表明，“/”文件系统中尚有24 420 460个可用的数据块（约11.75GB的可用存储空间），还可以创建1 768 787个新的目录或文件：

```
$ df
/ (/dev/dsk/c1d0s0 ):24420460 blocks 1768787 files
```

```
.....
/export/home      (/dev/dsk/c1d0s7   ):29692270 blocks  1787925 files
$
```

上述df命令仅仅给出了文件系统当前空闲的磁盘空间和文件信息节点的数量。为了获取更多的信息，如文件系统存储空间及信息节点的总容量等，可以使用“-t”选项。例如，下述命令针对每一个文件系统给出两行信息，其中第一行是空闲的磁盘空间容量（以512个字节的数
据块为单位）和空闲信息节点（文件）的数量；第二行是整个文件系统总的存储容量与信息节点总数：

```
$ df -t
/                  (/dev/dsk/c1d0s0   ): 24420460 blocks  1768787 files
                  total: 32449302 blocks  1956864 files
.....
/export/home      (/dev/dsk/c1d0s7   ): 29692270 blocks  1787925 files
                  total: 29722470 blocks  1787968 files
$
```

在较大的文件系统中，以数据块为单位的空间度量方法不太符合人的直观思维与阅读习惯，尚需经过一番换算才能准确地知道给定数值的意义。为此，可以使用“-k”选项，以KB为单位，显示每个文件系统的空间信息。其中包括每个文件系统的总容量，当前已经使用了多少存储空间，尚有多少可用的存储空间，实际使用的存储空间占整个文件系统总容量的百分比，文件系统的安装点等信息。

```
$ df -k
Filesystem          kbytes    used   avail capacity  Mounted on
/dev/dsk/c1d0s0      16224651 4014421 12047984    25%    /
.....
/dev/dsk/c1d0s7      14861235   15108 14697515     1%  /export/home
$
```

表11-3给出了df命令输出信息中各个字段的简单说明。

表11-3 df命令输出字段的说明

字段名	简单说明
kbytes	文件系统中可用数据存储空间的总容量
used	文件系统中已经占用的存储空间数量
avail	文件系统中可用的空闲存储空间数量
capacity (%used)	文件系统中已用存储空间数量占全部存储空间总量的百分比
Mounted on	安装点

如果df命令的输出信息表明文件系统的使用已经接近或超过整个容量的90%，通常需要清除不必要的文件，如删除临时文件或长期闲置不用的文件，或者使用后面将要介绍的各种系统工具，复制或备份部分文件，以增加可用的存储空间。

实际上，利用“-h”选项，还能够以更容易阅读的形式，即以KB、MB或GB等为计数单位，显示文件系统的存储空间信息，包括每个文件系统的总容量、当前已经使用了多少存储空间、尚有多少可用的存储空间、实际使用的存储空间占整个文件系统总容量的百分比，以及文

件系统的安装点等信息。例如（其中的size字段表示文件系统中全部存储空间的总容量）：

```
$ df -h
Filesystem              size  used  avail  capacity  Mounted on
/dev/dsk/c1d0s0         15G   3.8G   11G      25%      /
.....
/dev/dsk/c1d0s7         14G    15M   14G       1%    /export/home
$
```

在上述的几个例子中，df命令的输出数据实际上还包含一部分虚拟文件系统的信息，如proc等，而我们真正关心的是实际的文件系统，及其存储空间的实际使用情况。因此，为了避免输出其他干扰信息，可以使用“-F”选项，指定文件系统的类型，从而限定df命令的输出结果：

```
$ df -F ufs
/                (/dev/dsk/c1d0s0  ):24420460 blocks  1768787 files
/export/home     (/dev/dsk/c1d0s7  ):29692270 blocks  1787925 files
$ df -h -F ufs
Filesystem              size  used  avail  capacity  Mounted on
/dev/dsk/c1d0s0         15G   3.8G   11G      25%      /
/dev/dsk/c1d0s7         14G    15M   14G       1%    /export/home
$
```

11.1.3 使用du命令检查存储空间占用情况

du命令用于显示指定目录（或当前目录）中每个子目录或文件占用的存储空间数量。如果不加任何选项和参数，du命令通常会输出以1KB数据块为单位的统计数据。du命令的语法格式简写如下：

```
du [-adrs] [-k | -h] [directory] (Solaris)
du [-adr] [-k] [-m] [-g] [directory] (AIX)
```

表11-4给出了du命令的部分常用选项和参数。

表11-4 du命令的部分常用选项和参数

选项与参数	简单说明
-a	以512个字节的数据块为单位，显示指定目录或当前目录中每个文件及每个子目录（包括其中的每个文件）占用的存储空间数量，最终给出整个目录占用的存储空间总和。如果指定的参数是一个普通文件，则显示指定文件占用的存储空间
-h (Solaris)	以KB、MB、GB甚至TB为单位，显示指定目录或当前目录，以及其中所有子目录占用的存储空间数量。输出信息包括文件的大小和相应的目录名
-m、-g (AIX)	分别以MB和GB为单位，显示指定目录或当前目录，以及其中所有子目录占用的磁盘空间数量。输出信息包括文件的大小和相应的目录名
-k	以KB为单位，显示指定目录或当前目录，以及其中所有子目录占用的磁盘空间数量
-s	以512个字节的数据块为单位，显示指定目录或当前目录（包括其中的所有子目录和文件）占用的磁盘空间总量
directory	指定准备检查的目录。如果同时指定多个目录，目录之间需加空格分隔符

利用du命令，当发现较大的目录或文件时，可以视具体情况，确定是否保留、删除或备份，以节省磁盘的存储空间。下面是一个du命令输出的例子：

```
$ du /usr/share/sounds
52      /usr/share/sounds/gtk-events
74      /usr/share/sounds/panel
132     /usr/share/sounds/gataxx
138     /usr/share/sounds/gnibbles
292     /usr/share/sounds/gnrobots2
968     /usr/share/sounds/gnome-stones
132     /usr/share/sounds/iagno
1006    /usr/share/sounds/pidgin
6910    /usr/share/sounds
$
```

同样，为了避免按数据块形式列出统计数据，可以使用“-h”选项，以KB、MB或GB等便于阅读的、符合阅读习惯的形式，输出每个目录或文件占用的存储空间统计数据。下面的例子说明怎样利用“-h”选项，以适当的单位输出存储空间的使用情况。

```
$ du -h /usr/share/sounds
26K     /usr/share/sounds/gtk-events
37K     /usr/share/sounds/panel
66K     /usr/share/sounds/gataxx
69K     /usr/share/sounds/gnibbles
146K    /usr/share/sounds/gnrobots2
484K    /usr/share/sounds/gnome-stones
66K     /usr/share/sounds/iagno
503K    /usr/share/sounds/pidgin
3.4M    /usr/share/sounds
$
```

当输出的文件比较多，且文件的大小参差不齐时，可以采用管道机制，利用sort命令进行排序。其中，“-r”选项表示反向排序，按从大到小的顺序列出每一个目录和文件占用的磁盘空间数量。“-n”选项表示按数值而非字符顺序排序。例如：

```
$ du -k /usr/share/sounds | sort -rn
3455    /usr/share/sounds
503     /usr/share/sounds/pidgin
484     /usr/share/sounds/gnome-stones
146     /usr/share/sounds/gnrobots2
69      /usr/share/sounds/gnibbles
66      /usr/share/sounds/iagno
66      /usr/share/sounds/gataxx
37      /usr/share/sounds/panel
26      /usr/share/sounds/gtk-events
$
```

如果只想查询某个目录（包括其中的所有文件和子目录）占用的全部存储空间数量，可以使用带有“-s”选项的du命令。下面的例子仅仅统计了/boot目录占用的存储空间总和：

```
$ du -s /boot
156908  /boot
$
```


11.1.4 使用quot命令查询每个用户占用的存储空间

quot命令用于显示指定文件系统中每个用户当前占用的数据块数量，数据块的大小通常为1024个字节（AIX系统为512个字节）。quot命令的语法格式简写如下：

```
quot [-acfhv] [filesystem]
```

其中，“-a”选项用于列出指定文件系统，或所有已安装文件系统中属于每个用户的存储空间占有量，也即每个用户拥有的数据块数量。使用“-f”选项，可针对每个用户给出三列信息：占有的数据块数、文件计数和用户名。filesystem用于指定文件系统。默认情况下，quot命令将会检查所有已安装的文件系统。

例如，下列quot命令将会给出“/”文件系统中每个用户占有的数据块数量（其中第一列为数据块计数，第二列为注册的用户名）。

```
# quot /
/dev/rdisk/cld0s0:
3975192 root
  754  uucp
  188  bin
  115  svctag
   87  adm
.....
#
```

在下面的例子中，由于未指定文件系统，quot命令将会给出所有文件系统中每个用户占有的存储空间数量。

```
# quot -a
/dev/rdisk/cld0s0 (/):
3975192 root
  754  uucp
  188  bin
  115  svctag
   87  adm
.....
/dev/rdisk/cld0s7 (/export/home):
  33  gqxing
   9  root
   6  cathy
   6  hwang
#
```

利用“-f”选项，quot命令还会额外给出每个用户创建的文件（或目录）数量（其中第二列为文件数量）。

```
# quot -f /
/dev/rdisk/cld0s0:
3975192 187852 root
  754    44  uucp
  188    97  bin
  115     3  svctag
```

```

      87      10      adm
.....
#

```

11.1.5 使用find命令找出超大容量的文件

当存储空间紧张，需要尽快腾出磁盘空间时，删除超大容量的文件或把文件备份到其他存储介质上是一种快速有效的方法。为了找出超过指定大小的文件，可以使用find命令，其语法格式简写如下：

```
find directory -size +nnn -print
```

其中，*directory*是起始检索目录，“-size +nnn”选项表示大于指定数量的数据块（512个字节）的数量；“-print”选项表示输出find命令的检索结果。这个命令将会列出超过指定大小的所有文件。

例如，下面的例子说明怎样利用find命令，从指定的/var目录中找出大小超过2048个数据块（1MB）的所有文件。

```

$ find /var -size +2048 -print
/var/sadm/install/contents
/var/sadm/pkg/SUNWcslr/save/pspool/SUNWcslr/reloc/lib/libc.so.1
/var/sadm/wbem/logr/store
/var/sadm/smc/properties/classlist.txt
$

```

上述命令的输出结果仅仅给出了文件的名称，这些文件究竟有多大并不知道。同样，为了便于观察，最好能够按文件的大小输出最终的结果。为此，可以使用管道，把find、ls、awk和sort等命令组合起来，按从大到小的顺序显示文件列表。例如：

```

$ cd /var/sadm
$ find . -size +2048 -exec ls -s {} \+ | awk '{print $1 "\t" $2}' | sort -rn
35504  ./install/contents
24880  ./wbem/logr/store
2768   ./pkg/SUNWcslr/save/pspool/SUNWcslr/reloc/lib/libc.so.1
2272   ./smc/properties/classlist.txt
$

```

11.1.6 使用find命令找出长期闲置的文件

作为系统管理员，日常系统维护的一个主要工作就是清除系统中长期闲置不用的文件或垃圾文件，清除临时文件或目录。为了找出在指定的时间内一直没有使用过的文件，可以使用下列find命令：

```
find directory -type f [-atime +nnn] [-mtime +nnn] -print
```

其中，*directory*表示准备检索的起始目录。“-atime +nnn”选项用于找出指定天数（nnn）内没有访问过的文件。“-mtime +nnn”选项用于找出指定天数（nnn）内没有改动的文件。

注意：采用上述find命令检索长期闲置的文件，其输出结果可能包含一系列系统文件或用途不明的文件，而这些文件即使长时间未用也不应轻易删除，除非自己有绝对把握。在不确定其用途时，不能轻易地删除系统文件。即使确实没有用处，也不要直接删除，最好是利用/var/

sadm/install/contents文件，确定文件从属的软件包，然后再利用软件包管理工具予以删除。

因此，最保险的方法是先利用find命令，找出指定时间内一直没有用过的文件，把检索结果储存到一个临时文件（如filelist）中，确定是否应当删除。如果检索出来的文件均属无用文件（或剔除其中需要保留的文件后剩下的全部为无用文件），再使用下列命令删除检索出来的文件：

```
$ rm 'cat filelist'
```

下面的例子展示了怎样找出/export/home/guest目录及其子目录中半年来一直没有访问过的文件，并把这些文件列表存入/tmp/filelist文件中，经过确认之后，再使用rm命令予以删除：

```
$ cd /export/home/guest
$ find . -type f -atime +180 -print > /tmp/filelist
$ cat /tmp/filelist
./doc/TechnicalSpecifications.doc
./doc/TroubleShootingGuide.doc
./doc/NetworkSetup.doc
./doc/PrinterSetup.doc
$ rm 'cat /tmp/filelist'
$
```

11.1.7 使用find命令找出并删除core文件

在开发和测试期间，由于程序可能存在这样或那样的问题，开发系统中经常会存在大量的内存映像转储文件，也即core文件。这样的文件一多，既影响目录文件的清洁性，又会占用大量宝贵的存储空间。为此，作为系统管理员或开发者本人，应当经常清理系统中或开发者自己的工作目录中存在的core文件。

为了找出并删除这种core垃圾文件，可以使用下列find命令，在指定的目录或当前工作目录中，检索并删除core文件。

```
find directory -name core -exec rm {} \;
```

其中，directory是起始检索目录，“-name core”选项表示检索名为core的内存映像转储文件。“-exec rm {} \;”选项表示执行rm命令，删除检索出来的core文件。

下面的例子展示了怎样使用find命令找出并删除hwang用户主目录中的core文件。

```
$ cd /home/hwang
$ find . -name core -exec rm {} \;
$
```

11.1.8 使用ls命令检测文件的大小

前面曾经介绍过，ls命令的“-s”选项也可用于显示以KB为计数单位的文件大小，例如：

```
$ cd /usr/share/dict
$ ls -ls
total 8068
2880 -rw-r--r-- 1 root other 1463867 Nov 25 2004 sun_dict_e2c.dict
1152 -rw-r--r-- 1 root other 581143 Nov 25 2004 sun_dict_e2c.idx
2 -rw-r--r-- 1 root other 274 Nov 25 2004 sun_dict_e2c.ifo
2880 -rw-r--r-- 1 root other 1463980 Nov 25 2004 sun_dict_e2t.dict
```

```
1152 -rw-r--r-- 1 root other 580989 Nov 25 2004 sun_dict_e2t.idx
2 -rw-r--r-- 1 root other 275 Nov 25 2004 sun_dict_e2t.ifo
$
```

为了更便于观察，还可以组合使用**sort**命令，按照从大到小的顺序列出每个文件，例如：

```
$ ls -s | sort -rn
2880 sun_dict_e2t.dict
2880 sun_dict_e2c.dict
1152 sun_dict_e2c.idx
1152 sun_dict_e2t.idx
2 sun_dict_e2t.ifo
2 sun_dict_e2c.ifo
total 8068
$
```

11.2 采用标准工具备份与恢复数据

备份与恢复是两个互逆的数据处理过程，如果运用得当，能够防止系统丢失重要的数据。备份通常指的是从系统磁盘中把系统数据和业务数据等复制到另一个存储介质（通常为移动介质，如磁带、软盘、移动硬盘或CD/DVD等）的过程。恢复是一个逆过程，即从备份介质中把重要文件、文件系统或业务数据复制到系统中。

此外，还要考虑备份什么数据以及怎样备份，这涉及到备份的策略，也影响到数据的恢复。在UNIX系统中，通常可以备份单个文件、目录、文件系统或数据分区，这取决于系统中数据修改的频繁程度、文件的重要性以及其他因素。备份的频率可以采用每天一次、每周一次，甚至每月一次等备份方式。

从备份的数据考虑，可以采用下列备份方式之一。

- 文件备份——采用这种备份方式，可以备份文件系统中的重要系统文件和目录等。一旦系统文件出现问题，可以直接恢复单个或部分文件。

- 文件系统备份——采用这种备份方式，可以备份从根目录开始的整个文件系统，也可以备份/var、/home或用户自建的单个文件系统等。当文件系统出现问题时，可以恢复整个或单个文件系统。

- 数据分区备份——在某些数据库应用系统中，其数据通常采用原始分区而非文件系统存储数据。针对这种情况，可以采用数据分区的备份方式。当需要恢复这样的备份数据时，必须恢复整个磁盘分区。

从备份的方式考虑，可以采用下列备份方式之一。

- 完整备份——这种备份方式主要用于备份文件系统。利用这种备份方式，能够完整地恢复一个文件系统。

- 增量备份——增量备份是在完整备份或其他增量备份的基础上，仅仅备份自上次备份以来发生变动的文件。增量备份用于补充相应时间周期内的完整备份。在一个特定的时间周期内，数据的备份应当包括一个完整备份与0个或多个增量备份。

在备份与恢复数据文件或文件系统时，可以直接使用UNIX系统提供的标准工具软件实现，如**cpio**、**tar**或**dd**等。也可以采用UNIX系统提供的专用软件工具，如**ufsdump/ufsrestore**等。

cpio、**tar**和**dd**等命令是最基本的数据备份与恢复工具，不仅适用于UNIX系统，而且广泛应用于各种Linux系统。若想备份单个文件、部分文件或整个文件系统，完全可以使用UNIX系统的标准工具实现。因此，本章仅讨论UNIX系统提供的标准工具。

至于备份介质，则不外乎磁带、磁盘或光盘等。注意，在不同的UNIX系统上，存储介质的设备文件名并不相同，用户需根据自己的系统配置，确定其设备文件名，确保使用的设备文件名是正确的。这里采用的是Solaris系统中的设备文件名。

在选择备份工具时，也可以考虑各种应用软件提供的备份工具，如Oracle数据库的**exp**，MySQL数据库的**mysqldump**或**mysqlhotcopy**等。

为了备份一组文件、一个完整的目录或文件系统，如/export/home/gqxing/data目录，通常应使用**du**和**df**等命令，事先确定目录或文件系统的容量，以便能够确定生成的档案文件究竟有多大。如果需要把生成的档案文件写入外部的存储介质中，也可以据此选择存储介质的类型、规格和数量。示例如下：

```
$ cd /export/home/gqxing/data
$ du -s .
1251762 .
$
```

du命令的输出数据表示，上述目录中的文件总共占用了1 251 762个数据块（512字节）的存储空间。如果想要备份这些文件，至少需要610MB的存储容量。

在使用**cpio**或**tar**命令备份数据时，如果生成的档案文件过大，可以选用**compress**、**pack**、**zip**或**bzip2**等工具压缩档案文件，最终生成一个压缩的档案文件，以节省存储空间，同时也便于采用**ftp**等网络通信工具，以二进制数据的形式实现系统间的文件复制与交换。

11.2.1 利用cpio实现备份和恢复

cpio是一个常用的数据备份与恢复工具，能够复制单个文件、一组文件、一个完整的目录结构，甚至一个完整的文件系统，把选定的文件复制为一个档案文件，直接存储于磁带、磁盘或其他存储介质中。例如，**cpio**命令能够把一个磁盘分区中的文件系统完整地复制到另一个磁盘分区。

在复制过程中，**cpio**命令将会在每个文件之间（之前）插入一个头信息，以便能够恢复任何选定的文件。**cpio**命令也能够识别存储介质的结束标志，因而能够自动提示用户更换新的介质，根据备份文件的存储容量要求，把档案文件存储到多个介质上。当需要使用多个介质（如多个磁带）创建档案文件，备份文件或文件系统时，**cpio**是一个比**tar**等更灵活、更有效的工具。

cpio的工作原理是，首先利用**find**等命令生成一个文件列表，然后通过管道提交给**cpio**命令，**cpio**再把给定的文件，包括其属性信息，复制到指定的档案文件或存储介质上。

因此，在使用**cpio**命令时，可能需要频繁地使用**find**等命令，为**cpio**命令提供需要复制的文件或文件列表，并通过管道送交**cpio**命令。

cpio具有如下三种运行模式，或者说具有三种用途。

- 恢复，利用“-i”选项，把先前创建的档案文件恢复到系统中。
- 备份，利用“-o”选项和**find**命令等工具，从标准输入中读取文件列表，创建一个档案文件，把选定的文件，包括路径名及文件属性等信息，写入其中，以便能够把文件原样恢复到

系统中。

• 复制，利用“-p”选项和find命令等工具，从标准输入中读取文件列表，把选定目录或文件系统中的文件原样复制到另外一个目录位置或文件系统中，从而创建一个完整的目录或文件系统副本。

cpio命令的语法格式简写如下。表11-5给出了cpio命令的常用选项及其简单说明。

```
cpio -i [options] [-C bufsize] [-I file [-M message]] [pattern]
cpio -o [options] [-C bufsize] [-O file [-M message]]
cpio -p [options] directory
```

表11-5 cpio命令的常用选项

选项	说明
-i	输入模式，用于恢复档案文件
-o	输出模式，用于备份档案文件
-p	复制模式，用于复制一个完整的目录或文件系统
-a	在复制完成之后，复原输入文件的访问时间，使输入文件仍然保持原来的访问时间，就像cpio命令根本没有读过这些文件一样
-A	把新复制的文件附加到档案文件的后面。这个选项仅适用于输出模式，且要求与“-O”选项一起使用
-B	使用5120个字节作为数据块的读写单位。默认的数据块缓冲区为512个字节。注意，输入和输出时最好采用相同的数据块参数
-c	以ASCII字符格式读写头信息，确保不同的UNIX系统或Linux系统之间的兼容性
-C size	使用指定的字节数作为数据块的读写单位
-d	在复制过程中，根据需要创建必要的目录。注意，这个选项只能与“-i”或“-p”选项一起使用
-E file	用于指定一个输入文件，其中包含一系列需要从档案文件中抽取的文件名列表（每个文件名占一行）
-f	仅仅复制与指定模式不匹配的文件
-I file	使用指定的文件作为输入文件，而不是读取标准输入。如果指定的文件是一个字符设备特殊文件，且当前介质已经完全读入时，可根据提示更换新的介质，然后按下Enter键，以便继续读取下一个介质中的数据。注意，这个选项只能与“-i”选项一起使用
-k	尝试跳过部分损坏的档案文件头信息，以及可能遇到的I/O错误。如果想要从已经损坏或顺序不当的介质中复制文件，使用这个选项只能读取那些头信息尚未破坏的文件。注意，这个选项只能与“-i”选项一起使用
-m	使复制的文件仍然保持先前的修改时间。注意，这个选项对复制的目录无效
-M message	定义一个提示信息。当读写达到存储介质结尾，需要更换新的介质时，提示用户更换下一个介质。如果提示信息中包含字符串“%d”，系统将代之以当前的介质序号（介质从1开始编号）
-O file	使用指定的文件作为输出文件，而不是写到标准输出。如果指定的文件是一个字符设备特殊文件，且当前介质已经完全写满时，可根据提示更换新的介质，然后按下Enter键，以便继续写入下一个介质。注意，这个选项只能与“-o”选项一起使用
-t	读取并显示档案文件中的文件列表

(续表)

选项	说明
-u	无条件地强行复制文件。如果不加此选项，cpio命令的常规处理惯例是禁止同名的老文件替换或覆盖新文件
-v	显示方式。显示文件列表及其相关属性。当与“-t”选项一起使用时，其效果等同于“ls -l”命令的输出
-V	特殊显示方式。对于读写的每一个文件，仅仅输出一个句点“.”作为标记

假定我们想要备份/export/home/gqxing/data目录中的所有文件，把生成的档案文件写入磁盘中，可以使用下列cpio命令：

```
$ cd /export/home/gqxing/data
$ find . -print | cpio -oc > /backup/data.cpio
1250992 blocks
$
```

或

```
$ find /export/home/gqxing/data -print | cpio -oc > /backup/data.cpio
1250992 blocks
$
```

在上述例子中，find命令用于提供cpio命令需要复制的文件。由于“find . -print”命令使用当前目录“.”作为起始检索路径，其生成的文件列表为相对路径名，故前者将会采用相对路径名的形式，把所有文件备份到一个指定的data.cpio档案文件中。而第二个find命令采用的是绝对路径名，其生成的文件列表也为相对路径名，故后者生成的是一个包含绝对路径名的档案文件。

采用相对路径复制档案文件的最大好处是，在恢复档案文件时能够重新定位，抽取的文件可以集中存储在任何目录位置，不必把文件仍然恢复到原来的目录位置，故而能够避免覆盖原有的文件。如果先前复制的档案文件采用的是相对路径名，当使用cpio命令恢复档案文件时，读入的文件将会存放到当前目录中。因此，在使用cpio命令复制档案文件时，通常应事先进入指定的目录，然后在当前目录下执行find和cpio命令，使生成的档案文件不包含绝对路径名。

但是，如果复制档案文件时采用的是绝对路径名，在恢复档案文件时也会采用相同的绝对路径名读入文件，覆盖原有的目录和文件。而无法在恢复文件时重新定位，因而也就无法改变文件的存储位置，只能按照原来的目录结构原样恢复。由此可见，使用绝对路径名的文件备份方式具有潜在的危险性。如果现有的文件有所变动，恢复文件后将会使改动的文件丢失数据（当然，有时也许恰恰需要这样做——覆盖修改有误的文件）。

对于采用了相对路径名创建的cpio档案文件，如果想把整个档案文件全部恢复到一个新的目录中，可以事先进入一个目的目录，然后使用下列cpio命令：

```
$ cd /newdir
$ cpio -icdmu < /backup/data.cpio
1250992 blocks
$
```

其中，“-i”选项表示以输入模式执行cpio命令，从输入的档案文件中抽取文件。“-c”选

项表示读取ASCII字符格式的文件头信息。“-d”表示必要时可以在复制过程中创建相应的目录；“-m”选项表示复制的文件仍保持原先的访问时间。“-u”选项表示无条件地强行复制文件，即使存在同名的新文件，不管新旧与否，均强行覆盖，否则老的文件无法替代新文件。在原封不动地复制一个目录或文件系统时，这个选项尤其有用，因为同一文件之前可能已经存在于目的目录或文件系统中。

在完成文件的复制之后，**cpio**将会输出其读写的数据块（512个字节）数量，表示生成或读取的档案文件的大小。

除了备份与恢复功能之外，利用“-p”选项的复制功能，**cpio**命令还能够复制整个目录或文件系统。为了复制一个完整的目录或文件系统，把其中的文件全部复制到另外一个目录或磁盘分区（文件系统）中，应首先使用**cd**命令进入源目录或文件系统的根目录，然后再利用**find**和**cpio**命令实现整个目录或文件系统的复制，例如：

```
$ cd dir1
$ find . -depth -print | cpio -pdmu dir2
1250992 blocks
$
```

其中，**find**命令的“-depth”选项表示逐层深入各级子目录，自底向上依次检索所有的文件，“-print”选项表示输出检索出来的文件名。**cpio**命令的“-p”选项表示采用复制模式，复制一个完整的目录。

执行上述命令之后，**cpio**将会把dir1目录的文件，按照原有的目录层次结构，完全复制到新的目录dir2中。进入dir2目录之后，即可看到新复制的目录或文件系统与原来完全一样。

下面以磁带（其设备文件名为/dev/rmt/0）为例，说明怎样利用**cpio**和**find**命令备份或恢复文件、目录，甚至整个文件系统。实际上，这里介绍的备份与恢复过程同样也适用于其他存储介质，如移动硬盘或CD/DVD等，只需把输入或输出文件名改为相应的设备文件名即可。

1. 使用cpio命令把文件复制到磁带上

为了把某个目录下的所有文件复制到一个磁带上，可以采用**find**等命令提供需要复制的文件列表，然后通过管道提交给**cpio**命令，再由**cpio**把文件复制到磁带上。例如，假定我们希望备份/export/home/gqxing/data目录中的所有数据文件，可首先进入该目录，然后执行下列**cpio**命令：

```
$ cd /export/home/gqxing/data
$ find . -print | cpio -ocvB > /dev/rmt/0
.
data.0630
data.0731
data.0831
data.0930
data.1031
1250992 blocks
$
```

在上述例子中，**find**命令用于提供**cpio**命令需要复制的文件。**cpio**命令中的“-o”选项表示以备份方式执行**cpio**命令，创建一个档案文件。“-c”选项表示采用ASCII字符格式写入头信息，以便确保新建档案文件的兼容性。“-v”选项表示在屏幕（标准输出）上显示复制的文件名。

“-B”选项表示以5120个字节的数据块为读写单位。“> /dev/rmt/0”表示指定的输出文件为磁带机。

执行上述命令之后，当前目录中的所有文件将会归档并复制到磁带上。如果磁带中有数据，原有的数据将会被覆盖。命令输出的最后一行数据是复制的数据块（512字节）数量，表示生成的档案文件的大小。在这个例子中，生成的档案文件约为610MB。

为了验证数据是否确实已经复制到磁带上，可以使用下列命令：

```
$ cpio -ict < /dev/rmt/0
.
data.0630
data.0731
data.0831
data.0930
data.1031
1250990 blocks
$
```

其中，“-t”选项意味着仅仅读取并显示档案文件中的文件列表。“< /dev/rmt/0”表示指定的输入文件为磁带机。输出结果表示，读取的文件内容与上述档案文件制作过程的输出结果完全一致。

2. 使用cpio命令把文件复制到多个磁带上

如前所述，当使用cpio命令，把单个文件、一组文件或整个文件系统复制到磁带上时，应注意整个数据的容量。如果数据量过大，一个磁带无法容纳所有的数据，需要使用多个磁带介质时，应注意使用“-I”选项，以便在磁带复制完成之后，由系统提示用户更换一个新的磁带，从而把档案文件备份到多个磁带上。例如：

```
$ cd /home/gqxing/data
$ find . -depth -print | cpio -ocvB -M "The tape has reached end. Please
replace it with #2. Press Enter when ready." -O /dev/rmt/0
```

当复制到中途时，系统将会输出下列提示信息，提醒用户更换新的磁带：

```
The tape has reached end. Please replace it with #2. Press Enter when ready.
```

更换新的磁带后，按下Enter键，系统将会继续复制文件，直至把所有的文件均写入磁带中。

在上述例子中，“-M”选项用于指定更换介质时输出的提示信息；“-O /dev/rmt/0”选项表示指定的输出文件为磁带机。

3. 使用cpio命令从档案文件中恢复所有的文件

由于先前在复制磁带档案文件时采用了相对路径名，当使用cpio命令读取磁带档案文件时，读入的文件将会存放到当前工作目录中。因此，在恢复文件之前可以任选一个目录。

例如，为了把先前备份到磁带上的档案文件恢复到系统中，首先选择一个目录，然后执行下列cpio命令（其中，“-v”选项表示在屏幕上显示抽取的文件）：

```
# cd /newdir
# cpio -icduvB < /dev/rmt/0
.
```

```
data.0630
data.0731
data.0831
data.0930
data.1031
1250992 blocks
#
```

执行上述命令后，将会把先前复制的所有文件恢复到选定的目录中。

4. 使用cpio命令从档案文件中恢复指定的文件

上述cpio命令把档案文件中的文件全部恢复到系统中。事实上，也可以从复制的档案文件中抽取单个文件或一组文件。为此，可以使用下列cpio命令：

```
$ cpio -icduv "pattern" < /dev/rmt/0
```

其中，“pattern”表示从档案文件中抽取匹配指定模式的所有文件，并恢复到当前工作目录中（注意，如果需要同时指定多个模式，每个模式前后必须加双引号）。下列例子说明了怎样抽取文件名后缀为“0831”的文件：

```
# cd /newdir
# cpio -icduv "*0831" < /dev/rmt/0
data.0831
1250992 blocks
# ls -l *0831
-rw-r--r--  1 gqxing  other    128101124 Aug 31  2009 data.0831
#
```

5. 使用cpio命令查询先前备份的档案文件

为了查询先前备份的档案文件，可以使用下列cpio命令（其中，“-t”选项表示仅仅读取并显示档案文件中的文件列表）：

```
# cpio -ict < /dev/rmt/0
.
data.0630
data.0731
data.0831
data.0930
data.1031
1250992 blocks
#
```

如果使用“-v”选项，cpio命令还会给出更多的文件属性信息。例如：

```
# cpio -icvt < /dev/rmt/0
drwxr-xr-x  2 gqxing  other          0 Jun 22 18:27 2009, .
-rw-r--r--  1 gqxing  other    128099615 Jun 30 23:30 2009, data.0630
-rw-r--r--  1 gqxing  other    128100453 Jul 31 23:30 2009, data.0731
-rw-r--r--  1 gqxing  other    128101124 Aug 31 23:30 2009, data.0831
-rw-r--r--  1 gqxing  other    128101891 Sep 30 23:30 2009, data.0930
-rw-r--r--  1 gqxing  other    128102717 Oct 31 23:30 2009, data.1031
1250990 blocks
#
```

11.2.2 利用tar实现备份和恢复

tar命令可用于备份数据，把指定的文件、目录或文件系统及其中的所有文件组合后生成一个新的档案文件。生成的档案文件可以存储在磁带、磁盘或光盘等移动介质上，也可以存储到磁盘文件系统中。用户可以检索档案文件，必要时也可以把档案文件恢复到系统中。

tar具有5种运行模式，或者说具有5种用途。

- 创建，利用“-c”选项，创建一个包含多个文件的档案文件，实现文件的备份。
- 替换，利用“-r”选项，把新文件写入档案文件后部，以便能够在恢复文件时实现新老文件的替换。
- 显示，利用“-t”选项，显示档案文件中的文件列表。
- 更新，利用“-u”选项，更新档案文件中的指定文件。
- 抽取，利用“-x”选项，把档案文件恢复到系统中。

tar命令的语法格式简写如下。表11-6给出了常用的tar命令选项及其简单说明。

```
tar [-]{ c | r | u } [options] file-list
tar [-]{ x | t } [options] [file-list]
```

表11-6 tar命令的常用选项

选项	说明
-c	创建。组合指定的文件，创建一个新的档案文件。如果同名的档案文件已经存在，在创建新的档案文件之前将会清除原有的档案文件
-r	替换。把指定的文件写到档案文件的后部。采用这个选项时，原有的同名文件将会继续保留在档案文件中，并与新的文件共存于档案文件内。当使用“-x”选项抽取文件时，最新的同名文件将会取代原有的文件而保留下来
-t	显示。显示档案文件中的文件列表。文件列表的输出格式类似于“ls -l”命令。如果使用此选项未指定文件参数，tar将会显示档案文件中的所有文件。如果指定了文件参数，tar只会列出与文件参数匹配的文件
-u	更新。如果档案文件中不存在，或指定的文件比档案文件中的同名文件更新，则把指定的文件写到档案文件的后部。由于这个选项要求执行更多的检查，故其运行速度相对较慢
-x	抽取或恢复文件。从档案文件中抽取指定的文件，按照档案文件的目录结构，复制到当前工作目录中。如果使用这个选项时未指定文件参数，tar将会抽取档案文件中的所有文件。如果指定了文件参数，tar只会抽出与文件参数匹配的文件。如果指定的文件参数含有目录，tar除抽取匹配的目录之外，还将递归地抽取目录中的所有文件。如果档案文件中包含若干同名的文件，后抽取的文件将会覆盖先前抽取的文件。在抽取文件时，tar将会尽可能地使抽取文件的文件属主、修改时间和访问权限等属性与档案文件中的初始文件保持一致。注意，在使用“-c”选项创建档案文件时，应尽可能采用相对路径名，否则，tar命令可能无法正确地匹配文件
-b n	在访问存储介质时，采用“n×512”个字节的逻辑数据块（n的默认值为20）作为读写单位，创建或读取档案文件。通常，在读取档案文件时，tar将会自动地确定逻辑数据块的大小
-f tarfile	指定档案文件。档案文件可以是一个普通文件，也可以是一个设备文件（如磁带机等）
-k n	指定存储介质的容量或档案文件的大小，以KB为单位（即“n×1024”个字节）。如果复制的档案文件大于此选项指定的容量，当写出n-KB的数据时，tar将会提示用户更换新的存储介质（如磁带），从而能够把档案文件分布存储到多个介质上。对于具有固定容量的存储介质（如磁带和软盘），这个选项是非常有用的，可用于创建多卷形式的档案文件

(续表)

选项	说明
-m	采用抽取文件时的时间作为文件的修改时间。如果未指定此选项，tar将恢复文件在档案文件中的初始修改时间。这个选项仅适用于“-x”选项
-o	以执行tar命令的用户及其所属用户组作为文件恢复后的用户和用户组，而不采用档案文件中文件原有的用户和用户组属性。对于普通用户而言，这是tar命令的默认处理方式。如果是超级用户运行tar命令，且没有指定此选项，抽取的文件将采用档案文件中的文件原有的用户和用户组属性。这个选项仅适用于“-x”选项
q	当抽取第一个匹配的文件时，立即退出tar命令。如果不加此选项，tar命令的常规处理惯例是继续读取档案文件，直至遇到文件结束标志
-v	显示处理过程中读写的每一个文件名。与“-t”选项一起使用时，tar将会列出文件的详细属性信息，如文件属主、访问权限和文件大小等，其输出结果类似于“ls -l”命令的输出
-X file	表示归档或抽取指定文件中未列举的其他所有文件（即排除指定文件中包含的任何文件）。如果指定的文件是一个目录，则排除指定目录及其中的任何文件和子目录

假定我们想要备份/export/home/gqxing/src目录中的所有文件，把归档后生成的档案文件存放在/data目录中，可以使用下列tar命令：

```
$ cd /export/home/gqxing
$ tar -cvf /data/atmsrc.tar src
a src/ 0K
a src/atmmon.c 26K
a src/atmcom.c 26K
a src/listener.c 30K
a src/handler.c 51K
a src/makefile 1K
$
```

必要时，可以从备份的档案文件中恢复指定的文件，或恢复全部文件。如果需要恢复归档的全部文件，可以使用下列tar命令：

```
$ cd /newdir
$ tar -xvf /data/atmsrc.tar
x src, 0 bytes, 0 tape blocks
x src/atmmon.c, 26282 bytes, 52 tape blocks
x src/atmcom.c, 25852 bytes, 51 tape blocks
x src/listener.c, 30625 bytes, 60 tape blocks
x src/handler.c, 52125 bytes, 102 tape blocks
x src/makefile, 712 bytes, 2 tape blocks
$
```

下面以1.44MB软盘（其设备名为/dev/rdiskette）为例，说明怎样使用tar命令备份或恢复单个文件、一组文件，甚至整个文件系统。实际上，这里介绍的备份与恢复过程同样也适用于其他存储介质（如磁带、磁盘或CD/DVD等），只需把设备文件名改为相应存储介质的设备文件名即可。

1. 使用tar命令把文件归档备份到软盘上

在使用tar命令把文件复制到存储介质上时，应当注意下列事项。

- 在复制文件时，可以使用文件名生成元字符“*”、“?”以及“[...]”指定文件名。例如，若想复制以“.doc”为扩展名的所有文件，可以使用“*.doc”作为文件名参数。
- 但在抽取匹配的文件时，不能使用元字符“?”、“*”和“[...]”指定文件名。
- 使用tar命令的“c”选项创建档案文件时，将会毁灭存储介质上原有的任何文件。
- 包含tar档案文件的存储介质不能作为文件系统安装。

为了把档案文件复制到软盘上，首先应进入需复制文件所在的目录，然后使用下列tar命令复制文件：

```
tar cvf /dev/rdiskette filenames
```

其中，“c”选项表示创建一个档案文件。“v”选项表示在屏幕（标准输出）上显示正在处理的每一个文件及其相关的属性信息。“f /dev/rdiskette”表示把创建的档案文件写到软盘中。filenames是准备归档的文件或目录名。示例如下：

```
$ cd /export/home/gqxing
$ ls -l src
total 268
-rw-r--r--  1 gqxing  other      25852 Jun 18 15:31 atmcom.c
-rw-r--r--  1 gqxing  other      26282 Jun 18 15:31 atmmon.c
-rw-r--r--  1 gqxing  other      52125 Jun 18 15:33 handler.c
-rw-r--r--  1 gqxing  other      30625 Jun 18 15:32 listener.c
-rw-r--r--  1 gqxing  other        712 Jun 18 15:34 makefile
$ tar -cvf /dev/rdiskette src
a src/ 0K
a src/atmmon.c 26K
a src/atmcom.c 26K
a src/listener.c 30K
a src/handler.c 51K
a src/makefile 1K
$
```

为了验证指定的文件是否已经复制到软盘上，可以使用下列命令（其中，“-t”选项表示读取并显示档案文件中的文件列表）：

```
$ tar -tvf /dev/rdiskette
drwxr-xr-x 100/1      0 Jun 18 15:34 2009 src/
-rw-r--r-- 100/1      26282 Jun 18 15:31 2009 src/atmmon.c
-rw-r--r-- 100/1      25852 Jun 18 15:31 2009 src/atmcom.c
-rw-r--r-- 100/1      30625 Jun 18 15:32 2009 src/listener.c
-rw-r--r-- 100/1      52125 Jun 18 15:33 2009 src/handler.c
-rw-r--r-- 100/1        712 Jun 18 15:34 2009 src/makefile
$
```

2. 使用tar命令把文件归档备份到多个软盘上

当使用tar命令把档案文件写入软盘时，应注意整个档案文件的大小和软盘的容量。如果数据量过大，一个软盘无法容纳所有的数据，因而需要使用多个软盘时，可以使用“-k”选项指定软盘的容量，以便能够在在一个软盘复制完成之后，由系统提示用户更换新的软盘。这里使用“-k 1440”选项指定1.44MB软盘的容量。

```

$ cd /export/home/gqxing
$ tar -cvfk /dev/rdiskette 1440 doc
Volume ends at 1439K, blocking factor = 10K
a doc/ 0K
a doc/UserGuide 582K
a doc/RefManual 489K
tar: large file doc/TechSpecification needs 2 extents.
tar: current device seek position = 1072K
+++ a doc/TechSpecification 367K [extent #1 of 2]
tar: please insert new volume, then press RETURN.
+++ a doc/TechSpecification 14K [extent #2 of 2]
a doc/TechSpecification 381K (in 2 extents)
$

```

当复制到中途时，系统将会提醒用户更换新的软盘。在更换下一个软盘之后，按下Enter键，系统将会继续执行文件的复制，直至把所有的文件均写入软盘中。

3. 使用tar命令从档案文件中恢复所有的文件

为了从先前备份到软盘的档案文件中恢复所有的文件，首先应进入某个选定的目录，然后使用下列命令抽取其中的文件：

```
tar -xvf /dev/rdiskette [filenames]
```

其中，“-x”选项表示从指定的档案文件中抽取文件，并把抽取的文件恢复到当前目录中。

“-v”选项表示在屏幕（标准输出）上显示抽取的每一个文件及其相关的属性信息。filenames是准备抽取的文件。如果同时指定多个文件，文件名之间应加空格字符。通常，如果未指定文件名参数，tar命令将会抽取所有的文件。示例如下：

```

$ cd /var/tmp
$ tar -xvf /dev/rdiskette
x src, 0 bytes, 0 tape blocks
x src/atmmon.c, 26282 bytes, 52 tape blocks
x src/atmcom.c, 25852 bytes, 51 tape blocks
x src/listener.c, 30625 bytes, 60 tape blocks
x src/handler.c, 52125 bytes, 102 tape blocks
x src/makefile, 712 bytes, 2 tape blocks
$

```

4. 使用tar命令从档案文件中抽取指定的文件

如前所述，在“tar -x”命令中，如果命令行中未指定文件参数，则意味着抽取档案文件中的所有文件。为了抽取特定的文件，需要在命令行中指定文件名。指定文件名时必须严格匹配档案文件中的文件名。如果无法肯定文件名或路径名，可以使用下面即将介绍的“-t”选项，获取档案文件中的完整文件列表。

假定需要从先前备份的档案文件中抽取atmmon.c文件，可以使用下列命令：

```

$ cd /newdir
$ tar -xvf /dev/rdiskette src/atmmon.c
x src/atmmon.c, 26282 bytes, 52 tape blocks
$ ls -l src
total 52

```

```
-rw-r--r-- 1 gqxing other 26282 Jun 18 15:31 atmmon.c
$
```

注意：在抽取特定的文件时，指定的文件名必须与档案文件中的文件名完全匹配。否则，tar命令无法抽取预期的文件。例如，为了完全匹配“./thisfile”，抽取的文件名必须严格指定为“./thisfile”，而不能指定为“thisfile”。同样，为了抽取“src/atmmon.c”，给定的文件名必须严格指定为“src/atmmon.c”，而不能指定为“atmmon.c”。这是因为tar把整个目录和文件的路径名看做一体。

在tar命令中，如果想要恢复部分文件，需要逐一列出准备抽取的文件，不能使用元字符的文件名生成机制指定抽取的文件，但可以利用下列命令达到同样的目的：

```
tar -xvf /dev/rdiskette `tar -tf /dev/rdiskette | grep 'pattern'`
```

例如，为了抽取档案文件中所有以“.c”为后缀的文件，可以使用下列命令：

```
# tar -xvf /dev/rdiskette `tar -tf /dev/rdiskette | grep '.c$'`
x src/atmmon.c, 26282 bytes, 52 tape blocks
x src/atmcom.c, 25852 bytes, 51 tape blocks
x src/listener.c, 30625 bytes, 60 tape blocks
x src/handler.c, 52125 bytes, 102 tape blocks
#
```

5. 使用tar命令查询先前备份的归档文件

为了查询先前备份的档案文件，可以使用下列命令：

```
tar -tvf /dev/rdiskette
```

其中，“-t”选项表示读取并显示档案文件中的文件列表。“-v”选项表示在屏幕（标准输出）上显示每一个文件及其相关的属性信息。例如：

```
$ tar -tvf /dev/rdiskette
drwxr-xr-x 100/1 0 Jun 18 15:34 2009 src/
-rw-r--r-- 100/1 26282 Jun 18 15:31 2009 src/atmmon.c
-rw-r--r-- 100/1 25852 Jun 18 15:31 2009 src/atmcom.c
-rw-r--r-- 100/1 30625 Jun 18 15:32 2009 src/listener.c
-rw-r--r-- 100/1 52125 Jun 18 15:33 2009 src/handler.c
-rw-r--r-- 100/1 712 Jun 18 15:34 2009 src/makefile
$
```

11.2.3 利用dd实现数据的复制

简而言之，dd命令的基本功能是把标准输入复制到标准输出中。但其最大特点是能够采用原始读写方式，逐块逐道地把存储介质中的数据原封不动地复制到另一个存储介质上。此外还可以把一个文件系统完整地复制到另外一个磁盘分区中，从而实现文件系统的备份。

在复制过程中，dd命令还具有数据转换功能，可以把不同代码格式的数据转换为另外一种编码的数据，实现不同系统之间的数据交换。

与其他大多数命令相比，dd命令的语法格式明显不同。在dd命令中，命令选项采用“keyword=value”的形式。其中，keyword相当于其他命令中的选项，value是选项的参数。例如，当需要复制某个文件时，可以采用下列简单的命令语法格式：

`dd if=input-file of=output-file`

表11-7给出了dd命令的部分常用选项、参数及其说明。

表11-7 dd命令的部分常用选项

选项与参数	说明
<code>if=file</code>	指定输入文件。文件可以是普通数据文件，也可以是磁盘、磁盘分区、磁带、软盘或CD/DVD等设备文件。默认值为标准输入
<code>of=file</code>	指定输出文件。文件可以是普通数据文件，也可以是磁盘、磁盘分区、磁带、软盘或CD/DVD等设备文件。默认值为标准输出
<code>ibs=n</code>	指定输入数据块的大小，以字节为单位。如果未指定，默认值为512个字节
<code>obs=n</code>	指定输出数据块的大小，以字节为单位。如果未指定，默认值为512个字节
<code>bs=n</code>	指定读写数据块的大小，以字节为单位。如果未指定，默认值为512个字节。此选项可以强制替代ibs和obs两个选项
<code>skip=n</code>	在开始复制之前，先从输入文件的起始位置跳过指定数量的数据块（以ibs或bs定义的输入数据块为计数单位）
<code>seek=n</code>	在开始复制之前，先从输出文件的起始位置跳过指定数量的数据块（以obs或bs定义的输出数据块为计数单位）
<code>iseek=n</code>	在开始复制之前，先从输入文件的起始位置跳过指定数量的数据块（以指定的输入数据块大小为计数单位）
<code>oseek=n</code>	在开始复制之前，先从输出文件的起始位置跳过指定数量的数据块（以指定的输出数据块大小为计数单位）
<code>count=n</code>	指定复制的数据块数量。每个数据块的字节数以ibs、obs或bs的定义为准

例如，为了复制一个软盘，可以使用dd命令先把软盘复制到一个临时文件中：

```
$ dd if=/dev/rdiskette of=/tmp/tmpfile
2880+0 records in
2880+0 records out
$
```

然后，再使用dd命令，交换输入和输出文件的位置，把临时文件复制到一个新的软盘中：

```
$ dd if=/tmp/tmpfile of=/dev/rdiskette
2880+0 records in
2880+0 records out
$
```

运行结束之后，dd命令将会给出已经读写的数据块数量。其中，加号“+”之前的数字表示复制过程中读写的完整数据块数量。加号“+”之后的数字表示复制了多少个不完整的数据块。

为了把磁盘分区2中的文件系统复制到磁盘分区3中，可以使用下列命令：

```
# dd if=/dev/rdsk/c0t0d0s2 of=/dev/rdsk/c0t0d0s3 bs=1024K
xxxxxx+0 records in
xxxxxx+0 records out
#
```

然后，可以使用“fsck -f”命令，检测新复制的文件系统。最后再使用mount命令安装新

的文件系统。

```
# fsck -f /dev/rdisk/c0t0d0s3
.....
# mount /dev/dsk/c0t0d0s3 /mnt
#
```

在使用dd命令复制磁盘分区（文件系统）时，应注意下列事项。

- 确保源磁盘分区与目的磁盘分区具有相同的容量（且最好为同一型号的磁盘）。
- 使用fsck命令检测新复制的文件系统。然后再利用mount命令予以检验。
- 使用dd命令复制磁盘分区时，应确保系统处于单用户运行模式，从而保证源磁盘分区中的数据不会发生变化。

同样，dd命令也可用于复制磁带。如果系统配有两个磁带机，可以使用下列命令，实现从磁带到磁带的直接复制。否则，需要先把数据复制到磁盘上，然后再复制到新的磁带中。

```
# dd if=/dev/rmt/0 of=/dev/rmt/1
```

11.3 文件系统限额管理

本节以UFS文件系统为例，说明文件系统磁盘空间和信息节点的限额设置。

11.3.1 限额概述

1. 什么是限额

限额机制使系统管理员能够控制文件系统的空间分配与使用。利用限额可以限制每个用户能够使用的磁盘空间和信息节点数量（也即文件的数量）。由于这个原因，对于用户主目录所在的文件系统，限额是特别有用的。而在共用的文件系统（如/tmp）中建立限额则没有太多的实际意义。

即使已经设定了限额，也可以随时调整用户可用的磁盘空间与信息节点的数量，而且可以根据系统需求的变化增加甚至删除限额。

另外，限额的状态也可以监控。作为系统管理员，可以利用repquota和quota命令显示文件系统的限额信息，或检索超限的用户。

2. 限额的软性设置与硬性设置

在设置磁盘空间的限额时，可以采用软性限制和硬性限制两种方式。系统不允许用户超越其硬性限制，但允许用户临时超越其软性限制。软性限制必须小于硬性限制，一旦用户超越了软性限制，系统将会给予一个宽限周期，同时启动一个限额时钟。在宽限周期之内，允许用户暂时超越软性限额进行操作，但不能超越硬性限额。一旦由于清理文件使得磁盘空间低于软性限制，限额时钟将会置零。但是，如果在超过软性限制后的宽限周期之内用户一直未采取任何措施，一旦限额时钟到期，软性限制将强行变为硬性限制。默认情况下，超越软性限制的宽限周期为7天时间。

repquota和quota命令中的timeleft字段将给出宽限周期的设置。例如，假定用户的软性限制为10 000个数据块，硬性限制为12 000个数据块。如果已使用了10 000多个数据块，且7天的宽限周期也已到期，在清理文件直至其占用的空间低于软性设置之前，用户不能继续申请文件系

统中的磁盘存储空间，分配更多的数据块。

3. 数据块与文件数量限制之间的差别

文件系统能够为用户提供两种资源：用于存储文件数据的数据块，用于创建文件的信息节点。每个文件均占用一个信息节点，而文件的实际数据将存储在数据块中。在限额管理中，数据块通常以1KB为单位。

假定不考虑目录，一个用户即使不占用任何数据块，如完全创建一系列空文件，也可能超过其信息节点的限制。另外，即使仅用一个信息节点，只需令文件大得足以耗尽用户限额规定的所有数据块数量，也可能超过其存储空间限额。

4. 设置限额的步骤

为了设置文件系统的限额，可以参照下列步骤进行。

(1) 设置文件系统，使之支持限额控制。为了在文件系统中设置限额，需要编辑/etc/vfstab文件，在文件系统的安装选项中增加一个“rq”限额标志，以确保每次引导系统时都会强制实行文件系统的限额控制。

(2) 此外，还需要在文件系统的顶级目录中创建一个空的quotas文件，确保只有超级用户才能读取此文件。

(3) 使用edquota命令，为每个用户设置磁盘空间和信息节点限额，包括软性限制、硬性限制和宽限周期。

(4) 在启用限额设置之前，使用quotacheck命令，检测每个文件系统的限额设置与实际使用情况是否一致，确保两者之间没有冲突。如果系统很少关机重启，需要周期地运行quotacheck命令。

(5) 使用quotaon命令启用文件系统的限额设置。在启用文件系统的限额控制之前，上述限额设置并不会立即发生作用。但是，如果已经适当地设置了/etc/vfstab文件，并在准备实行限额控制的文件系统根目录中创建了quotas文件之后，则每当重新引导系统，在安装了已经实行限额控制的文件系统之后，系统将会自动地启用限额控制。

5. 设置与维护限额的工具

表11-8给出了可用于设置和维护文件系统与用户限额的工具及其简单说明。

表11-8 限额的设置与维护工具

命令	基本功能
edquota	调用默认的vi编辑器，设置、修改和关闭指定用户的磁盘空间与信息节点限额，包括硬性限制和软性限制
quotacheck	考察已安装的每个文件系统，检测文件系统的限额设置与实际使用情况的一致性，初始化限额数据库
quotaon	激活指定文件系统的限额设置与控制
quotaoff	关闭指定文件系统的限额设置与控制
quota	查询用户的限额设置和实际磁盘使用情况，检查是否存在超限的用户，也可据以确认限额是否已经正确地设置
repquota	查询文件系统的限额设置，显示用户限额设置与使用情况（包括磁盘空间和文件数量）的汇总信息

6. 设置限额的原则

在设置限额之前，系统管理员首先需要确定究竟应为每个用户分配多少磁盘空间和信息节点。如果想要绝对保证所有用户占用的磁盘总量不会超过文件系统的全部空间，可以按用户数平均分配每个用户的可用文件系统存储空间。例如，如果4个用户共享一个1GB的文件系统，且每个用户需要的磁盘空间大体相同，则可以按均分的方式为每个用户分配256MB的磁盘空间。

在一个实际的应用系统环境中，并非所有的用户都会同样地竞相使用或超限使用磁盘存储空间，因而可以分别为用户设置限额，并分配一个大于平均数的存储空间，使设定的全部用户存储空间大于文件系统的实际容量。例如，如果4个用户共享一个1GB的文件系统，则可以为每个用户分配300MB的磁盘空间。

11.3.2 设置限额

如前所述，为了在文件系统中实行限额控制，需要编辑/etc/vfstab文件，创建适当的限额文件，启用文件系统限额控制，以及设置用户限额等。

1. 设置文件系统限额标志

编辑/etc/vfstab文件，对于准备实行限额控制的每一个文件系统，在其安装选项字段中增加一个“rq”限额标志，重新安装相应的文件系统，或重新启动系统，以便使文件系统的限额设置发生作用。例如，为了在“/export/home”文件系统中实行限额控制，可修改/etc/vfstab文件，把下列行：

```
/dev/dsk/c1d0s7    /dev/rdisk/c1d0s7  /export/home    ufs    2    yes    -
```

改为：

```
/dev/dsk/c1d0s7    /dev/rdisk/c1d0s7  /export/home    ufs    2    yes    rq
```

2. 创建限额文件

使用下列touch命令，在相应文件系统的根目录（如export/home）中创建一个空的quotas文件：

```
# touch /export/home/quotas
#
```

使用下列命令，修改新建限额文件的访问权限，使得只有超级用户才能够读写此限额文件：

```
# chmod 600 /export/home/quotas
#
```

3. 维护限额文件

在系统的引导过程中，通过/etc/init.d/ufs_quota启动脚本，系统将会自动地运行quotacheck命令。quotacheck命令用于考查每一个文件系统的使用情况，必要时创建、初始化限额文件，检测与维护文件系统的限额设置与限额文件的一致性，更新限额文件。

对于新增的空文件系统，即使设定了限额，通常也不必运行quotacheck命令。但是，如果在一个已经建有用户文件的文件系统中设置限额，则需要运行quotacheck命令，检查每个文件系统中用户已经占用的磁盘空间和信息节点，然后对限额文件进行同步与更新。而且还应记住，在较大的文件系统中运行quotacheck命令是一项非常耗时的处理任务。

限额初始化的最终目的是生成一个限额文件。限额文件quotas中包含了相应文件系统的用户限额设置、用户已经使用的存储空间，以及用户已经创建的文件或目录数量等信息。在用户访问系统期间，UNIX系统会自动和透明地更新这个文件，并根据其中的限额设置，确定用户是否有权使用更多的存储空间，或创建更多的文件。

为了初始化或更新限额文件，或者在之后定期地检测文件系统的限额设置与限额文件的一致性，可以使用下列quotacheck命令：

```
quotacheck [-a [-v]] | [-v filesystem]
```

其中，“-v”选项表示检测指定文件系统中每个用户的空间限额。“-a”选项表示检测与维护/etc/vfstab文件中具有限额标志的所有文件系统的限额设置。filesystem用于指定需要检测限额设置的文件系统。

下面的例子说明了如何检测/export/home文件系统（其磁盘分区为/dev/rdisk/c0t0d0s7）的限额设置（假定“/export/home”是/etc/vfstab文件中唯一具有限额标志的文件系统）：

```
# quotacheck -va
*** Checking quotas for /dev/rdisk/c1d0s7 (/export/home)
#
```

注意：为了确保磁盘数据的准确性，在手工运行quotacheck命令时，应尽量确保文件系统处于静止状态。

4. 设置用户限额

在设置用户限额时，可以首先选定一个用户，参照下列步骤执行。

(1) 使用“edquota username”命令调用vi编辑器，创建一个临时文件，对于实行了限额控制的每一个文件系统，edquota命令将会自动增加一行限额信息，其初始数据内容如下（假定整个系统中只有“/export/home”文件系统实行了限额控制）：

```
# edquota gqxing
fs /export/home blocks (soft = 0, hard = 0) inodes (soft = 0, hard = 0)
```

(2) 针对实行了限额控制的每个文件系统，利用vi编辑命令修改限额文件，把存储空间的软性和硬性限制分别设为适当的限额值（如20000和204800个1KB的数据块），把信息节点的软性和硬性限制分别设为适当的限额值（如为4000和4096个文件）。保存限额设置后退出edquota命令。示例如下：

```
fs /export/home blocks (soft = 20000, hard = 204800) inodes (soft = 4000, hard = 4096)
```

(3) 使用“quota [-v] username”命令，验证指定用户的限额是否已经适当地设置。其中，“-v”选项用于显示指定用户在实行了限额控制的文件系统中的限额信息。

```
# quota -v gqxing
Disk quotas for gqxing (uid 100):
Filesystem      usage quota  limit  timeleft   files  quota  limit  timeleft
/export/home      0  20000 204800          0   4000   4096
#
```

在上述输出信息中，每个字段的意义说明如下：

- Filesystem 文件系统的安装点。

- **usages** 用户当前使用的磁盘空间数量（以1KB的数据块为单位）。
- **quota** 数据块限额的软性限制。当超越软性限额设置时，用户将会收到警告信息。超过宽限周期后如果一直未采取措施，系统将会禁止用户使用附加的磁盘空间。
- **limit** 数据块或信息节点数量限额的硬性限制。如果这个数值为0，则意味着未加限制。用户能够超过其软性限额设置，但不能超过其硬性限额设置。
- **timeleft** 当数据块限额时钟因超限开始计时后，按天数计算的剩余时间。
- **files** 用户当前已有的文件数量（当前信息节点的实际使用情况）。
- **quota** 信息节点数量限额的软性限制。当超越软性限额设置时，用户将会收到警告信息。超过宽限周期后如果一直未采取措施，系统将会禁止用户创建新的文件。
- **limit** 信息节点数量限额的硬性限制。如果这个数值为0，则意味着未加限制。用户能够超过其软性限额设置，但不能超过其硬性限额设置。
- **timeleft** 当信息节点限额时钟因超限开始计时后，按天数计算的剩余时间。

5. 为其他用户设置限额

在为选定的第一个用户设置了限额之后，可以用此限额设置作为模板，利用下列**edquota**命令为其他用户设置限额：

```
edquota -p prototype-user username
```

其中，**prototype-user**是用做限额设置模板的用户名。**username**是尚未设置限额的其他用户名（如果同时指定多个用户名，用户名之间需加空格分隔符）。

下面的例子说明了怎样利用用户**gqxing**的限额设置模板，为**cathy**、**david**与**sarah**等其他用户设置限额：

```
# edquota -p gqxing cathy hwang
#
```

6. 启用限额控制

为了启用文件系统的限额控制，可以使用下列命令：

```
quotaon [-a [-v]] | [-v filesystem]
```

其中，“-v”选项意味着显示激活限额设置的每一个文件系统。“-a”表示激活/etc/vfstab文件中具有限额标志的所有文件系统的限额设置。**filesystem**表示需要激活限额设置的文件系统。如果同时指定多个文件系统，文件系统名之间应加空格分隔符。

在设置了文件系统的限额标志，以及设置**quotas**文件之后，每当重新启动系统时，都会自动启用文件系统的限额控制。下面的例子说明了怎样在必要时手工激活某个指定文件系统（如/export/home）的限额设置：

```
# quotaon -v /export/home
/export/home: quotas turned on
#
```

11.3.3 限额的维护

在设置限额、启用存储空间限额和信息节点限额控制之后，系统管理员可以检查超过限额设置的用户，检查文件系统的限额控制信息。在合理的情况下，通过调整用户能够继续使用的

存储空间或信息节点数量，可以修改和删除限额设置。如果必要，也可以关闭单个用户，甚至关闭整个文件系统的限额控制。

下面具体列出了文件系统限额维护的若干任务及其简单说明。

- 检测用户的限额设置。使用**quota**命令，针对已经实行限额控制的文件系统，检查指定用户的限额设置及其实际使用情况。

- 检测文件系统的限额设置。使用**repquota**命令，针对已经实行限额控制的文件系统，检查所有用户的限额及其实际使用情况。

- 修改宽限周期的默认值。使用**edquota**命令，针对已经超过存储空间和信息节点限额限制的用户，修改其剩余时间长度的限制。

- 修改用户的限额设置。使用限额编辑器**edquota**，修改指定用户的限额控制。

- 关闭用户的限额设置。使用限额编辑器**edquota**，关闭指定用户的限额控制。

- 关闭文件系统的限额设置。使用**quotaoff**命令，关闭指定文件系统的限额控制。

1. 检测用户的限额设置

作为系统管理员，可以使用下列**quota**命令，针对已经实行限额控制的文件系统，检查指定用户的限额设置，确定是否存在超限使用的情况。

```
# quota [-v] username
```

其中，“-v”选项表示显示指定用户在实行限额控制的文件系统上的限额。**username**是用户名或用户ID。

下面的例子说明，用户**cathy**的软性和硬性存储空间限额分别为200000和204800个1KB的数据块，目前已经使用了11200个1KB的数据块。软性和硬性信息节点限额分别为4000和4096个，目前已经创建了1866个文件。示例如下：

```
# quota -v cathy
Disk quotas for cathy (uid 101):
Filesystem      usage quota    limit   timeleft  files   quota  limit  timeleft
/export/home      0 20000  204800          0    4000  4096
#
```

2. 检测文件系统的限额设置

利用**repquota**命令，系统管理员可以检查所有用户的文件系统限额设置和存储空间的实际使用情况。

```
repquota [-a [-v]] | [-v filesystem]
```

其中，“-v”选项意味着显示所有用户的限额设置，包括没有占用任何资源的用户。“-a”选项意味着显示所有文件系统的限额设置。**filesystem**表示仅显示指定文件系统的限额设置。

下面是一个执行**repquota**命令时的输出实例（系统中只有“/”文件系统实行了限额控制）：

```
# repquota -av /export/home
/dev/dsk/c1d0s7 (/export/home):

          Block limits
User      used   soft  hard   timeleft   used   File limits
          --    --    --    --    --    --    soft  hard   timeleft
gqxing    --     0   20000 204800          0    4000 4096
cathy     --     0   20000 204800          0    4000 4096
```

```

hwang      --      0    20000 204800      0      4000 4096
#

```

其中，“Block limits”部分各个字段的说明如下：

- **used** 当前实际使用的数据块数量；
- **soft** 数据块的软性限制；
- **hard** 数据块的硬性限制；
- **timeleft** 宽限周期。当超过限额设置时，这个字段为宽限周期剩余的天数。

“File limits”部分各个字段的说明如下：

- **used** 当前实际使用的信息节点；
- **soft** 信息节点的软性限制；
- **hard** 信息节点的硬性限制；
- **timeleft** 宽限周期。当超过限额设置时，这个字段为宽限周期剩余的天数。

第二列的两个字符位置分别表示是否超过数据块和信息节点的限额设置。其中“-”表示两者均未超过限额设置。如果某一项超过了限额设置，相应的字符位置为加号“+”。

3. 修改限额的宽限周期

限额的宽限周期是在超过软性限制之后系统强行禁止用户继续使用系统资源的缓期时间。默认情况下，在超过软性限额但未超过宽限周期规定的一周时间内，用户能够超限使用文件系统的资源。超过宽限周期之后，系统将会禁止用户继续申请存储空间或信息节点。

如果需要，可以使用**edquota**命令和**vi**的编辑命令，把系统默认的宽限周期改为一个适当的时间限制。在修改宽限周期时，设定的时间限制可以包含一个数值和一个表示时间单位的关键字，如**days**、**hours**、**minutes**和**seconds**，分别表示天、小时、分和秒等。注意，数字和关键字之间没有空格。

运行“**edquota -t**”命令（“-t”选项表示编辑每个文件系统的宽限周期）之后，将会进入**vi**编辑器，打开一个临时文件，同时给出当前的宽限周期，其中的数值0意味着系统默认的宽限周期为一周。

```
fs /export/home blocks time limit = 0 (default), files time limit = 0 (default)
```

下面是经过修改后的同一临时文件，其中把超过数据块限额的时间限制改为2周。同时，把超过文件数限额的时间限制改为16天。

```
fs /export/home blocks time limit = 2 weeks, files time limit = 16 days
```

增加宽限周期之后，用户能够继续超限使用存储空间或信息节点。注意，这个过程并不影响当前已经超限的用户。

4. 修改与关闭用户的限额

如果限额设置不当，可能会影响用户的正常工作，必要时可以修改其限额。为了修改指定用户的限额设置，可以使用**edquota**命令，针对实行了限额控制的每一个文件系统，按照软性和硬性限制，重新设定其1KB数据块的磁盘空间和信息节点的数量。

为了关闭用户的限额设置，仍然可以使用**edquota**命令，针对已经实行了限额控制的每个文件系统，只需把其中已设定的软性和硬性磁盘空间限额，以及软性和硬性信息节点限额均设置为0即可。注意，最好不要删除相应的文本行。

5. 关闭文件系统的限额设置

作为系统管理员，为了关闭文件系统的限额设置，可使用下列命令：

```
quotaoff [-a [-v]] | [-v filesystem]
```

其中，“-v”选项意味着显示关闭限额设置的每一个文件系统。“-a”选项表示关闭所有文件系统的限额设置。**filesystem**用于指定欲关闭其限额设置的文件系统。如果同时给出多个文件系统，文件系统名之间应加空格分隔符。

下面的例子说明了怎样关闭“/export/home”文件系统的限额设置：

```
# quotaoff -v /export/home  
/export/home: quotas turned off  
#
```


第12章 TCP/IP网络管理

网络与通信是操作系统的一个重要组成部分，UNIX系统提供丰富的网络通信功能。本章主要介绍IPv4（即传统的TCP/IP），以及网络接口、网关和路由等的设置，讨论TCP/IP网络的管理与维护。

12.1 TCP/IP简介

12.1.1 TCP/IP协议的层次结构

ISO的OSI网络参考模型描述了一种理想的网络层次结构及相应的通信协议，但TCP/IP与OSI的层次模型并不完全吻合。TCP/IP或者没有相应的结构层次，或者干脆把若干层合并为一层。表12-1展示了TCP与OSI模型的对照关系。

表12-1 TCP/IP层次结构与OSI层次模型对照表

OSI 分层	OSI 层次模型	TCP/IP 层次结构	TCP/IP协议栈
7	应用层	应用层	Telnet、SSH、FTP、TFTP、SFTP、SCP、SMTP、DNS、NIS、LDAP、SNMP、NFS等 套接字、端口
6	表示层		
5	会话层		
4	传输层	传输层	TCP、UDP
3	网络层	Internet层	IP、ICMP、RIP
2	数据链路层	网络访问层	ARP、RARP IEEE 802.2、PPP、设备驱动程序 Ethernet、IEEE 802.3、FDDI、ATM、Token Ring等
1	物理层		

表12-1也给出了TCP/IP协议栈中每个协议层支持的协议，了解这些协议与协议层的对应关系，有助于加深对TCP/IP协议的理解，也有助于在应用过程中分析与排查网络故障。

1. 网络访问层

在TCP/IP协议中，网络访问层涵盖了OSI层次模型的物理层、数据链路层和部分网络层的功能。网络访问层的主要功能如下：

- 利用地址解析协议ARP（Address Resolution Protocol），把IP地址映射为介质访问控制地址，即MAC（Media Access Control）地址，实现IP地址与MAC地址之间的转换。
- 把来自Internet层的数据报封装为网络上实际传输的数据帧，并增加数据帧的头信息。其中包括源和目的MAC地址和CRC循环冗余校验等字段。
- 利用设备驱动程序及实际的网络设备和传输介质，提供错误控制，以及按帧发送与接收数据的功能，以数据帧的形式及其错误控制机制，实现数据的传输。

2. Internet层

Internet层也称做**IP**层，其主要功能是接收和发送来自传输层或网络访问层的数据。**Internet**层包括强有力的**IP**协议和**ICMP**协议。

IP协议（**Internet Protocol**）。在**TCP/IP**协议系列中，**IP**协议及其有关的路由协议也许是最重要的协议。**IP**支持下列功能。

- **IP地址**——**IP**地址是**IP**协议的一个重要组成部分，用于确定数据传输的源和目的主机，确定数据传输的路由。
- 负责网络访问层与传输层之间的数据传输——基于接收系统的**IP**地址，**IP**协议负责确定分组数据到达目的系统必须经由的路径，实现主机与主机间的通信。
- 定义和封装数据报——数据报是**Internet**数据传输的基本单位，是**Internet**协议定义的分组数据格式。传输数据时，**IP**协议层把来自传输层的分组数据进一步细分为较小的数据报（**datagram**），然后通过网络访问层实现数据的传输。
- 数据的分片与装配——如果分组数据太大，发送系统的**IP**协议层将会把数据报分成较小的数据片（**fragment**）。而接收系统中的**IP**协议层则需要把数据片重组为初始的数据报。
- 利用路由信息协议**RIP**和**ICMP**路由发现（**Router Discovery**）协议等，收集、转发和维护路由表，为数据的传输提供路由服务。

ICMP协议（**Internet Control Message Protocol**）。**ICMP**协议的主要功能是控制、检测和报告网络错误，其中包括以下内容。

- 数据流控制——当分组数据到达得太快，接收系统来不及处理，或因网络通信问题导致分组数据中途丢失时，目的主机或中间的路由器将会向发送主机回送“**ICMP Source Quench Message**”报文，通知发送主机暂时停止发送数据。
- 检测网络或主机的连接问题——因网络硬件或协议软件故障，导致分组数据无法到达目的系统时，**ICMP**协议将会向发送分组数据的主机回送“**Destination Unreachable Message**”报文，表明无法联系远程主机。
- 重定向路由——通过**ICMP**重定向报文（**Redirect Message**），通知发送主机使用另外一个路由。
- 检测远程主机——通过发送**ICMP**回显报文（**Echo Message**），本地主机可以检测远程主机的网络协议是否已处于工作状态。当远程主机收到回显报文时，将向发送主机回送同样的分组数据。**UNIX**系统中的**ping**命令就是利用**ICMP**回显报文协议实现的。

3. 传输层

传输层的协议包括传输控制协议（**Transmission Control Protocol, TCP**）和用户数据报协议（**User Datagram Protocol, UDP**）。通过建立主机与主机之间的通信连接、接收数据后的确认，以及丢失分组数据后的重传等机制，**TCP**确保分组数据能够准确无误地按顺序到达接收系统。这种类型的数据通信称做面向连接的、端到端的通信。**TCP**能够提供可靠的、端到端的通信服务，而**UDP**只能提供非可靠的数据报服务。

TCP协议。**TCP**使应用程序能够实现主机到主机的通信。当收到应用层提供的数据流后，**TCP**协议将会把数据流分为一系列**TCP**协议能够识别的数据段（**segment**），并在每个数据段之前增加一个头信息。头信息包含源和目的端口号、数据段的序号以及校验和等字段，以便目

的主机能够正确地接收数据，并把数据传输到相应的应用程序中。

通过在源和目的主机之间建立端到端的连接，TCP能够确保准确无误地把数据发送到目的主机。故通常把TCP称做“可靠的、面向连接的”协议。

UDP协议。UDP协议提供数据报传输服务。UDP协议并不负责建立和检验发送主机与接收主机之间的通信连接。因此，只有传输少量数据的应用程序才使用UDP协议。实际上，由于协议层的开销较少，在网络条件和接收主机性能比较好的情况下，使用UDP协议比使用TCP协议的效率要高。

4. 应用层

应用层定义任何用户均可使用的标准Internet服务和网络应用。这些服务通过传输层发送或接收数据。TCP/IP应用层提供的服务和应用如下：

- 标准的TCP/IP服务，如文件传输协议FTP、TFTP和Telnet等；
- 名字服务DNS和NIS等；
- 目录服务LDAP；
- 网络文件服务NFS；
- 简单邮件传输协议SMTP；
- 简单网络管理协议SNMP。

12.1.2 TCP/IP协议如何处理数据通信

当用户输入一个TCP/IP应用的命令时，将会触发一系列处理过程。用户的命令或数据将会穿过本地系统的TCP/IP协议栈，经由网络介质到达远程系统的相应协议层。发送系统的每一层协议将会在初始的数据中增加协议控制头信息，而接收系统的每个协议层将会逐层剥离相应的协议控制头信息。两个系统之间的同一层协议按照协议的约定进行交互，实现数据的封装、传输和剥离，如图12-1所示）。

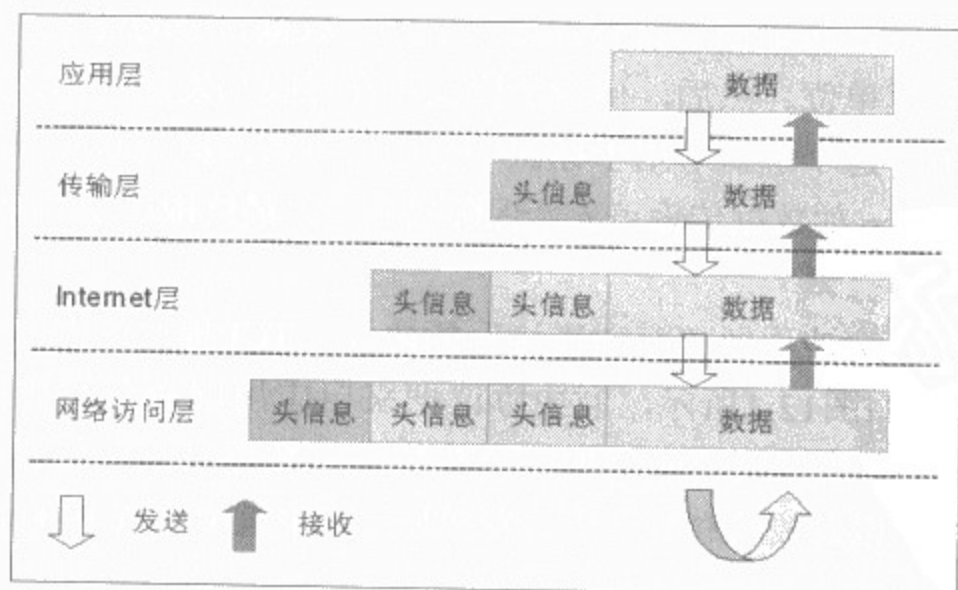


图12-1 分组数据经由TCP/IP协议栈的处理过程

1. 数据封装与TCP/IP协议栈

分组是网络传输的基本数据单位。一个分组数据是由发送和接收主机地址等头信息和实际数据组成的。当分组数据经过TCP/IP协议栈时，每个协议层会增加或者删除必要的协议字段。在TCP/IP协议中，通常把发送系统每个协议层增加头信息的处理过程称做封装。注意，每个协

议层所用数据单位的术语并不完全相同（在不致引起混淆的情况下，通常统称为分组数据），如图12-2所示。

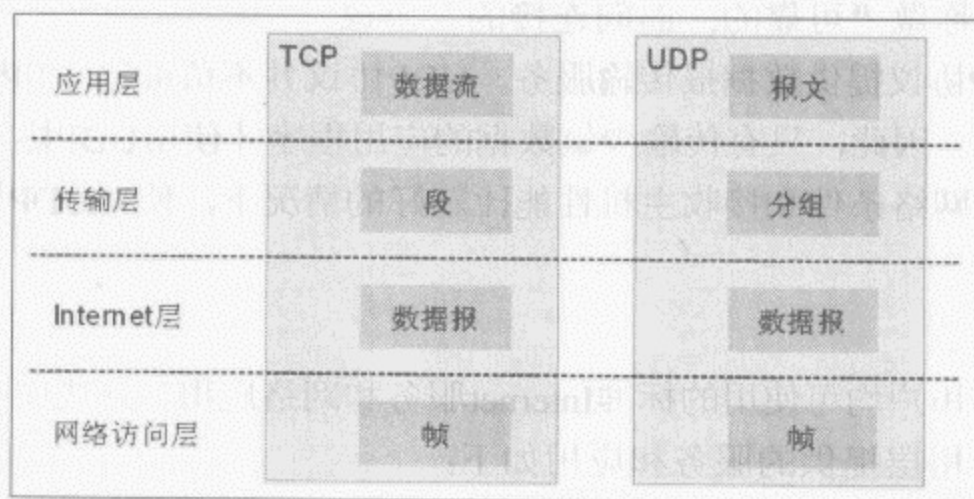


图12-2 数据结构及术语

下面简单说明分组数据经由TCP/IP处理的生命周期，也即数据的封装、传输和剥离等处理过程。当用户输入一个TCP/IP命令或发送一个数据时，标志分组数据生命周期的开始。当接收系统的相应应用收到分组数据时，分组数据的生命周期即告结束。

2. 应用层：数据通信的发起者

如上所述，当用户输入一个必须访问远程主机的TCP/IP命令时，即启动了TCP/IP的分层处理过程。应用层协议依据选用的传输层协议是TCP或UDP，把数据转换为数据流（stream）或报文（message）格式。

假定用户输入一个telnet命令，准备注册到远程主机系统。telnet使用的底层协议是传输层的TCP，而TCP期望收到的数据是由一系列字节组成的数据流，其中包含telnet命令的所有信息。因此，telnet需要以TCP数据流的形式向传输层协议发送数据。

3. 传输层：数据封装的起始点

当数据到达传输层时，传输层协议将启动一个执行数据封装的进程，把应用数据封装成若干传输层能够处理的数据单位。然后，传输层协议将在接收与发送应用之间建立一个由传输端口号标识的虚拟数据流。这个端口号用于标识一个内存位置，以便接收或发送数据。另外，传输协议层亦提供其他服务，如数据传输可靠性保障措施等，最终取决于采用TCP，还是采用UDP处理数据传输。

TCP数据段。TCP经常被称做“面向连接”的协议。TCP提供的保障措施能够确保数据成功地传输到接收系统。如图12-1所示，TCP协议把来自应用层命令的数据流分成若干数据段（segment），然后在每个数据段前面增加一个头信息。数据段的头信息包含发送和接收的端口号、数据段的顺序号，以及校验和等字段。发送和接收主机中的TCP协议均使用这个校验和，确定数据的传输是否有误。

建立TCP连接。TCP采用一个传输控制块（Transmission Control Block, TCB），维护每个活动连接的状态信息，用以确定接收系统是否能够接收数据。当发送主机的TCP协议准备建立连接时，TCP将向接收主机的TCP协议发送一个称做SYN的数据段。如果接收主机已经就绪，接收TCP协议将返回一个称做ACK的数据段，确认已经成功地收到了连接请求数据段。

在收到接收主机的ACK确认信息之后，发送主机中的TCP协议也需要发送一个ACK数据段，然后才能开始实际的数据发送。这种交互传输控制信息的方式称做“三次握手（three-way

handshake) ”。

UDP分组。UDP是一种无连接的协议。与TCP不同，UDP并不检查数据是否已经到达接收主机。而且，UDP协议把来自应用层的报文分成UDP分组（packet），然后在每个分组前面增加一个头信息。分组的头信息包含发送和接收的端口号、分组数据长度以及校验和等字段。

UDP发送进程只管尝试向接收主机的UDP进程发送分组数据，由应用层确定UDP接收进程是否已经确实收到了分组数据。UDP不要求接收方确认，也不事先使用上述握手方式建立连接。

4. Internet层：传输分组数据的准备阶段

传输层的TCP和UDP把其分装的数据段或分组数据传输到Internet层之后，由Internet层的IP（或ICMP）协议处理收到的数据段或分组数据。在开始传输之前，IP协议首先把数据再进一步细分为IP数据报，然后，IP协议开始确定数据报的IP地址，以便把数据传输到接收主机。

在实际传输之前，IP协议将在TCP或UDP协议增加的数据段或分组头信息之前再增加一个IP头信息。IP头信息包括发送和接收主机的IP地址、数据报的长度以及数据报的顺序号等。注意，如果数据报的大小超过了网络允许的字节数，IP协议层还需要把数据报分解为数据片（fragment）。

5. 网络访问层：数据分帧，以及按帧传输数据

在收到IP数据报之后，数据链路层协议（如PPP）将会把数据报组装成数据帧（frame），并增加第三个头信息。数据帧的头信息包含由IP地址转换成的MAC地址（源和目的），以及CRC循环冗余校验字段。经过网络介质传输之后，接收主机可以使用这个校验字段检查传输后的数据帧是否存在错误。最后，数据链路层把组装好的数据帧传输到物理层。

当发送主机的物理层收到数据帧时，物理层将会依据目的MAC地址，通过网络传输介质，把数据帧发送到接收系统的物理层硬件设备中。

6. 接收主机对分组数据的处理过程

当分组数据到达接收主机时，分组数据将以相反的顺序逐层穿越接收主机的TCP/IP协议栈，如图12-1。而且，接收主机中的每一个协议层将会剥离发送主机对应层依次加到分组数据前部的头信息。其详细处理过程如下。

（1）在收到帧格式的分组数据时，网络访问层将会计算帧格式数据的CRC。当确认数据帧的CRC正确无误时，网络访问层将会剥离数据帧的头信息，包括CRC。然后利用反向地址解析协议RARP（Reverse Address Resolution Protocol），把MAC地址映射为IP地址，实现MAC地址与IP地址之间的转换。最后把剥离后的数据发送到Internet层。

（2）Internet层读取头信息中的信息，确认传输的目的系统无误之后，Internet层还需要确定接收的数据是否为数据片，如果确实如此，IP协议层需要把数据片组装成最初的数据报。最后再剥离IP头信息，把数据段或分组数据上传到传输层协议。

（3）传输层协议（TCP或UDP）读取头信息，确定应当由哪一个应用层协议接收这一数据。然后，TCP或UDP协议层将剥离相应的传输层头信息，把数据流或报文发送到相应的目的应用中。

12.2 网络接口设置

在UNIX系统中，网络接口的参数设置通常都是利用配置文件实现的。但不同的系统使用

的配置文件，而且其名字和数量也不相同，具体的实现方式也不尽相同。有的系统需要直接编制配置文件，有的系统提供了一定的工具。

例如，Solaris系统采用/etc/hostname.ifname文件的形式配置每一个网络接口。其中ifname可以是ceN、geN、hmeN、eriN或iprbN等形式的网络接口名（N是0、1、…等数字，分别表示同一类型的第1个、第2个、…网络接口）。

```
# cat /etc/hostname.iprb0
192.168.90.100
#
```

但不管采用哪一种实现方式，提供什么配置工具，实际上，网络接口的参数配置都是在系统生成时，利用/etc/rcN.d目录中的Shell脚本，根据接口配置文件完成的。如果仔细观察一下这些Shell脚本可以发现，其底层采用的基本工具都是一致的，那就是ifconfig命令。

ifconfig命令主要用于配置网络接口的各种参数，包括IP地址、广播地址和网络掩码等。因此，为了使网络设置总是生效，必须在系统的生成过程中使用ifconfig命令配置系统中的每个网络接口。如果想要测试或维护网络接口，也可以随时使用ifconfig命令，显示、修改或重新设置网络接口的IP地址等参数。注意，只有超级用户才能修改网络接口的配置参数。ifconfig命令的一般语法格式简写如下：

```
ifconfig [interface | -a] [addr_family] [addr [/prefix_length] [dest_addr] ]
[parameters]
```

其中，interface是网络接口的设备文件名。“-a”选项表示指定的动作适用于所有的网络接口，通常主要用于显示系统配置的所有网络接口及其状态信息。addr_family表示协议或地址类型，如inet（IPv4）或inet6（IPv6）等。addr是分配给网络接口的IP地址。prefix_length表示地址中用做网络掩码的位数。parameters是ifconfig命令支持的各种参数，参见表12-2。

表12-2 ifconfig命令支持的部分参数

参数	简单说明
up	启用网络接口，使得用户程序能够通过相应的网络接口访问TCP/IP应用和服务。利用ifconfig命令为网络接口分配IP地址时，也借机启用相应的网络接口。当使用这个参数启用网络接口时，也会把相应的网络接口设置为UP和RUNNING状态。使用down参数临时关闭网络接口之后，也可以使用这个参数重新启用网络接口
down	关闭网络接口，禁止任何程序通过相应的网络接口访问TCP/IP应用和服务，同时把网络接口标记为DOWN状态。注意，设置这个参数时也会自动删除与当前网络接口相应的路由表项
arp	启用地址解析协议（Address Resolution Protocol, ARP），实现网络地址与物理地址之间的映射，以便检测主机网络接口的物理地址
-arp	禁止使用ARP地址解析协议。如果禁用ARP，ifconfig命令的输出信息中将会存在一个NOARP标志
netmask mask	指定网络接口的子网掩码。子网掩码是一个32位的二进制数值，其中1表示网络地址部分，0表示主机地址部分。其表示方法可以采用8个十六进制的数值，加上0x前缀；也可以采用4组十进制的数值，中间加句点“.”分隔符。注意，指定子网掩码时至少应包括标准的网络地址部分
broadcast addr	指定网络的广播地址。默认的广播地址是网号不变，主机地址部分全为1的IP地址。如果设置了广播地址，同时也会设置网络接口的BROADCAST标志

(续表)

参数	简单说明
mtu n	使用指定的数值，设置网络接口的最大传输单位（Maximum Transmission Unit, MTU），即一次数据传输最多能够处理的字符数量，这个数值也限制了分组数据的最大尺寸。不同类型的网络接口，MTU的上限值也不相同。例如，Ethernet默认的MTU值为1500
metric number	用于指定网络接口的路由度量值，其默认值为0。RIP等路由服务进程使用这个度量值建立路由表。如果没有启用RIP等路由服务进程，无需设置这个参数（即使启用了路由服务进程，通常也不必关心这个参数值的设置）

在配置网络接口时，首先需要指定一个网络接口，然后指定必要的参数（其中至少必须包含IP地址）。例如，为了设置系统中的以太网接口iprb0，可以使用下列ifconfig命令：

```
# ifconfig iprb0 192.168.90.100
#
```

如果仅仅指定了网络接口（或“-a”选项），而未提供任何参数，ifconfig命令将会显示指定网络接口（或所有网络接口）的状态信息。例如，为了查询上述设置是否已经生效，可以使用下列命令：

```
# ifconfig iprb0
iprb0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
    inet 192.168.90.100 netmask ffffffff broadcast 192.168.90.255
    ether 0:1:4a:3:c3:c
#
```

12.3 主机名字解析

在利用TCP/IP协议进行系统间的网络通信时，其中主要以IP地址标识数据的来源和目的。为了便于用户访问网络服务器，TCP/IP提供名字服务，以便实现从主机名到IP地址的转换。其中包括/etc/hosts文件、DNS、NIS和LDAP等。

在实际的应用中，/etc/hosts文件是最简单也是最常见的名字解析方法。因此，本章主要说明怎样使用hosts文件实现主机名字解析，而不打算讨论DNS、NIS或LDAP等主机名字服务，有关内容可以参考系统提供的相关文档。

与DNS等名字解析方法相比，hosts文件相对比较简单，其文件格式如下：

```
IP-address hostname [aliases...]
```

其中，IP-address为网络接口的IP地址。hostname是主机的名字。aliases是主机的别名。例如，下面是iscas系统中的一个hosts文件实例：

```
$ cat /etc/hosts
127.0.0.1          localhost
192.168.90.100     iscas
192.168.90.101     sinosoft
192.168.90.102     winxp
.....
$
```

12.4 网络路由设置

当网络中的所有主机均位于同一网段时，通过IP地址或主机名，系统之间可以随意进行访问。当准备访问的主机位于不同的网段时，则无法直达，必须通过网关或路由器才能访问。因此，为了访问外部网络，必须考虑路由设置。

根据网络的规模、网络拓扑结构的稳定性，以及主机在网络中扮演的角色——提供路由功能的主机或普通主机，可以分两种方式设置路由。

注意：在使用命令或配置文件设置网络路由时，应尽量使用IP地址或网络接口的设备名，而不要使用主机名或网络名。

12.4.1 静态路由

通常，在一个小型或拓扑结构保持稳定不变的网络中，作为数据服务器或客户机的普通主机，在设置了IP地址和hosts文件之后，还应设置网关或路由。考虑到路由守护进程的开销，及对计算机性能的影响，作为普通主机，建议使用静态路由。即使主机配有多个网络接口（如数据中心中数据库服务器），也以使用静态路由为最佳选择。

同网络接口一样，在不同的UNIX系统中，设置网关或路由的具体方法并不相同。通过观察系统启动过程中执行的Shell脚本可以发现，用于设置静态路由的基本工具均为route命令。

route命令的主要功能是维护网络路由表。利用route命令，可以增加、修改、删除、显示和监控路由表。其中，关于增加或删除路由的route命令，其语法格式简写如下：

```
route [-fnvq] add [modifiers] destination gateway [args]
route [-fnvq] delete [modifiers] destination gateway [args]
```

其中，destination表示欲访问的目的主机或网络。gateway表示通过哪一个系统（IP地址）或网络接口转发访问外部网络的数据。modifiers用于说明后续参数的意义。其中常用的两个修饰选项如下：

- “-host” 把指定的目的地址强制解释为主机。
- “-net” 把指定的目的地址强制解释为网络。

在Solaris系统中，默认静态路由的设置是根据/etc/defaultrouter配置文件，在系统生成时由/etc/rcN.d目录中的脚本文件实现的。例如：

```
# cat /etc/defaultrouter
192.168.90.1
#
```

假定本地系统的网络接口（其IP地址为192.168.90.101）连接到主机（其内部网络接口的IP地址为192.168.90.100），主机的另外一个网络接口（其IP地址为153.78.26.145）连接到外部网络。为了使本地系统能够访问153.78.26.0网络中的任何主机，可在内网的任何主机（192.168.90.101）中，使用下列命令增加静态路由（如图12-3）：

```
# route add -net 153.78.26.0 192.168.90.100
#
```

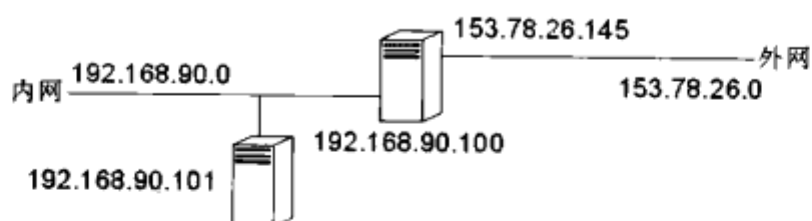


图12-3 增加静态路由示意图

上述设置方式只是临时性的，一旦重新启动系统，路由的设置也就不复存在了。在UNIX系统中，为了确保系统重启后能够重建路由，永久性地设置静态路由，可以把上述命令直接加到/etc/gateways（或/etc/gateways.eth）文件，或把提供路由服务的主机IP地址直接加到/etc/defaultrouter文件中。

12.4.2 动态路由

在一个大型的、拓扑结构动态变化的网络中，可以考虑使用动态路由。除此之外，如果一个主机需要转发数据、提供路由服务，则至少必须配置两个网络接口。其中的两个网络接口必须分别连接到两个不同的网段上。

为了设置动态路由，使主机能够转发数据，提供路由服务，至少应完成下列三个步骤：

- (1) 使用ifconfig命令分别设置两个（或多个）网络接口；
- (2) 启动in.routed或其他路由守护进程，提供动态路由功能；
- (3) 使用ndd或其他等价的命令设置TCP/IP软件的ip_forwarding参数，使系统能够转发IP数据。

例如，为了设置上述主机中的两个网络接口，我们可以使用下列ifconfig命令：

```
# ifconfig en0 192.168.90.100
# ifconfig en1 153.78.26.145
#
```

下一步是设置动态路由。在上一节中，使用route命令设置的路由实为静态路由。对于小型网络中的普通主机来讲，使用静态路由已经足够了。但对于提供路由服务的主机来讲，静态路由是不可思议的。为此，我们需要使用in.routed等路由守护进程，实现动态路由。

in.routed守护进程的主要功能是采用路由信息协议（Routing Information Protocol, RIP）发现网络路由，根据网络的变化不断更新路由，自动维护一个最新的核心路由表，并通过广播地址，以30秒的频率，定时向直接连接的主机或网络设备提供其维护的路由表副本。

通常，in.routed守护进程是在系统生成过程中，由/etc/rcN.d目录中的Shell脚本，根据配置文件的设置或要求启动的。在Solaris系统中，in.routed守护进程的启动与否，取决于是否存在/etc/defaultrouter文件（还有其他条件）。如果文件存在，且含有默认路由器的地址，则按文件中的IP地址（或主机名）设置静态路由。如果文件不存在，或为空，则启动in.routed守护进程，实现动态路由，自动维护核心路由表。

in.routed守护进程的语法格式简写如下：

```
in.routed [-dghmnqSStv]
```

其中，“-s”选项表示强制in.routed守护进程提供路由信息，不管其所在的主机是否作为路由器。例如，为了启动in.routed守护进程，可以使用下列命令：

```
# /usr/sbin/in.routed -s
#
```

如果主机是一个路由器，在启动in.routed守护进程时，还需要检查IP转发功能的核心变量ip_forwarding。在Solaris系统中，可以使用下列ndd命令，查询ip_forwarding参数的当前设置状态：

```
# /usr/sbin/ndd -get /dev/ip ip_forwarding
0
#
```

其中0表示关闭IP转发功能，1表示启用IP转发功能。如果需要，可以使用下列命令启用IP转发功能：

```
# /usr/sbin/ndd -set /dev/ip ip_forwarding 1
#
```

12.5 配置网络服务

当需要使用各种传统的网络服务时，需要用到inetd守护进程，设置其配置文件/etc/inetd.conf。inetd是一个超级守护进程，负责集中管理许多标准的传统Internet服务，如telnet、ftp、finger和talk等，这些Internet服务的相应服务程序telnetd、ftpd、fingerd和talkd都是由inetd负责调度运行的。inetd同时支持UDP和TCP协议。

inetd守护进程用于监听Internet套接字的连接请求。当从套接字中发现一个连接请求时，inetd需要确定与套接字相关联的网络服务，调用服务程序以响应连接请求。在服务程序结束运行之后，inetd将继续监听其负责管理的套接字。采用一个inetd守护进程调用多个服务程序的目的是为了减少系统的负载。

inetd守护进程主要依赖于两个配置文件：一为/etc/inetd.conf，其中包括当前支持的所有服务程序列表，服务程序的路径名及其运行时使用的参数等。第二个是/etc/services文件，用于维护服务名与端口号的映射。这个文件确保系统能够使用正确的端口激活相应的服务。

在开始运行之后，inetd首先会读取/etc/inetd.conf文件中的配置信息。当收到SIGHUP信号时，inetd守护进程会重新读取inetd.conf配置文件。因此，在修改inetd.conf文件，增加、删除或修改服务程序之后，需要使用kill命令，向inetd守护进程发送SIGHUP信号，或发送SIGTERM信号，强行终止之后重新启动inetd守护进程，使其重新读取inetd.conf文件，才能确保修改后的配置文件立即生效。例如：

```
# kill -s HUP `pgrep inetd`
#
```

inetd.conf配置文件包含需由inetd守护进程调度运行的所有网络服务、服务程序的路径名及运行时使用的参数等，每个服务占用一行，其格式定义如下（字段之间以空格或制表符作为分隔符）：

```
srv_name socket_type protocol wait/nowait user srv_prog arguments
```

表12-3给出了inetd.conf配置文件每个字段的简单说明。

表12-3 inetd.conf文件及其说明

字段	简单说明
srv_name	/etc/inet/services文件中定义的有效服务名
socket_type	套接字类型可以是下列关键字之一： <ul style="list-style-type: none"> • stream: 表示STREAMS套接字 • dgram: 表示数据报套接字 • raw: 表示原始套接字
protocol	/etc/inet/protocols文件中定义的有效协议，如tcp或udp等
wait/nowait	这个字段的有效值是wait或nowait，说明当inetd调用相应的服务程序时是否需要等待服务进程运行终止。对于数据报类型的多线程服务进程，这个字段应为nowait。对于数据报类型的单线程服务进程，这个字段应为wait。采用TCP协议的服务程序通常应设为nowait。但是，如果单个服务进程需要处理多个连接请求，则应设为wait
user	指定一个用户名，表示以什么用户身份运行相应的服务程序。服务程序能够以普通用户的身份运行
srv_prog	指定服务程序的完整路径名，以便inetd能够正确地调用，使之执行请求的服务。如果请求的服务是由inetd本身提供的，这个字段应为internal
arguments	如果调用服务程序时需要提供命令行参数，必须在这个字段中指定整个命令行（包括命令名及其所有参数）。如果服务是由inetd内部提供的，这个字段应为internal

下面是一个inetd.conf文件的实例：

```
# cat /etc/inet/inetd.conf
# srv_name socket_type      protocol wait_status user srv_prog arguments
ftp      stream  tcp    nowait  root    /usr/sbin/in.ftpd      in.ftpd
telnet   stream  tcp    nowait  root    /usr/sbin/in.telnetd   in.telnetd
shell    stream  tcp    nowait  root    /usr/sbin/in.rshd      in.rshd
login    stream  tcp    nowait  root    /usr/sbin/in.rlogind   in.rlogind
exec     stream  tcp    nowait  root    /usr/sbin/in.rexecd    in.rexecd
talk     dgram   udp    wait    root    /usr/sbin/in.talkd     in.talkd
finger   stream  tcp    nowait  nobody  /usr/sbin/in.fingerd   in.fingerd
tftp     dgram   udp    wait    root    /usr/sbin/in.tftpd     in.tftpd -s /tftpboot
time     stream  tcp    nowait  root    internal
time     dgram   udp    wait    root    internal
echo     stream  tcp    nowait  root    internal
echo     dgram   udp    wait    root    internal
.....
#
```

上述文件说明，telnet服务是基于TCP实现的，以telnetd的用户身份运行。in.telnetd服务程序的完整路径名为/usr/sbin/in.telnetd。一旦启用了in.telnetd服务程序，任何用户，只要拥有注册的账号，就可以利用telnet访问远程主机系统。

实际上，inetd.conf文件反映的是主机系统当前支持的各种Internet服务。如果想要封锁某个Internet服务，只需删除相应的服务项，或在服务项前面增加一个注释符“#”即可。

因此，为了确保数据库服务器或业务主机的系统安全，严禁任何用户使用telnet远程访问主机系统，可以利用编辑器，在inetd.conf文件的telnet定义前面增加一个注释符“#”，禁止inetd守护进程调用telnetd，封锁telnet服务，从而关闭主机系统的远程访问功能，防止telnetd响应用

户的telnet访问请求。

12.6 网络管理与维护

对于新安装或首次接入网络的系统，如果无法与网络中的其他任何主机进行通信，其主要问题也许是网络设置不当。如果主机一直工作正常，突然出现网络故障，那么网络连接等硬件环节出故障可能是主要原因。如果主机能够与同一网络中的任何主机通信，但无法联系外部网络中的主机，问题肯定出在路由器上：可能主机中的路由设置不当，或者路由器工作不正常。当然，也可能是网络的硬件连接存在问题。

如果网络通信存在问题，可以采用下列方法，利用ifconfig、netstat、ping等命令检测网络的硬件连接和软件配置，确定问题的原因。原因不明的问题轻则影响网络的性能，次则丢失数据，重则无法通信，无法传输任何数据。

- (1) 使用ping命令界定问题的性质，是丢失数据包，还是网络根本就不通；
- (2) 使用ifconfig命令，检查网络接口的状态信息，确定网络接口工作是否正常；
- (3) 使用netstat命令检查网络接口、路由表和协议统计数据等网络状态信息；
- (4) 使用traceroute命令跟踪路由信息；
- (5) 使用ps等命令，确定相关的守护进程（如telnetd等）是否已经启动。

12.6.1 使用ifconfig命令维护网络接口

1. 使用ifconfig命令获取网络接口的状态信息

利用ifconfig命令，可以手工地配置、修改或删除网络接口的IP地址，也可以设置其他网络参数，获取网络接口的基本配置信息等。一个简单的ifconfig查询（未指定任何选项和参数）能够给出如下信息：

- 系统配置的所有网络接口及其设备名；
- 网络接口的MAC地址以及链路协议封装类型；
- 赋予网络接口的IP地址、广播地址以及子网掩码等；
- 网络接口的运行状态，如启用标志以及MTU设置等。

如果指定了网络接口的设备名，ifconfig命令将会给出特定网络接口的各种设置与状态信息。例如，下列ifconfig命令将会给出第一个以太网络接口的相关信息：第一行是网络接口的设备名（如iprb0）、启用状态（如UP和RUNNING等）以及MTU设置（如1500）等；第二行是IP地址信息、广播地址以及子网掩码等；第三行是网络接口的MAC地址。

```
# ifconfig iprb0
iprb0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
    inet 169.254.78.100 netmask ffffffff00 broadcast 169.254.78.255
    ether 0:1:4a:3:c3:c
#
```

表12-4给出了ifconfig命令输出字段的简单解释。

当一个系统配置多个网络接口时，可以使用“ifconfig -a”命令显示系统中配置的所有网络接口的状态信息，包括IP地址分配等，如下列代码所示：


```
# ifconfig -a
lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232
index 1
    inet 127.0.0.1 netmask ff000000
iprb0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
    inet 169.254.78.100 netmask ffffffff00 broadcast 169.254.78.255
    ether 0:1:4a:3:c3:c
#
```

表12-4 ifconfig命令的部分输出字段

输出字段	屏幕显示	简单描述
网络接口名	iprb0	网络接口的设备名
网络接口状态	flags=1000843<UP	网络接口的工作状态及其相关标志。表示网络接口当前是否已经启用，或是否已经初始化。例如，UP表示网络接口已经初始化，DOWN表示网络接口尚未初始化
广播状态	BROADCAST	广播标志。表示网络接口是否支持IPv4广播
传输状态	RUNNING	传输标志。表示网络接口是否已经开始传输分组数据
多播状态	MULTICAST	多播标志。表示网络接口是否支持多播传输
网络协议	IPv4	协议标志。表示网络接口支持的网络协议，如IPv4或IPv6
最大传输单位	mtu 1500	最大传输单位。表示网络接口的最大传输单位为1500个字节
IP地址	inet 169.254.78.100	网络接口的IP地址。例子中en0接口的IP地址为10.0.0.112
子网掩码	netmask ffffffff00	网络接口的子网掩码
广播地址	broadcast 169.254.78.255	网络接口的广播地址
MAC地址	ether 0:1:4a:3:c3:c	网络接口的MAC地址

12.6.2 使用netstat命令监控网络状态

netstat命令可以给出各种网络状态信息，包括网络接口设置、IP路由，以及各种网络协议的分类统计数据。netstat命令的基本语法格式简写如下：

```
netstat [-a] [-p] [-s] [-m] [-i [-I ifname]] [-r] [-n] [-f addr-family]
        [-P protocol] [interval] [count]
```

其中，“-a”选项表示显示所有套接字、路由表以及网络接口的状态信息。“-p”选项用于显示网络IP地址与MAC地址的映射关系。“-s”选项用于显示各种网络协议的分类统计数据。“-i”选项用于显示网络接口的状态信息。“-I”选项用于指定具体的网络接口。“-r”选项用于显示核心路由表信息。“-n”选项表示只需显示IP地址，而不必把IP地址解析成相应的主机名或网络名。“-f”选项用于选择协议地址类型，如inet或inet6等。“-P”选项用于限定显示哪一个协议（如ip、tcp或udp等）的套接字统计与状态信息。interval表示显示抽样统计数据的时间间隔（以秒为时间单位）。count表示显示抽样统计数据的次数。

1. 显示套接字的状态信息

如果不加任何选项和参数，直接运行netstat命令能够显示各种网络协议（如TCP）当前活动套接字的状态信息。例如：

```
$ netstat
TCP: IPv4
  Local Address      Remote Address      Swind Send-Q  Rwind Recv-Q  State
-----
iscas.telnet         winxp.1044          65494      1 49640      0 ESTABLISHED
iscas.ftp            winxp.1052          65443      0 49640      0 ESTABLISHED

Active UNIX domain sockets
Address Type      Vnode      Conn  Local Addr      Remote Addr
d5b06620 stream-ord 00000000    00000000          /tmp/.X11-unix/X0
d5b06758 stream-ord 00000000    d5527d80          /tmp/.X11-unix/X0
d5b06890 stream-ord 00000000    00000000          /tmp/.X11-unix/X0
.....
$
```

表12-5解释了上述输出信息中部分字段的意义。

表12-5 netstat命令的部分输出字段及说明

输出字段	简单说明
Local Address	本地系统与远程主机两端的地址和端口号。除非使用了“-n”选项，主机地址通常采用主机名或规范域名，端口号采用相应的服务名。地址的表示形式为“主机:端口号”，或“网络:端口号”。其中，“主机”是源或目的主机的名字或IP地址（如/etc/hosts文件中定义的主机名或IP地址）。“网络”是源或目的网络的名字或网络地址（如/etc/networks文件中定义的网络名或网络地址）。“端口号”表示一个网络服务，既可以是/etc/services文件中定义的端口号，也可以是相应的服务名（如telnet）。在上述地址格式中，可以存在星号“*”通配符
Remote Address	
Send-Q	本地主机发送队列中远程主机尚未确认是否已经读取的数据字节计数
Recv-Q	本地主机接收队列中用户程序尚未读取的数据字节计数
State	<p>表示网络连接的状态。下面是可能出现的部分常见状态（注意，raw模式与UDP协议不提供网络连接的状态信息，故这一列通常为空）：</p> <p>CLOSED 网络连接已经关闭</p> <p>CLOSE_WAIT 本地主机已经收到远程主机断开连接的请求，且已经给予确认，正在等待本地应用程序关闭连接</p> <p>CLOSING 在经过LAST_ACK和TIME_WAIT两个状态之后，本地与远程系统均进入CLOSING状态，开始正式关闭网络连接</p> <p>ESTABLISHED 网络连接已经建立，可以双向传输分组数据</p> <p>FIN_WAIT1 本地主机已向远程主机发送断开连接的请求，正在等待远程主机的响应</p> <p>FIN_WAIT2 本地主机在向远程主机发出断开连接的请求之后，已经收到远程主机的确认</p> <p>LAST_ACK 本地主机在收到应用程序关闭连接的请求之后，再次向远程主机发出确认信息</p> <p>LISTEN 本地主机处于监听状态，正在监听来自远程主机的连接请求</p> <p>SYN_RECV 本地主机已经收到远程主机的连接请求，且已经给予确认，现正等待远程主机的最终确认</p> <p>SYN_SENT 本地主机已经发出连接请求，正在尝试建立套接字连接，并等待远程主机的确认</p> <p>TIME_WAIT 本地主机在第二次收到远程主机断开连接的确认之后，接着向远程主机发出最终的确认；然后开始着手关闭网络连接</p>

2. 显示所有套接字的状态信息

使用netstat命令的“-a”选项，可以查询本地系统所有套接字（包括正在监听和尚未监听的套接字）的状态信息。下面是“netstat -a”命令的部分输出结果：

```
$ netstat -a
UDP: IPv4
  Local Address      Remote Address      State
-----
    *.sunrpc          Idle
    *.*               Unbound
.....
TCP: IPv4
  Local Address      Remote Address      Swind Send-Q Rwind Recv-Q      State
-----
    *.*               *.*                0      0 49152      0 IDLE
    *.sunrpc          *.*                0      0 49152      0 LISTEN
.....
    *.telnet          *.*                0      0 49152      0 LISTEN
    *.ftp             *.*                0      0 49152      0 LISTEN
    *.finger          *.*                0      0 49152      0 LISTEN
    *.login           *.*                0      0 49152      0 LISTEN
    *.shell           *.*                0      0 49152      0 LISTEN
    *.ssh             *.*                0      0 49152      0 LISTEN
.....
iscas.telnet         winxp.1044          64411    1 49640      0 ESTABLISHED
iscas.ftp            winxp.1052          65443    0 49640      0 ESTABLISHED
*.login
.....
$
```

根据套接字的状态信息，可以确定某个网络服务进程是否已经启动。例如，为了查询是否已经启动了FTP服务器守护进程，可以使用下列命令：

```
$ netstat -a | grep ftp
    *.ftp             *.*                0      0 49152      0 LISTEN
    *.ftp             *.*                0      0 49152      0 LISTEN
$
```

上述输出信息的状态字段为LISTEN，说明FTP服务器守护进程已经启动。

3. 显示网络接口的状态信息，检测网络主机的可靠性

为了检测网络主机的可靠性和数据通信能力，可以利用netstat命令的“-i”选项，显示网络接口的状态信息，确定本地系统发送和接收的分组数据数量，发送或接收分组数据出错的次数等。下面是“netstat -i”命令输出的一个例子，其中给出了网络接口的分组数据流量与出错统计等数据：

```
$ netstat -i
Name Mtu Net/Dest Address Ipkts Ierrs Opkts Oerrs Collis Queue
lo0 8232 loopback localhost 9810 0 9810 0 0 0
iprb0 1500 iscas iscas 3230 0 2010 0 0 0
$
```

表12-6给出了“netstat -i”命令的输出结果及简单说明。

表12-6 netstat命令的部分输出字段及其解释

输出字段	简单说明
Name	网络接口的名字
Mtu	网络接口当前支持的最大数据传输单位。MTU是IP模块的设备驱动程序一次收发时能够处理的最大字节数量。对于以太网网络接口，MTU的默认值为1500。对于回环网络接口（loopback），其数值为8232。对于IEEE 802.3接口，其数值为1492
Net/Dest	网络接口连接的网络或主机名（或者IP地址）。对于点对点的网络接口而言，这个字段是链路另一端的主机名或IP地址
Address	网络接口的IP地址或相应的主机名
Ipkts	经由网络接口发送的分组数据数量
Opkts	经由网络接口接收的分组数据数量
Ierrs	接收分组数据时出现错误的次数
Oerrs	发送分组数据时出现错误的次数
Collis	碰撞统计计数

上述输出信息给出了当前主机中每个网络接口已经发送和接收的分组数据数量。利用发送和接收的分组数量，可以判定网络的运行是否正常。在一个正常运行的网络系统中，反映网络流量的Ipkts和Opkts字段应当连续不断地增长。

例如，假定客户系统正在尝试请求服务器引导自己的系统时，服务器接收的输入分组数据计数（Ipkts）不断增加，但发送的输出分组数据计数（Opkts）保持不变。这一事实表明服务器正在接收和处理来自客户系统含有引导请求的分组数据，但服务器不知道如何响应，故只有请求输入，而没有数据输出。这个结果说明网络的设置存在问题：可能是客户系统的配置文件，如hosts文件中的服务器地址设置不正确，或者服务器的远程引导服务没有启动。

此外，如果输入输出分组数据计数字段的值一直保持稳定不变，说明系统根本没有接收和发送任何分组数据。这个结果表明：可能是网络连接存在问题，或者软件设置存在问题，总之，本地系统与其他系统之间没有任何数据通信。

为了计算网络的碰撞率，可以使用碰撞计数字段（Collis）的数值除以发送分组数据数量字段（Opkts）的数值。在一个正常运行的网络环境中，如果碰撞率超出5%~10%的范围，表明存在一定的问题。

为了计算接收分组数据的错误率，可以使用接收错误计数字段（Ierrs）的数值除以接收分组数据合计字段（Ipkts）的数值，即Ierrs/Ipkts。同样，发送分组数据的错误率等于发送错误计数字段（Oerrs）的数值除以发送分组数据合计字段（Opkts）的数值，即Oerrs/Opkts。如果接收错误率较高，超过0.25%，说明主机已经开始出现丢包现象。

4. 显示当前路由的状态

netstat命令的“-r”选项用于显示本地主机维护的路由信息，其中反映了本地主机已知的所有路由及其当前状态，示例如下：

```

$ netstat -r

Routing Table: IPv4
  Destination          Gateway                Flags  Ref    Use    Interface
-----
default                169.254.78.1          UG      1      0
169.254.78.0          iscas                  U        1      1 iprb0
224.0.0.0              iscas                  U        1      0 iprb0
localhost              localhost              UH      25    1130 lo0
$

```

表12-7给出了“netstat -r”命令输出信息的简单说明。

表12-7 “netstat -r”命令的部分输出字段及其描述

输出字段	简单描述
Destination	表示路由的目的主机或网络
Gateway	表示把分组数据转发到目的主机或网络需要用到的网关
Flags	表示路由的当前状态。其中可能出现的状态标志如下： <ul style="list-style-type: none"> • U 表示相应的路由已经就绪； • G 表示路由是利用网关实现的； • H 表示路由的目的地是一个主机，如回环网络接口表示的主机（127.0.0.1）
Ref	当前使用相应网关或路由的数量
Use	表示自网络开始运行以来通过相应网关发送的分组数据数量
Interface	表示传输路由数据时使用的本地网络接口名

5. 按协议显示统计信息

“netstat -s”命令用于显示不同协议（IP、UDP、TCP或ICMP等）的分类统计数据。利用这些统计数据，可以确定哪个网络协议存在问题。如果再使用netstat命令的“-P”选项，还可以显示特定传输协议的状态。

在利用“netstat -s”命令获取统计数据时，要重点考察输入输出错误、校验和错误、重传次数、分组数据丢失统计等字段。这些数据能够反映网络和主机的运行状态与性能，以及硬件连接和软件安装是否存在问题等。

12.6.3 使用ping命令测试远程主机的连通性

为了测试远程主机的连通性，最常见、最简单的方法是采用ping命令。运行时，ping命令将会利用ICMP协议向指定的远程主机发送ECHO_REQUEST请求信息，期望远程主机回以ECHO_REPLY响应信息。利用ping命令，可以检查与指定的远程主机是否已经建立了TCP/IP连接。ping命令的语法格式简写如下：

```

ping host [timeout]
ping -s [-I interval] host [packet size] [count]

```

其中，host是远程系统的主机名或IP地址。timeout是以秒为单位的超时值，使ping命令能够在指定的时间内连续地尝试连接远程主机。默认的超时值为20秒。“-s”选项表示每秒一次，连续发送分组数据。“-I”选项表示发送每个分组数据之间的时间间隔，默认值为1秒。

packetsize表示分组数据的大小，默认值为56个字节（加上8个ICMP头字节，共64个字节）。**count**表示发送ECHO请求的次数。

1. 确定与远程主机的连通性

ping命令的主要用途是测试本地系统与远程主机之间的网络连通性，以及远程主机是否正在运行。如果指定的远程主机能够接收并响应ICMP请求，**ping**命令的运行结果如下：

```
$ ping iscas
iscas is alive
$
```

如果指定的远程主机已经关机，或收不到ICMP响应信息，**ping**命令的运行结果如下：

```
$ ping sinosoft
no answer from sinosoft
$
```

2. 确定网络通信是否丢失分组数据

利用**ping**命令的“-s”选项，不仅可以确定与远程主机之间的连通性，还可以检测与远程主机之间的数据通信是否丢失分组数据。“-s”选项表示向指定的远程主机每秒钟发送一次分组数据，直至用户使用中断键、或超时才能终止命令的执行。示例如下：

```
$ ping -s iscas
PING iscas: 56 data bytes
64 bytes from iscas (192.168.90.100): icmp_seq=0. time=0.130 ms
64 bytes from iscas (192.168.90.100): icmp_seq=1. time=0.164 ms
64 bytes from iscas (192.168.90.100): icmp_seq=2. time=0.153 ms
^C
----iscas PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max/stddev = 0.130/0.149/0.164/0.017
$
```

在上述的输出信息中，“packet loss”字段是分组数据丢失统计计数。如果**ping**命令失败，或丢失的分组数据数量较大，需要利用**ifconfig**和**netstat**等命令进一步检查网络的工作状态。

此外，还可以指定分组数据的大小，进一步检测网络和远程主机的响应能力。

```
$ ping -s iscas 1024
PING iscas: 1024 data bytes
1032 bytes from iscas (192.168.90.100): icmp_seq=0. time=0.135 ms
1032 bytes from iscas (192.168.90.100): icmp_seq=1. time=0.0940 ms
1032 bytes from iscas (192.168.90.100): icmp_seq=2. time=0.0960 ms
^C
----iscas PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max/stddev = 0.0940/0.108/0.135/0.023
$
```

12.6.4 使用ftp命令检测网络主机的传输性能

利用**ftp**命令也可以检测网络主机的数据传输性能。进入**ftp**会话之后，可以使用下列**ftp**子命令，以/dev/zero作为输入，以/dev/null作为输出，传输一个较大的文件。这将避免使用磁盘

(磁盘读写本身也有开销)，也不必使用内存缓存整个文件，因而能够纯粹测试网络的数据传输性能：

```
put "dd if=/dev/zero bs=32k count=10000" /dev/null
```

在下面的例子中，**put**命令最后一行的统计数据可以给出网络传输的速度。在执行实际的测试时，数据块的大小与传输的次数可由用户视具体情况而定：

```
$ ftp iscas
.....
ftp> put "dd if=/dev/zero bs=32k count=10000" /dev/null
200 PORT command successful.
150 Opening BINARY mode data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
local: dd if=/dev/zero bs=32k count=10000 remote: /dev/null
327680000 bytes sent in 0.83 seconds (387878.46 Kbytes/s)
ftp> put "dd if=/dev/zero bs=64k count=10000" /dev/null
150 Opening BINARY mode data connection for /dev/null.
10000+0 records in
10000+0 records out
226 Transfer complete.
local: dd if=/dev/zero bs=64k count=10000 remote: /dev/null
655360000 bytes sent in 1.6 seconds (396110.52 Kbytes/s)
ftp>
```

12.6.5 使用tracert命令跟踪路由信息

在查询指定的主机时，**tracert**命令利用IP协议的TTL (Time-to-Live) 字段，要求途经的每个路由器或目的主机返回一个“ICMP TIME EXCEEDED”响应信息。在**tracert**命令的跟踪过程中，TTL字段从默认值1开始，每经过一个中间路由器，TTL字段将依次加1，直至接收到一个“ICMP PORT UNREACHABLE”信息（表示已经到达指定的主机），或已经达到TTL的最大值，跟踪过程立即结束。针对每一个TTL值，也即途经的每一个中间路由器，**tracert**命令将会发送三个UDP数据报或“ICMP ECHO”查询，因而期望得到三个“ICMP TIME EXCEEDED”响应信息。

tracert命令的主要功能是跟踪IP分组数据到达目的主机时经由的整个路径，用于显示相互通信的两个系统之间，IP分组数据从源主机到目的主机经过的所有中间路由。另外一个功能就是发现任何不合理的路由和路径不正确的路由。如果无法到达指定的主机，可以使用**tracert**命令，检查IP分组通过什么路径联系远程主机，中间哪一个环节可能存在问题。

tracert命令的语法格式如下：

```
tracert [-dnv] [-f first-ttl] [-w wait] [-m max-ttl] [-p port] [-q queries]
dest-host
```

其中，“-d”选项表示启用套接字级的调试功能。“-n”选项表示仅显示中间路由器或目的主机的IP地址，无需把IP地址解析成主机名。“-v”选项表示显示收到的ICMP分组数据而非“ICMP TIME EXCEEDED”或“ICMP PORT UNREACHABLE”响应信息。“-f”选项用于指定起始TTL值，默认值为1。“-w”选项表示等待中间路由器或目的主机返回ICMP响应信

息的时间，默认值为5秒。“-m”选项用于指定TTL的最大值（也即最大的中转路由数量），默认值为30。“-p”选项表示tracert命令使用的UDP端口号，默认值为33434。“-q”选项表示针对每一个TTL，中间路由器或目的主机应返回几个ICMP响应信息，默认值为3。dest-host表示查询的远程主机。

tracert命令将会显示到达目的主机期间途经的每一个路由器（包括目的主机）的IP地址，以及分组数据往返传输耗费的时间。这个时间信息对于分析两个主机之间任何环节的网络流量或通信能力是非常有用的。如果在发出UDP数据报或“ICMP ECHO”查询信息的5秒钟之内一直没有接收到ICMP响应信息，tracert命令将会在相应的中间路由器或目的主机的返回时间字段中输出一个星号“*”字符。

tracert命令的每一行输出信息通常包含下列三个字段：

- TTL值（默认情况下从1开始，最大值为30）；
- 中间路由器或目的主机的IP地址；
- 每一个ICMP响应信息的返回时间。

例如，tracert命令的下列输出信息表明，一个分组数据从本地主机到达远程主机www.google.cn，中间跨越9个网络路径。这个输出信息同时也给出了分组数据途经每个网段时耗费的时间。

```
$ tracert www.google.cn
tracert to www.google.cn (203.208.37.104), 30 hops max, 40 byte packets
 1  * * *
 2  202.106.57.29 (202.106.57.29)  17.161 ms  19.006 ms  20.952 ms
 3  61.148.155.81 (61.148.155.81)  22.324 ms  24.341 ms  25.900 ms
 4  61.148.156.221 (61.148.156.221)  28.560 ms  30.234 ms  32.406 ms
 5  202.96.12.49 (202.96.12.49)  35.071 ms  37.101 ms  38.352 ms
 6  219.158.11.90 (219.158.11.90)  40.676 ms  25.791 ms  *
 7  219.158.32.226 (219.158.32.226)  26.538 ms  27.036 ms  26.440 ms
 8  203.208.62.27 (203.208.62.27)  27.771 ms  25.818 ms  26.805 ms
 9  203.208.62.121 (203.208.62.121)  33.355 ms  30.822 ms  29.088 ms
10  203.208.37.104 (203.208.37.104)  26.597 ms  26.135 ms  26.572 ms
$
```

第13章 TCP/IP网络应用

UNIX系统提供了丰富的网络功能，有助于用户实现各种数据通信与文件传输。本章主要介绍TCP/IP网络应用，其中包括OpenSSH、Telnet和FTP等。

13.1 OpenSSH

目前，OpenSSH是最流行的远程系统注册与文件传输应用，也是传统Telnet、FTP与R系列网络应用的换代产品。其中，SSH（Secure Shell）可以替代Telnet、rlogin和rsh，SCP（Secure Copy）与SFTP（Secure FTP）能够替代FTP。

OpenSSH不仅适用于UNIX系统（如Solaris、AIX和HP-UX等），也适用于各种Linux系统，且其中的SSH、SCP与SFTP等客户端软件也可用于Windows等系统。必要时，Windows用户可以下载Windows版的OpenSSH客户端软件，如WinSCP，在Windows系统与UNIX系统之间实现文件的传输，详见<http://winscp.net/eng/download.php>等网站中的内容。

OpenSSH采用密钥的方式对数据进行加密，确保数据传输的安全。在正式开始传输数据之前，双方首先要交换密钥。当收到对方的数据时，再利用密钥和相应的程序对数据进行解密。这种加密的数据传输有助于防止非法用户获取敏感的数据信息。

OpenSSH采用随机的方式生成公私密钥。密钥通常只需生成一次，必要时也可以重新制作。

当使用ssh命令注册到远程系统时，OpenSSH服务器的sshd守护进程将会发送一个公钥，OpenSSH客户端程序ssh随之会提请用户确认是否接受对方发送的公钥。同时，OpenSSH客户端也会向服务器回送一个密钥，使得OpenSSH连接双方的每个系统都拥有对方的密钥，因而能够解密对方经由加密链路发送的加密数据。

OpenSSH服务器的公钥与私钥均存储在/etc/ssh目录中。在OpenSSH客户端，用户收到的所有公钥，以及提供密钥的OpenSSH服务器的IP地址均存储在用户主目录下的~/.ssh/known_hosts文件中（注意，.ssh是一个隐藏的目录）。如果密钥与IP地址不再匹配，OpenSSH将会认为某个环节出了问题。例如，重新安装操作系统或升级OpenSSH时都会导致系统再次生成新的密钥。当然，恶意的网络攻击也会造成密钥的变动。因此，当密钥发生变化时，应当了解密钥变化的原因，以确保网络访问期间的数据安全。

13.1.1 sshd_config配置文件

/etc/ssh/sshd_config是OpenSSH服务器默认的配置文件的，sshd守护进程根据其中的定义，规范其处理动作。如果需要，也可以在启动sshd的命令行中使用“-f”选项指定其他配置文件。

sshd_config配置文件的每一行可能是一个注释行（起始字符为注释符“#”），又或者是一个参数设置。sshd_config配置文件的语法格式如下：

```
parameter value
```

表13-1给出了部分重要的配置参数及其简单说明。

表13-1 sshd的部分配置参数

配置参数	简单说明
AllowGroups	设定这个配置参数时，可以列举多个用户组的名字或模式，中间以空格作为分隔符。如果设定了这个配置参数，只允许其用户组（包括附加用户组）匹配指定用户组名或模式的用户注册。注意，这里只能指定用户组名，不能使用用户组ID。通常，所有用户组的成员均允许注册。sshd将会按照DenyUsers、AllowUsers、DenyGroups与AllowGroups的顺序依次处理用户注册的限定策略
AllowTcpForwarding	指定是否启用TCP转发功能，默认值为yes。注意，禁止TCP转发功能并不能改善系统的网络安全问题，除非也同时拒绝用户访问Shell，否则用户总是能够利用Shell安装自己的转发软件
AllowUsers	设定这个配置参数时，可以列举多个用户名或模式，中间以空格作为分隔符。如果设定了这个配置参数，只允许匹配指定用户名或模式的用户注册。同样，这里只能指定用户名，不能使用用户ID。通常，所有用户均可注册。如果用户名模式采用USER@HOST形式，需要单独检测USER与HOST，从而能够确保特定主机中的特定用户注册
AuthorizedKeysFile	指定包含用户认证公钥的文件。AuthorizedKeysFile配置参数的值可以包含%T形式的标记符，这些标记符能够在建立连接期间予以替换，其中，%%是单个百分号“%”，%h可以替换成认证用户的主目录，%u可以替换为注册用户的用户名。经过上述替换后，AuthorizedKeysFile指定的文件将会成为一个绝对路径名，或相对于用户主目录的相对路径名。默认值是.ssh/authorized_keys
Compression	用于控制服务器是否允许客户系统协商使用压缩方式，默认值是yes
DenyGroups	设定这个配置参数时，可以列举多个用户组的名字或模式，中间以空格作为分隔符。如果设定了这个配置参数，则禁止其用户组（包括附加用户组）匹配指定用户组名或模式的用户注册。同样，这里只能指定用户组名，不能使用用户组ID。通常，所有用户组的成员均允许注册
DenyUsers	设定这个配置参数时，可以列举多个用户名或模式，中间以空格作为分隔符。如果设定了这个配置参数，则禁止匹配指定用户名或模式的用户注册。同样，这里只能指定用户名，不能使用用户ID。通常，所有用户均可注册。如果用户名模式采用USER@HOST形式，需要单独检测USER与HOST，从而能够确保正确地限制特定主机中的特定用户注册
HostKey	用于指定包含主机私钥的文件，默认值是/etc/ssh/ssh_host_key（OpenSSH协议版本1），或/etc/ssh/ssh_host_rsa_key与/etc/ssh/ssh_host_dsa_key（OpenSSH协议版本2）。注意，sshd将会拒绝使用同组或其他用户均可访问的文件作为主机私钥文件。此外，OpenSSH允许使用多个主机私钥文件
KeepAlive	指定sshd是否向客户系统发送TCP keepalive消息。如果发送keepalive消息，服务器便能及时了解客户系统的网络连接状态、关机或系统崩溃等情形。但这也意味着，如果网络路由临时失灵，导致连接暂时断开，有可能会造成不必要的麻烦，而这也许并非是设置此参数之本意。从另外一方面讲，如果不发送TCP keepalive消息，sshd可能会无限期地维持当前的网络会话，空耗服务器的资源。这个配置参数的默认值是yes，即发送TCP keepalive消息，以便服务器能够及时了解网络连接是否已经断开，对方的系统是否已经关机等，从而避免无限期地维持无效的网络会话。为了禁止发送TCP keepalive消息，可以把服务器与客户系统中的KeepAlive配置参数均设置为no
ListenAddress	指定sshd应当监听的本地IP地址。在指定监听地址时，可以采用下列形式： <ul style="list-style-type: none"> • ListenAddress host IPv4_addr IPv6_addr • ListenAddress host IPv4_addr:port

(续表)

配置参数	简单说明
	<ul style="list-style-type: none"> • ListenAddress [host IPv6_addr]:port <p>其中, host表示主机名, IPv4_addr表示IPv4地址, IPv6_addr表示IPv6地址, port表示端口号。如果ListenAddress配置参数中未指定端口(第一种定义形式), sshd将会监听指定的地址及其所有端口, 除非Port配置参数之前另有限定。因此, 任何Port配置参数(如果存在)必须位于未加端口限制的ListenAddress配置参数之前。在sshd_config配置文件中, 允许同时指定多个ListenAddress配置参数。通常, sshd会监听本地系统的所有IP地址</p>
LoginGraceTime	如果用户未能成功地注册, sshd将会在这个配置参数指定的时间过后断开连接。如果参数值为0, 表示没有时间限制, 默认值为120秒
LogLevel	指定sshd采用的日志信息级别, 可取的参数值是SILENT、QUIET、FATAL、ERROR、INFO、VERBOSE、DEBUG、DEBUG1、DEBUG2和DEBUG3, 默认值是INFO。DEBUG与DEBUG1是等同的, DEBUG2与DEBUG3表示记录更高级别的调试信息。采用DEBUG级别有可能会侵犯用户的隐私, 因而不建议使用
PasswordAuthentication	指定是否允许使用密码认证, 默认值是yes
PermitEmptyPasswords	当采用密码认证方式时, 指定sshd是否允许使用空的密码注册, 默认值是no
PermitRootLogin	指定超级用户是否能够仍以超级用户的身份注册远程系统。选用的参数值必须是yes、no、without-password或forced-commands-only之一, 默认值为yes (UNIX系统的实际设置为no, 即禁止超级用户注册)
PermitUserEnvironment	指定sshd是否应处理位于服务器中的用户文件~/.ssh/environment, 以及AuthorizedKeysFile配置参数指定文件中的环境选项, 其默认设置为no。注意, 启用环境处理功能有可能会提供用户绕过某些访问限制的机会。此外, 仅当用户采用公钥认证方法认证之后, sshd才会处理AuthorizedKeysFile配置参数指定文件中的环境设置。在这两个文件中, 如果存在同名的环境变量设置, 以~/.ssh/environment文件中设置的变量值为准
PidFile	指定包含sshd守护进程PID的文件, 默认值是/var/run/sshd.pid
Port	指定sshd监听的端口号, 默认值是22。配置文件允许同时指定多个Port配置参数, 参见ListenAddress配置参数的说明
PrintLastLog	指定以交互方式注册时, sshd是否应当输出用户上一次注册的日期和时间, 默认值是yes
PrintMotd	指定当用户以交互方式注册时, sshd是否应当显示/etc/motd文件中的内容。在有些UNIX系统中, 这一处理动作是由Shell或/etc/profile实现的。默认值是yes
Protocol	指定sshd支持的OpenSSH协议版本, 有效的取值是1、2或者1与2。如果同时支持两个协议版本, 两个数字之间需加逗号“,”分隔符, 而且, 数字的排列顺序表示sshd尝试使用的先后顺序。这个配置参数的默认值是“2,1”(UNIX系统的实际设置为2), 这意味着首先尝试使用协议版本2, 如果失败, 再尝试使用协议版本1
PubkeyAuthentication	指定是否允许采用公钥认证, 默认值是yes。注意, 这个配置参数仅适用于协议版本2
RSAAuthentication	指定是否允许使用纯RSA认证, 默认值是yes。这个配置参数仅适用于SSH协议版本1
StrictModes	指定sshd是否应在接受用户注册之前检测文件的访问权限, 以及用户的文件与主目录的属主等属性。默认值是yes
Subsystem	用于配置一个外部服务程序, 如文件传输服务器sftp-server。配置参数的值应是一个系统名与命令(如果命令需要选项与参数, 必须同时给出), 能够基于客户系统的请求开始运行。sftp-server命令实现了sftp文件传输子系统。注意, 这个配置参数仅适用于OpenSSH协议版本2

下面是取自Solaris系统中/etc/ssh/sshd_config配置文件的部分内容：

```
$ cat /etc/ssh/sshd_config
.....
# Only v2 (recommended)
Protocol 2
.....
# Listen port (the IANA registered port number for ssh is 22)
Port 22
.....
# IPv4 only
#ListenAddress 0.0.0.0
# IPv4 & IPv6
ListenAddress ::
.....
# KeepAlive specifies whether keep alive messages are sent to the client.
# See sshd(1) for detailed description of what this means.
# Note that the client may also be sending keep alive messages to the server.
KeepAlive yes
.....
# Host private key files
# Must be on a local disk and readable only by the root user (root:sys 600).
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
.....
# To disable tunneled clear text passwords, change PasswordAuthentication to no.
PasswordAuthentication yes
.....
# Are root logins permitted using sshd.
# Note that sshd uses pam_authenticate(3PAM) so the root (or any other) user
# maybe denied access by a PAM module regardless of this setting.
# Valid options are yes, without-password, no.
PermitRootLogin no

# sftp subsystem
Subsystem      sftp /usr/lib/ssh/sftp-server
.....
$
```

根据表13-1中的说明，以及系统提供的/etc/ssh/sshd_config配置文件可知，sshd将会监听本地系统所有IP地址的22号TCP端口，支持OpenSSH协议版本2，使用密码认证方式，不允许超级用户root注册，采用单独的子系统sftp，以sftp-server作为服务器，实现文件的安全传输等。

13.1.2 ssh_config配置文件

/etc/ssh/ssh_config是OpenSSH客户端的配置文件，通常无需修改此文件。如果确实想要修改默认的配置，或增加自己的特定设置，可以参考表13-2给出的说明。

表13-2 ssh的部分配置参数

配置参数	简单说明
BindAddress	在配有多个网络接口的系统中，指定使用哪一个网络接口进行网络通信
CheckHostIP	如果这个配置参数为yes，ssh还需要进一步检测known_hosts文件中的主机IP地址，使ssh能够发现是否存在由于DNS欺骗而导致的主机密钥变动。否则不检测
Compression	指定是否采用压缩通信方式，其默认值为no
CompressionLevel	如果启用了压缩通信方式，这个配置参数用于指定当前采用的压缩级别，其有效值范围是0~9之间的一个整数，默认的压缩级别是6。对于大多数应用而言，这一设置是比较适当的。注意，这个配置参数仅适用于OpenSSH协议版本1
ConnectionAttempts	用于指定在终止执行或退出之前尝试连接主机系统的次数（每秒1次）。这个配置参数的取值必须是一个整数，默认值为1。在使用脚本连接主机系统的情况下，这个配置参数是比较有用的
EscapeChar	用于设置转义字符，其默认值为波浪号“~”。用户也可以利用命令行设置转义字符。设置的转义字符应当是一个上箭头“^”，后面紧跟一个字母，也可以为none，表示完全禁用转义字符（使网络连接能够透明地传输二进制数据）
GlobalKnownHostsFile	用于指定一个替代/etc/ssh/ssh_known_hosts的文件
HostKeyAlgorithms	指定客户系统准备优先选用的协议版本2主机密钥算法，其默认值为ssh-rsa,ssh-dss
HostName	用于指定注册的实际主机名或IP地址。如果设置了这个配置参数，连接时可以指定主机的别名或缩写名。默认值是命令中指定的主机名或IP地址
IdentityFile	用于指定一个文件，可以从中读取用户的RSA或DSA身份认证信息。在OpenSSH协议版本1中，默认的文件是\$HOME/.ssh/identity；协议版本2的默认文件是\$HOME/.ssh/id_rsa和\$HOME/.ssh/id_dsa
KeepAlive	指定ssh是否向服务器发送TCP keepalive消息。其默认值为yes，即发送TCP keepalive消息，以便服务器能够及时了解客户系统的网络连接状态，是否已经关机，从而避免无限期地维持无效的网络会话。为了禁止发送TCP keepalive消息，可以把服务器与客户系统中的KeepAlive配置参数均设置为no
LogLevel	指定ssh采用的日志信息级别，可取的参数值是FATAL、ERROR、QUIET、INFO、VERBOSE、DEBUG、DEBUG1、DEBUG2和DEBUG3，默认值是INFO。DEBUG与DEBUG1是等同的，DEBUG2与DEBUG3表示记录更高级别的调试信息
NumberOfPasswordPrompts	指定在放弃密码和键盘交互认证方法之前可以尝试进行认证的次数，默认值为3。注意，每一种认证方法是单独计算的
PasswordAuthentication	指定是否使用密码认证，其默认值为yes。这个配置参数适用于OpenSSH协议版本1和2
Port	指定连接远程的端口号，默认值为22
PreferredAuthentications	在协议版本2支持的认证方法中，指定ssh尝试使用的先后顺序。这允许客户系统选用自己最喜欢使用的认证方法，这个配置参数的默认值为“hostbased,publickey,keyboard-interactive,password”
Protocol	指定ssh支持的OpenSSH协议版本。有效的取值是1、2或者1与2。如果同时支持两个协议版本，两个数字之间需加一个逗号“,”分隔符，而且，两个数字的排列顺序表示ssh尝试使用的先后顺序。这个配置参数的默认值是“2,1”，意味着首先尝试使用协议版本2，如果失败，再尝试使用协议版本1

（续表）

配置参数	简单说明
ProxyCommand	用于设置一个连接sshd服务器的命令，设置命令必须能够由/bin/sh解释执行，如果需要，设置命令只能从标准输入中接收数据，也只能把输出数据写到标准输出。如果存在宏定义，命令行中的%h可以替换成服务器的主机名，%p可以替换成实际的TCP端口
PubkeyAuthentication	指定是否尝试使用公钥认证，其默认值为yes。这个配置参数仅适用于OpenSSH协议版本2
StrictHostKeyChecking	如果这个配置参数为yes，ssh不会把主机密钥自动加到\$HOME/.ssh/known_hosts文件中，而且还会拒绝连接主机密钥发生变动的主机系统。这也意味着需要用户手工增加每一个新的主机系统，这种做法能够提供一定的安全防护，避免特洛伊木马病毒的攻击。但是，如果/etc/ssh/ssh_known_hosts文件配置不当，需要频繁地连接新的主机系统，由此也会带来极大的不便。这个配置参数的有效取值是yes、no和ask，默认设置为ask，表示经过用户的确认后，新的主机系统将会自动加到known_hosts文件中。但在任何情况下，ssh都会自动验证已知系统的主机密钥
User	指定以哪一个用户身份注册。如果在不同的服务器系统中具有不同的用户名，这个配置参数是非常有用的，可以避免在命令行中输入用户名
UserKnownHostsFile	指定一个替代\$HOME/.ssh/known_hosts的文件

13.1.3 使用SSH注册到远程系统

在OpenSSH中，ssh是一个重要的客户端应用程序。利用ssh，可以采用加密通信方式，注册到远程系统。ssh命令的语法格式简写如下：

```
ssh [-l login_name] [-p port] hostname | user@hostname [command]
```

其中，“-l”选项用于指定用户名，表示以哪一个用户身份注册到远程系统。如果不提供用户名，则以当前用户的身份注册到远程系统。hostname表示远程系统的主机名或IP地址。例如，下列命令形式表示仍以本地系统的gqxing用户身份，采用默认的端口22，注册到远程系统：

```
$ ssh iscas
Password:
Last login: Mon Jul 20 15:59:51 2009 from sinosoft
Sun Microsystems Inc.    SunOS 5.10        Generic January 2005
$
```

利用“-l”选项，可以使用不同的用户身份访问远程系统。例如，下列命令意味着本地系统的当前用户以cathy的用户身份注册到远程系统iscas：

```
$ ssh -l cathy iscas
Password:
Last login: Mon Jul 20 15:59:51 2009 from sinosoft
Sun Microsystems Inc.    SunOS 5.10        Generic January 2005
$
```

除了“-l”选项之外，还可以采用“user@hostname”的表示形式，以指定用户的身份注册到远程系统，例如：

```
$ ssh gqxing@iscas
Password:
```

```
Last login: Mon Jul 20 16:06:40 2009 from sinosoft
Sun Microsystems Inc.    SunOS 5.10        Generic January 2005
$
```

当第一次使用ssh注册到远程系统时，ssh将会给出一个警告信息，提示用户确认连接的远程系统是否正确。如果回答yes，ssh将会在用户主目录的~/.ssh/known_hosts文件中存储远程系统的密钥，同时也会把客户端用户的密钥发送到远程系统，例如：

```
$ ssh iscas
The authenticity of host 'iscas (169.254.78.100)' can't be established.
RSA key fingerprint is 6d:de:44:d1:ea:47:54:de:f6:67:94:71:5f:1d:fc:a0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'iscas,169.254.78.100' (RSA) to the list of known
hosts.
Password:
Last login: Mon Jul 20 15:59:51 2009 from sinosoft
Sun Microsystems Inc.    SunOS 5.10        Generic January 2005
$
```

此后，当用户再次注册到同一远程系统时，就不会再出现上述提示信息了。

13.1.4 使用SSH执行远程系统中的命令

除了具有Telnet远程终端仿真功能之外，ssh还能够在注册后执行远程系统中的单个命令之后立即返回。具体的用法是在ssh命令后面增加一条命令，命令前后使用双引号括起来。在下面的例子中，假定当前系统中的用户想要了解远程系统iscas的操作系统版本，因而需要在远程系统中运行“uname -r”。示例如下：

```
$ ssh iscas "uname -r"
Password:
5.10
$
```

这种一次性地注册，执行远程命令，然后立即返回的功能是非常有用的。结合使用后面将要介绍的无密码注册功能，无论何时都能非常容易地访问远程系统，获取远程系统的各种状态信息。

13.1.5 使用SCP替代FTP

从网络通信的角度来看，FTP的数据传输方式是不安全的，这是因为在网络中传输用户名、密码和数据时，FTP协议没有采取任何加密措施。比较安全的方法是采用OpenSSH的SFTP（Secure FTP）和SCP（Secure Copy）。

SCP是OpenSSH中另一个重要的客户端应用程序，能够在不同的系统之间复制文件，其语法格式类似于常规的cp命令：

```
scp [-P port] [[user@]host1:]file1 [[user@]host2:]file2
```

其中，“-P”选项用于指定TCP端口。随后的第一个参数是源文件，第二个参数是目的文件。当需要复制远程系统中的文件时，SCP首先需要注册到远程系统，注册成功后才能开始传输文件，因此要求提供远程系统的名字、用户名和密码。在引用远程系统中的文件名时，文件

名前面应加用户名与系统名前缀（两者之间需要插入一个“@”字符，之后再加一个冒号“:”分隔符）。远程文件名或目录名的表示形式如下：

```
username@servername:filename
username@servername:directoryname
```

其中，**servername**既可以是远程系统的主机名，也可以是**IP**地址。另外，除非引用的是用户主目录中的文件，指定文件时应给出绝对路径名或针对主目录的相对路径名。例如，如果远程系统的**IP**地址为192.168.0.1，为了访问其中的/etc/profile文件，可以使用“192.168.0.1:/etc/profile”的形式表示远程系统中的文件。为了访问远程系统**guest**用户主目录中的.profile文件，可以使用“guest@servername:.profile”的形式表示其中的文件。

1. 利用scp下载文件

例如，为了把远程系统中的/etc/profile文件复制到本地系统的指定目录/tmp中，可以使用下列命令：

```
$ scp gqxing@iscas:/etc/profile /tmp
Password:
profile                                100% |*****|          712          00:00
$
```

2. 利用scp上传文件

反之，把本地UNIX系统中的文件复制到远程系统中也是一样的。例如，为了把本地系统中的/etc/hosts文件复制到远程的/tmp目录中，可以使用下列命令：

```
$ scp /etc/hosts gqxing@iscas:/tmp
Password:
hosts                                100% |*****|          89          00:00
$
```

13.1.6 使用SFTP替代FTP

OpenSSH也提供一个**SFTP**程序。**SFTP**采用加密的**SSH**会话，实现了**FTP**的文件传输功能。由于**SFTP**也具有**FTP**的目录浏览功能，故**SFTP**比**SCP**更方便实用，尤其是在不知道欲复制文件的确切位置时。但请注意，**SFTP**并不支持传统**FTP**的匿名注册功能。

下面的例子说明了怎样使用**SFTP**注册，查询命令用法，显示以及下载文件等。

```
$ sftp iscas
Connecting to iscas...
The authenticity of host 'iscas (169.254.78.100)' can't be established.
RSA key fingerprint is 6d:de:44:d1:ea:47:54:de:f6:67:94:71:5f:1d:fc:a0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'iscas,169.254.78.100' (RSA) to the list of known
hosts.
Password:
sftp> help
Available commands:
cd path                Change remote directory to 'path'
lcd path               Change local directory to 'path'
chgrp grp path         Change group of file 'path' to 'grp'
```

```

chmod mode path      Change permissions of file 'path' to 'mode'
chown own path       Change owner of file 'path' to 'own'
help                 Display this help text
.....
sftp> ls -l
.....
drwxr-xr-x    0 100    1          512 Jul 15 18:07 backup
drwxr-xr-x    0 100    1          512 Jun 29 17:13 bin
drwxr-xr-x    0 100    1          512 Jun 18 15:41 conf
drwxr-xr-x    0 100    1          512 Jun 30 12:58 doc
-rw-r--r--    0 100    1        6108 Jul  7 13:39 filelist
drwxr-xr-x    0 100    1          512 Jun 19 12:57 incl
drwxr-xr-x    0 100    1        2048 Jul 20 14:53 script
drwxr-xr-x    0 100    1          512 Jun 18 15:34 src
sftp> lcd /tmp
sftp> get filelist
Fetching /home/gqxing/filelist to filelist
sftp> exit
$

```

13.1.7 SSH与SCP的无密码注册

常规情况下，每次使用ssh命令注册远程系统或使用scp命令复制文件时，都需要提供密码，然后才能做进一步的处理。为了省略输入密码这一步骤，可以采用Shell脚本实现，但这种方法需要把手工输入的密码（也即明码）放到脚本文件中。

实际上，适当地利用密钥配置文件，OpenSSH能够省略ssh远程注册与scp文件复制操作过程中的密码验证环节。为了实现无密码注册，客户端应首先建立OpenSSH连接，然后自动向服务器发送其密钥（公钥）。之后，服务器即可根据相应用户主目录中预定义的密钥列表，对收到的密钥进行比较。如果存在匹配的密钥，服务器将会允许ssh或scp自动注册。

用于远程系统注册与数据传输的密钥文件需要事先单独生成，生成后的密钥文件存储在服务器用户主目录的~/.ssh子目录中。其中，私钥和公钥分别存储在id_dsa和id_dsa.pub文件中，authorized_keys文件则用于存储所有授权的远程客户系统的公钥，使得远程客户系统能够以此用户身份注册到本地系统而无需提供密码。

按照上述方法实现无密码注册，唯一的限制是需要把密钥绑定到两个系统的IP地址上。之后，服务器即可使用预安装的密钥，对每一次的远程注册和文件传输进行验证。但这一功能特性并不适合于利用DHCP协议动态分配IP地址，致使IP地址经常发生变化的情况。

由于只需知道用户名，即可实现远程系统注册与文件传输，而无需提供密码，故这种做法存在一定的安全风险。因此，在具体实现时，服务器与客户端之间最好采用普通用户账号，以免造成较大的破坏。

下面以gqxing用户为例，详细说明在实现无密码注册之前，OpenSSH的客户端与服务器双方事先都需要做哪些准备工作。

1. OpenSSH客户端配置

为了实现无密码的系统注册，OpenSSH客户端需要完成下列准备工作。

首先，在客户端与服务器系统中分别创建一个同名的gqxing用户，然后以gqxing用户的身份注册到客户端系统，使用ssh-keygen命令生成一对密钥。当系统提示输入一个与密钥相关联

的密码时，直接按下Enter键。示例如下：

```
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/export/home/gqxing/.ssh/id_dsa): <Enter>
Enter passphrase (empty for no passphrase): <Enter>
Enter same passphrase again: <Enter>
Your identification has been saved in /export/home/gqxing/.ssh/id_dsa.
Your public key has been saved in /export/home/gqxing/.ssh/id_dsa.pub.
The key fingerprint is:
fc:4e:5a:9c:0f:99:bb:82:98:d6:bf:52:d8:b6:5a:0a gqxing@sinosoft
$
```

其次，验证上述步骤生成的两个密钥文件是否均存储在gqxing用户主目录的.ssh子目录中。其中，id_dsa文件中存储的是私钥，id_dsa.pub文件中存储的是公钥（用于提交远程服务器，解密客户端传输的加密数据）。

```
$ ls -l .ssh
total 6
-rw----- 1 gqxing other 668 Jul 20 16:22 id_dsa
-rw-r--r-- 1 gqxing other 602 Jul 20 16:22 id_dsa.pub
-rw-r--r-- 1 gqxing other 215 Jul 20 16:13 known_hosts
$
```

最后，采用下列命令，把生成的公钥文件id_dsa.pub复制到远程系统的gqxing用户主目录中：

```
$ cd .ssh
$ scp id_dsa.pub gqxing@iscas:pub_key
Password:
id_dsa.pub 100% |*****| 602 00:00
$
```

2. OpenSSH服务器配置

在OpenSSH服务器中，以gqxing用户的身份注册到系统，然后使用cat命令和“>>”重定向符号，把pub_key文件中的数据附加到authorized_keys文件的后面：

```
$ cat ~/pub_key >> .ssh/authorized_keys
$ rm ~/pub_key
$
```

authorized_keys文件包含所有OpenSSH客户端系统的公钥列表，如果表中列举的客户系统中的gqxing用户仍以同一用户身份连接到服务器，则无需提供密码。

3. 示例

在完成上述的全部设置之后，当OpenSSH客户端的gqxing用户仍以gqxing的用户身份，使用ssh、scp或sftp等命令连接到远程主机时，就不会再提示用户输入密码了。例如：

```
$ ssh iscas
Last login: Mon Jul 20 16:25:21 2009 from sinosoft
Sun Microsystems Inc. SunOS 5.10 Generic January 2005
$
```


下面是一个利用scp命令复制文件的例子，其情况同ssh命令一样，远程服务器也不要求用户提供密码：

```
$ scp iscas:/etc/hosts .
hosts          100% |*****| 89      00:00
$
```

但这一方法对客户端的其他用户（包括超级用户）无效，即使他们也以gqxing用户的身份注册到远程服务器，仍需提供密码，示例如下：

```
$ ssh -l gqxing iscas
Password:
Last login: Mon Jul 20 17:25:22 2009 from sinosoft
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$
```

13.1.8 OpenSSH的安全考虑

从先前的介绍和表13-1中可知，sshd守护进程通常都会监听服务器所有IP地址的22号TCP端口。为了安全起见，可以修改sshd_config配置文件，把默认的22号TCP端口改为其他约定的空闲端口号（如435）。同时还要把/etc/services文件中的下列两行端口定义：

```
ssh      22/tcp
ssh      22/udp
```

改为

```
ssh      435/tcp
ssh      435/udp
```

修改TCP端口之后，可在OpenSSH客户端中使用下列命令，注册远程系统：

```
$ ssh -p 435 iscas
Password:
Last login: Sat Sep 19 00:07:49 2009 from iscas
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$
```

类似地，为了使用新的TCP端口，把远程系统中的文件复制到本地系统的指定目录中，可以使用下列命令：

```
$ scp -P 435 gqxing@iscas:/etc/profile /tmp
Password:
profile          100% |*****| 712      00:00
$
```

此外，也可以使用AllowUsers、AllowGroups、DenyGroups以及DenyUsers配置参数，或者这些参数的组合，限定用户或用户组的访问。例如，为了限定只有gqxing和cathy两个用户能够访问系统，可在/etc/ssh/sshd_config配置文件中增加下列配置参数：

```
AllowUsers      gqxing cathy
```

重新启动sshd守护进程之后，除了gqxing与cathy两个用户之外，系统将会拒绝接受其他用户的注册，同时输出一个拒绝访问的信息，示例如下：

```
$ ssh -l guest iscas
Password:
Password:
Password:
Permission denied (gssapi-keyex,gssapi-with-mic,publickey,keyboard-interac-
tive).
$
```

修改OpenSSH配置文件之后，为使新的设置立即生效，需要重新启动sshd守护进程。在Solaris系统中，只需采用下列命令终止sshd守护进程后即可自动重启该进程：

```
# kill -9 `pgrep sshd`
#
```

13.2 Telnet远程系统注册

提供交互式远程访问是多用户操作系统的一个重要特征，通常，UNIX系统均支持telnet和ftp等应用服务。telnet命令的语法格式简写如下：

```
telnet [options] hostname-or-ipaddr [port]
```

其中，hostname-or-ipaddr是远程系统的主机名或IP地址，port是TCP端口号。当使用下列telnet命令连接远程UNIX系统时，系统将会显示一系列信息，提示用户输入用户名与密码。如果输入的用户名和密码通过验证之后，即可进入UNIX系统，打开默认的注册Shell，输出命令提示符。至此，用户即可输入命令，交互访问远程系统。如果用户名和密码不匹配，就会拒绝用户的注册尝试。例如：

```
$ telnet 169.254.78.100
Trying 169.254.78.100...
Connected to 169.254.78.100.
Escape character is '^]'.
login: gqxing
Password:
Last login: Sat Jul 25 02:29:25 from iscas
Sun Microsystems Inc. SunOS 5.10 Generic January 2005
$
```

如果输入telnet命令是没有提供远程系统的主机名或IP地址，可以使用telnet的open子命令连接远程主机：

```
$ telnet
telnet> open (也可以把open与随后的IP地址合并为一个命令)
(to) 169.254.78.100
Trying 169.254.78.100...
Connected to 169.254.78.100.
Escape character is '^]'.
login: gqxing
Password:
Last login: Sat Jul 25 02:36:37 from iscas
Sun Microsystems Inc. SunOS 5.10 Generic January 2005
$
```

但是，这种注册访问仅限于普通用户，即只能使用普通用户账号注册到UNIX系统，UNIX系统不允许用户在控制台之外的其他终端上（或远程系统中），以超级用户的身份直接注册。若想直接使用root注册，需要在控制台上修改/etc/default/login文件，在“CONSOLE=/dev/console”之前增加一个注释符“#”，即可取消此限制。

实际上，telnet充其量只是一个终端仿真程序，其主要功能无非就是支持用户访问远程系统而已。但是，如果熟悉TCP/IP的各种协议，则可以利用telnet和相应的服务端口，连接相应的TCP/IP服务程序，如FTP服务器、Web服务器或电子邮件服务器，与之直接对话，测试服务器是否已经启动，工作是否正常等。

13.3 FTP文件传输

inetd提供的另外一个常用服务就是文件传输协议FTP。FTP允许用户在本地系统与远程系统之间传输文件。利用ftp命令，可以从远程系统中下载或上传文件。ftp命令的语法格式简写如下：

```
ftp [-dginv] [hostname [port]]
```

其中，hostname是远程系统的主机名或IP地址。port是ftp服务的端口号。表13-3给出了部分选项的简单说明。

表13-3 ftp命令的部分选项

选项	简单说明
-d	启用调试功能
-g	禁止在文件名中使用通配符生成文件名
-i	在传输多个文件期间，关闭交互提示与确认模式
-n	禁止初始连接时使用“自动注册”功能。如果未禁止自动注册，ftp将会检测用户主目录中的.netrc文件，检索其中提供的远程系统账号信息。如果检索失败，或.netrc文件的访问权限设置不正确，ftp将会提示用户输入远程系统的用户名或密码
-v	显示远程系统提供的所有响应信息，以及数据传输的统计信息。采用交互方式运行ftp时，这是一个默认的选项

ftp命令提供许多内部子命令，借助于这些子命令，可以实现各种文件传输功能。为了查询ftp提供了哪些可用的内部子命令，可在ftp命令提示符“ftp>”下输入“?”命令，然后按Enter键。表13-4给出了部分ftp内部子命令及其解释。

表13-4 部分ftp内部子命令

ftp内部子命令	简单说明
! [cmd]	在本地系统的Shell环境中执行指定的命令。如果未提供命令，则进入交互式的Shell运行环境
append local-file [remote-file]	把本地文件附加到远程主机的文件中。如果未指定远程文件，则表示附加到远程系统的同名文件中
ascii	以ASCII代码形式传输文件。这是默认的传输方式

(续表)

ftp内部子命令	简单说明
bell	文件传输命令完成后以响铃方式提醒用户
binary	以二进制代码形式传输文件
bye	终止与远程FTP服务器的会话，并退出ftp命令
case	用于控制在执行mget命令期间是否对远程文件名进行大小写转换。当case设置为on（默认值为off）时，远程文件名中的所有大写字母在复制到本地目录时均改为小写字母
cd remote-dir	把远程系统上的当前工作目录改换到指定的目录
cdup	把远程主机上的当前工作目录改换到其父目录
close	终止与远程FTP服务器的会话，清除先前定义的宏，并退出ftp命令
cr	用于控制下载文件时是否删除回车字符。当cr为on（默认值）时，将删除回车字符，以便与UNIX系统中仅以换行字符为行终止符的惯例保持一致
delete remote-file	删除远程系统中的指定文件
debug	打开或关闭调试模式。当打开调试模式时，ftp将会显示发送到远程主机的每一个命令，同时在命令前面冠以“->”标志
dir [remote-dir] [local-file]	列出远程主机指定目录下的文件和目录，并把输出结果保存在指定的本地文件中。如果未指定远程目录，则默认为远程主机的当前工作目录。如果未指定本地文件，或指定的本地文件为“-”，则输出结果将送到终端中显示
get remote-file [local-file]	下载远程文件，并储存在本地系统上。如果未指定本地文件名，则采用与远程文件相同的名字命名本地文件
glob	针对mdelete、mget和mput命令，表示启用或禁用文件名生成机制。如果禁用文件名生成机制，则文件名中的通配符将会按普通字符处理
hash	确定是否对传输的每一个数据块都输出一个井号（#）字符。数据块的长度为8192字节
help [cmd]	显示指定内部命令的说明信息。如果未指定任何命令，ftp将会列出所有可用的内部命令
lcd [dir]	改变本地系统上的当前工作目录。如果未指定目录，则使用用户的主目录
ls [-al] [remote-dir] [local-file]	通常，ls命令将会以最简单的形式列出远程主机中的指定目录或当前工作目录下的文件和子目录。通过修改/etc/default/ftp文件，把其中的FTP_LS_SENDS_NLST变量设置为no，可以使ls与dir命令的输出形式一致。其中，“-a”选项意味着列出所有文件，包括以句点“.”为首字符的隐藏文件。“-l”选项意味着以长列表格式列出每一个文件，给出文件的类型与访问权限、链接数、文件属主、用户组、文件的字节数以及最后一次修改时间等属性。其效果类似于UNIX系统中的“ls -l”命令。如果未指定远程目录，默认值为当前工作目录。如果未指定本地文件名，或者本地文件名是“-”，输出内容将直接送到终端
mdelete remote-files	可以同时删除远程主机中的多个文件
mget remote-files	按文件名生成机制展开指定的远程文件名，对于每个文件执行一次get操作。下载的文件将存于本地工作目录或之前由“lcd dir”命令指定的目录中。如果需要，也可以使用“! mkdir dir”命令创建新的本地目录
mkdir dir-name	在远程系统上创建指定的目录

(续表)

ftp内部子命令	简单说明
<code>mput local-files</code>	一次上传多个文件。其具体实现是展开由通配符表示的本地文件名，使之作为put命令的参数，对每个文件执行一次put操作
<code>open host [port]</code>	与指定的FTP服务器建立连接。如果给出了端口号，ftp将会尝试按指定的端口号联系FTP服务器。如果“自动注册”选项的状态为on（默认设置），ftp还会尝试自动注册到FTP服务器中
<code>prompt</code>	启用或禁用交互提示处理方式。在传输多个文件期间，交互提示处理方式允许用户选择下载或上传的文件。在默认情况下，ftp启用交互提示处理方式。如果禁用交互提示处理方式，mget或mput将会不加提示地传输所有的文件
<code>put local-file [remote-file]</code>	把本地文件上传到远程主机。如果未指定远程文件名，则使用本地文件名作为远程文件名
<code>pwd</code>	显示远程主机上的当前工作目录
<code>quit</code>	同bye
<code>reget remote-file [local-file]</code>	其功能类似于get，不同的是，如果本地文件存在，且小于远程文件，则假定本地文件是远程文件部分传输的副本，文件应当从断点处开始继续传输。当传输较大的文件，中途经常因故断开网络连接，致使文件传输中断时，这个命令是非常有用的
<code>remotehelp [cmd-name]</code>	获取远程FTP服务器的帮助信息（服务器命令列表）。如果指定了具体的命令，仅给出相应命令的使用说明（命令的语法格式）
<code>rename from to</code>	重新命名远程主机中的文件
<code>restart [marker]</code>	从get或put设定的标记之后开始传输。在UNIX系统中，这个标记通常是文件的偏移位置
<code>rmdir dir-name</code>	删除远程主机中的目录
<code>status</code>	显示ftp的当前运行状态
<code>user user-name [password]</code>	以给定的用户名注册FTP服务器。如果未同时给出密码，ftp将会进一步提示用户输入密码，并禁止本地回显
<code>verbose</code>	启用或禁用详细显示模式。在详细显示模式中，来自FTP服务器的所有响应信息均输出到用户终端。此外，当文件传输完成时，还将显示文件传输效率的统计信息。默认情况下，如果ftp命令来自终端输入，则启用详细显示模式。否则禁用详细显示模式
<code>? [command]</code>	同help

13.3.1 连接FTP服务器

连接FTP服务器主要有两种方式。常用的连接方式是在ftp命令后面直接提供FTP服务器的主机名或IP地址。例如：

```
$ ftp iscas
Connected to iscas.
220 iscas FTP server ready.
Name (iscas:root): gqxing
331 Password required for gqxing.
```

```

Password:
230 User gqxing logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>

```

另外一种连接方式是利用ftp的open子命令，连接FTP服务器。

```

$ ftp
ftp> open          (也可以把open与随后输入的IP地址合并为一个命令)
(to) 192.168.90.100
Connected to 192.168.90.100.
220 iscas FTP server ready.
Name (192.168.90.100:root): gqxing
331 Password required for gqxing.
Password:
230 User gqxing logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>

```

当完成文件传输，准备退出FTP服务器时，可以使用quit或bye子命令：

```

ftp> quit
221-You have transferred 983040000 bytes in 2 files.
221-Total traffic for this session was 983040526 bytes in 2 transfers.
221-Thank you for using the FTP service on localhost.
221 Goodbye.
$

```

13.3.2 FTP应用

为了传输文件，可以使用ftp的get（下载）或put（上传）子命令。为了查询FTP服务器当前目录中存在的文件，然后下载其中的某个文件，可以使用ftp的ls或dir子命令。例如：

```

ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
bin
conf
doc
incl
memo
script
src
226 Transfer complete.
43 bytes received in 0.00016 seconds (256.35 Kbytes/s)
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
总数 18
-rw-r--r--  1 gqxing  other      144  7月 25日 02:14 .profile
-rw-----  1 gqxing  other      602  7月 25日 15:28 .sh_history
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 bin

```



```

drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 conf
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 doc
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 incl
-rw-r--r--  1 gqxing  other      986  7月 25日 15:27 memo
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 script
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 src
226 Transfer complete.
588 bytes received in 0.0002 seconds (2873.15 Kbytes/s)
ftp> get memo
200 PORT command successful.
150 Opening BINARY mode data connection for memo (986 bytes).
226 Transfer complete.
local: memo remote: memo
986 bytes received in 0.00098 seconds (986.93 Kbytes/s)
ftp>

```

从上述ls和dir子命令的输出结果可以看到，ls子命令的输出信息太少。为了改变ls子命令的输出形式，可以删除/etc/default/ftp文件中“#FTP_LS_SENDS_NLST=NO”之前的注解字符“#”。当再次进入ftp执行ls子命令时，即可看到与dir子命令同样的输出结果。

下面的例子说明了修改/etc/default/ftp文件之后，ls子命令输出结果的变化：

```

ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
总数 18
-rw-r--r--  1 gqxing  other      144  7月 25日 02:14 .profile
-rw-----  1 gqxing  other      614  7月 25日 15:31 .sh_history
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 bin
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 conf
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 doc
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 incl
-rw-r--r--  1 gqxing  other      986  7月 25日 15:29 memo
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 script
drwxr-xr-x  2 gqxing  other      512  7月 25日 15:23 src
226 Transfer complete.
588 bytes received in 0.00023 seconds (2515.32 Kbytes/s)
ftp>

```

当需要一次传输多个文件时，可以使用ftp的mget或mput子命令。但在使用mget或mput命令时，ftp通常总是在传输每一个文件之前提请用户确认一次。如果文件太多，将不胜其烦。为了避免这一麻烦，可以使用ftp的“-i”选项，关闭提示与确认模式。例如：

```

$ ftp -i iscas
Connected to iscas.
220 iscas FTP server ready.
Name (iscas:gqxing): gqxing
331 Password required for gqxing.
Password:
230 User gqxing logged in.
Remote system type is UNIX.
Using binary mode to transfer files.

```

```
ftp> cd /usr/include/sys/fs
250 CWD command successful.
ftp> lcd /tmp
Local directory now /tmp
ftp> mget *.h
200 PORT command successful.
150 Opening BINARY mode data connection for autofs.h (8337 bytes).
226 Transfer complete.
local: autofs.h remote: autofs.h
8337 bytes received in 0.11 seconds (74.91 Kbytes/s)
.....
200 PORT command successful.
150 Opening BINARY mode data connection for zfs.h (18287 bytes).
226 Transfer complete.
local: zfs.h remote: zfs.h
18287 bytes received in 0.0006 seconds (29671.87 Kbytes/s)
ftp> quit
221-You have transferred 420958 bytes in 48 files.
221-Total traffic for this session was 429948 bytes in 49 transfers.
221-Thank you for using the FTP service on iscas.
221 Goodbye.
$
```

13.3.3 FTP访问控制

为了实现FTP的访问控制，除了使用前述的inetd.conf文件关闭FTP服务程序之外，还可以使用/etc/ftpusers文件限制用户级的FTP访问。在安装UNIX系统之后，系统提供的ftpusers文件通常包含超级用户root，以及daemon、bin和sys等系统管理用户。

为了限制其他用户访问FTP服务，可以把相应的用户名加到ftpusers文件中。其中，每个用户名占用一行。为了允许用户使用FTP服务，可以在用户名前增加一个注解符号“#”，或直接删除相应的用户名。

下面是一个ftpusers文件的例子。其中，由于在root前增加了一个注解字符“#”，因而允许超级用户root访问FTP服务器，但禁止daemon和guest等用户访问：

```
$ cat /etc/ftpd/ftpusers
# root
daemon
.....
guest
$
```

13.3.4 FTP自动注册

每次使用ftp命令传输文件之前，都需要提供用户名和密码，才能建立与FTP服务器之间的连接。为了启用FTP的自动注册功能，简化操作步骤起见，用户可以在自己的主目录中创建一个.netrc文件，把远程主机名、用户名和密码等信息加到其中。.netrc文件的语法格式如下：

```
machine hostname login username [password password]
```

其中，关键字machine用于定义远程主机名。关键字login用于指定远程主机中的用户名。

关键字password用于提供密码（为安全起见，这个字段可以省略）。一旦设置了这个文件，即可实现自动注册。

例如，为了便于用户gqxing能够自动注册到iscas、host2以及host3三个FTP服务器，我们可以创建下列.netrc文件（假定gqxing在这三个FTP服务器中的用户名也是gqxing）：

```
$ cat $HOME/.netrc
# $HOME/.netrc file
#
machine iscas login gqxing password hereiam
machine host2 login gqxing password hello
machine host3 login gqxing password itsme
$
```

注意：如果.netrc文件中提供了密码，且由于密码是以明文形式给出的，FTP要求用户必须使用下列命令设置.netrc文件的访问权限，确保只有用户自己能够读写此文件。否则，FTP将拒绝执行自动注册过程：

```
$ chmod 400 $HOME/.netrc （或chmod 600 $HOME/.netrc）
$
```

在完成上述两个步骤之后，只要输入ftp命令，立刻就会建立FTP连接，进入ftp命令状态，从而省略输入用户名和密码这两个步骤。示例如下：

```
$ ftp iscas
Connected to iscas.
220 iscas FTP server ready.
331 Password required for gqxing.
230 User gqxing logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

第14章 网络文件系统

网络文件系统（Network File System, NFS）是一种分布式文件系统，能够使用户透明地访问远程资源。利用NFS，用户能够共享文档、数据和程序等。对于用户而言，远程资源无非就是本地文件系统中的—个目录层次，与其他目录和文件没有任何差别。本章将讨论怎样设置NFS服务器和NFS客户系统，以及怎样共享远程资源。

14.1 NFS简述

在NFS网络文件系统中，客户系统与服务器用于描述每个系统在文件系统共享中各自扮演的角色。通常把通过网络提供目录或文件系统等共享资源的主机称做NFS服务器，访问共享资源的系统称做NFS客户系统。

NFS是一种基于远程过程调用（Remote Procedure Call, RPC）协议，采用客户端/服务器结构实现的分布式文件系统。在NFS的网络环境中，一个UNIX系统既可以作为NFS服务器，提供共享目录或文件系统，供NFS客户系统访问，也可以作为NFS客户系统，安装其他NFS服务器提供的远程资源，访问其中的目录和文件资源。

远程资源可以安装在本地系统的任何目录位置。一旦安装了NFS服务器提供的共享目录或文件系统，只要给予足够的访问权限，客户系统用户就可以像访问本地文件系统一样，透明地访问服务器中的远程目录和文件，可以读写、创建和删除其中的文件与目录。

NFS是一种强有力的集中数据存储与客户访问工具，使客户系统能够快速访问大批量的数据，如存储在中央数据库中的业务数据。尤其是，还可以采用NFS的方式，利用NFS服务器中存储的CD/DVD影像数据，直接安装各种系统软件，如采用NFS方式安装Solaris系统等。另外，还可以利用NFS服务器存储引导程序，直接引导客户系统。

当前可用的NFS主要有三个版本，即版本2、3和4（版本1只是一个原型）。NFS版本2的主要功能特点如下：

- 数据传输单位的上限为8KB；
- 处理的文件最大不能超过2GB；
- 使用UDP作为底层传输协议；
- 仅当把数据完全写到NFS服务器的磁盘之后才算完成数据的写请求。

NFS版本3是在版本2基础上的增强与改进，其主要的功能特点如下：

- 数据传输单位的上限为32KB（UDP）或1MB（TCP），实际的数据传输限制是客户系统与服务器协商后共同确定的一个优化值；
- 能够处理大于2GB的文件；
- 支持TCP和UDP，并以TCP作为首选的底层传输协议；
- 支持异步处理，把数据写到NFS服务器缓冲区之后就算完成了数据的更新，客户系统不需要等待，因而改善了响应时间；

• NFS服务器能够以批处理的方式，集中、统一处理客户系统的访问请求，因而提高了NFS服务器的数据处理性能。

NFS版本4又是在版本3基础上的增强与改进，其主要的功能特点如下：

- 支持Internet与Web等网络应用环境，改善并提高了网络访问的性能与安全性；
- 扩展了NFS文件系统的文件属性，支持访问控制表ACL；
- 采用强制性的文件锁（之前是选用的），文件的加锁与远程安装均集成到NFS的后台守护进程；

- 使用单一的TCP端口，用于增强网络安全；
- 增加了基于RPCSEC_GSS（Remote Procedure Call Security - Generic Security Services）的API，采用GSS-API（Generic Security Services Application Programming Interface），从客户系统与服务器中选用双方均支持的安全机制。

注意：客户系统与服务器中运行的NFS版本必须匹配和兼容，才能确保NFS网络文件共享的顺利实现。

14.2 配置NFS服务器

当NFS服务器需要提供共享资源，对外公布某个目录或文件系统时，可以使用share命令实现。share命令主要用于提供或查询本地系统的共享资源，其语法格式简写如下：

```
share [-F nfs] [-d description] [-o specific_options] pathname
```

其中，“-F”选项说明共享的文件系统类型为NFS。“-d”选项用于描述共享资源，以便用户参考。“-o”选项用于指定文件系统或目录的共享方式与访问限制，其中可以包括一系列形如“key=value”的参数，参数之间加逗号分隔符。如果未指定此选项，默认的共享方式和访问权限是任何用户均可以读写共享资源。表14-1给出了部分常用的选项及其参数。

表14-1 share命令的部分选项及其参数

参数	简单说明
anon=uid	设置匿名或未知用户的有效用户ID。通常，匿名或未知用户的有效用户ID为UID_NOBODY。如果把有效用户ID设置为“-1”，意味着拒绝匿名或未知用户访问共享文件系统
ro	以只读方式提供共享目录或文件系统，禁止客户系统在远程共享资源中写入文件
ro=access_list	用于限定访问共享资源的客户系统。等号“=”右边可以采取主机名列表的形式给出部分客户系统，表示只有限定的客户系统才能以只读方式访问相应的共享资源
root=access_list	NFS服务器不允许任何客户系统中的超级用户仍以超级用户的身份访问共享资源。而且，客户系统中的超级用户通常均被映射为匿名用户。如果在等号“=”右边以主机名列表的形式给出部分客户系统，表示只有限定客户系统中的超级用户访问共享资源时才具有超级用户的权限
rw	允许读写远程目录或文件系统等共享资源
rw=access_list	用于限定访问共享资源的客户系统。等号“=”右边可以采取主机名列表的形式给出部分客户系统，表示只有限定的客户系统才能读写相应的共享资源

（续表）

参数	简单说明
nosub	用于防止客户系统安装共享目录的子目录（注意，nosub选项仅适用于NFS V2和NFS V3）。例如，如果共享/export目录时使用了nosub选项，则NFS客户系统不能使用下列命令安装其中的子目录： mount -F nfs iscas:/export/customer /mnt
nosuid	通常，客户系统可以在共享资源中创建具有setuid或setgid标志位的文件。如果指定了nosuid选项，NFS服务器的文件系统将会拒绝上述企图，并忽略任何设置setuid或setgid标志位的尝试

假定存在4个系统，其主机名分别为iscas、support、product及marketing，iscas为NFS服务器。为了便于support和product客户系统中的用户访问/export/contract共享目录中的文件，可在iscas系统中使用下列命令提供共享资源：

```
# share -F nfs -d "contract" -o ro=support:product /export/contract
#
```

为使marketing系统中的用户能够读写共享目录/export/customer中的文件。可在iscas系统中使用下列命令提供共享资源：

```
# share -F nfs -d "customer" -o rw=marketing /export/customer
#
```

如果想要查询系统中当前都公布了哪些共享资源，可以不加任何选项，直接运行share命令：

```
# share
-      /export/contract      ro=support:product "contract"
-      /export/customer      rw=marketing      "customer"
#
```

注意：利用不加任何选项的share命令，只能查询本地系统提供的共享资源，如果想要了解其他NFS服务器都提供哪些共享资源，可以使用dfshares命令，其语法格式简写如下：

```
dfshares [-F nfs] [-h] [server]
```

其中，“-h”选项表示禁止输出标题栏。server表示查询指定的NFS服务器。

例如，为了查询任何指定的NFS服务器都提供哪些共享资源，可以使用下列命令：

```
# dfshares iscas
RESOURCE                                SERVER      ACCESS      TRANSPORT
      iscas:/export/contract            iscas       -            -
      iscas:/export/customer            iscas       -            -
#
```

使用share命令行方法提供共享资源只是一种临时性的设置，仅对当前系统有效。一旦关机或重新启动系统，share命令设定的共享资源就会取消。为了在每次启动系统后都能自动提供共享的目录或文件系统，可把上述share命令加到/etc/dfs/dfstab文件中，从而实现本地文件系统或目录的自动共享。

```
share -F nfs -d "contract" -o ro=support:product /export/contract
share -F nfs -d "customer" -o rw=marketing /export/customer
```


一旦需要临时取消某个共享的目录或文件系统，可以使用unshare命令：

```
# unshare -F nfs /export/contract
# unshare -F nfs /export/customer
#
```

在取消共享之前，首先应检查一下是否有NFS客户机仍在继续使用共享资源。为此，可以使用dfmounts命令进行检查。其语法格式简写如下：

```
dfmounts [-F nfs] [-h] [server]
```

dfmounts命令用于显示通过NFS方式共享的本地资源，以及已经安装了本地资源的客户系统列表。如果运行dfmounts命令未加任何选项，表示显示安装到本地系统的所有远程资源。dfmounts命令的输出信息通常提供下列标题：

```
RESOURCE SERVER PATHNAME CLIENTS
```

其中，RESOURCE字段只是一个减号“-”，用做位置占用符，并无实际的意义。SERVER字段表示安装的远程资源是由哪一个服务器提供的。PATHNAME字段是一个与share命令行相对应的路径名。CLIENTS字段给出的是一系列安装了远程资源的客户系统列表，客户系统名字之间由逗号分隔。例如：

```
# dfmounts -F nfs iscas
RESOURCE      SERVER  PATHNAME      CLIENTS
-             iscas  /export/contract  support,product
-             iscas  /export/customer  marketing
#
```

上述输出结果表明，NFS客户系统marketing已经安装了共享资源/export/customer，support和product已经安装了共享资源/export/contract，因此需要事先通知客户系统的系统管理员先行卸载。

此外，为了查询共享目录或文件系统资源的公布与使用情况，也可以使用showmount命令。showmount命令主要用于查询NFS服务器提供的共享资源、运行状态以及客户系统的安装信息。其语法格式简写如下：

```
showmount [-ade] [hostname]
```

其中，“-a”选项表示以“host:directory”形式显示已经安装了远程资源的NFS客户主机名（或IP地址）以及安装的目录或文件系统。“-d”选项仅用于显示NFS客户系统安装的目录或文件系统。“-e”选项用于显示NFS服务器公布的共享目录或文件系统。

例如，除了运行share命令以及其查询/etc/dfs/dfstab文件之外，也可以使用下列showmount命令显示NFS服务器已经公布的共享目录或文件系统：

```
# showmount -e
export list for iscas:
/export/contract      support,product
/export/customer      marketing
#
```

下列命令将会列出客户系统已经安装的远程目录或文件系统：

```
# showmount -d
/export/contract
/export/customer
#
```

14.3 配置NFS客户系统

配置NFS客户系统通常主要包括两项任务：采用mount命令，手工安装NFS服务器提供的远程目录或文件系统等共享资源；利用/etc/vfstab文件，自动安装远程共享资源。必要时，还需要创建本地目录，作为远程共享资源的安装点。

14.3.1 安装远程文件系统

在NFS客户系统中，可以使用mount命令，直接安装NFS服务器提供的远程目录或文件系统。mount命令的语法格式简写如下：

```
mount [-F nfs] [generic_opts] [-o specific_opts] resource mount-point
```

其中，“-F”选项表示安装NFS网络文件系统，“resource”表示远程共享资源的目录或文件系统名，由形如“host:pathname”的远程主机名与路径名组成。“mount-point”为本地系统的安装点，即安装文件系统的目录位置。mount命令的通用安装选项、NFS的特定安装选项及其说明如表14-2所示。

表14-2 mount命令的部分安装选项

选项	简单说明
bg fg	在安装远程目录或文件系统时，如果第一次安装失败，客户系统将会尝试以后台或前台方式重新安装远程目录或文件系统，直至安装成功，或者因为达到指定的尝试安装次数而放弃。默认值为fg，即以前台方式尝试安装
exec noexec	表示允许或禁止执行远程目录或文件系统程序，其默认值为exec
hard soft	表示一旦NFS服务器在规定的超时值范围内没有响应NFS客户系统的安装与访问请求时如何处理，其默认值为hard。hard选项表示，当NFS安装失败时将会不断地尝试安装，直至服务器响应访问请求为止，因而能够确保数据的完整性和一致性。soft选项意味着，当NFS安装失败时立即放弃，然后返回一个错误信息。由此可见，采用soft选项安装远程目录或文件系统无法保证数据的完整性和一致性。当以读写形式访问NFS远程资源，或安装一个包含可执行文件的远程目录或文件系统时，应当总是使用hard选项
intr nointr	指定是否允许使用中断键（Ctrl-C）或kill命令终止因等待NFS服务器响应而挂起的文件访问进程。对于采用hard选项安装的远程目录或文件系统而言，一旦出现了超时，允许用户使用中断键或kill命令终止调用的程序
noac	禁止使用文件数据与属性的缓冲处理机制。这个选项虽然会影响NFS的性能，但却能保证数据的一致性，使同时访问同一服务器资源的不同NFS客户系统能够得到一致的准确结果。为了改善性能，NFS客户系统通常都会缓存文件的数据与属性信息，定时查询服务器，更新文件的属性信息。noac选项表示把数据立即写入NFS服务器，防止NFS客户系统缓存文件的数据与属性信息，使得应用程序能够更快地检测到NFS服务器中的文件属性变化，同时也强制应用程序采用同步方式实现文件的写操作，确保服务器总是能够维护最新的文件数据与属性信息，其他客户系统也能够及时了解文件的最新变动。由此可见，采用noac选项能够确保数据的一致性，但以降低性能为代价

(续表)

选项	简单说明
nosuid suid	如果指定了nosetuid选项，则表示忽略远程目录或文件系统中二进制文件的setuid与setgid标志位，进程的访问权限与运行二进制文件的用户完全相同。如果指定了setuid选项，系统内核将会按照常规处理二进制文件的setuid与setgid标志位。如果两者均未明确指定，默认的安装选项是suid。nosuid选项能够提供更多的安全保障，尤其是在访问信任度较低的NFS服务器时，可以减少设置了setuid标志位的不良二进制文件侵袭客户系统的机会
port=n	指定NFS服务器的端口号，其默认值是2049
retrans=n	用于指定NFS支持的数据重传次数，其默认值为5。当底层协议为面向连接的TCP时，这个选项没有任何意义
retry=n	指定在安装远程目录或文件系统失败时，尝试重新安装的次数，对于mount命令而言，其默认值为10000；对于automountd守护进程而言，其默认值为1。当NFS服务器的负载较重时，增加这个数值，可以避免出现“server not responding”之类的错误信息
ro rw	指定以只读或读写方式安装与访问远程目录或文件系统，其默认值为“rw”
rsiz=n	设置文件读缓冲区的最大字节数。默认值取决于NFS的版本与底层协议，对于NFS V3和V4，当底层协议为面向连接的TCP时，其默认值为1048576；当底层协议为无连接的UDP时，其默认值为32768。NFS V2的默认值为32768（或8192）。但是，这个数值经过双方协商后可以下调，最终的实际数值是NFS服务器与客户系统能够支持的最大值经过协商后的结果。通常，这个数值的设置不能小于NFS服务器与客户系统双方都支持的最大值，否则将会影响NFS的性能
timeo=n	指定NFS超时值（以1/10秒为单位），对于无连接的UDP底层传输协议，其默认值为1.1秒。对于面向连接的TCP底层传输协议，默认的超时值为60秒
vers=n	使用指定的NFS版本尝试安装远程目录或文件系统。通常，NFS客户系统与服务器最终采用的NFS版本是双方系统均支持的高版本，其主要目的是与其他许多操作系统保持兼容
wsiz=n	设置文件写缓冲区的最大字节数。默认值取决于NFS的版本与底层协议，对于NFS V3和V4，当底层协议为面向连接的TCP时，其默认值为1048576；当底层协议为无连接的UDP时，其默认值为32768。NFS V2的默认值为32768（或8192）。同样，这个数值经过双方协商后可以下调，最终的实际数值是NFS服务器与客户系统能够支持的最大值经过协商后的结果。而且，这个数值的设置不能小于NFS服务器与客户系统双方都支持的最大值，否则将会影响NFS的性能

假定NFS服务器的设置已经就绪，NFS客户系统可以在本地文件系统的目录层次结构中确定一个安装点，或创建一个新的目录，然后利用mount命令，把NFS服务器提供的远程资源安装到自己的文件系统中。例如，为了安装iscas服务器提供的共享资源，我们可以建立下列目录，然后把远程目录安装到新建的安装点：

```
# mkdir /customer
# mount -F nfs iscas:/export/customer /customer
# ls -l /customer
total 46
-rw-r--r-- 1 root root 6268 Jul 28 21:30 abc
-rw-r--r-- 1 root root 4672 Jul 28 21:28 boc
-rw-r--r-- 1 root root 4629 Jul 28 21:32 ccb
-rw-r--r-- 1 root root 5470 Jul 28 21:33 icbc
#
```

如果远程目录或文件系统已经安装，想要了解安装时使用的安装选项（或默认安装选项），如NFS版本、`rsiz`以及`wsiz`等，可以使用下列`nfsstat`命令：

```
# nfsstat -m
/customer from iscas:/export/customer
Flags:          vers=4,proto=tcp,sec=sys,hard,intr,link,symlink,acl,
rsiz=1048576,wsiz=1048576,retrans=5,timeo=600
Attr cache:    acregmin=3,acregmax=60,acdirmin=30,acdirmax=60
#
```

14.3.2 设置/etc/vfstab文件

上述安装方法属于临时性的安装，重新启动系统之后，原来安装的远程文件系统将不复存在。在许多情况下，用户可能需要长期安装一定的远程共享目录或文件系统。为了一劳永逸地自动安装远程资源，需要借助于UNIX系统的`/etc/vfstab`文件。按照`vfstab`文件的格式要求，把上述命令经过适当地修改之后再加到文件中。读者可参见第17章中有关`vfstab`文件的介绍。

以上述远程共享资源为例，为了自动安装远程目录或文件系统，可在NFS客户系统的`vfstab`文件中增加下列一行内容（前两行仅做字段说明）：

# device	device	mount	FS	fsck	mount	mount
# to mount	to fsck	point	type	pass	at boot	options
iscas:/export/customer	-	/mnt	nfs	-	yes	rw

注意：在NFS的所有版本中，目前最流行的仍是版本2，因为大多数NFS客户系统均使用这一版本，但较新的NFS服务器一般都支持NFS版本4。为了确保具有较好的兼容性与适应性，最好在NFS客户系统的`/etc/vfstab`文件中增加`vers=2`安装选项，从而强制NFS服务器按照版本2提供共享目录或文件系统。此外，此时无需指定“`device to fsck`”和“`fsck pass`”字段，因为这是一个NFS文件系统。

之后，每当重新启动NFS客户系统时，都会读取`/etc/vfstab`文件，在执行`mountall`命令期间，自动安装`iscas`服务器提供的`/home/customer`共享目录。一旦在NFS客户系统中安装了远程目录或文件系统之后，用户就可以像使用本地文件系统一样地访问远程目录或文件系统。

在修改`vfstab`文件之后，为了立即安装新增的远程共享目录资源，可以使用下列`mountall`命令：

```
# mountall
# ls /customer
abc boc ccb icbc
#
```

不需要继续访问远程共享资源时，可以利用下列`umount`命令卸载远程目录或文件系统：

```
# umount /customer
#
```

14.4 NFS自动安装

远程文件系统的永久性安装存在一定的缺陷。由于每个UNIX服务器中的`/etc/vfstab`文件是唯一的，因而需要编辑每一个客户系统中的`vfstab`文件。因此，对NFS客户系统的管理与维护便

是一个非常困难的问题。从资源利用方面来看, 不管是否访问NFS服务器, 这种永久性的安装对NFS客户系统与服务器都是一种CPU与网络带宽等的极大资源浪费。

利用NFS特定的映射文件与自动安装功能, 可以绕过/etc/vfstab文件, 直接实现NFS远程资源的自动安装, 因而也就克服了上述的诸多问题。当再次收到NFS文件访问请求时, automountd守护进程将会按照映射文件的定义, 把远程资源重新安装到指定的安装点, 而安装点是由automountd守护进程根据需要自动创建的。

14.4.1 主映射文件

在NFS的自动安装机制中, 除了automountd等NFS守护进程之外, NFS服务器几乎可以不知道、也不关心NFS客户系统究竟是采用mount命令, 还是采用automountd守护进程安装共享目录或文件系统的。而在NFS客户端, 除了automountd等NFS守护进程之外, 至少还要准备下列3个映射文件:

- 主映射文件auto_master;
- 间接映射文件(如auto_home, 文件名可由主映射文件指定);
- 直接映射文件(如auto_direct, 文件名可由主映射文件指定)。

主映射文件auto_master是一个总的映射配置, 其中定义其他映射文件, 以便进一步说明安装点与远程资源之间的映射关系, 使得automountd守护进程能够根据主映射文件的定义, 按照需要自动安装远程资源。主映射文件的语法格式如下:

```
mount-point map-file [mount-options]
```

其中, mount-point用于定义一个具有绝对路径名的目录, 以便在此目录中安装NFS服务器提供的远程共享资源。对于直接映射而言, 这个字段为“/”, 实际的安装点是直接映射文件中第一个字段的值。对于间接映射而言, 这个字段只是安装点的根目录, 实际的安装点应是这个字段值与间接映射文件第一个字段值的组合。如果指定的目录不存在, autofs将会尽可能地创建一个目录。如果目录存在, 且目录不为空, 一旦安装后将会隐藏其中的内容。在此情况下, autofs将会发出一个警告信息。第二个字段的map-file用于指定直接映射文件或间接映射文件。第三个字段的mount-options是mount命令或automountd守护进程可用的安装选项。

在下列auto_master示例文件中, /home目录是NFS客户系统中的一个安装点。最终的安装点是/home与间接映射文件/etc/auto_home中第一个字段所列目录的组合。最后一列是安装选项, 参见表15-2。如果没有特别指定, 这个安装选项将会作为/etc/auto_home文件中所有安装项的默认值:

```
$ cat /etc/auto_master
.....
# Master map for automounter
#
+auto_master
/net          -hosts          -nosuid,nobrowse
/home         auto_home       -nobrowse
$
```


14.4.2 直接映射文件

顾名思义，直接映射文件用于定义客户系统中的安装点与NFS服务器上的远程资源（目录或文件系统）之间的直接映射关系。其语法格式如下：

```
mount-point [mount-options] remote-resource
```

其中，**mount-point**是NFS客户系统中的一个目录，用于安装NFS服务器提供的远程共享资源，指定的目录必须是一个绝对路径名。**mount-options**是**mount**命令或**automountd**守护进程可用的安装选项。**mount-options**字段是选用的，如果存在，第一个安装选项前应加一个减号“-”字符，多个选项之间需加逗号“,”分隔符。**remote-resource**是NFS服务器提供的一个远程共享资源，其表示形式为“**server:pathname**”。

在下面的例子中，**/docs**是客户系统中的一个目录，用于安装NFS服务器**iscas**提供的远程共享资源**/export/docs**。

```
$ cat /etc/auto_direct
.....
/docs -rw      iscas:/export/docs
$
```

直接映射文件用于定义相对独立的远程安装要求，任何不规则的安装点或远程共享资源均可采用直接映射文件的配置方式。

直接映射关系也可以采用**/etc/vfstab**文件实现。例如，为了在系统启动过程中自动安装上述直接映射关系的远程资源，可以在**vfstab**文件中增加下列安装项：

```
iscas:/export/docs - /docs nfs - yes rw
```

14.4.3 间接映射文件

间接映射文件用于定义主映射文件给出的同一安装点下的子目录（相对路径），以及从NFS服务器上安装的远程目录或文件系统。其语法格式如下：

```
mount-point [mount-options] remote-resource
```

其中，**mount-point**是NFS客户系统上的一个相对目录。**mount-options**是**mount**命令或**automountd**守护进程可用的安装选项。同样，**mount-options**字段是选用的，如果存在，第一个安装选项前应加一个减号“-”，多个选项之间需加逗号“,”分隔符。**remote-resource**是NFS服务器提供的一个远程共享资源，其表示形式为“**server:pathname**”。

在下面的例子中，**cathy**、**gqxing**和**hwang**是三个子目录，组合**auto_master**文件定义的共同根目录**/home**，可以构成三个完整的安装点，分别用于安装NFS服务器**iscas**中的三个用户主目录：

```
$ cat /etc/auto_home
.....
cathy  iscas:/home/cathy
gqxing iscas:/home/gqxing
hwang  iscas:/home/hwang
$
```


在间接映射文件中，`mount-point`与`remote-resource`字段中间可以使用星号“*”和“&”通配符。其中星号“*”表示所有可能的目录，“&”表示使用相应的目录替换“&”字符位置。据此，我们可以利用“&”通配符，把`/etc/auto_home`间接映射文件改写如下：

```
$ cat /etc/auto_home
.....
cathy   iscas:/home/&
gqxing  iscas:/home/&
hwang   iscas:/home/&
$
```

在上述间接映射文件中，第一个安装点的相对路径名是`cathy`，因而需要把“&”解释为`cathy`，这意味着远程资源为`iscas:/home/cathy`。同样，当安装点（相对路径名）为`gqxing`和`hwang`时，也需要把“&”分别解释为`gqxing`和`hwang`，表示远程资源为`iscas:/home/gqxing`和`iscas:/home/hwang`。

如果配合使用星号“*”与“&”通配符，还可以进一步简化，最终可以把`/etc/auto_home`间接映射文件改写如下：

```
$ cat /etc/auto_home
.....
*       iscas:/home/&
$
```

在上述间接映射文件中，第一个字段中的星号“*”表示主映射文件指定目录（`/home`）中所有可能的子目录。如果NFS客户系统引用的是`/home/cathy`，则“&”应解释为`cathy`，因而需要安装的远程资源就是`iscas:/home/cathy`。如果NFS客户系统访问的是`/home/gqxing`，需要安装的远程资源就是`iscas:/home/gqxing`，如此类推。

在完成上述配置，重新启动NFS服务器和客户系统的相应守护进程之后，`automountd`守护进程将会根据用户对远程资源的访问请求，自动安装或卸载远程目录或文件系统。例如，如果NFS客户系统中的用户想要查阅`/home/cathy/docs/readme.doc`文件，`automountd`守护进程将会检查其中引用的目录是否涉及到`/etc/auto_master`文件中列举的安装点，以及`/home/cathy`目录是否由间接映射文件`auto_home`控制。如果确乎如此，`automountd`守护进程将会拦截用户的访问请求，检查`/home/cathy`目录是否已经安装。如果尚未安装，则自动安装。至此，用户即可访问指定的文件了。

注意：如果修改了主映射文件，需要重新启动`automountd`守护进程，才能使修改后的配置立即生效。

第15章 系统启动与关机

UNIX系统的启动是一个复杂的内部引导过程。本章将从磁盘分区开始，讨论系统的引导与生成过程，最后介绍各种系统关机方法。掌握系统启动过程的工作原理，不仅有助于理解如何修复系统启动过程中出现的问题，也有助于了解从何处入手，定制应用程序的启动脚本。最后本章还给出一个应用实例。

UNIX系统的启动涉及到系统的磁盘分区、加电引导、运行初始引导程序及UNIX系统引导程序等一系列复杂的引导与处理过程。在加电自检之后，计算机开始加载并执行系统硬盘第0号扇区中的初始引导程序，根据0号扇区的定义，确定活动的磁盘分区，从而确定引导哪一个操作系统，然后加载并启动操作系统提供的引导程序，检测系统的硬件配置，设置系统内核的硬件运行环境，加载操作系统内核，执行系统初始化，加载软件模块，组织系统的内部数据结构，运行系统启动脚本，建立多用户运行环境，直至最终完成系统的启动，其间经过一系列复杂的处理过程。尤其是操作系统的初期引导过程，因涉及到具体的硬件系统，不同UNIX系统的厂家也有自己的具体考虑，因而存在较大的差别。

本章以UNIX System V在Intel x86硬件系统上的实现为例，分两个阶段，即系统初始化阶段和用户初始化阶段，说明系统的整个启动过程。

在系统的初始引导过程中，涉及到具体硬件系统的初始化，而且，UNIX内核模块的初始化过程主要是在核心模式下执行的，故称做系统初始化阶段。在此阶段，UNIX系统的初始引导程序开始设置UNIX内核的硬件运行环境，构造进程0，加载并启动UNIX内核，执行各种软件模块的初始化，直至生成进程1，即/sbin/init进程。

init进程主要依据/etc/inittab文件，调度运行各种系统守护进程，直至完成UNIX系统的生成。这一阶段的初始化过程主要是在用户模式下执行的，因此通常把这一过程称做用户初始化阶段。

15.1 磁盘分区与初始引导

15.1.1 磁盘分区

在Intel x86体系结构的计算机中，传统UNIX系统的磁盘划分如图15-1所示。其中，位于0号扇区的512字节数据块为主引导记录（Master Boot Record，MBR）。从1号扇区开始是操作系统分区。

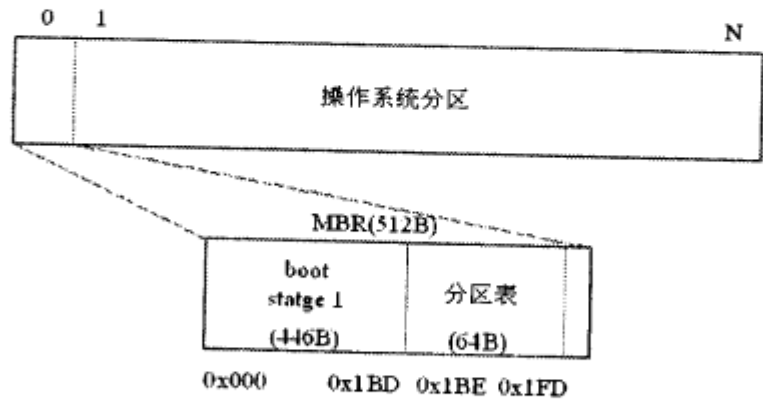


图15-1 系统磁盘布局

512个字节的主引导记录可以分为三部分内容，其中前446个字节为计算机系统的初始引导程序，也称第一阶段引导程序。中间的64个字节为磁盘分区表，用于描述4个操作系统的分区信息，包括分区引导标志（说明是否可以引导）、每个操作系统分区的起始位

置与容量，以及分区的系统类型等。由此可见，在Intel x86系列的系统中，一个计算机可以同时支持4个不同的操作系统。最后一部分只占2个字节，用于存储主引导记录扇区的标志代码，其中的0xAA55表示当前的0号扇区即主引导记录，如图15-2。

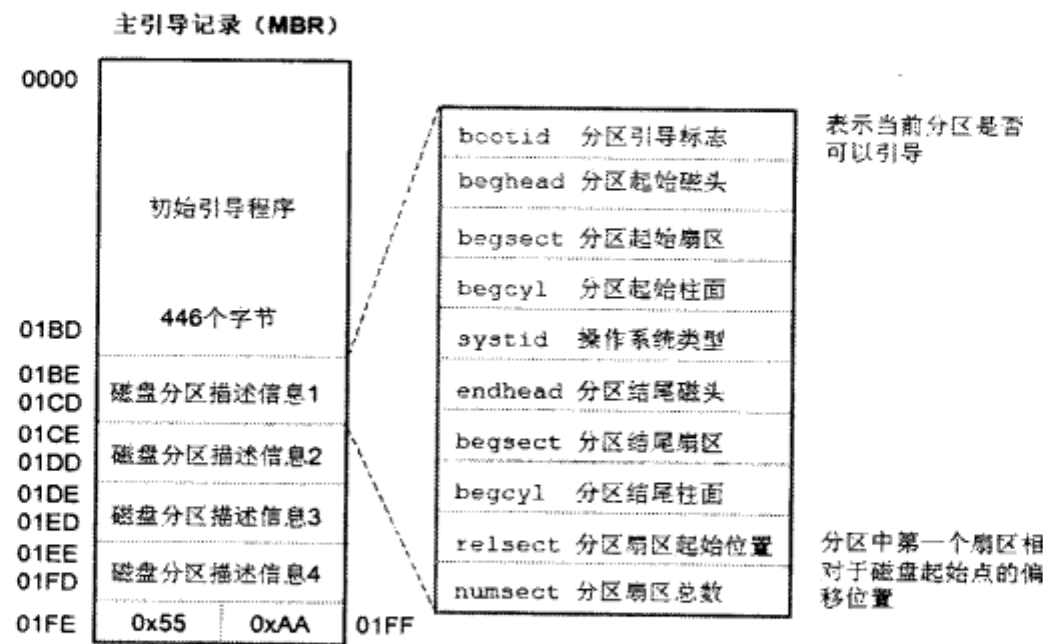


图15-2 系统硬盘主引导块

在磁盘分区表中，如果分区引导标志为0x0，说明相应的分区是非活动分区；如果引导标志为0x80，说明相应的磁盘分区是可引导的。至于究竟启动哪一个磁盘分区中的操作系统，取决于分区表中的分区引导标志，或用户在初始启动过程中做出的选择。操作系统类型字段表示磁盘分区的系统类型，如0x07表示Windows NTFS分区，0x08为AIX引导分区，0x09为AIX数据分区，0x0b为Win95 FAT32分区，0x82为Linux swap分区（或Solaris分区），0x83为Linux分区，0xbe为x86 Solaris引导分区，以及0xbf为Solaris UNIX分区等。有关主引导记录的详细定义，参见/usr/include/sys/fdisk.h文件中的定义。

为了叙述简便起见，我们假定计算机中仅安装了一个UNIX System V操作系统，即除了零号扇区之外，其他所有存储空间均为UNIX系统分区，其典型的磁盘空间布局结构与引导序列如图15-3所示。

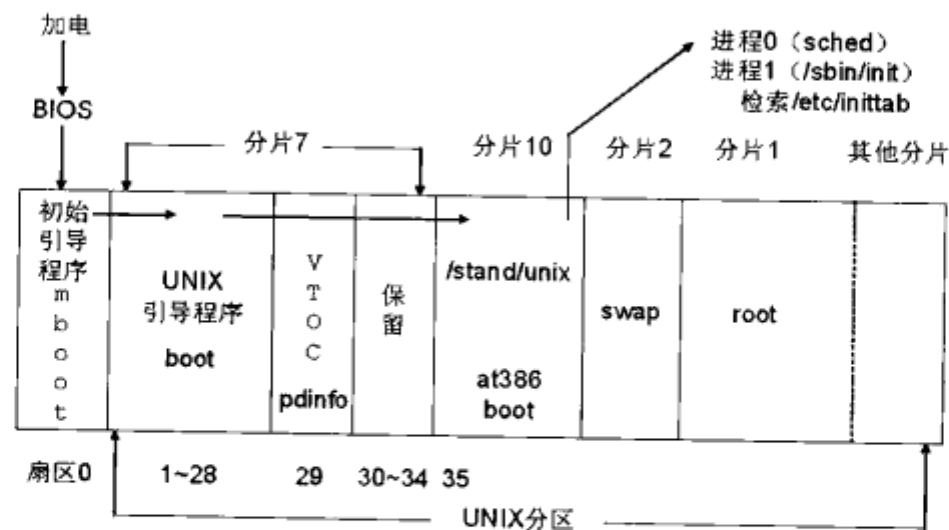


图15-3 UNIX系统磁盘分区布局结构与引导序列

从图15-3可知，第1至第28号扇区为UNIX系统引导程序区，其中包含一个引导程序，俗称boot程序。第29号扇区由两部分内容组成，一是VTOC，其中包含UNIX系统的磁盘分片信息；二是pdinfo，其中包含磁盘参数信息。图15-4和图15-5分别是VTOC与pdinfo的简单示意图。

v_sanity	VTOC完整性标志
v_version	布局版本
v_volume[8]	磁盘卷名字
v_nparts	UNIX分片数量
v_part[V_NUMPAR]	UNIX分片1
v_part[V_NUMPAR]	UNIX分片2
.....	
v_part[V_NUMPAR]	UNIX分片8
timestamp[V_NUMPAR]	分片时戳

p_tag	分片标识字段
p_flag	访问权限标志
p_start	分片起始扇区号
p_size	分片数据块容量

图15-4 VTOC与UNIX系统的磁盘分片信息

driveid	设备类型标识
sanity	设备信息完整性标志
version	设备版本号
serial[12]	设备序列号
cyls	磁盘的柱面总数
tracks	每个柱面的磁道数
sectors	每个磁道的扇区数
bytes	每个扇区的字节数
.....	

图15-5 pdinfo信息

从35号扇区开始是UNIX系统的/stand分片，也即UNIX系统的bfs文件系统。最初，UNIX系统中并没有分片（slice）的概念，一直把现在所谓的分片称做分区（partition）。从移植到Intel x86体系结构的硬件系统开始，为区别于fdisk的操作系统分区，才把UNIX系统的磁盘分区改称磁盘分片（或简称分片）。在后面的讨论过程中，在不致引起混淆的情况下，我们仍把UNIX系统中的分片称做分区。

需要说明的是，在每个UNIX系统的具体实现中，上述定义并不完全一样，读者可以参考自己系统中的数据结构定义文件（如/usr/include/sys/vtoc.h）。例如，Solaris系统的磁盘分区不仅与上述划分不同，即使在SPARC和Intel x86两个系列的系统中也不尽相同。图15-6给出了Solaris SPARC系统的磁盘布局与引导序列示意图。

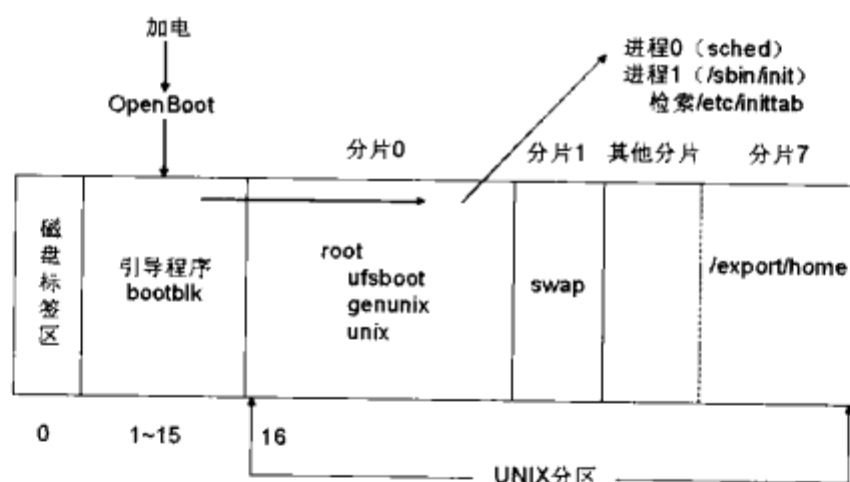


图15-6 Solaris SPARC系统磁盘布局与引导序列

由图15-6可见，零号扇区为磁盘标签区，其中包括VTOC磁盘分片与磁盘参数信息。参见/usr/include/sys/dklabel.h文件。第1号至第15号扇区为UNIX系统引导程序区，其中包含主引导程序bootblk。辅助引导程序ufsboot位于“/”文件系统中。UNIX内核分为两部分：一为与平台无关的genunix，另一个则是涉及具体平台的unix。

Solaris Intel x86系列的系统磁盘布局与引导序列如图15-7所示。

由图15-7可知，零号扇区的pboot相当于UNIX System V Intel x86系列计算机中的主引导块mboot。而第1号~第2号扇区为磁盘标签区，其中包括VTOC磁盘分片与磁盘参数信息。参见/usr/include/sys/dklabel.h文件。第3号至第15号扇区为UNIX系统引导程序区，其中包含主引导程序bootblk。辅助引导程序boot.bin、UNIX内核genunix与unix位于“/”文件系统中。

目前，UNIX系统也开始使用GRUB作为引导程序，如最新的Solaris 10 UNIX系统。在此情况下，其磁盘分区布局如图15-8所示。其中，位于第0号扇区的512字节数据块为主引导记录MBR。GRUB使用这个扇区存储第一阶段引导程序（stage1）。紧随第0号磁道的1号~62号扇

区（31KB）为GRUB引导程序区，可用于存储第1.5阶段引导程序，如ufs_stage1_5。从63号扇区开始是操作系统分区。

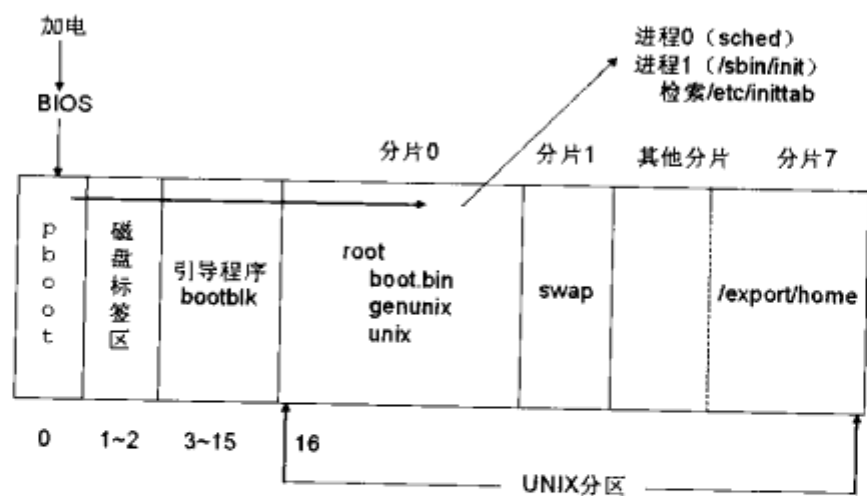


图15-7 Solaris Intel x68系统磁盘布局与引导序列

如果系统中仅仅安装了一个Solaris UNIX操作系统，即除了零号磁道之外，其他所有的存储空间均为UNIX系统分区，其典型的磁盘空间布局结构如图15-9所示（假定在安装Solaris UNIX系统时采用的是默认磁盘分区布局，即没有额外划分其他文件系统分区）。

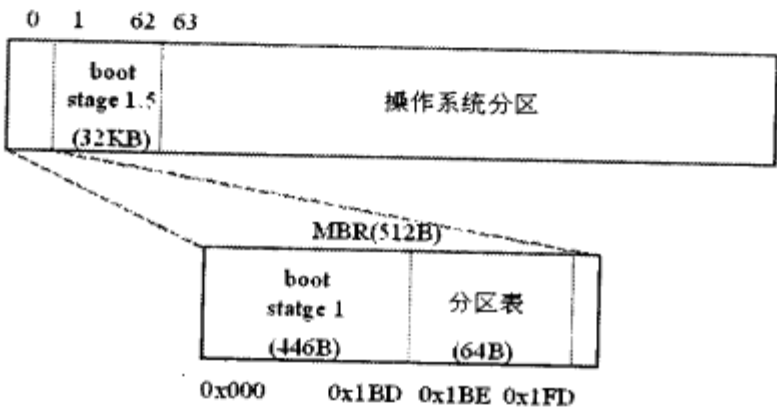


图15-8 系统磁盘布局

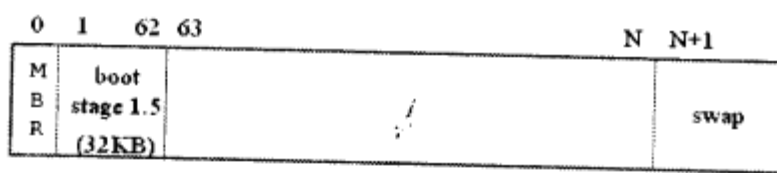


图15-9 UNIX系统磁盘分区布局结构（1）

如果计算机中安装了多个系统，如安装Windows之后又安装了Solaris系统，其系统磁盘分区布局的典型结构如图15-10所示（假定在安装Solaris UNIX系统时采用的是默认磁盘分区布局，即没有额外划分其他文件系统分区）。

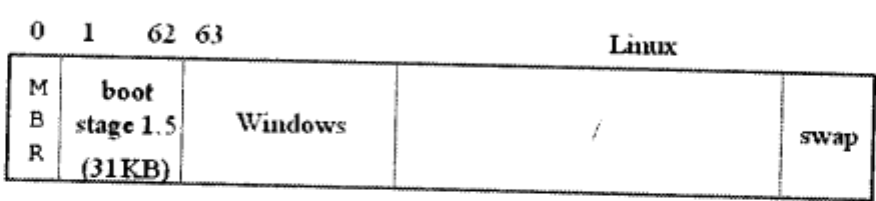


图15-10 UNIX系统磁盘分区布局结构（2）

15.1.2 初始引导过程

在计算机加电或重新启动时，系统将会自动跳到ROM BIOS中的加电引导程序入口点开始运行。加电引导程序的基本功能是检查系统的硬件配置，运行硬件诊断程序，执行加电自检。然后找出系统磁盘，把系统盘0号扇区中的初始引导程序读入内存，执行初始的系统引导。

初始引导程序的主要功能是确定活动的分区（当系统中装有多多个操作系统时），找出活动分区中由操作系统提供的系统引导程序（即boot程序），加载并运行相应的系统引导程序。

对于UNIX操作系统而言，boot引导程序的主要目的一是建立UNIX内核的运行环境，二是

根据VTOC表中的信息，加载UNIX内核（在System V UNIX系统中，UNIX内核是bfs文件系统中的/stand/unix文件），然后把控制权交给UNIX内核的main()主程序，执行各种模块初始化程序。

在UNIX内核真正开始运行之前，boot程序还需要调用mlsetup()，执行底层的硬件初始化设置。其中包括禁止硬件中断，设置和初始化CPU的虚拟地址转换机制，并使CPU处于保护模式，组织和初始化UNIX内核的虚拟地址空间，初始化部分全局变量、数据结构以及高精度的系统时钟。

调用dispin()函数，针对系统配置的每一个进程调度类别，调用其特定的函数，对进程调度类别的控制机制进行初始化，为进程调度队列分配内存空间等。

至此，boot开始组织和初始化第一个进程，即进程0，分配程序控制块pcb、user和proc结构，建立内存页表，建立和初始化进程控制表。在初始化上述结构数据之后，进程0基本上已经就绪，但还有一个非常重要的问题，即进程0此时尚没有必需的代码、数据和栈段，因而也无法运行。

boot程序在结束执行之前，调用UNIX内核的main()主程序，UNIX系统从此开始投入运行。系统引导过程的第二阶段开始。

15.1.3 系统初始化

在系统初始化阶段，UNIX内核开始运行自己的初始化程序。首先，在main()主程序内部（或startup()函数中）执行最后一部分硬件系统的初始化。其中包括SCSI控制器和DMA控制器等外部设备的初始化。期间，UNIX内核将调用各种I/O设备的初始化程序，设置硬件接口，使外部设备处于可工作状态。

之后，UNIX内核开始检查系统的内存配置，根据内存空间的大小，适当地分配系统内存（初始化程序通常会在控制台上显示系统配置的内存，以及经过初始化过程后剩余的可用内存，二者之差即UNIX内核占用的内存），从而确定负责页面调度的系统内核参数设置，如lotsfree、desfree和minfree等。

启动系统硬件时钟，以便clock()系统例程能够处理时钟中断信号。打开先前禁用的硬件中断。初始化utsname数据结构，把操作系统版本号及计算机类型等数据（“uname -a”命令的输出信息）赋予utsname结构。

然后，通过调用每个软件模块的初始化例程，开始执行一系列软件初始化过程，其中包括组织与初始化系统内核维护的数据、变量和表格等。建立基本的内部数据结构，构造系统表格和缓冲区。摘要列举如下：

- I/O系统缓冲区链表；
- 系统打开文件表；
- STREAMS子系统；
- 每个文件系统类型的vfs结构；
- 进程通信系统；
- 目录名字检索缓冲区；
- 核心内存分配器等。

此时，UNIX内核开始安装“/”文件系统，配置交换分区。然后通过newproc()系统调用（内

核版的fork()系统调用), 创建进程1, 也即/sbin/init进程的原型。经过一系列处理步骤之后, 包括分配和初始化用户地址空间, 最终把磁盘文件系统中的/sbin/init可执行文件调入内存, 且赋予较高的优先权开始运行。

接着, UNIX内核利用newproc()系统调用, 创建一个负责内存页面调度的pageout进程, 以及负责把文件I/O的内存页面数据写入磁盘文件系统的fsflush进程, 并以核心模式开始运行。至此, 除了进程0等少数核心进程之外, 系统中只有一个有用的进程, 即init进程是活动的。

最后, UNIX内核再把自己转化为进程0, 通称sched (或swapper) 进程。在其整个系统生命周期中, 进程0一直以核心模式工作, 并与pageout进程一起, 监控系统的空闲内存。pageout负责进程的页面调度, 当系统的空闲内存达到系统可调参数desfree或minfree规定的最低限度时, pageout负责把选定进程的部分页面 (而非整个进程) 转储到磁盘的交换分区中。当pageout无法为用户进程提供必要的空闲内存时, sched将会以进程 (而非页面) 为单位, 把按一定原则选定进程的整个内存映像转储到磁盘交换分区, 直至内存的可用空间大于系统可调参数lotsfree定义的数量。

至此, 系统初始化阶段已告完成, UNIX系统已基本成形。但还远未到完全可用的工作状态, 还有大量系统生成方面的任务需要由init进程完成。init进程需要在用户模式下, 根据/etc/inittab文件的定义, 调度运行其他系统服务, 提供附加的系统功能。

上述工作过程可以简要地概述如下 (如图15-11) :

(1) ROM BIOS——加电自检。一经加电, 计算机中的BIOS即开始检测系统的硬件配置, 执行硬件诊断程序, 然后确定系统引导设备, 如内置磁盘、软盘、CD/DVD、网络设备或USB移动盘等。找出零号扇区的主引导记录, 根据其中的最后两个字节确定是否MBR。一旦找到引导设备, BIOS将会立即加载其零号扇区MBR中的初始引导程序, 把系统引导的控制权交由初始引导程序, 开始执行初始的系统引导。

(2) mboot/pboot——初始引导程序。根据FDISK表确定活动分区, 加载并运行活动分区中的系统引导程序 (1号~28号扇区)。

(3) boot系统引导程序。检查、设置UNIX内核的硬件运行环境, 配置CPU、内存、磁盘及I/O设备等, 加载必要的驱动程序模块, 设置系统时钟, 构造运行环境, 设置进程调度类别, 构造进程0, 分配程序控制块, 建立内存页表和进程控制表。根据VTOC表, 加载并执行UNIX内核。

(4) /stand/unix——UNIX系统内核。初始化部分I/O设备, 检查内存配置, 组织各种数据结构, 设置系统缓冲区, 安装“/”文件系统, 配置磁盘交换区, 创建进程1、pageout和fsflush等进程, 运行/sbin/init程序, 把引导过程的控制权转交init进程, 由init进程完成系统的生成过程。

(5) 进程0 (sched) ——内存监控。与pageout进程一起, 监控系统的空闲内存, 采用内存页面交换与进程交换的方式, 确保用户进程能够公平地获得运行机会。

(6) 进程1 (init) ——系统生成。读取/etc/inittab文件, 确定系统运行级, 根据运行级调度运行/sbin/rcN脚本, 依次启动/etc/rcN.d目录中的Shell脚本 (包括用户定义的服务程序), 启动各种系统服务的守护进程, 设置网络环境, 直至在控制台上显示一个注册界面。

Solaris SPARC系统的引导过程如图15-12所示。

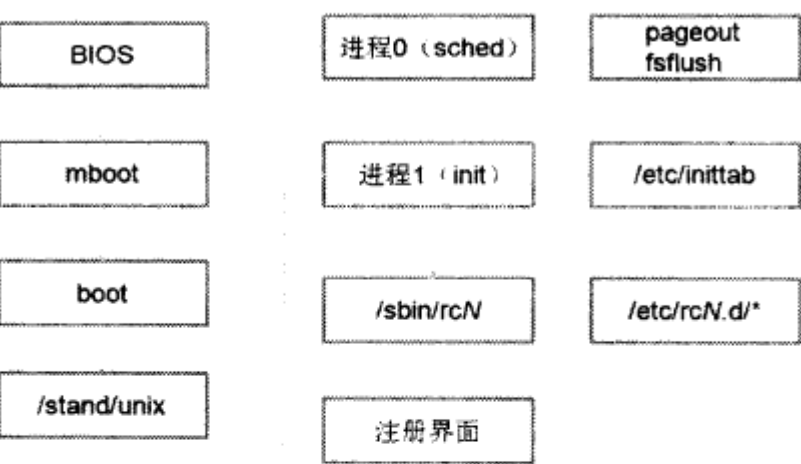


图15-11 UNIX系统引导过程示意图

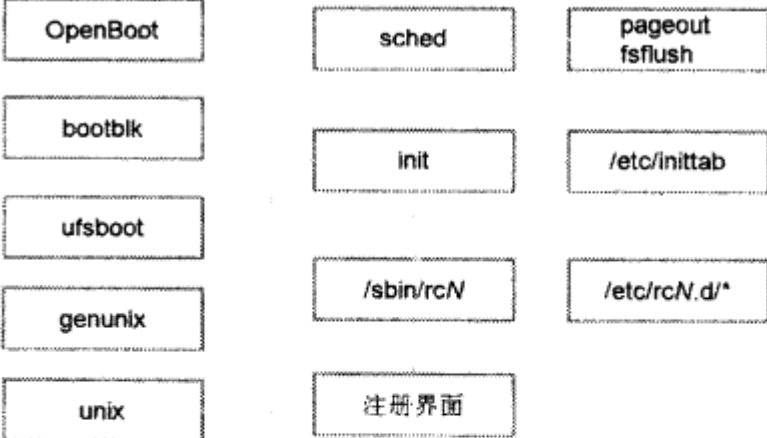


图15-12 Solaris SPARC系统引导过程示意图

Solaris x86系统的引导过程如图15-13所示。

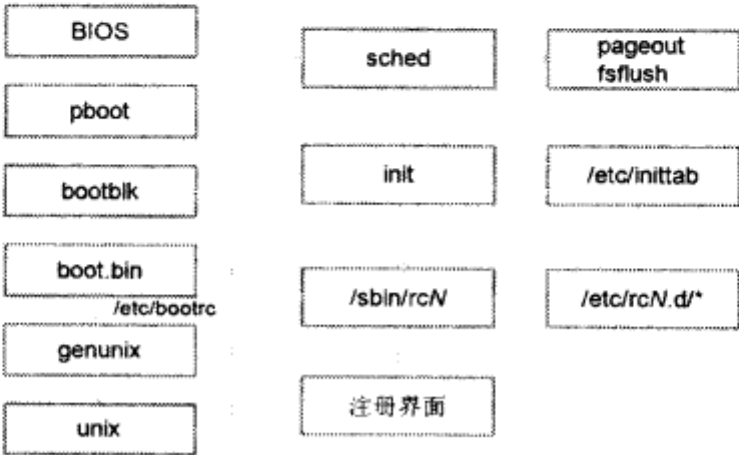


图15-13 Solaris x86系统引导过程示意图

15.2 . init进程与系统生成

系统启动过程的最后一个阶段是系统生成。在此阶段中，init进程将会依据/etc/inittab文件的定义，自动完成系统的后续启动过程。init进程的基本功能是进程调度，以及改变系统的运行级。除了sched、pageout和fsflush等少数几个核心进程，系统中几乎所有的进程都是init进程的直接或间接子进程。

init是UNIX系统中最具特色的一个进程，是一个解决和处理系统启动问题的极好方法与范例。在系统的初始化过程中，需要启动大量的系统服务进程，完成诸如安装文件系统，输出注册提示信息，引导用户注册，以及设置与启动TCP/IP网络等功能。init进程是一个通用的触发机制，能够灵活地启动各种必要的服务进程，由软件厂商和用户共同决定究竟需要启动哪些服务进程。

一经启动，init进程将会读取/etc/inittab文件，检查其中是否含有boot、bootwait、sysinit或initdefault等处理方式的定义项，而与之相应的进程必须在处理inittab文件中其他定义项之前执行，确保在用户注册访问系统之前，能够预先执行任何必要的系统初始化任务，如启动网络功能，以及安装文件系统等。

initdefault用于确定系统的初始运行级，如果initdefault定义项存在，init进程将会根据initdefault的定义，使用其中指定的运行级，确定系统的最终工作状态，即单用户、多用户，还是多用户加网络等工作模式，调度运行目的运行级中需要执行的所有启动脚本，从而启动各种

必要的服务进程，配置系统和网络，使系统处于可用的状态，最终完成系统的整个启动过程。

如果initdefault一项不存在（甚至/etc/inittab文件都不存在），系统将会通过控制台提示用户输入运行级。如果initdefault定义项存在，但未指定运行级，默认的运行级为0123456。此时，init进程将会按照运行级6引导系统。由于运行级6意味着重新启动系统，结果将会使系统陷于连续地关机、启动、再关机、再启动的无限循环怪圈。

为了调度运行/etc/inittab文件中指定的进程，init将会逐行读取其中内容。对于需要调度运行的每一个进程，init将利用fork()系统调用创建一个新的子进程。在提交了inittab文件中指定的所有进程之后，init将会不断地监控或等待进程的终止信号、电源故障信号或其他改变系统运行级的信号。当这三种情况之一出现时，init进程将会重新检查inittab文件。

为了详细讨论init进程的工作情况，首先需要简单地介绍运行级以及/etc/inittab文件。

15.2.1 运行级

运行级（Run Level）实际上是系统定义的一种运行模式或运行状态，故运行级有时也称做运行状态（Run State）。在不同的运行级中，系统能够提供不同的功能或服务。从系统管理的角度来看，运行级实际上是一种软件配置方式。针对每一个运行级，UNIX系统中都定义了一组选定的服务程序，使系统能够满足不同的工作需要。

定义运行级的主要目的是组织和划分系统进程的适用场合，根据不同的工作模式和功能需求确定不同的进程配置。每个运行级都表示系统的一种工作模式，因而具有不同的功能要求，需要运行一组选定的进程。每个系统进程既可属于一个特定的运行级，也能适用于多个运行级。这就需要使用运行级划分系统的工作模式，相应地配置一组选定的系统进程。

UNIX系统可以处于不同的运行状态，但在任何时刻，UNIX系统只能处于8种可能的运行级之一（init进程可以识别0~6等7个主要运行级，及一个内部定义的单用户运行级s或S）。例如，常见的多用户与网络工作模式，或用于维护目的的单用户工作模式等。在任何一个运行级中，只有选定的一组进程才能够运行。

在UNIX系统中，需要由init调度执行的进程均可在/etc/inittab文件中定义。对于每一个运行级，/etc/inittab文件中定义了在进入指定的运行级时，系统应执行哪些命令或进程。在系统的运行过程中，超级用户可以利用“init n”形式的命令，使系统从一个运行状态转入另一个运行状态。

在UNIX的标准中，运行级并没有一个严格的规定，只是作为惯例，大家都遵守这样的约定，如果需要，完全可以赋予它们新的意义。表15-1给出的是UNIX System V中的典型运行级定义。

表15-1 运行级定义

运行级	简单说明
0	运行级0用于停止UNIX操作系统，执行系统关机。进入此运行级后，系统将会终止所有的进程，卸下所有的文件系统，完全停止整个系统的运行，然后即可断开电源（在具有软关机功能的系统中，系统会自动断电）
1	单用户命令行界面运行模式。顾名思义，单用户运行模式仅允许一个用户，即超级用户root注册进入系统。安装（或保持安装）多用户工作模式需要的所有文件系统。此时只有一部分最基本的内核进程在运行。当系统直接进入运行级1时，只有控制台可以工作，其他多用户

(续表)

运行级	简单说明
	模式的服务均不可用，管理员只能通过控制台访问系统。但请注意，当从多用户模式进入运行级1时，系统并不会终止所有的用户进程。在需要执行某些系统维护方面的功能时，通常要求进入单用户工作方式。如安装软件包，增加和测试新设备等
2	运行级2为多用户工作方式。早期，在加电引导、初始化与生成过程完成之后，根据/etc/inittab文件的要求，系统将会自动进入此运行状态。此时，系统将会根据/etc/vfstab文件的要求，安装所有的文件系统，运行/etc/rc2.d目录中的所有后台服务进程，为每一个终端用户调度有关的ttymon和login等进程，使用户能够注册到系统中
3	运行级3为多用户加网络工作方式。现在，在系统启动过程完成之后，根据/etc/inittab文件的要求，系统将会自动进入这一运行状态。此时，根据/etc/fstab文件的要求，系统将会安装所有的文件系统，运行/etc/rc3.d目录中的所有后台服务进程，包括网络服务，如广播本地资源，安装远程文件系统等，使UNIX系统既可以作为多用户的分时系统，又可作为网络服务器或客户系统。最后，调度运行login等进程，提供命令行注册界面，使用户能够注册访问系统
4	暂未定义，可供用户定义其他多用户环境配置
5	固件工作方式。用于执行rc5脚本，实际上还执行rc0.d目录中的Shell脚本，关闭所有的系统服务，使系统停止运行。然后进入固件诊断方式，或断电关机
6	运行级6用于重新启动系统。这个运行级包括两个工作过程：首先停止UNIX系统，其处理步骤与运行级0完全相同；然后重新引导UNIX系统，使系统进入/etc/inittab文件定义的默认运行级，其效果如同加电启动系统。这种先关机后启动的做法相当于常规意义的热启动。在进行系统配置、调整系统参数或生成新的UNIX系统时，通常需要用到此运行级
s,S	系统内部定义的单用户工作方式。这也是唯一不要求/etc/inittab文件必须存在的运行级。在系统引导过程中，如果/etc/inittab文件不存在，系统将会自动进入这一运行级。进入S（或s）运行级时，除“/”文件系统之外，系统不会安装其他任何文件系统，且只有最基本的内核进程在运行。当系统从其他较高的运行级下转到S（或s）运行级时，已经安装的所有文件系统保持安装状态不变，只能在原运行级下运行的进程（包括用户进程）将会停止运行，但并非由init直接启动的进程仍将保持其运行状态。虽然运行级S（或s）也为单用户工作模式，但与运行级1不同的是，S（或s）是系统内部定义的，与/etc/inittab文件的存在与否无关。一旦进入此运行级，系统管理员只能通过控制台访问系统。注意，除了initdefault一行尚可使用之外，一般不要在/etc/inittab文件中使用这个运行级，总之，最好不要在/etc/inittab文件中乱用S（或s）运行级
q,Q	其主要目的是让init进程重新检查/etc/inittab文件
a,b,c	仅仅处理/etc/inittab文件中运行级字段为a、b或c的相应进程。这三个运行级主要用于定义需要不定期运行的用户程序或进程。进程一经启动，除非进入单用户或关机，相应的进程可在其他任何运行级中运行，不受运行级改变的影响。使用运行级a、b或c作为参数运行init命令时，不会引起当前运行级的变化，故称其为伪运行级

如果运行级字段设置为a、b或c，仅当使用a、b或c作为参数运行init命令时，才会开始运行相应的进程（不管系统当前处于哪一个运行级）。而且，即使改变系统的运行级（运行级0、1和6除外），也不影响这些进程的继续运行。

如果需要，超级用户可以运行init命令，改变系统的运行级。在改变运行级之前，有时也许需要了解当前的运行级。为此，可以使用下列命令：

```
# who -r
.      run-level 3  Apr  4 08:33      3      0  S
#
```


在上述命令的输出信息中，“run-level 3”字段表示当前的运行级为3。最后一个字段S表示先前的运行级为S，说明从哪一个运行级转入当前的运行级。

当改变系统的运行级时，通常需要终止在新的运行级中未定义的活动进程。终止活动的进程分为两个步骤：**init**首先向进程发送**SIGTERM**警告信号，然后给予5秒钟的等待时间，使进程能够做适当的善后处理（编写程序时可以捕捉这一信号，根据需要可以自行终止执行或者置之不理）。在超过等待时间之后，第二次发送**SIGKILL**信号，强行终止进程。这个信号不能捕捉，也不容忽略。

当**init**进程收到一个信号，通知某个进程已经终止运行时，**init**将会把这一事件及其终止的原因记录在/var/adm/utmpx和/var/adm/wtmpx文件中。注意，/var/adm/utmpx文件仅记录当前活动的进程，/var/adm/wtmpx文件会记录进程调度的历史信息。

15.2.2 /etc/inittab文件

/etc/inittab是一个非常重要的系统管理文件。**inittab**文件是一个ASCII文本文件，每行最长不能超过512个字符，至于文件的行数，即文件的大小并无限制。该文件每一行由四个字段组成，中间以冒号“:”作为分隔符。其语法格式定义如下：

```
id:run-level:action:command
```

表15-2给出了每个字段的简单说明。

表15-2 /etc/inittab文件

字段	简单说明
id	id是由1至4个字符组成的字符串，用以唯一地标识一个控制与命令信息
run-level	运行级字段用于定义在哪一个运行级中执行命令字段指定的进程。运行级字段可由表示运行级的字符串组成。当系统进入此字段定义的运行级时，需要执行命令字段中指定的进程。表示运行级的字符串可以是单个数字，也可以是由0~6组合的任意一组数字，例如，2345说明在2、3、4和5等四个运行级中都要执行相应的进程。运行级字段也可以为空，意味着其运行级字符串为0123456，表明在任何运行级中都需要执行命令字段指定的进程。此外，运行级字段还可以使用a、b或c三个伪运行级。当超级用户以a、b或c作为运行级参数运行 init 命令时，将会在原运行级中执行相应的进程，而不改变系统的当前运行级
action	指定处理方式。处理方式字段中的关键字告诉 init 进程怎样处理命令字段中指定的进程。 init 进程认可的处理方式关键字详见表15-3
command	指定执行的命令。 init 进程根据运行级和处理方式两个字段的的要求，通过 fork() 或 exec() 系统调用，以“ sh -c 'exec command' ”的形式，运行指定的命令。由此可见，用户可以采用任何合法的Shell语法格式指定命令字段

15.2.3 处理方式

处理方式（**action**）字段的主要目的是告诉**init**，当系统处于指定的运行级时应当怎样处理相应命令字段中的进程。有效的处理方式关键字如表15-3所示。

当系统处于一个确定的运行级时，**init**将会根据处理方式字段的要求，执行相应的进程。如果进程只能在一个运行级中运行，在改变运行级之前，需要终止相应进程的运行。如果某个进程可以在多个运行级中运行，在改变运行级时，相应的进程会继续保持运行状态不变。

表15-3 处理方式字段可用的关键字

处理方式	简单说明
boot	调度运行命令字段中指定的进程，而且不必等待进程运行结束，即可继续处理/etc/inittab文件中的其他项。进程执行结束后也无需再次启动。通常，仅在系统引导过程完成之后，init进程第一次开始处理inittab文件时才调度运行此类的进程
bootwait	在引导过程完成之后，init进程从单用户转入多用户，第一次开始处理inittab文件时，启动相应的进程，且需等待进程运行结束。在进程终止之后方可继续处理inittab文件中的其他项。此类进程终止后也不需要重新启动
initdefault	指定系统启动过程最终进入的运行级或工作模式，也即在系统引导过程完成后，系统最终应处的运行状态。如果运行级字段指定的运行级不止一个时，取其中的最大数值（通常也是最后一个）作为最终运行状态。如果inittab文件中的initdefault未指定运行级，即其运行级字段为空，则取运行级字段默认值0123456的最大数值6作为系统的最终运行级，这意味着系统将会连续不断地重新启动。如果inittab文件中不存在initdefault一行，在启动过程中，系统将会在控制台上询问用户进入哪一个运行级。另外，不管inittab文件中的initdefault是否指定了默认运行级，如果在系统引导期间无法创建utmpx文件，系统最终会进入运行级s。如果/var是一个独立的文件系统，且由于文件系统损坏导致无法正常安装时，也会出现这一状况。注意，initdefault一行没有相应的进程字段
off	表示不执行相应的进程。如果与此有关的进程当前正在运行，则按前述的步骤终止进程的执行。off处理方式仅用于暂时禁止某一进程的执行，在inittab文件中保留一个定义
once	启动相应的进程，无需等待进程结束，终止后也不需要再次启动
ondemand	等同于respawn，主要用于伪运行级a、b和c
powerfail powerwait	在相应的运行级中，一旦收到电源故障信号（SIGPWR），init需要立即启动相关的进程同上，但需等待进程结束，在进程终止之后方可继续处理inittab文件
respawn	启动命令字段中的相应进程。如果进程没有运行，需要立即启动，使之开始运行。然后，继续逐行处理inittab文件。一旦终止执行，init还要负责重新启动进程。如果相应的进程当前正在运行，init不必做任何事情——跳过当前行，继续处理inittab文件中的后续内容
sysinit	执行相应的进程，并等待进程的完成。在进程结束运行之后，再继续处理inittab文件中的其他项。与之相关的进程应在init确定运行级，访问系统控制台（也即在系统控制台出现“Console Login:”注册提示信息）之前启动并运行，然后等待进程的终止。此类进程通常仅用于系统生成早期阶段的各种软硬件初始化任务
wait	执行相应的进程，并等待进程的完成。在进程结束运行之后，再继续处理inittab文件中的其他项

在系统运行期间，如果收到电源故障信号SIGPWR，init将会考察/etc/inittab文件的处理方式字段，检查其中是否存在powerfail或powerwait定义项。如果存在，且当前运行级允许执行，在采取其他任何处理动作之前，init将会调度运行相关的命令。采用这种处理方式，init进程能够在系统的关机过程中执行各种文件清理与日志处理工作。

15.2.4 /etc/inittab文件举例

从理论上讲，我们可以在inittab文件中定义任何需要在系统生成过程中自动调度运行的进程，但根据实际的应用，inttab文件中定义的进程，大体上可以分为四类：一是在系统完全就绪之前，换言之就是在进入指定的运行级之前尚需执行的最后一部分初始化任务（其处理方式字段通常为boot、bootwait或sysinit）；二是系统启动后最终应处于哪一个运行级，也即系统的目

的工作模式；三是在进入指定的运行级之后需要启动的各种系统服务，也即执行/etc/rcN.d目录中的Shell脚本，其中N是一个表示实际运行级的数字；四是启动必要的终端服务进程或应用程序。

在不同的UNIX系统中，/etc/inittab文件的内容并不相同（而且，有的UNIX系统现在开始淡化inittab文件的作用）。下面的代码取自Solaris 10系统中inittab文件的前两行：

```
ap::sysinit:/sbin/autopush -f /etc/iu.ap
sp::sysinit:/sbin/soconfig -f /etc/sock2path
```

其中的autopush命令将根据/etc/iu.ap文件的定义，对于每一个驱动程序，配置自动加载的STREAM模块。soconfig按照/etc/sock2path文件的定义，配置套接字使用的传输层接口，建立socket()函数使用的协议系列（如IPv4和IPv6）、类型（如TCP和UDP）和其他协议参数与传输层接口（如/dev/tcp）之间的映射关系等。

在完成上述处理任务后，init开始检查inittab文件中initdefault一行的运行级字段。在下面的例子中，系统默认的运行级为3，说明系统最终应进入多用户加网络工作模式。

```
is:3:initdefault:
```

第三部分说明，当确定了系统的最终运行级之后，init进程将会根据运行级调度运行相应的/sbin/rcN脚本，通过执行/etc/rcN脚本，完成最后的系统生成过程。注意，在正常的系统启动过程中，下列第一行通常不会处理。但如果在系统运行期间（包括系统生成过程中）出现电源故障，则需要执行相应的shutdown程序。

```
p3:s1234:powerfail:/usr/sbin/shutdown -y -i5 -g0 >/dev/msglog 2<>/dev/msglog
sS:s:wait:/sbin/rcS                2>&1 </dev/console
s0:0:wait:/sbin/rc0                2>&1 </dev/console
s1:1:respawn:/sbin/rc1             2>&1 </dev/console
s2:23:wait:/sbin/rc2               2>&1 </dev/console
s3:3:wait:/sbin/rc3                2>&1 </dev/console
s5:5:wait:/sbin/rc5                2>&1 </dev/console
s6:6:wait:/sbin/rc6                2>&1 </dev/console
```

例如，当系统启动后进入运行级3时，即进入多用户加网络工作方式时，init进程将会依次调用/sbin/rc2（其运行级字段为23）和/sbin/rc3脚本。而rc2和rc3分别用于调度运行/etc/rc2.d和/etc/rc3.d目录中以“Sxx”为起始文件名的所有启动脚本，并按序依次启动相应的系统守护进程，从而完成最后的系统生成过程。其中，rc2脚本的内容如下：

```
# cat /sbin/rc2
.....
case $action in
    start)
        for f in /etc/rc2.d/S*; do
            case $f in
                *.sh)    . $f ;;
                *)       /sbin/sh $f start ;;
            esac
        done
        ;;
    stop)
```

```

        for f in /etc/rc2.d/K*; do
            case $f in
                *.sh)      . $f ;;
                *)          /sbin/sh $f stop ;;
            esac
        done
        ;;
    *)
        echo "Invalid argument"
        exit 1
    esac
    exit 0
#

```

/etc/rc2.d和/etc/rc3.d目录中的脚本文件说明了在进入多用户或网络运行级之后都会启动哪些系统服务或应用程序。注意，在不同厂家的UNIX系统中，甚至在同一厂家不同版本的UNIX中，rcN.d目录中的文件并不完全一样。这一点实际上并不重要，我们关心的只是这样一种机制，以及怎样利用这种机制，了解系统的启动过程，以及把应用程序的启动脚本加到系统的生成过程中，使应用程序能够随着系统的启动自动运行。

在上述启动脚本中，大部分均采用case分支控制语句，按照start和stop两种情况，确定怎样启动或停止相应的系统守护进程。下面是一个取自corn脚本文件的例子。

```

# cat /etc/init.d/cron
.....
case "$1" in
    'start')
        if [ -p /etc/cron.d/FIFO ]; then
            if /usr/bin/pgrep -x -u 0 -P 1 cron >/dev/null 2>&1; then
                echo "$0: cron is already running"
                exit 0
            fi
        fi
        if [ -x /usr/sbin/cron ]; then
            /usr/bin/rm -f /etc/cron.d/FIFO
            /usr/sbin/cron &
        fi
        ;;
    'stop')
        /usr/bin/pkill -x -u 0 -P 1 cron
        ;;
    *)
        echo "Usage: $0 { start | stop }"
        exit 1
        ;;
esac
exit 0
#

```

系统生成过程的第四部分主要是终端设置。

```
sc:234:respawn:/usr/lib/saf/sac -t 300
co:234:respawn:/usr/lib/saf/ttymon -g -h -p "Console login: " -T ansi -d /dev/
console -l console -m ldterm,ttcompat
```

上述第一行告诉init，当系统处于运行级2、3或4时，应调度运行终端服务监控程序sac，按照终端服务配置文件的定义，启动必要的通信服务程序ttymon进程，以使用户能够通过终端进入系统。

第二行告诉init，当系统处于运行级2、3或4时，应直接启动终端通信服务程序ttymon进程，在系统控制台上输出“Console login:”注册提示信息，以使用户能够通过控制台终端访问系统。

至此，系统的生成过程已全部完成。

15.2.5 启动用户定义的应用程序

除了创建启动脚本，利用init进程和/etc/rcN.d目录，实现应用程序的自动启动之外，系统管理员也可以把需要执行的应用程序加到/etc/inittab文件中，由init进程负责调度执行。但是，把命令加到inittab中并不等于init会立即调度执行新增加的进程。

实际上，除了系统生成阶段，只有在init调度的进程终止执行，收到电源故障信号或其他改变系统运行级的信号之后，init才会重新考察/etc/inittab文件。

为了解决这个问题，可以使用“init Q”或“init q”命令，唤醒init进程，使之立即重新考察/etc/inittab文件。

15.3 用户注册过程

15.3.1 用户注册的处理过程

UNIX System V Release 4.0之前，在完成系统的启动，进入多用户工作模式之后，UNIX系统采用“init-getty-login-sh”的调用过程，使用户能够注册、访问UNIX系统。

之后，新版UNIX系统的用户注册过程改为“init-sac/ttymon-login-sh”调用过程。init进程根据/etc/inittab文件的定义，通过sac或直接调度运行ttymon进程，设置终端通信接口，输出注册提示。当输入用户名之后，ttymon将会利用exec()系统调用，调用login进程，由login进程引导用户输入密码。

login进程根据/etc/passwd和/etc/shadow文件，验证用户输入的用户名和密码正确与否，然后调用指定的命令解释程序。sh进程开始运行之后，将会执行/etc/profile和\$HOME/.profile等初始化文件，设置用户的工作环境。

最后，sh输出命令提示符，开始与用户会话。其内部处理步骤简述如下（注意，这里不涉及图形界面注册的内部处理过程）。

- (1) 根据inittab文件的定义，init进程直接或间接（通过sac）调度运行ttymon终端服务程序。
- (2) ttymon的主要用途是设置与监控终端通信接口，根据/etc/ttydefs文件的定义，设置通信参数，在终端屏幕上输出“Console login:”或“login:”等注册提示信息，然后等待用户输入注册的用户名。一旦读取用户名，立即建立utmp记录，接着使用用户名作为参数调用login

进程。

(3) **login**进程主要负责处理用户的系统注册。开始运行之后，**login**进程首先在终端屏幕上输出“**password:**”提示信息，等待用户输入密码。在读取密码之后，根据**/etc/passwd**和**/etc/shadow**文件，检验用户提供的注册用户名和密码是否正确。如果用户名和密码通过了验证，**login**进程将会按照**/etc/passwd**文件的定义，确定用户的初始工作目录及注册Shell。然后调用指定的命令解释程序。

(4) **sh**进程首先执行**/etc/profile**初始化文件，设置系统范围内的运行环境，如**PATH**和**LOGNAME**等变量。如果**/etc/motd**文件非空，输出文件中的内容。然后进入用户主目录，执行其中的用户初始化文件**.profile**，进一步定制用户自己的运行环境。当一切准备工作就绪之后，取决于用户的身份，在终端屏幕上输出UNIX系统的命令提示符“**\$**”或“**#**”，Shell会话过程开始。

在完成上述四个步骤之后，用户即可进入UNIX环境，如图15-14所示。

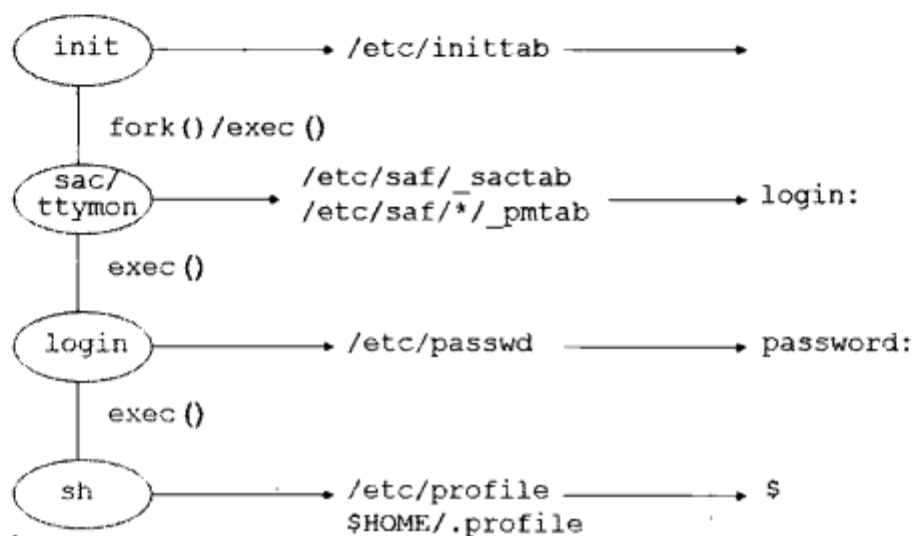


图15-14 用户注册过程

15.3.2 utmpx和wtmpx文件

在调度各种守护进程时，系统将会自动记录每个进程的起始和终止运行时间。对于所有活动的进程，这些信息均记录在**/var/adm/utmpx**文件中。当收到一个信号，说明调度的进程已经终止运行时，如果**/var/adm/wtmpx**文件存在，系统将会把相应的进程记录到**wtmpx**文件中，说明进程是什么时间启动和终止的，以及终止的原因等。也就是说，**wtmpx**文件将会记录整个进程调度的历史。

/var/adm/utmpx和**/var/adm/wtmpx**文件中记录的信息如下（详见**/usr/include/utmpx.h**文件的定义）：

- 注册用户名；
- **/etc/inittab**文件的id字段；
- 缩写的终端设备名（如**console**、**pts/2**等）；
- 进程ID；
- 进程类型（其中，1为运行级改变时间，2为系统引导时间，5为**init**调度的进程，6为**login**进程，7为用户进程，8为终止的进程）；
- 进程终止/出口状态；

- 记录时间（也即进程或会话的起止运行时间）；
- 远程系统的主机名或IP地址（远程注册时）。

为了查询用户的注册历史信息，以及用户访问系统时使用的终端设备或远程系统等信息，可以使用last（或fwtmp）命令，其语法格式简写如下：

```
last [-a] [-n number] [-f filename] [name | tty]
```

last命令可以直接读取wtmpx文件，如果再利用其“-f”选项，指定其他输入文件（如备份的wtmpx文件），还可以查询早期的用户注册信息。例如：

```
$ last
root      console      :0      Wed Jul 22 12:21    still logged in
reboot    system boot    Wed Jul 22 12:20
reboot    system down    Mon Jul 20 18:09
root      console      :0      Mon Jul 20 18:02 - 18:09    (00:06)
reboot    system boot    Mon Jul 20 18:01
reboot    system down    Mon Jul 20 17:40
gqxing    pts/6          localhost Mon Jul 20 17:34 - 17:39    (00:05)
gqxing    sshd           localhost Mon Jul 20 17:34 - 17:39    (00:05)
.....
$
```

15.4 系统关机过程

15.4.1 使用shutdown命令关闭系统

shutdown命令能够采用一种比较安全的方式关闭系统。在真正关闭系统之前，系统将会不断地通知或提醒已经注册到系统中的所有用户，使用户能够早做准备，及时保存尚未完成的处理工作，同时封锁其他用户再注册到系统。shutdown命令也会向所有的进程发送SIGTERM信号，使进程能够执行善后处理。最终，shutdown命令将会调用init进程，请求改变运行级。除了在指定的时间延迟之后实施实际的关机动作之外，shutdown命令也能够立即关机。shutdown命令的语法格式如下：

```
shutdown [-y] [-g grace-period] [-i init-state]
```

其中，“-y”选项表示事先确认系统关机提示，以便shutdown命令直接运行。“-g”选项用于指定关机前的等待时间，默认的关机等待时间为60秒。“-i”选项用于指定init命令的运行级参数，以便使系统进入指定的运行级。默认的运行级为0（或S）。

在关机实际开始之前，系统将会按照一定的时间间隔向所有注册的用户发出警告信息。下列例子表示在输入shutdown命令1分钟之后开始执行系统关机，并在实际执行关机之前输出警告信息：

```
# shutdown -y -i 0
Shutdown started.      Sun Aug 30 00:56:16 CST 2009
Broadcast Message from root (pts/3) on iscas Sun Aug 30 00:56:16...
The system iscas will be shut down in 1 minute
Broadcast Message from root (pts/3) on iscas Sun Aug 30 00:56:46...
```

```
The system iscas will be shut down in 30 seconds  
Broadcast Message from root (pts/3) on iscas Sun Aug 30 00:57:06...  
THE SYSTEM iscas IS BEING SHUT DOWN NOW !!!  
Log off now or risk your files being damaged  
Changing to init state 0 - please wait
```

15.4.2 使用init命令关闭系统

若想快速地关机，可以直接使用“init 0”命令。在使用“init 0”命令关闭系统时，系统将会以0作为参数，执行/etc/init.d/rc脚本，然后执行/etc/rc0.d目录中以“Knn”为起始文件名的所有Shell脚本，最终关闭系统。示例如下：

```
# init 0
```

15.4.3 使用其他命令关机

除shutdown和init命令之外，还可以使用uadmin等命令关闭或重新启动系统。uadmin命令提供最基本的系统控制功能，仅供超级用户在无法正常关闭或重新引导系统等特殊情况时使用，其语法格式如下：

```
uadmin cmd fcn [mdep]
```

其中，cmd和fcn是一个系统定义的整数常量，分别表示一个特定的命令与功能。可用的部分命令参数如下（参见/usr/include/sys/uadmin.h文件）：

- 1 表示重新启动。终止系统的运行，系统的下一步动作由随后的功能参数确定。
- 2 表示关机。终止所有的用户进程，把系统缓冲区中的数据写入磁盘，卸载外“/”文件系统。系统的关机方式由随后的功能参数确定。
- 5 强制系统处于瘫痪状态，把当前的系统内存映像写入转储设备，系统的下一步动作由随后的功能参数确定。

可以指定的部分功能参数如下：

- 0 终止系统的运行；
- 1 多用户重新引导系统；
- 2 多用户交互方式重新引导系统，提示用户指定引导程序的文件名；
- 3 单用户重新引导系统；
- 4 单用户交互方式重新引导系统，提示用户指定引导程序的文件名；
- 6 软关机（终止系统的运行，切断系统的电源）。

例如，当无法使用shutdown或init等常规命令关机或重新启动时，为了重新启动系统，可以使用下列命令：

```
# uadmin 2 1
```

注意：输入上述命令之后，系统将会立即开始关机，然后重新启动系统。因此，通常应当慎用此命令。

15.5 应用实例

在安装Oracle数据库时，通常要求以oracle用户的身份，注册到UNIX系统，使用Oracle提供的dbstart命令，启动Oracle数据库。使用dbshut命令关闭Oracle数据库。这种做法显然是不方便的。为了在启动UNIX时系统自动启动Oracle数据库，在关闭UNIX系统时自动关闭Oracle数据库，可以制作一个简单的Shell脚本，利用init进程和运行级的概念，自动启动和关闭Oracle数据库。

在UNIX系统中，与init进程和运行级有关的启动和关闭脚本通常均存储在/etc/init.d目录中。根据需要，可以使用ln命令，以硬链接或符号链接的方式，在相应的rcN.d目录中创建一个链接文件。这样做的好处是显而易见的——只需维护一个脚本副本，如果需要修改，也只需编辑一个文件，从而可以保证数据的一致性。

根据在启动系统时还是在关闭系统时使用脚本文件，文件名之前，应分别增加一个Sxx或Kxx前缀。其中xx是一个两位数字的序号。序号大小的确定，取决于应用程序的对系统服务的依赖程度。通常，序号越小，启动和关闭的顺序越靠前。反之，序号越大，启动和关闭的顺序越靠后。因此，在确定脚本序号的时候，一般应采取下列原则：

- 用于启动时的脚本应采用较大的序号，以便在系统服务都启动之后再启动应用程序；
- 用于关闭时的脚本应采用较小的序号，确保在系统服务都停止运行之前，先行关闭应用程序，以免出现不可预知的问题。

另外，当利用init命令改变运行级时，init进程会利用start或stop作为参数，运行/sbin/rcN脚本，从而依次运行/etc/rcN.d目录中的脚本。因此，我们可以利用这一惯例，以Oracle数据库为例，采用case分支结构语句，编制一个集启动与关闭功能为一体的脚本。然后，把写好的脚本文件放置在/etc/init.d目录中。

下面是一个针对Oracle版本9的启动脚本，相信稍加修改以后也可适用于其他版本的Oracle数据库：

```
# cat /etc/init.d/oracle
#!/bin/sh
ORACLE_HOME=/oracle/OraHome1
ORACLE_USER=oracle

if [ ! -f $ORACLE_HOME/bin/dbstart ]
then
    echo "Oracle startup: cannot start Oracle"
    exit 1
fi

case "$1" in
    'start')
        echo "Start Oracle database ..."
        su - $ORACLE_USER -c $ORACLE_HOME/bin/dbstart &
        su - $ORACLE_USER -c $ORACLE_HOME/bin/tnslsnr &
        ;;
    'stop')
        echo "Shutdown Oracle database ..."
```

```
su - $ORACLE_USER -c $ORACLE_HOME/bin/dbshut &
;;
esac
exit 0
#
```

写好上述脚本之后，可以采用下列步骤，在/etc/rc3.d和/etc/rc0.d目录中建立链接文件。

```
# cd /etc/rc3.d
# ln ../init.d/oracle S99oracle
# chmod 744 S99oracle
# chgrp sys S99oracle
# cd /etc/rc0.d
# ln ../init.d/oracle K00oracle
# chmod 744 K00oracle
# chgrp sys K00oracle
#
```

完成上述步骤之后，Oracle数据库即可随系统的启动而启动，随系统的关闭而关闭了。

第16章 文件系统内部组织

本章主要介绍文件系统内部的组织结构，以System V和UFS文件系统为例，详细讨论UNIX文件系统的核心——超级块，文件系统的信息节点，信息节点的分配与释放过程，数据块的分配与释放过程，以及信息节点与目录文件的关系，使读者对文件系统能有一个深入的了解和认识。

16.1 文件系统的组织结构

自UNIX System V 4.0引入虚拟文件系统之后，可以同时支持多个不同的文件系统，如System V（简称S5）、UFS、JFS、HSFS（CD-ROM, ISO 9660）以及PCFS（FAT/FAT32）等文件系统。

文件系统主要是在磁盘存储介质上实现的。磁盘通常是由一组盘片和磁头组成的，每个盘片围绕圆心划分为若干磁道。根据磁盘控制器及其驱动程序一次能够读写的数据量，再把每个磁道划分为若干扇区，每个扇区拥有512个字节，称做一个物理数据块，简称物理块或数据块。为了提高数据传输的吞吐能力和访问速度，文件系统通常均以1KB、2KB、4KB甚至8KB作为数据存储与处理的基本单位，称做逻辑数据块（在不至于混淆的情况下也简称做数据块）。

一个磁盘可以划分为若干个分区，每个分区可以创建一个文件系统。S5等文件系统将磁盘或磁盘分区中的数据块看做一个连续的线性存储区，其中的每个数据存储块均可通过数据块地址进行访问。

在一个磁盘或磁盘分区中，位于不同盘片上的同一磁道可以构成一个柱面。为了减少磁头的移动，提高文件的访问速度，UFS文件系统以柱面为基础组织磁盘的存储空间，把磁盘的所有柱面划分为若干个柱面组。然后以柱面组为单位，构建文件系统。其优点是能够减少磁头的移动，因而减少读写数据的等待时间，提高数据的传输性能，加快数据访问的响应时间。

磁盘上的部分存储区用于存储文件系统的管理与维护信息，其他大部分存储空间用于存储实际的数据。在不同的文件系统中，内部的组织形式并不一样。但在每个文件系统中，至少都具有四个组成部分。图16-1给出了S5与UFS文件系统的内部组织结构。

S5文件系统是UNIX采用的传统文件系统。在UNIX支持的所有文件系统中，S5是最具有代表性的文件系统，其内部的组织结构也比较简单直观。其他文件系统大都是基于S5文件系统的概念发展或演变出来的。其中的UFS文件系统是为提高数据传输的性能，减少数据访问的响应时间而设计的一种文件系统，两者的工作原理基本相同，但后者的内部组织结构则相对比较复杂。

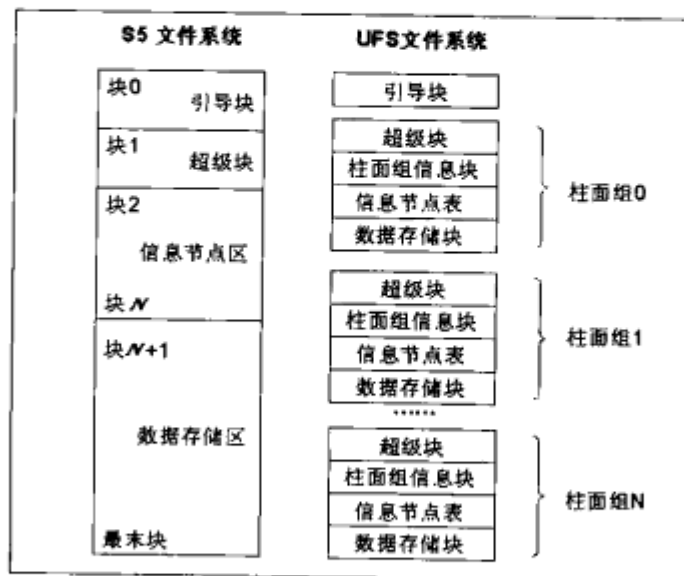


图16-1 S5与UFS文件系统的组织结构

为了能够把问题说清楚，我们先以S5文件系统为例子，介绍其内部组织结构。等到熟悉了S5文件系统之后，再学习其他文件系统就比较容易了。

一个S5文件系统通常包含四部分：引导块、超级块、信息节点区和数据存储区。

- 引导块：引导块用于引导UNIX系统，通常位于磁盘的起始部分。即使文件系统所在的磁盘不是启动盘，引导块的存储位置仍然保留。

- 超级块：超级块是文件系统的核心，其中包含文件系统的大小，空闲信息节点和数据块链表，供创建文件申请磁盘空间时使用。当启动UNIX系统，把文件系统安装到系统之后，超级块将被读入内存。磁盘上的超级块信息是静态的，而位于内存中的超级块是动态的。在系统运行期间，系统将会不断地更新位于内存中的超级块，并定时（可调内核参数，通常为60秒）把更新后的超级块写回磁盘中。当系统因意外事件而未正常关机时，有可能会造成两个超级块中的信息不一致，因而引起文件系统的损坏。出现此种情况后，将会引起UNIX在系统启动过程中修复文件系统，严重时甚至无法启动系统。

- 信息节点区：在信息节点区里，由于UNIX系统不允许使用0作为信息节点号（例如，在目录文件中，如果信息节点号为0，则表示相应的目录项为空），信息节点从1开始编号，但信息节点1不用，信息节点2总是用做根目录的信息节点。根目录的信息节点是在建立文件系统时分配的。根目录的第一个数据块总是位于数据存储区的第一个数据块中。

- 数据存储区：信息节点区之后即为数据存储区。数据区的大小取决于磁盘分区的容量和文件系统的类型，以及创建文件系统时指定的参数。一旦创建文件系统，确定了信息节点区的大小之后，也就确定了数据存储的起始位置，从而也就确定了数据存储区的大小。数据存储区用于存储文件或目录的实际数据。

图16-2给出了S5文件系统各个组成部分的划分与布局。

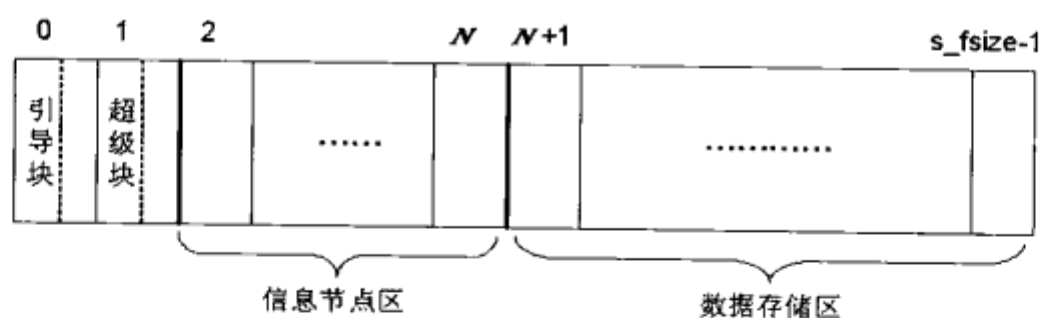


图16-2 S5文件系统布局

下面以1.44MB的3.5英寸软盘为例，说明文件系统的内部组织结构及其布局。为叙述简单起见，我们把整个软盘作为一个UNIX分区，即使用1.44MB的整个存储空间创建一个逻辑数据块为1KB的文件系统。数据块0的第一个物理块保留为引导区，数据块1的第一个数据块为文件系统的超级块。

引导块与超级块的位置是固定的，而且只占两个数据块的存储空间。现在的问题是怎样确定信息节点区和数据存储区的划分。实际上，只要确定了信息节点区的长度，也就确定了数据存储区的大小。确定信息节点区的大小可以有两种方法：一是在使用mkfs命令建立文件系统时指定，参见第17章“文件系统管理”；二是采用系统默认的分配方式。如果在创建文件系统时没有指定信息节点的数量，系统默认的信息节点数量与文件系统中全部数据块的数量之比为1:4。

根据上面的说明，一个1.44MB软盘的全部数据块为1440个，默认的信息节点数量应为360个（ $1440 \div 4$ ）。一个信息节点占用64个字节（参见后面的说明），故每个数据块可以容纳16

个 $(1024 \div 64)$ 信息节点，大约需要23个 $(360 \div 16)$ 数据块。因此，新建文件系统中的信息节点区从第二个数据块开始，占用23个数据块。数据存储区的长度为1415个 $(1440 - 23 - 2)$ 数据块，如图16-3。

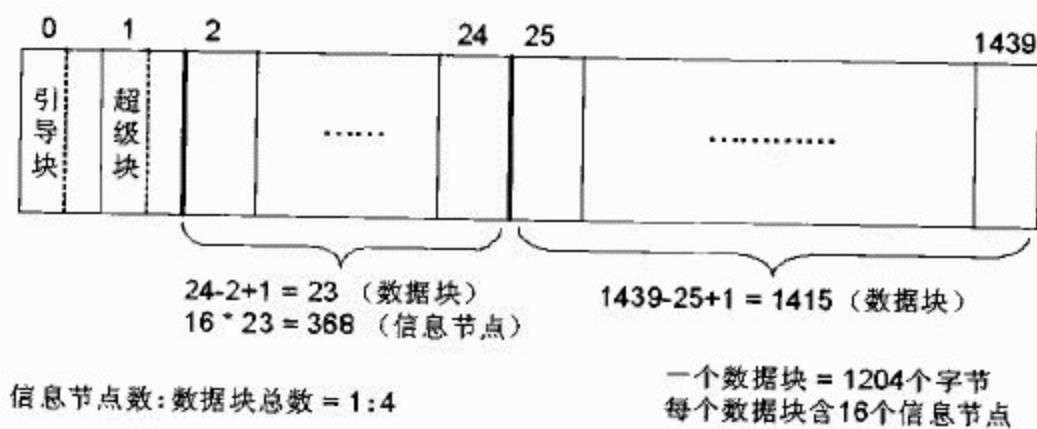


图16-3 在1.44MB软盘上创建的文件系统

按照上述划分，数据块2~24为信息节点区，数据块25~1439是数据存储区，其中第一个数据块（此处为25）总是根目录的第一个数据块。整个文件系统可以容纳368个文件或目录，总的文件或目录的数据容量不超过1415KB。

16.2 超级块

超级块是UNIX文件系统的核心，UNIX系统利用超级块实现文件系统的管理。对文件系统的所有操作都是通过超级块实现的。超级块存储的文件系统信息包括下列内容：

- 文件系统的大小；
- 信息节点区的大小；
- 空闲信息节点的数量，空闲信息节点数组，以及指向下一个空闲信息节点的指针；
- 空闲数据存储块的数量，空闲存储块数组，以及指向下一个空闲块的指针；
- 文件系统名和卷名；
- 文件系统逻辑数据块的大小；
- 超级块最后一次更新的日期和时间；
- 文件系统的状态与修改标志；
- 文件系统的类型（512B、1KB和2KB等）。

图16-4给出了超级块的结构定义（参见/usr/include/sys/fs/filsys.h）。

字节数	超级块	说明
2	s_size	文件系统的大小（块数）
4	s_fsize	空闲数据块地址数组的指针
2	s_nfree	空闲数据块地址数组
200	s_free[NICFREE]	空闲信息节点数组的指针
2	s_ninode	空闲信息节点数组
200	s_inode[NICINOD]	对空闲数据块数组进行操作时的加锁标志
1	s_flock	对空闲信息节点数组进行操作时的加锁标志
1	s_iloc	超级块修改标志
1	s_fmod	文件系统以只读方式安装的标志
1	s_ronly	超级块最后一次同步更新的时间
4	s_time	设备信息，其中第一个字节表示旋转周期，第二个字节表示每个柱面含有多少个数据块
8	s_dinfo[4]	文件系统中全部空闲数据块的统计计数
4	s_tfree	文件系统中全部空闲信息节点的统计计数
2	s_tinode	文件系统的名字
6	s_fname[6]	文件系统的卷标
6	s_fpack[6]	把超级块调整为512字节的填充字符（48个字节）
48	s_fill[12]	文件系统的状态
4	s_state	文件系统的识别号
4	s_magic	文件系统类型，实际上并无意义
4	s_type	

整个超级块占512个字节

图16-4 超级块的结构定义

16.3 信息节点

在UNIX系统中，每当创建一个文件、目录或特殊文件时，文件系统都会为其分配一个信息节点。也就是说，每个文件（包括目录和特殊文件等）都对应一个信息节点。信息节点位于信息节点区中。根据信息节点的结构定义（参见/usr/include/sys/fs/s5ino.h），每个信息节点占用64个字节。因此一个逻辑数据块可以存放16个信息节点，如图16-5。

除了文件名以及信息节点号存储于目录文件之外，信息节点包含了文件的所有信息，其中包括文件的属性、文件的实际数据在数据区中的存储位置等。列举如下：

- 文件的类型，如普通文件、目录文件、块特殊文件、字符特殊文件、管道（FIFO）文件、符号链接文件，以及套接字文件等；
- 文件读、写和执行的访问权限；
- 文件的链接计数；
- 文件属主的用户ID；
- 用户组ID；
- 文件的字节数；
- 拥有13个数据块地址的数组；
- 最后一次访问文件的日期和时间；
- 最后一次修改文件的日期和时间；
- 创建文件的日期和时间。

根据信息节点结构中di_mode的定义，文件的类型和访问权限字段占用两个字节（16位），其中的详细定义及意义如图16-6所示。

字节数	信息节点	说明
2	di_mode	文件的类型和访问权限
2	di_nlink	文件连接计数
2	di_uid	用户标识号
2	di_gid	用户组标识号
4	di_size	文件的大小，以字节为单位
39	di_addr[39]	数据块地址数组，每个地址占3个字节，13个地址总共占39个字节。其中前10个地址为直接地址，第11个地址为一级间接地址，第12个地址为二级间接地址，第13个地址为三级间接地址
1	di_gen	文件生成号，实际上并无意义
4	di_atime	最后一次访问文件的时间
4	di_mtime	最后一次修改文件的时间
4	di_ctime	创建文件的时间

整个信息节点占64个字节

图16-5 信息节点的结构定义

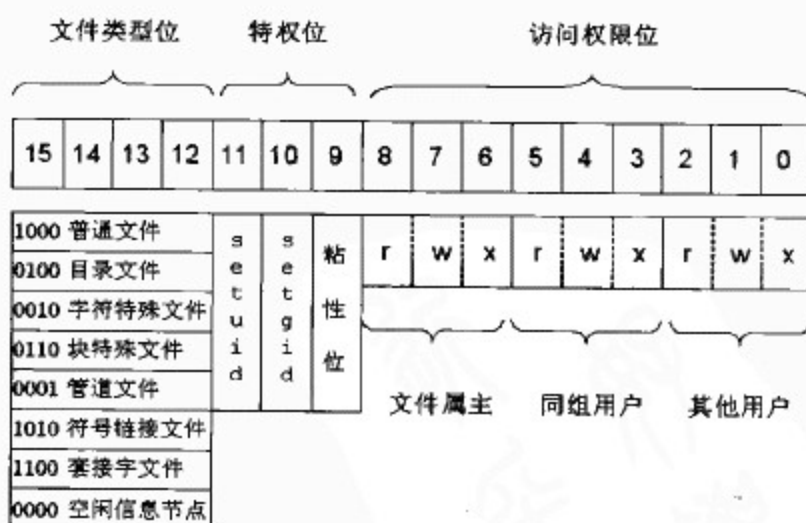


图16-6 文件的类型和访问权限定义

如果类型位为0000，表示当前的信息节点是空闲的。当创建一个文件时，根据文件的类型，文件系统将会设置文件的类型位。当使用chmod命令设置文件的访问权限、UID、GID和粘性位时，将会设置相应的二进制数据位。

16.3.1 特权标志位

特权标志位的设置使系统在调度执行相应的程序时能够给予特殊的关注。UNIX系统提供

三种特权标志位。

(1) **setuid** (设置用户ID)：当设置了**setuid**特权标志位时，程序调用者的有效用户ID将会在程序执行期间临时改为文件属主的用户ID。采取这种方式，用户能够执行一部分通常仅限于文件属主（通常为超级用户）才有权执行的操作。

例如，当普通用户想更换自己的密码时，需要使用**passwd**命令，修改/etc/shadow文件。出于系统安全的考虑，UNIX系统规定，只有超级用户才能访问shadow文件。而从用户安全的角度考虑，系统又鼓励用户经常修改自己的密码。

为解决此矛盾，UNIX系统把**passwd**命令的**setuid**特权标志位设置为1，使普通用户在运行**passwd**命令时，其有效用户ID能够临时变为passwd文件属主，也即超级用户root的用户ID，因而具有超级用户的访问权限，自然也能够修改shadow文件。如果使用下列ls命令显示passwd文件，可以看到文件访问权限字段中存在一个字符“s”：

```
$ ls -l /usr/bin/passwd
-r-sr-sr-x  1 root      sys          22628 Aug 15  2007 /usr/bin/passwd
$ ↑
```

文件属主执行权限位置中的“s”表示，在运行passwd命令期间，普通用户也拥有超级用户的访问权限。在更换密码之后，也即退出passwd命令之后，其有效用户ID又恢复到自己的实际用户ID。

为了设置一个文件的**setuid**特权标志位，可以使用下列命令：

```
# chmod u+s file
```

(2) **setgid** (设置用户组ID)：与**setuid**类似，当设置了**setgid**特权标志位时，程序调用者的有效用户组ID将会在程序执行期间临时成为文件所属用户组的ID。同样，采取这种方式，只要与文件同属一个用户组，普通用户也能执行部分特权操作。在下面的例子中，同组用户执行权限的“s”表示wall文件设置了**setgid**特权标志位：

```
$ ls -l /usr/sbin/wall
-r-xr-sr-x  1 root      tty          10592 Jan 23  2005 /usr/sbin/wall
$ ↑
```

为了设置一个文件的**setgid**特权标志位，可以使用下列命令：

```
# chmod g+s file
```

(3) 粘性位 (sticky)：粘性位表示共享代码段 (text) 方式。如果设置了粘性位，则当第一次调用相应的程序时，一经执行，系统就会把程序的代码段副本存储在交换区中，即使程序已经终止执行。这种处理方式使得一旦再次执行程序，便能够减少加载程序代码段的时间。这是因为访问交换区要快于访问文件系统程序文件。

如果某个共享目录设置了粘性位，则普通用户只能删除该目录下属于自己的文件。对于目录中的其他文件，即使其访问权限允许文件属主和同组用户之外的其他用户写文件，其他用户也无法写和删除，这将防止用户无意中覆盖或删除共享目录中属于每个用户的文件。注意，在新的UNIX系统中，粘性标志位只用于目录，文件已不再使用。

如果某个目录或文件设置了粘性位特权标志，利用下列ls命令，可以看到其访问权限字段中存在一个字符“t”：

```
$ ls -ld /tmp
drwxrwxrwt  8 root      sys      1044 Jun 12 20:16 /tmp
$
```

为了设置一个目录的粘性位特权标志，可以使用下列命令：

```
# chmod +t dirname
```

16.3.2 数据块地址数组

按照信息节点的定义，数据块地址数组`di_addr[]`共占用39个字节位置，其中每三个字节表示一个数据块地址，因而可以拥有13个数据块地址（0~12），用以指向存储文件内容的数据块。其中前10个地址是直接地址，直接指向文件内容的前10个逻辑数据块。后三个地址分别为一级、二级和三级间接索引。如果文件大于10个逻辑数据块的容量，则第11个地址指向一个间接地址块。间接地址块包含的是直接数据块的地址而非文件内容。第12个地址指向一个二级间接地址块，其中包含一级间接地址块的地址。第13个地址指向一个三级间接地址块的地址。图16-7给出了从信息节点开始的地址指针、间接地址块与数据块的指向关系。

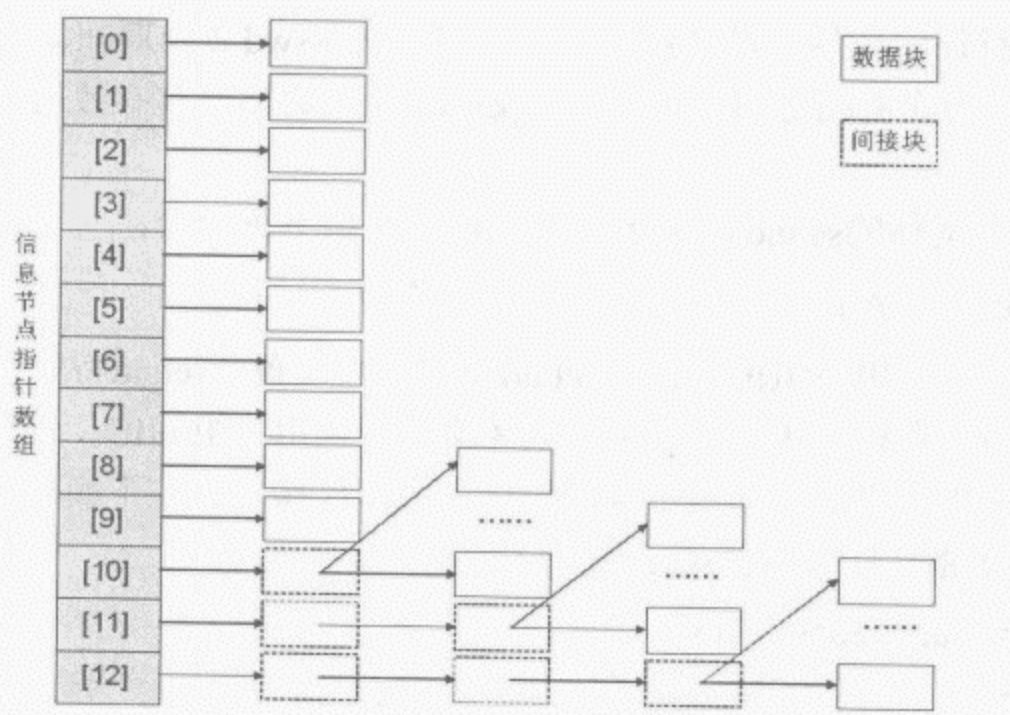


图16-7 信息节点中的地址指针与间接地址块和数据块的指向关系

在间接索引块中，每四个字节引用一个数据块地址（数据块号）。以逻辑数据块为1024字节的文件系统为例，一个间接索引块可以容纳256个数据块号。因此，在S5文件系统中，一个文件最大可以具有 $(10 + 256 + 256^2 + 256^3) \times 1 \text{ KB} = 16\,843\,018 \text{ KB}$ ，约为16GB。表16-1给出了文件大小与信息节点地址数组的理论计算关系。

表16-1 文件大小与地址数组的理论计算关系

1024字节数据块	直接寻址	一次间接寻址	二次间接寻址	三次间接寻址
数据块数	10	256	65 536	16 777 216
字节数	10 240	262 144	67 108 864	17 179 869 184
累计字节数	10 240	272 384	67 381 248	17 247 250 432

16.4 数据区与空闲数据存储块的组织

在使用mkfs命令创建文件系统时，系统将会把位于数据存储区中的所有空闲数据块组织起来，每50个空闲数据块为一组，使用其中的第一个数据块作为链表数据块。链表数据块中存有50个数据块的地址（数据块号），每个数据块地址为4个字节。其中的一个地址依次指向下一个链表数据块。另外，链表数据块中还包含一个空闲数据块计数字段。数据存储区中的空闲数据块组织形式如图16-8所示。

其中，第一组的50个数据块位于超级块的空闲数据块地址数组s_free[]中。数组的第0号元素s_free[0]容纳的是链表数据块，指向下一组空闲数据块的链表数据块。作为超级块中空闲数据块地址数组的指针，其中也含有数组中空闲数据块地址的数量。s_nfree的初始值为50。

注意：在使用mkfs命令建立文件系统，组织空闲数据块链表时，UNIX系统将会尽量把位于同一柱面的空闲数据块组织在一起。另外还会考虑磁盘的旋转间隙，采用交错方式，把相隔一定间隙的空闲数据块组合在一起，以便减少磁头的移动次数，减少读写数据块的等待时间，从而提高数据的访问效率，如图16-9所示。

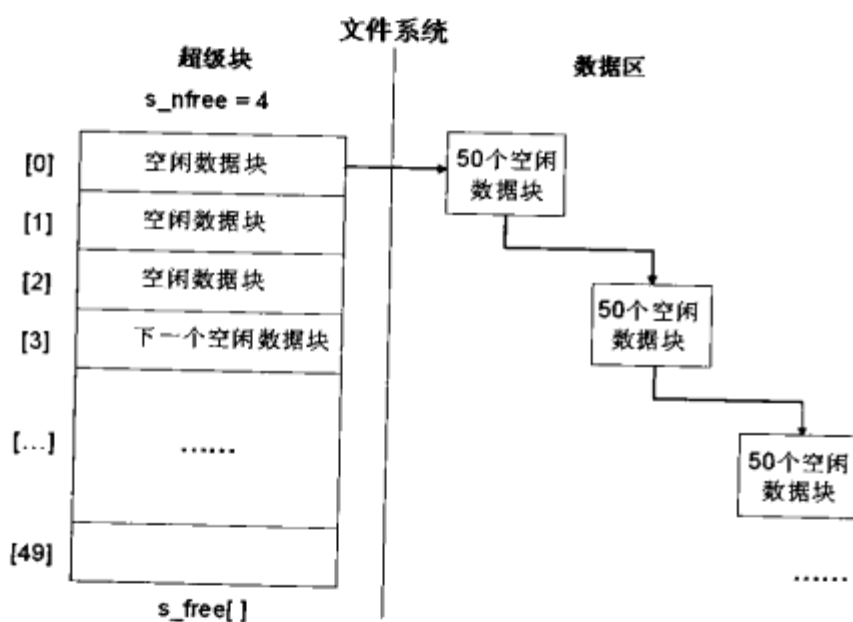


图16-8 空闲数据块的组织形式 (1)

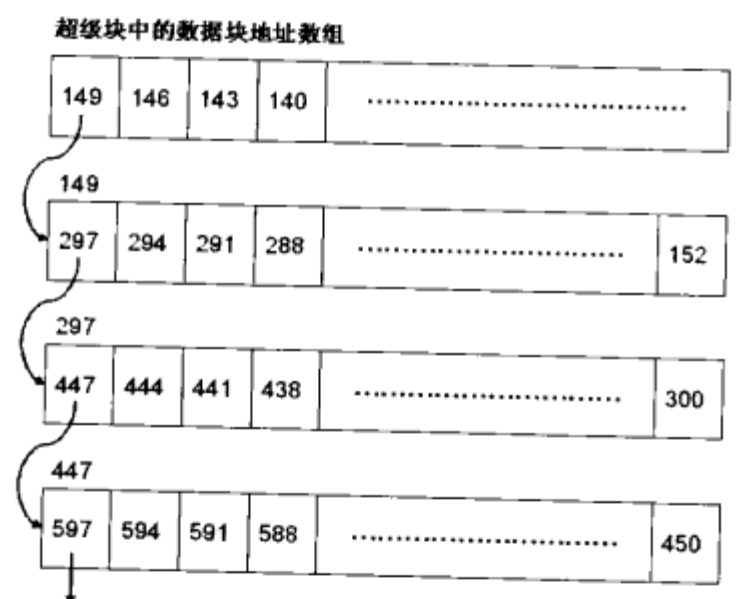


图16-9 空闲数据块的组织形式 (2)

16.5 信息节点的分配与释放

创建和删除文件或目录时，都会涉及到信息节点的分配与释放。当使用mkfs命令创建文件系统时，UNIX系统将会把前100个信息节点号置于空闲信息节点地址数组中。同时把s_ninode设置为100。

当申请一个空闲的信息节点时，首先需要把超级块中表示空闲信息节点计数的s_tinode字段值减1，然后再把信息节点地址数组指针s_ninode字段中的数值减1。如果s_ninode等于0，系统将开始进行垃圾收集，把收集到的100个（如果文件系统空闲信息节点不足时，则以实际收集到的信息节点数量为准）空闲信息节点（其di_mode字段等于0）依次置于s_inode[99]、s_inode[98]，直至s_inode[0]中。然后再执行下述过程。

如果s_ninode减1后仍然大于0，则使用s_ninode作为指针，从空闲信息节点地址数组中取出

一个信息节点号作为新分配的信息节点。然后按照下列计算公式求出信息节点所在的数据块：

数据块号 = [(信息节点号 - 1) ÷ 每个数据块中含有的信息节点数量] + 信息节点区的起始数据块号

假定取出的信息节点号为100，每个数据块中含有的信息节点数量为16，信息节点区的起始数据块号为2，则可以计算出信息节点所在的数据块号：

数据块号 = [(100 - 1) ÷ 16] + 2 = 6 + 2 = 8

确定了信息节点所在的数据块之后，还要根据下列公式计算出信息节点起始位置在数据块中的偏移值（其中，信息节点的长度为64）：

偏移值 = [(信息节点号 - 1) mod 每个数据块中含有的信息节点数量] × 信息节点长度

按照上述公式，可计算出信息节点起始位置在数据块中的偏移值：

偏移值 = [(100 - 1) mod 16] × 64 = 3 × 64 = 192

当删除一个文件或目录，需要释放一个信息节点（和相应的数据块）时，首先需要把超级块中表示空闲信息节点计数的s_tinode字段加1，然后再把表示信息节点地址数组指针的s_ninode字段值加1。如果s_ninode字段小于100，则把释放的信息节点置于相应的信息节点地址数组位置中。

如果s_tinode字段加1后等于100，则需要检查信息节点地址数组s_inode[0]中的信息节点号。如果释放的信息节点号低于s_inode[0]中的信息节点号，使用释放的信息节点号替换s_inode[0]中的信息节点。否则什么也不做。

16.6 数据块的分配与释放

数据块地址数组指针s_nfree含有数组s_free[]中空闲数据块的数量。当分配或释放数据块时，超级块中的s_free将相应地执行减1或加1运算，以反映文件系统中当前空闲数据块的数量。同时，s_nfree字段也会相应地执行减1或加1运算。

当用户因创建文件而需要分配数据块时，系统将把s_nfree字段中的数值减1。如果s_nfree字段的数值大于0（如5），则返回s_free[s_nfree]中的数据块（如161），如图16-10所示。

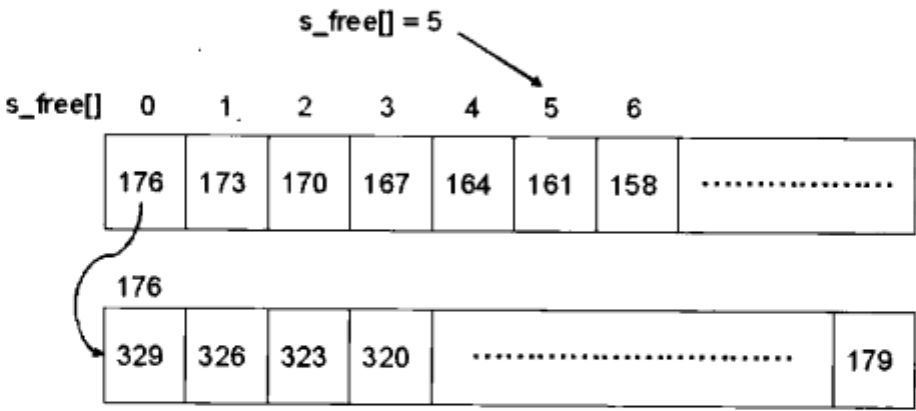


图16-10 分配空闲数据块

如果s_nfree字段等于0，封锁超级块，防止其他超级块操作，然后读取s_free[0]，把其中包含的50个数据块号依次置于s_free[]数组中，再把数值50赋予s_nfree字段。接着返回包含50个地址的数据块（如176）。最后解除对超级块的封锁，如图16-11所示。

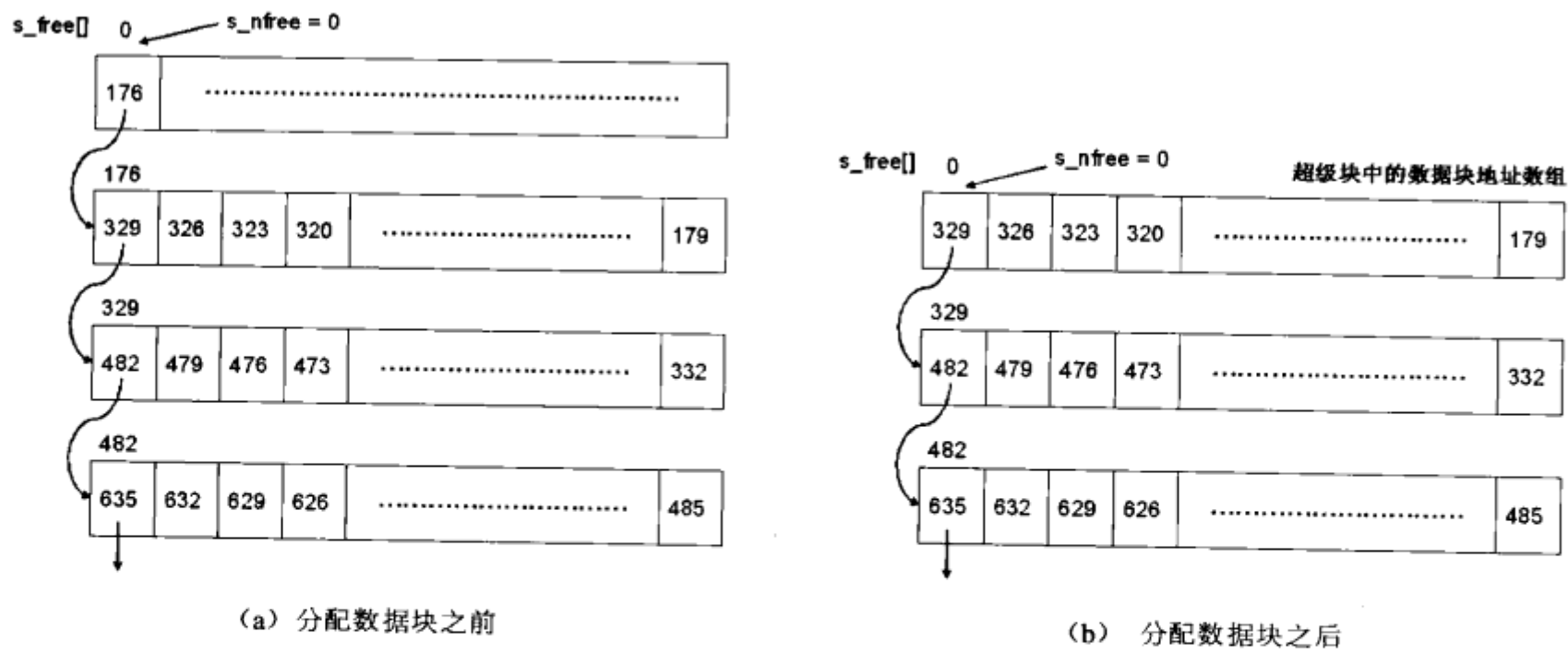


图16-11 分配空闲数据块时的极端情况

当删除文件或目录因而需要释放数据块（包括信息节点）时，如果s_nfree字段小于50（如2），则将释放的数据块号（假定为986）置于s_free[s_nfree]中。然后把s_nfree字段中的数值加1，如图16-12所示。

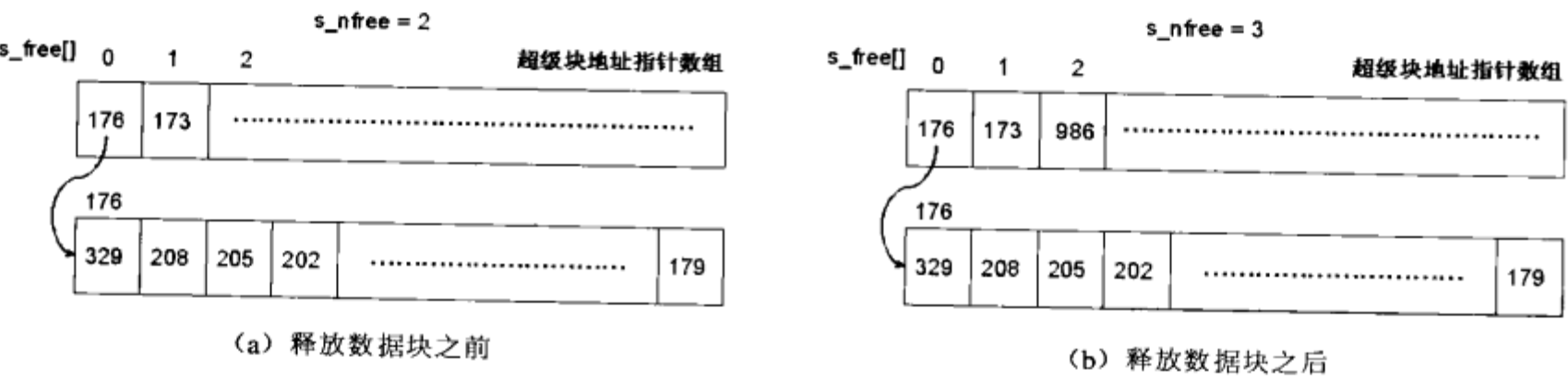


图16-12 释放数据块

如果s_nfree字段等于50，封锁超级块，防止其他超级块操作，然后把s_free[]数组中的50个数据块号写到刚释放的数据块中，接着让s_free[0]指向这个刚释放的数据块，再把s_nfree字段设置为1。最后解除对超级块的封锁，如图16-13所示。

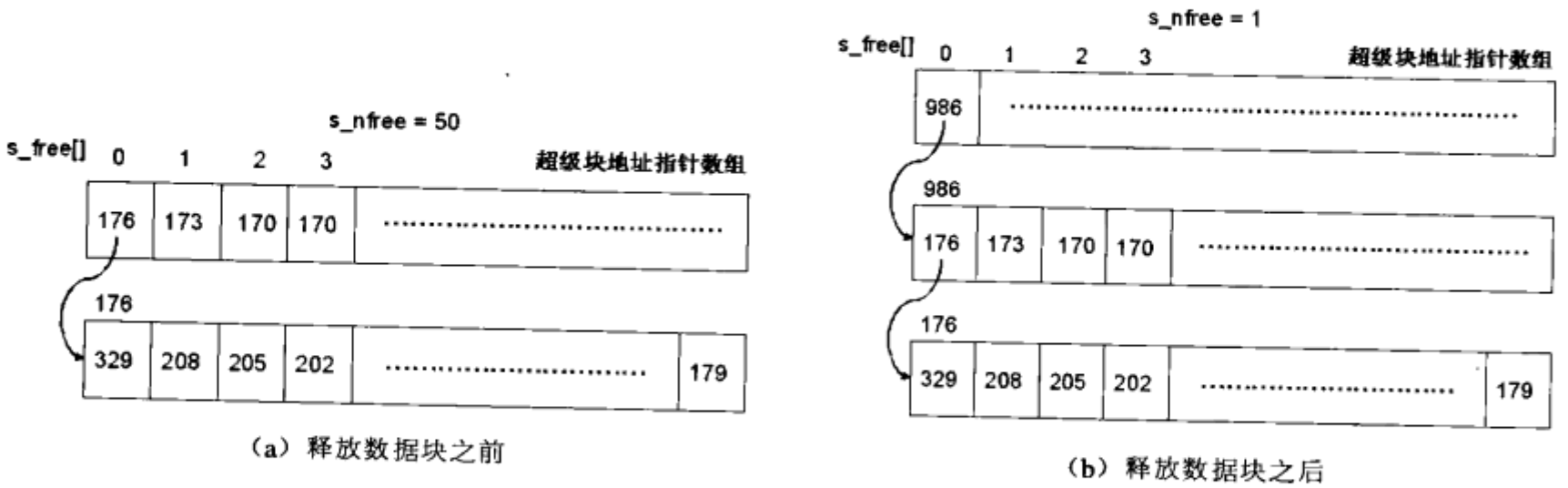


图16-13 释放数据块时的极端情况

16.7 信息节点与目录和文件的关系

每个文件（包括目录和特殊文件）总有一个与之对应的信息节点，信息节点中的信息确定了文件的属性（文件属主和文件大小等），以及文件内容所在数据块的地址等。实际上，除了文件名与信息节点号，文件的所有信息均可从信息节点中找到。文件名存放在所属目录的目录文件中，其关系如图16-14所示。

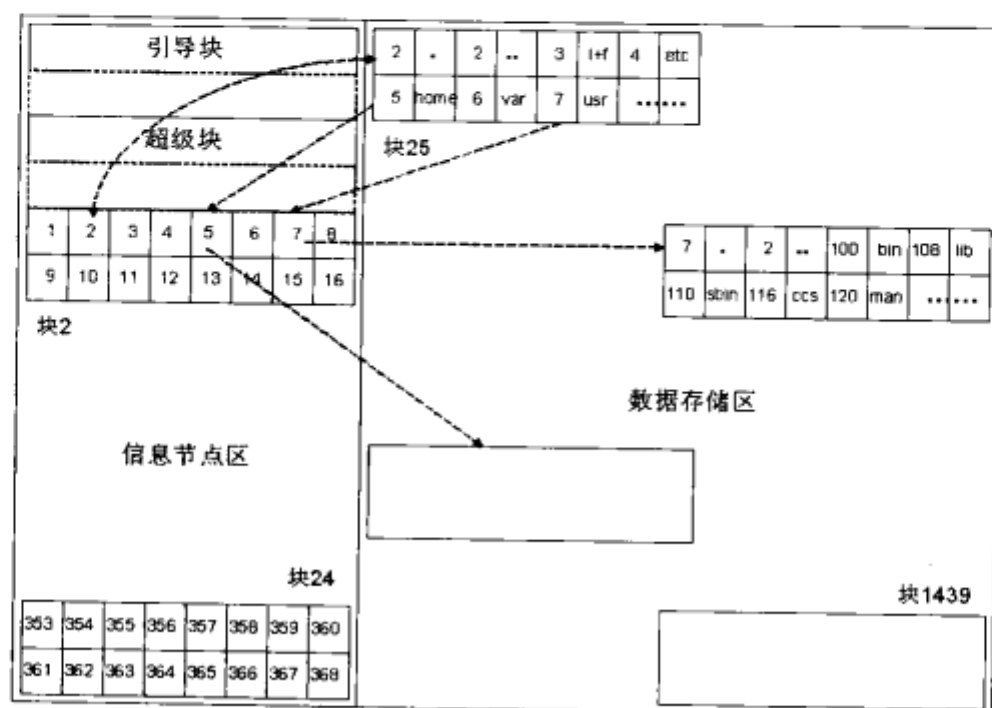


图16-14 信息节点与目录或文件之间的关系

16.8 UFS文件系统

16.8.1 UFS文件系统的组织结构

UFS文件系统是某些UNIX系统采用的传统文件系统，也是最有代表性的文件系统。UFS文件系统源于伯克利版UNIX的快速文件系统（Fast File System, FFS），与S5文件系统相比，UFS文件系统具有三个特点：采用较大的逻辑数据块（4KB和8KB），借以提高数据传输的吞吐能力；以柱面组为单位组织文件系统，减少磁头的移动距离，从而提高数据的访问速度；采用多个超级块，每个柱面组存储一个副本，以提高文件系统的可靠性。

UFS文件系统把一个磁盘或磁盘分区中的磁盘柱面分为若干柱面组，每个柱面组由若干相邻的柱面构成。然后再以柱面组为单位组建文件系统。在UFS文件系统中，柱面组的数量是在创建文件系统时确定的。图16-15给出了一个典型的UFS文件系统的组织结构。

由图16-2可知，一个UFS文件系统是由多个柱面组组成的。除第一个柱面组包含一个额外的8K字节的引导块之外，每个柱面组均由下列四部分组成：

- 超级块——文件系统的核心，其中含有文件系统的各种结构与汇总信息。
- 柱面组信息块——其中含有超级块的辅助管理与控制信息，以及柱面组的汇总信息。
- 信息节点区——用于存储信息节点，其中包括除文件名和信息节点号之外的所有文件属性信息，如文件的类型、访问权限和大小等。

UFS文件系统的组织结构

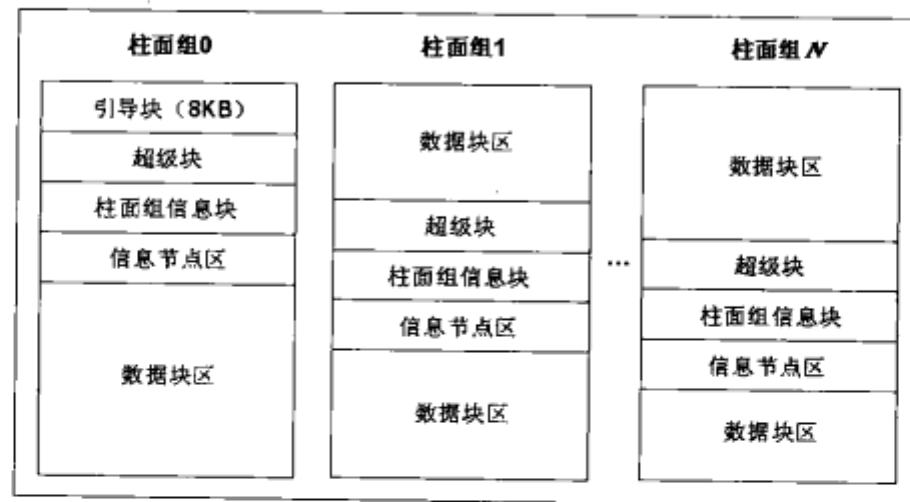


图16-15 UFS文件系统的典型组织结构

- 数据块区——用于存储文件的实际数据。

UFS文件系统以8KB（或4KB）作为逻辑数据块。由于大的数据块容易造成存储空间的浪费，UFS文件系统又把逻辑数据块进一步细分为512B（逻辑数据块为4KB时）、1KB、2KB或4KB的数据片（fragment）。默认的数据片为1KB。

下面将进一步说明各种组成部分的基本功能与文件系统的组织结构。

16.8.2 引导块

每个UFS文件系统，其磁盘分区的起始位置都有一个8KB字节的引导块，用于存储引导系统时使用的引导程序或数据。同S5文件系统（即System 5）中的引导块一样，UNIX系统并不使用这个引导块启动系统。按照原来的设计，引导块主要用于存储引导UNIX系统的机器指令代码和数据。即使文件系统所在的磁盘不是系统盘，并不用于引导系统，引导块的位置仍然保留，只是其中并不存储任何信息。在一个拥有多个柱面组的文件系统中，引导块仅存在于第一个柱面组（柱面组0）中，占用文件系统所在磁盘分区最前面的8KB字节位置。

16.8.3 超级块

超级块是文件系统的核心，其中包含磁盘分区的关键数据和总体结构信息。操作系统利用超级块对文件系统进行管理，所有的文件操作都是通过超级块实现的。为了保证文件系统的可靠性，每个柱面组均保存一个超级块副本。其中，位于柱面组0中的超级块称做主超级块，其他超级块称做次超级块或备份超级块。

磁盘上的超级块信息是静态的，而位于内存中的超级块是动态的。在UNIX系统工作期间，系统将会不断地更新位于内存中的超级块，并定时把更新后的超级块写回磁盘中。当系统因意外事件没有正常关机时，将会造成两个超级块中的信息不一致，因而引起文件系统的损坏。这种情况将会造成在系统在引导过程中修复文件系统，严重时甚至无法启动系统。

在安装UFS文件系统时，系统将会把第一个柱面组（也即柱面组0）中的主超级块读入内存。在文件系统工作期间，系统仅更新位于内存中的超级块。由于UFS文件系统的超级块大部分为静态数据，故不存在太大的问题。当卸载文件系统时，对超级块的任何更新将会同时写到磁盘上的每个超级块中。

超级块内存储的文件系统信息主要包括下列内容（参见第17章“文件系统管理”）：

- 在每个柱面组中，超级块、柱面组信息块、信息节点区和数据块存储区相对于起始地址的偏移值（柱面组0中的主超级块通常总是位于引导块之后的固定位置）；
- 文件系统中所有存储块的数量（文件系统的大小）；
- 文件系统中所有数据块的数量；
- 文件系统名和卷名；
- 文件系统中逻辑数据块的大小；
- 文件系统中逻辑数据片的大小；
- 超级块最后一次更新的时间；
- 文件系统中柱面组的大小和数量；
- 每个柱面组中的柱面、信息节点以及数据块的数量；
- 每个柱面的磁道数量；
- 每个磁道的扇区数量；
- 每个柱面的扇区数量；
- 文件系统汇总信息区，其中包括现有目录的数量，以及空闲信息节点、空闲数据块与空闲数据片的数量；
- 文件系统的状态与修改标志；
- 最后一次安装时安装点的路径名。

注意：超级块和柱面组各包含一个汇总信息区。其中，超级块汇总信息区存有文件系统级的汇总信息，反应的是整个文件系统的目录、信息节点、数据块以及数据片的统计数据。而柱面组汇总信息区仅存有柱面组一级的汇总信息，反应的是柱面组的相应统计数据，如图16-16所示。

文件系统汇总信息	
字段	说明
cs_ndir	所有柱面组中目录的数量
cs_nbfree	所有柱面组中空闲数据块的数量
cs_nifree	所有柱面组中空闲信息节点的数量
cs_nffree	所有柱面组中空闲数据片的数量

图16-16 文件系统汇总信息区

16.8.4 柱面组信息块

柱面组信息块用于维护其所在柱面组的信息。一旦安装了文件系统之后，柱面组信息块中的信息将会随着对文件系统的操作不断地更新，以反映文件系统的最新状态。柱面组信息块主要包括下列信息：

- 柱面组中的柱面数量；
- 柱面组中的信息节点数量；
- 柱面组中的数据块数量；
- 柱面组中每个柱面拥有的数据块数量；
- 柱面组汇总信息区，其中包括柱面组中现有目录的数量，以及空闲数据块、空闲数据片和空闲信息节点的数量；

- 数据块分配位图，用于标记柱面组中已用的或空闲的数据块与数据片；
- 信息节点分配位图，用于标记柱面组中的空闲信息节点；
- 指向空闲存储块链表中第一个空闲块的指针。

柱面组信息块的结构定义如图16-17所示。

其中，柱面组汇总信息区的结构定义如图16-18所示。

字段	说明
cg_link	保留为柱面组链表指针
cg_magic	标识码(0x090255)，用于区别于不同的实体
cg_time	最后一次更新柱面组信息块的时间
cg_cgix	标记当前柱面组为文件系统中的第几个柱面组
cg_ncyl	当前柱面组中拥有的柱面数量
cg_niblk	当前柱面组中拥有的信息节点数量
cg_ndblk	当前柱面组中拥有的数据块数量
cg_cs	当前柱面组的汇总信息
cg_rotor	最后一次分配的数据块的地址
cg_frotor	最后一次分配的数据片的地址
cg_itor	最后一次分配的信息节点的地址
cg_freeuse[MAXF]	可用数据片计数
cg_btrotor	每个柱面拥有的数据块总数
cg_boff	空闲存储块链表中第一个空闲存储块的地址
cg_iusedoff	信息节点分配位图(用于标记信息节点的分配与空闲情况)
cg_freeoff	数据块分配位图(用于标记数据块的分配与空闲情况)
cg_nextfreeoff	下一个可用的空间
cg_sparecon[16]	保留位置，供将来使用
cg_spec[1]	用于柱面组位图的空间

图16-17 柱面组信息块

柱面组汇总信息	
字段	说明
cs_ndir	柱面组中目录的数量
cs_nbfree	柱面组中空闲数据块的数量
cs_nifree	柱面组中空闲信息节点的数量
cs_nffree	柱面组中空闲数据片的数量

图16-18 柱面组汇总信息

在柱面组信息块中，最重要的数据是数据块分配位图与信息节点分配位图。在创建UFS文件系统时，系统将会根据每个柱面组中拥有的存储块数量（包括信息节点存储块和数据存储块），在每个柱面组中组织一个数据块分配位图，和一个信息节点分配位图。

根据数据块区的大小，可以确定数据块分配位图的长度。根据数据片的大小，可以确定每个位图的有效数据位。同样，根据信息节点区的大小，也可以确定信息节点分配位图的长度。

在一个逻辑数据块为8192字节，逻辑数据片为1024字节的UFS文件系统中，每个逻辑数据块对应一个一字节的位图，用于标记一个数据块及其数据片的分配与空闲状态。位图中的所有数据位均初始化为1。如果数据块或数据片已经分配，位图中的相应数据位为0，如图16-19所示。

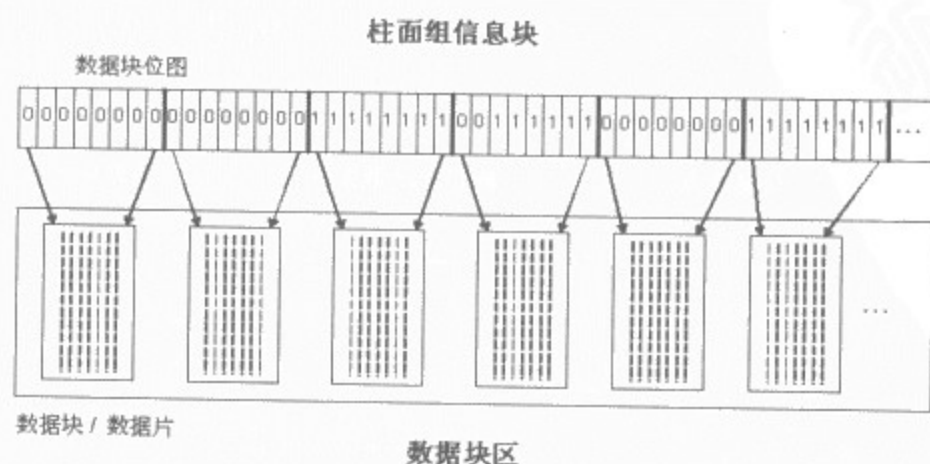


图16-19 数据块位图与数据块的关系

数据块分配位图用于标记数据块或数据片的分配与空闲状态。例如，如果某个位图的8个数据位均为1，表示相应的数据块及其中的数据片均为空闲的。如果其中的第一个数据片已经

分配，则相应的位图为01111111，这也意味着相应的数据块已经占用。当第二个数据片被分配，相应的位图为00111111。重复这个过程，直至8个数据片完全被用。此时，所有的数据片将被复制到一个新的数据块中。复制数据之后，原来的数据片又返回空闲数据片链表，数据块分配位图又重置为11111111。

信息节点也使用位图表示其分配与空闲状态，只是其中仅用一个数据位。数值0表示相应的信息节点已经分配，数值1表示相应的信息节点是空闲的。

16.8.5 信息节点区与信息节点

每个柱面组均存在一个由一系列存储块组成的信息节点存储区，用于存储文件的信息节点。除了文件名位于目录文件之外，信息节点包含了文件的所有信息。一个信息节点占用128个字节，其中包含下列信息：

- 文件的类型，如普通文件、目录文件、块特殊文件、字符特殊文件、管道（FIFO）文件、符号链接文件以及套接字文件；
- 文件读、写和执行的访问权限；
- 文件的硬连接计数（多个文件名共享同一信息节点和数据内容）；
- 文件属主的用户ID；
- 文件所属的用户组ID；
- 文件的大小（字节数）；
- 一个拥有15个数据块地址的数组，用于引用分配给文件的数据块。在字符和块特殊文件的情况下，数据块中的内容为特殊文件的主次设备号；
- 分配给文件的数据块数量；
- 最后一次访问文件的日期和时间；
- 最后一次修改文件的日期和时间；
- 首次创建文件的日期和时间。

数据块地址数组（0~14）指向存储文件内容的数据块。其中前12个地址是直接地址，也就是说，它们直接指向文件内容的前12个逻辑数据块。如果文件大于12个逻辑数据块的容量，则第13个地址指向一个间接地址块，其中包含的是直接数据块地址而非文件内容。第14个地址指向一个二次间接地址块，其中包含一次间接地址块的地址。第15个地址指向一个三次间接地址块的地址（三次间接寻址并未实现）。图16-20给出了从信息节点开始，地址指针、间接存储块与数据块之间的指向关系。

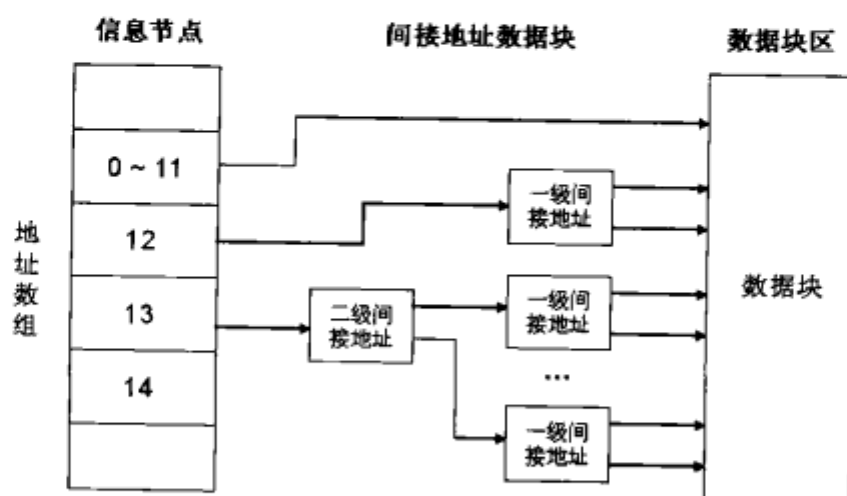


图16-20 UFS文件系统的典型组织结构

在间接索引块中，每四个字节引用一个数据块号。以8192字节的逻辑数据块为例，一个间接索引块可以容纳2048个数据块号。因此，在UFS文件系统中，一个文件最大可以具有 $(12+2048+20482) \times 8KB = 33\,570\,912KB$ ，约为32GB。表16-2给出了文件大小与信息节点地址数组的理论计算关系。

表16-2 文件大小与地址数组的理论计算关系

8192字节数据块	直接寻址	一次间接寻址	二次间接寻址	三次间接寻址（尚未实现）
数据块数	12	2048	4 194 304	
字节数	98 304	16 777 216	34 359 738 368	
累计字节数	98 304	16 875 520	34 376 613 888	

16.8.6 数据块区

数据块区包括柱面组中除分配给超级块、柱面组信息块和信息节点区之外的所有存储空间。数据块用于存储文件的实际数据。对于目录文件而言，数据块存储的是文件的名字和赋予文件的信息节点号。UFS文件系统将会尽可能地把相关数据块分配到同一柱面内。在UFS文件系统中，尚未分配的存储块称做空闲存储块（简称空闲块）。

16.8.7 数据块的分配与释放过程

1. UFS数据块的分配过程

在讨论UFS文件系统数据块的分配过程中，我们假定UFS文件系统的逻辑数据块为8192字节，逻辑数据片为1024字节。

每当创建一个新文件或扩充现有的文件时，文件系统都需要为文件分配磁盘空间。UFS文件系统分配磁盘空间的方式可以分为三种情况：一次分配一个逻辑数据片；一次分配若干连续的逻辑数据片；或一次分配整个逻辑数据块。这个过程是由write()系统调用完成的，其步骤如下：

- (1) 如果文件系统上的空闲数据块或数据片的数量为0，系统将会返回一个写错误，其错误代码为EIO，表示出现了I/O错误。
- (2) 如果当前用户并非超级用户，且文件系统的剩余磁盘空间与整个存储空间的比率低于创建文件系统时设定的一个百分比（通常为10%），系统将会返回一个磁盘I/O写错误。
- (3) 如果启用，系统还要检查当前用户的数据块限额是否超限。如果已经超限，系统仍会返回一个磁盘I/O写错误。
- (4) 如果一切正常，文件系统将基于特定的数据写请求，决定分配一个数据片、若干数据片或整个数据块。
- (5) 根据确定的磁盘空间需求量，按照下述原则分配数据块或数据片（这里假定文件系统采用每次分配一个数据片的方式）：

- 按照性能优化的原理，从与文件现有数据位于同一柱面的数据块中分配一个数据片。
- 如果找不到这样的空闲数据块，则从同一柱面组的数据块中分配一个数据片。
- 如果整个柱面组均找不到可用的空闲数据块或数据片，则从文件系统的其他柱面组的数据块中分配一个数据片。

• 如果是一个新建的文件，则从与信息节点位于同一柱面组的空闲数据块中分配一个数据片。

(6) 更新柱面组信息块中的数据块分配位图。

如前所述，数据块位图中的数据位均初始化为1。如果数据块或数据片已经分配，则相应的数据位为0。在下列逻辑数据块为8192字节，逻辑数据片为1024字节的例子中，数据块1已经完全分配，数据块2有6个数据片是空闲的，数据块3整个都是空闲的。

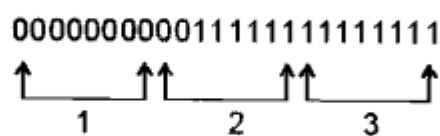


图16-21

当需要附加存储空间时，文件系统总是先分配一个数据片。随着文件的扩展，文件系统不断地从数据块中依次分配数据片，直至分配7个数据片为止。当文件还需要额外的存储空间，即还需要增加数据片时，文件系统将会从

空闲数据块区中分配一个完整的数据块，同时把数据块的地址写入信息节点的地址数组中。然后把前7个数据片中的数据复制到新的数据块中，接着再写入附加的数据。最后释放前7个数据片，把数据块位图中的数据位置1，供新的I/O请求使用。

2. 数据块的释放过程

当删除整个文件，或删除文件的一部分数据内容时，文件系统将会释放文件占用的存储空间，包括数据块和数据片。数据块或数据片的释放过程相对比较简单，其步骤如下。

(1) 在柱面组信息块中，更新数据块位图。

(2) 如果释放的是一个完整的数据块，或释放数据片后使整个数据块均处于空闲状态（数据块位图中的所有数据位均为1），需要更新柱面信息块中的汇总信息，包括空闲数据块和空闲数据片计数。

(3) 如果启用了限额控制，还要更新用户的存储空闲限额信息。

16.8.8 信息节点的分配与释放过程

1. UFS信息节点的分配过程

前面讨论了数据块的分配过程。实际上，每当创建一个新的文件时，除了分配数据块之外，文件系统首先需要为新建文件分配一个信息节点。信息节点的分配过程和采用的性能优化准则与数据块的分配过程类似。其步骤如下：

(1) 在分配信息节点时，首先需要确保文件系统中具有可用的信息节点，并且用户具有足够的信息节点限额。否则将会返回错误信息。

(2) 如果创建的文件是目录文件，文件系统将会从信息节点数量较多，目录数最少的柱面组中选取信息节点。

(3) 如果创建的文件不是目录文件，文件系统将会从其父目录所在的柱面组中分配信息节点。

(4) 如果其父目录所在柱面组中的信息节点已全部分配，则从其他柱面组中分配一个信息节点。

(5) 更新柱面组信息块中的空闲信息节点位图，把相应的数据位设置为0。空闲信息节点位图的用法与数据块位图基本相同。1表示空闲，0表示已经占用。但不同的是，在每个信息节点位图中，只有第一个数据位有效，其余7位没有意义。

(6) 更新柱面组信息块中的空闲信息节点计数。

2. 信息节点的释放过程

信息节点的释放过程相对比较简单，其步骤如下：

- (1) 从柱面组信息块中，清除相应信息节点位图的数据位。
- (2) 更新柱面信息块中的汇总信息，即空闲信息节点计数。
- (3) 如果启用了限额控制，还需要更新用户的文件限额信息。

第17章 文件系统管理

本章主要以UFS文件系统为例，介绍文件系统的管理，说明怎样创建文件系统，怎样安装和卸载文件系统，怎样确定存储设备的文件系统类型，最后讨论怎样检测与修复受损的文件系统，确保系统能够正常启动。

当加装新的磁盘等存储设备时，首先需要确保系统能够识别新增的设备，自动生成相应的设备文件。然后需要利用format等命令格式化磁盘设备，利用fdisk命令划分磁盘分区（如果整个磁盘仅用于UNIX系统，也可以把磁盘划分为一个UNIX分区）。

如果磁盘的容量较大，必要时可以把UNIX分区划分为若干分片（slice），在每个分片中创建一个文件系统，然后再把每个文件系统安装到UNIX系统中。一个UNIX分区究竟支持多少分片，不同厂家的UNIX系统也不一致，但其总的范围约在8至16之间。这里需要说明一点，当存储设备只有一个UNIX分区时，在不至于引起混淆的情况下，我们也把UNIX分区中的分片称做分区。

最后，还需要执行下列步骤，才能以文件系统的形式使用磁盘等存储设备。

1. 利用mkfs或newfs等命令在指定的UNIX磁盘分区中创建文件系统；
2. 利用labelit命令，对新建的文件系统以及卷标进行命名（这一步是选用的）；
3. 利用mount等命令把新建的文件系统安装到系统的指定目录位置。

注意：磁盘设备如何格式化、分区以及分片，不同UNIX系统的实现方法并不完全一样。因此，本章不打算详细讨论这方面的内容，建议读者查阅相关厂家提供的手册或文档。

17.1 创建文件系统

一个磁盘分区既可以用做原始设备，如swap交换空间，也可以用做文件系统。在实际应用过程中，当需要增加新盘或改变现有的磁盘分区结构时，有可能需要用户自己手工创建文件系统。

在创建文件系统时，主要有两种方法：一是直接使用最基本的mkfs命令，在选定的硬盘分区中创建文件系统；二是利用专门工具，如newfs等，在选定的硬盘分区上创建文件系统。

17.1.1 使用mkfs命令创建UFS文件系统

1. mkfs命令介绍

mkfs命令可用于直接创建UFS文件系统，其语法格式简写如下：

```
mkfs -F ufs [-mV] [-o opt1=n,opt2=n,.....optN=n] device size
```

表17-1给出了mkfs命令的选项及说明。

此外，还可以采用由一系列数值组成的参数表作为mkfs命令的参数。参数值之间由空格分隔，每个数值的意义由其所在的位置确定。在此情况下，“-o”选项及关键字均可忽略，整个参数表只需直接写在设备分区容量参数的后面，参见后续mkfs与newfs命令的介绍。

表17-1 mkfs命令选项

选项	简单说明
-F ufs	表示创建一个UFS类型的文件系统
-m	用于显示创建文件系统时可用的命令行参数，实际上并不真正地创建文件系统。尤其是，使用“-m”选项可以获取命令行参数需要的size等数据
-V	显示当前输入的mkfs命令行及其参数
-o	用于指定UFS文件系统特定的选项。注意，多个选项之间应加逗号分隔符，中间没有空格
nsect=n	磁盘设备中每个磁道包含的扇区数量。默认值为32
ntrack=n	磁盘设备中每个柱面包含的磁道数量。默认值为16
bsize=n	文件系统逻辑数据块的大小（以字节为单位）。有效的逻辑数据块是4096个字节或8192个字节，默认值为8192个字节
fragsize=n	文件系统逻辑数据片的大小（以字节为单位）。这个数值确定了能够分配给文件的最小磁盘存储空间。数据片必须是2的N次方，且bsize/fragsize等于1、2、4或8。这意味着，如果逻辑数据块是4096，则合法的数据片为512、1024、2048和4096。当逻辑数据块为8192时，合法的数据片为1024、2048、4096和8192。默认值是1024
cgsize=n	每个柱面组包含的柱面数量。其有效值或默认值总是位于16和256之间
free=n	用于维护文件系统而保留的最小空闲空间的百分比。当达到这个数值时，只有超级用户才能继续请求分配磁盘空间。取决于磁盘的容量，其默认值为1%~10%
rps=n	磁盘的旋转速度（即每秒多少转）。默认值为60。rps等价于newfs命令中rpm参数
nbpi=n	指定信息节点的字节数。这个数值反映了文件系统平均文件大小。如果希望生成的信息节点数量较少（也即希望创建大型文件），应把这个参数设置成一个较大的数值。为了创建较多数量的信息节点，应当把这个参数设置成一个较小的数值。默认值为2048
opt=s t	指定文件系统的组织原则。文件系统的组织可以按花费时间最少为原则分配数据块，也可以按磁盘空间碎片最小为原则组织文件系统。默认值是按时间优化文件系统
apc=n	指定每个柱面应保留的替换扇区数量，默认值为0。这种情况仅适用于SCSI磁盘设备的坏块替换
gap=n	旋转延迟。不管输入数值如何，这个值总是设置为0
nrpos=n	用于划分柱面组的不同旋转位置。默认值为8
maxcontig=n	一个文件可连续分配的最大逻辑数据块数量。默认值的计算公式如下： maxcontig=磁盘驱动器传输的最大字节数量/磁盘数据块的大小
device	块或字符特殊文件名，表示准备创建文件系统的磁盘分区，如/dev/dsk/c0t0d0s6
size	指定整个磁盘分区的大小，即物理数据块（512个字节）的数量，也是文件系统的容量。另外，这个参数必须是位于磁盘分区设备文件名后面的第一个参数。默认值为整个磁盘分区的扇区数量

2. 确定设备分区的容量

为了创建一个文件系统，不管采用什么方式和方法，其中最核心、最基本的工具仍然是mkfs命令。但是，使用mkfs命令需要用户提供一系列参数，而针对不同型号、不同容量的磁盘设备，其参数也不尽相同。另外，即使采用默认值，至少还有两个参数仍然必须由用户提供：一是磁盘分区的设备文件名；另外一个为磁盘分区的容量。尤其是第二个参数，可能让人感到无从下手。

因此，在使用mkfs命令创建文件系统时，除了需要知道磁盘分区的设备文件名之外，关键是如何确定磁盘分区的大小。其实，这个数字可以从两个方面得到：一是利用prtvtoc等命令查询磁盘的UNIX分区表；二是利用mkfs命令本身或newfs命令给出的磁盘分区容量（及其他参数）。

（1）使用prtvtoc命令获取设备分区的容量。

UNIX系统采用卷标目录（VTOC）维护磁盘设备的分区与文件系统信息。为了查询磁盘分区信息，可以使用prtvtoc命令，其语法格式如下：

```
prtvtoc [-fhs] [-t vfstab] [-m mnttab] device
```

其中，“-f”选项用于显示磁盘的空闲存储空间，其中包括空闲存储空间的起始数据块地址，数据块的数量，以及磁盘设备中尚未使用的分区。“-h”选项表示禁止输出任何说明信息及磁盘分区的标题，如每个扇区有多少字节，每个磁道有多少扇区等信息。“-s”选项用于禁止输出除磁盘分区标题之外的其他说明信息。“-m”选项表示使用指定的文件替代文件系统安装表/etc/mnttab文件。“-t”选项表示使用指定的文件替代默认的/etc/vfstab文件。

例如，为了获取一个磁盘分区的容量，可以使用下列prtvtoc命令：

```
# prtvtoc -s /dev/rdisk/c1t3d0s6
*
*                               First   Sector      Last
* Partition  Tag  Flags   Sector      Count        Sector    Mount Directory
      0        2    00         0      264576      264575
      1        3    01      264576      264576      529151
      2        5    01         0  143349312  143349311
      6        4    00     529152  142820160  143349311
#
```

从上述输出信息中可以得知，磁盘分区1和2均拥有264 576个扇区，磁盘分区6拥有142 820 160个扇区，整个磁盘的容量为143 349 312个扇区（在Solaris系统中，分区2表示整个磁盘，在传统的System V UNIX系统中，通常以分区0表示整个磁盘）。

（2）使用mkfs命令的“-m”选项获取设备分区的容量。

利用mkfs命令的“-m”选项，可以获取磁盘分区中已有文件系统的参数，尤其是磁盘分区的容量。注意，这个命令实际上并不创建文件系统。

```
# mkfs -m /dev/rdisk/c1t3d0s6
mkfs -F ufs -o nsect=424,ntrack=24,bsize=8192,fragsize=1024,
cgsiz=10,free=1,rps=167,nbpi=8275,opt=t,apc=0,gap=0,nrpos=8,maxcontig=128 /dev/dsk/
c1t3d0s6 142820160
#
```

但是，对于一个刚刚做过格式化，尚未创建任何文件系统的磁盘分区，上述命令根本无法执行。例如，假定/dev/rdisk/c1t4d0s6是一个新划分的磁盘分区，使用mkfs命令的输出结果如下：

```
# mkfs -m /dev/rdisk/c1t4d0s6
[not currently a valid file system - bad superblock]
Arithmetic Exception - core dumped
#
```

（3）使用mkfs命令的“-o N”选项获取设备分区的有关参数。

另外，还可以利用mkfs命令的“-o N”选项，获取磁盘分区的有关参数信息。与“-m”

选项不同的是，这个选项并不要求磁盘分区已经事先创建过文件系统。但是，使用这个命令时必须提供磁盘分区的容量，而我们需要的恰恰就是磁盘分区的容量。注意，这个命令也不创建文件系统。

```
# mkfs -o N /dev/rdisk/c0d0s7 20478528
Warning: 448 sector(s) in last cylinder unallocated
/dev/rdisk/c0d0s7: 20478528 sectors in 39998 cylinders of 16 tracks, 32 setors
9999.3MB in 2500 cyl groups (16 c/g, 4.00MB/g, 1920 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
32, 8256, 16480, 24704, 32928, 41152, 49376, 57600, 65824, 74048,
Initializing cylinder groups:
.....
super-block backups for last 10 cylinder groups at:
20398432, 20406656, 20414880, 20423104, 20431328, 20439552, 20447264,
20455488, 20463712, 20471936
#
```

如果未指定磁盘分区的容量参数，mkfs命令将会给出下列错误信息，同时也会给出语法格式及部分建议的参数值：

```
# mkfs -o N /dev/rdisk/c1t3d0s6
size not specified
ufs usage: mkfs [-F FSType] [-V] [-m] [-o options] special size(sectors) \
[nsect ntrack bsize fragsize cpg free rps nbpi opt apc gap nrpos maxcontig]
-m : dump fs cmd line used to make this partition
-V :print this command line and return
-o :ufs options: :nsect=32,ntrack=16,bsize=8192,fragsize=1024
-o :ufs options: :cgsiz=0,free=0,rps=60,nbpi=2048,opt=t
-o :ufs options: :apc=0,gap=0,nrpos=0,maxcontig=0
NOTE that all -o suboptions: must be separated only by commas so as to
be parsed as a single argument
#
```

(4) 使用newfs命令获取设备分区的容量。

当然，我们还可以利用后面将要介绍的newfs命令，获取使用mkfs命令时需要提供的参数数据。例如：

```
# newfs -Nv /dev/rdisk/c0d0s7
mkfs -F ufs -o N /dev/rdisk/c0d0s7 20478528 63 32 8192 1024 264 1 60 8192 t 0
-18 7 n
/dev/rdisk/c0d0s7: 20478528 sectors in 10158 cylinders of 32 tracks, 63 setors
9999.3MB in 208 cyl groups (49 c/g, 48.23MB/g, 6080 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
32, 98880, 197728, 296576, 395424, 494272, 593120, 691968, 790816, 889664,
19559648, 19658496, 19757344, 19856192, 19955040, 20053888, 20152736,
20251584, 20350432, 20449280
#
```

3. 使用mkfs命令创建文件系统

现在，假定一切准备工作就绪，可以开始使用mkfs命令创建文件系统了。

下面的例子说明怎样利用mkfs命令，以及通过上述各种方法获取的磁盘分区容量，在给定

的磁盘分区中创建一个UFS文件系统。其中, 由于没有明确指定, 逻辑数据块将采用默认值8KB, 逻辑数据片采用默认值1KB。执行mkfs命令之后, 创建的文件系统具有一个根目录和一个lost+found子目录:

```
# mkfs -F ufs /dev/rdisk/c0d0s7 20478528
/dev/rdisk/c0d0s7: 20478528 sectors in 10158 cylinders of 32 tracks,
63 sectors 9999.3MB in 208 cyl groups (49 c/g, 48.23MB/g, 6080 i/g)
super-block backups (fsck -F ufs -o b=#)
32, 98880, 197728, 296576, 395424, 494272, 593120, 691968, 790816, 889664,
19559648, 19658496, 19757344, 19856192, 19955040, 20053888, 20152736,
20251584, 20350432, 20449280
#
```

下面的例子说明怎样利用mkfs命令, 以及newfs命令给出的每一个参数, 在给定的磁盘分区中创建一个UFS文件系统:

```
# mkfs -F ufs -o nsect=424,ntrack=24,bsize=8192,fragsize=1024,\
cgsiz=203,free=1,rps=167,nbpi=8192,opt=t,apc=0,gap=-1,\
nrpos=8,maxcontig=128 /dev/rdisk/clt3d0s6 142820160
/dev/rdisk/clt3d0s6: 142820160 sectors in 14035 cylinders of 24 tracks, 424
sectors
69736.4MB in 1404 cyl groups (10 c/g, 49.69MB/g, 6016 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
32, 102224, 204416, 306608, 408800, 510992, 613184, 715376, 817568, 919760,
.....
```

其中, 逻辑数据块为8KB, 逻辑数据片为1KB。当空闲存储空间达到1%时, 只有超级用户才能继续申请磁盘空间, 扩充文件或创建新的文件。执行mkfs命令之后, 创建的文件系统具有一个根目录和一个lost+found子目录。

另外一种方法是按照后面即将介绍的新fs命令给出的mkfs命令行(包括一系列数值参数), 直接运行此命令即可创建一个UFS文件系统。注意, 下列参数必须严格按照“newfs -Nv /dev/rdisk/clt3d0s6”命令给出的数据顺序依次列出:

```
# mkfs -F ufs -o N /dev/rdisk/clt3d0s6 142820160 424 24 \
8192 1024 203 1 167 8192 t 0 -1 8 128
/dev/rdisk/clt3d0s6: 142820160 sectors in 14035 cylinders of 24 tracks, 424
sectors
69736.4MB in 1404 cyl groups (10 c/g, 49.69MB/g, 6016 i/g)
super-block backups (for fsck -F ufs -o b=#) at:
32, 102224, 204416, 306608, 408800, 510992, 613184, 715376, 817568, 919760,
.....
```

17.1.2 使用newfs命令创建文件系统

在创建UFS文件系统时, newfs命令是一种更常用的工具。实际上, newfs命令只是mkfs命令的一个外包装, 真正负责创建文件系统的仍然是mkfs命令。newfs命令能够读取磁盘分区表中的信息, 如每个柱面有多少磁道, 每个磁道有多少扇区等, 再根据用户提供的参数, 经过适当的计算之后, 传递给mkfs命令, 由mkfs命令执行实际的文件系统创建任务。

从实际应用来讲, 由于不必提供使用mkfs命令创建文件系统时必须提供的磁盘分区容量参

数，newfs是一个比mkfs更好用的工具。newfs可以自己计算应当使用什么参数合适，然后再利用计算的结果调用mkfs命令，完成文件系统的创建。如果以交互方式运行，newfs命令还会在开始创建文件系统之前，提示用户确认下一步的处理动作。newfs命令的语法格式简写如下：

```
newfs [-Nv] [mkfs-options] device
```

其中，“-N”选项表示仅仅输出创建文件系统时将会采用的参数，而不是立即创建文件系统。“-v”选项用于显示newfs执行的处理动作，其中包括传递给mkfs命令的参数。device是准备磁盘分区的字符特殊文件名。

表17-2给出了mkfs命令特定的选项，这些选项可用于强制替代默认的参数（有关mkfs命令选项的详细说明参见表17-1）。

表17-2 mkfs命令特定的选项

选项	简单说明
-b bsize	指定文件系统逻辑数据块的大小（以字节为单位）
-c cgsiz	指定每个柱面组包含的柱面数量
-f fragsize	指定文件系统逻辑数据片的大小（以字节为单位）
-o space time	指定文件系统的组织原则
-s size	指定文件系统的全部容量（以扇区或物理数据块为单位）
-t ntrack	磁盘每个柱面的磁道数量

下面的例子只是说明怎样利用newfs命令，获取在一个磁盘分区中创建文件系统时需要的各种参数。注意，这个命令实际上并非真的创建文件系统。

```
# newfs -Nv /dev/rdisk/clt3d0s6
mkfs -F ufs -o N /dev/rdisk/clt3d0s6 142820160 424 24 8192 1024 203 1 167 8192
t 0 -l 8 128 /dev/rdisk/clt3d0s6: 142820160 sectors in 14035 cylinders of
24 tracks, 424 sectors 69736.4MB in 1404 cyl groups (10 c/g, 49.69MB/g, 6016 i/g)
super-block backups (fsck -F ufs -o b=#)
32, 102224, 204416, 306608, 408800, 510992, 613184, 715376, 817568, 919760,
.....
```

现在，我们就利用newfs命令，在一个73GB的磁盘分区上创建UFS文件系统。创建文件系统时采用的具体参数如上述“newfs -Nv /dev/rdisk/clt3d0s6”命令的输出完全一致。

```
# newfs /dev/rdisk/clt3d0s6
newfs: construct a new file system /dev/rdisk/clt3d0s6: (y/n)? y
/dev/rdisk/clt3d0s6: 142820160 sectors in 14035 cylinders of 24 tracks, 424
sectors 69736.4MB in 1404 cyl groups (10 c/g, 49.69MB/g, 6016 i/g)
super-block backups (fsck -F ufs -o b=#)
32, 102224, 204416, 306608, 408800, 510992, 613184, 715376, 817568, 919760,
.....
```

注意：除非是在软盘上创建UFS文件系统，否则只有超级用户才能使用newfs命令。

17.2 使用labelit命令命名文件系统

使用mkfs和newfs命令创建文件系统时，并未设定文件系统的名字和卷标。从下列命令的输出信息中可以确认这一事实：

```
# labelit /dev/dsk/c0d0s7
fsname:
volume:
#
```

labelit命令的主要功能是显示和命名文件系统的名字和卷标，其语法格式简写如下：

```
labelit [-F ufs] [options] device [fsname volume]
```

其中，device是磁盘分区的设备文件名，如/dev/dsk/c0d0s7等。fsname表示文件系统的名字或安装点，如/var等。volume是文件系统的卷标。如果未指定fsname和volume，labelit命令将会输出之前设定的文件系统名与卷标。注意，在指定文件系统名与卷标时，不能超过6个字符。例如：

```
# labelit -F ufs /dev/dsk/c1t3d0s6 oracle datav1
fsname: oracle
volume: datav1
# labelit /dev/dsk/c1t3d0s6
fsname: oracle
volume: datav1
#
```

注意：设置文件系统的名字与卷标并非是必须执行的处理动作，有的UNIX系统并不要求执行这一步骤。

17.3 安装、卸载文件系统

这一节介绍怎样在UNIX系统中安装和卸载文件系统。

17.3.1 安装文件系统

在成功地创建了文件系统之后，需要把新建文件系统安装到UNIX文件系统目录层次结构中的任何一个安装点上，然后才能开始使用新建的文件系统。

当需要手工安装文件系统或NFS远程资源时，可以使用mount或mountall命令。其中，mount是最基本的命令，mountall是基于mount命令实现的。当需要同时安装多个文件系统时，可以使用mountall命令，安装/etc/vfstab文件中列举的，其“mount at boot”字段为yes的所有文件系统（当进入多用户模式时，系统将会自动运行mountall命令）。mount命令的基本语法格式如下：

```
mount [-F fstype] [generic-opts] [-o specific-opts] [-O] device mount-point
```

其中，“-F”选项用于指定文件系统的类型，如ufs等。针对不同类型的文件系统，可以使用“-o”选项提供文件系统特定的选项。以UFS文件系统为例，表17-3给出了使用“-o”选项时常用的命令选项。如果需要同时指定多个命令选项，可以使用逗号“,”分隔每个选项（注意，

选项中间不能有空格），如 “-o ro,nosuid”。

表17-3 常用的“-o”命令选项

-o命令选项	简单说明
atime noatime	允许或禁止对文件的访问时间进行更新，除非确实是由于用户修改文件而引起的文件状态变化。对于文件修改时间并不重要的情况，这个选项将会减少文件系统的磁盘处理活动。默认值为atime
intr nointr	指定是否允许使用键盘的中断键中止一个因等待安装文件系统的响应而挂起的进程。默认值是intr，即允许中断
logging nologging	启用或禁用文件系统的日志功能。UFS文件系统的日志处理过程是在执行实际的文件系统更新之前先把文件处理的全部操作步骤写入日志文件中。日志功能有助于保证UFS文件系统的完整性，避免由于系统瘫痪导致文件系统损坏，以至于需要使用fsck命令检测和修复文件系统，影响系统的引导过程。日志的空间分配取自文件系统的空闲数据块，通常，每1GB容量的文件系统约需保留1MB的日志空间，最多不超过64MB。默认值是logging
quota	启用文件系统的限额控制
remount	重新安装一个已安装的文件系统。这个选项的主要目的或常见用法是改变已安装文件系统的安装标志，特别是把一个以只读方式安装的文件系统重新安装为可读写的文件系统。注意，使用这个选项时既不能更换设备文件，也不能改变安装点
ro rw	以只读（ro）或读写（rw）方式安装文件系统。默认值为rw。CD/DVD（ISO 9660）文件系统的默认值为ro
rq	以读写方式安装文件系统，同时启用文件系统的限额控制，相当于同时选用“rw,quota”两个安装选项

在使用mount或mountall命令时应注意下列事项：

- 除非安装软盘或USB移动磁盘等文件系统，在使用mount命令安装文件系统时，必须具有超级用户的身份（必要时还需要使用mkdir命令，事先创建适当的安装点）。
- mount和mountall命令不能安装已经受损但又需要以读写方式安装的文件系统。如果在运行mount或mountall命令时出现错误信息，通常需要使用fsck命令检测与修复文件系统。
- 使用remount选项，可以把以只读方式安装的文件系统重新安装为读写方式的文件系统。但是，不能把以读写方式安装的文件系统重新安装为只读方式的文件系统。

17.3.2 /etc/vfstab文件

在UNIX操作系统中，/etc/vfstab是一个重要的系统文件。vfstab文件的主要功能是维护系统中需要安装的默认文件系统。如果需要在系统引导过程中自动安装，必须把文件系统加到vfstab文件中。vfstab文件也可用于指定交换（swap）分区，以便在系统引导过程中使用/sbin/swap命令自动增加系统交换区。

/etc/vfstab文件的格式如下，其中每一行包含7个字段（表17-4给出了每个字段及其说明）：

device	device	mount	FS	fsck	mount	mount
to mount	to fsck	point	type	pass	at boot	options

表17-4 /etc/vfstab文件中的每个字段及其说明

字段	简单说明
device to mount	这个字段表示准备安装的设备，其设备文件名可以是下列之一： <ul style="list-style-type: none"> • 本地UFS文件系统的块设备文件名（如/dev/dsk/c1t0d0s6） • 远程文件系统的资源名（如iscas:/export/home） • 交换分区的块设备文件名（如/dev/dsk/c0t3d0s1） • 虚拟文件系统的某个目录
device to fsck	相应于块设备的原始设备文件名（字符特殊文件），如/dev/rdisk/c1t0d0s6。这个字段确定了fsck命令使用的原始设备接口。“-”表示没有相应的原始设备，无需检测相应的文件系统，如只读文件系统或远程文件系统等
mount point	表示把文件系统安装到指定的目录位置（如“/”）
FS type	标识文件系统的类型
fsck pass	fsck命令用以决定是否需要检测文件系统，以及如何检测文件系统。如果“device to fsck”与此字段均为“-”，表示不需要自动检测相应的文件系统。如果“device to fsck”字段存在，但此字段为“-”，表示在安装之前需要检测文件系统是否能够安装，如果无法正常安装，将会在系统引导过程中显示一条警告信息，说明文件系统有误，但fsck不会自动修复文件系统。也就是说，系统管理员必须自行修复文件系统，然后才能安装文件系统。如果这个字段是一个大于0的数值，则意味着总是需要检测文件系统，其中1表示需要按照文件系统在vfstab文件中出现的顺序，依次进行检测。如果这个字段的数值大于1，意味着可以使用“fsck -o p”命令并行检测不同磁盘分区中的文件系统，从而提高检测与修复文件系统的效率
mount at boot	这个字段可设置为yes或no，表示在引导系统时，是否应当使用mountall命令自动安装相应的文件系统。注意，“/”、/usr和/var文件系统，以及虚拟文件系统（如/proc）并非是根据vfstab文件安装的，因此，这个字段应当总是为no
mount options	安装文件系统时使用的一系列以逗号“,”分隔的mount命令选项（中间没有空格）。减号“-”表示安装选项字段为空。参见mount命令选项的说明

在/etc/vfstab文件中，每个字段之间必须使用空白字符（空格或制表字符）隔开；每个字段都不能空缺。如果某个字段确实为空，必须确保在相应的字段位置加一个减号“-”。否则，系统可能无法成功地启动。另外，每个字段的值中也不能有空格。如果需要，可为准备安装的文件系统创建一个安装点（安装点是一个目录。为了安装一个文件系统，本地系统上必须有一个安装点）。

例如，下面是一个取自Solaris UNIX系统中的vfstab文件实例：

```
$ cat /etc/vfstab
#device      device      mount      FS      fsck  mount  mount
#to mount    to fsck     point      type    pass  at     boot options
#
fd           -           /dev/fd    fd      -      no     -
/proc       -           /proc      proc    -      no     -
/dev/dsk/c0d0s1 -          -          swap    -      no     -
/dev/dsk/c0d0s0 /dev/rdisk/c0d0s0 /          ufs     1      no     -
/dev/dsk/c0d0s7 /dev/rdisk/c0d0s7 /export/home ufs     2      yes    -
.....
$
```

17.3.3 安装文件系统

这一节主要介绍怎样使用mount命令手工安装文件系统，或通过/etc/vfstab文件，自动安装文件系统。

1. 确定系统中已安装了哪些文件系统

为了确定系统当前已安装了哪些文件系统，可以使用下列mount命令：

```
# mount [-v]
```

其中的“-v”选项意味着显示已安装文件系统的详细信息。

下面的例子说明了怎样利用mount命令显示当前已安装的文件系统及有关信息：

```
# mount | grep "/dev/dsk"
/ on /dev/dsk/c0d0s0 read/write/setuid/devices/intr/largefiles/logging/
xattr/onerror=panic /dev=1980000 on Sun Aug 23 00:18:01 2009
/export/home on /dev/dsk/c0d0s7 read/write/setuid/devices/intr/largefiles/
logging/xattr/ onerror=panic/dev=1980007 on Sun Aug 23 00:18:28 2009
#
```

另外，还可以使用cat或more等命令，查阅文件系统安装表/etc/mnttab，获取文件系统的安装信息，确定系统中当前已经安装了哪些文件系统。无论何时安装或卸载文件系统，系统都会自动修改文件系统安装表。因此，/etc/mnttab文件总是能够反映当前最新的文件系统安装信息（注意，不要直接修改这个文件）。例如：

```
$ cat /etc/mnttab
/dev/dsk/c0d0s0 /          ufs      rw,intr,largefiles,logging,xattr,...
.....
/dev/dsk/c0d0s7 /export/home  ufs      rw,intr,largefiles,logging,xattr,...
.....
$
```

2. 文件系统的自动安装

如果想在系统启动过程中自动安装文件系统，必须编辑/etc/vfstab文件，增加文件系统安装项，把相应的文件系统加到/etc/vfstab文件中，同时确保其“mount at boot”字段为yes。

下面的例子说明怎样修改/etc/vfstab文件，以便在启动系统时能够自动地把磁盘分区/dev/dsk/c1d0s6中的文件系统安装到安装点/backup处。其中，“device to fsck”字段的原始设备名为/dev/rdisk/c1d0s6，“fsck pass”字段为2，意味着可以并行检测与修复文件系统。“mount at boot”字段为yes，这意味着在系统引导过程自动安装指定的文件系统。修改后的vfstab文件如下，其中最后一行即新增加的安装项：

```
$ cat /etc/vfstab
#device          device          mount          FS          fsck  mount  mount
#to mount        to fsck         point          type        pass  at boot options
#
fd               -               /dev/fd        fd          -      no      -
/proc            -               /proc          proc        -      no      -
/dev/dsk/c0d0s1  -               -              swap       -      no      -
/dev/dsk/c0d0s0  /dev/rdisk/c0d0s0 /              ufs        1      no      -
```

```

/dev/dsk/c0d0s7    /dev/rdisk/c0d0s7  /export/home  ufs      2      yes    -
.....
/dev/dsk/c1d0s6    /dev/rdisk/c1d0s6  /backup      ufs      2      yes    -
$

```

3. 使用mount命令安装文件系统

在安装文件系统时，可以使用下列mount命令：

```
# mount [-F fstype] [-o mount-opts] device-name mount-point
```

其中，“-F”选项用于指定文件系统的类型。在Solaris等UNIX系统中，如果安装的是UFS文件系统，“-F”选项可以省略。“-o”选项用于指定安装文件系统时使用的各种安装选项，可参见表17-3中的相关说明。device-name用于指定文件系统的设备文件名，如/dev/dsk/c2d0s6。mount-point用于指定安装文件系统的目录位置，也即安装点。

下面的例子说明了怎样利用mount命令，把磁盘分区/dev/dsk/c2d0s6中的UFS文件系统安装到/filsys1目录上：

```
# mount -F ufs /dev/dsk/c2d0s6 /filsys1
#
```

4. 安装PCFS文件系统

Solaris UNIX系统支持pcfs文件系统（即Windows系统的FAT16/FAT32文件系统），如果需要在两个系统之间交换数据，可以使用下列mount命令，把Windows系统的逻辑盘作为pcfs文件系统直接安装到UNIX系统中：

```
# mount -F pcfs device-name:logical-drive mount-point
```

其中，device-name是Windows系统磁盘的设备文件名，如/dev/dsk/c1d0p0。logical-drive可以是逻辑盘c~z，也可以是设备编号1~24（相当于逻辑盘c~z）。注意，device-name与logical-drive之间必须加冒号“:”分隔符。

下面的例子说明了怎样把Windows系统的逻辑盘C安装到/mnt目录中：

```
# mount -F pcfs /dev/dsk/c1d0p0:c /mnt
#
```

5. 使用mount命令安装NFS文件系统

详见第14章“NFS网络文件系统”。

17.3.4 卸载文件系统

卸载文件系统意味把文件系统从安装点移走，删除/etc/mntab文件中的安装项。在文件系统的管理与维护任务中，部分处理任务只能在未安装的文件系统中执行。此外，当临时安装的文件系统不再需要时，应及时卸载。因此，当出现下列情况时，应考虑卸载文件系统：

- 文件系统用过之后已不再需要；
- 文件系统受损，因而需要使用fsck命令检测与修复文件系统；
- 为了增加卷标，因而需要使用labelit命令调整文件系统；
- 误删文件，因而需要使用fsdb命令调试文件系统；

• 在执行完整的文件系统备份之前，通常也需要卸载文件系统。

卸载文件系统（或NFS远程资源）时，可以使用umount或umountall命令。同样，umount是最基本的命令，umountall是基于umount命令实现的。当需要同时卸载多个文件系统时，可以使用mountall命令卸载/etc/vfstab文件中列举的，其“mount at boot”字段为yes的所有文件系统。umount命令的语法格式简写如下：

```
umount [-fV] device | directory
umount -a [-fV] [directory]
```

其中，“-f”选项表示强制卸载指定的文件系统；“-a”选项表示卸载/etc/vfstab文件中列举的，其“mount at boot”字段为yes的所有文件系统，其功能相当于直接运行umountall命令；device是文件系统的设备文件名或NFS远程资源名等；directory是准备卸载的文件系统安装点。

通常，umount或umountall命令不能卸载当前正在使用的文件系统，例如，用户正在访问文件系统中的文件或目录，打开了文件系统中的文件；文件系统正在共享等。但在紧急情况下，也可以使用“umount -f”命令强制卸载一个尚在工作的文件系统。除非不得已，正常情况下一般不建议采取这种卸载方式，因为这种强制卸载会致使已打开的文件丢失数据。另外，“-f”选项仅适用于UFS和NFS文件系统。

注意：作为系统关机过程的一部分，文件系统将会自动卸载。

1. 卸载文件系统前的准备工作

在卸载文件系统之前，通常需要做一定的准备工作，或具备一定的条件，其中包括：

- (1) 通常情况下，只有超级用户才能卸载文件系统。
- (2) 文件系统必须处于空闲状态才能卸载。通常，不能卸载一个尚处于工作状态的文件系统。所谓文件系统处于工作状态，意味着可能还有用户正处于或正在访问文件系统中的某个目录或文件，用户或进程打开了文件系统中的某个文件，或者文件系统正处于共享状态。为了确保文件系统能够正常卸载，可以采取下列措施：
 - (3) 改换到不同文件系统中的目录；
 - (4) 退出UNIX系统；
 - (5) 使用fuser命令找出访问文件系统的所有进程，然后通知用户停止使用准备卸载的文件系统，或由超级用户停止运行相应的进程（如果必要或可能）；
 - (6) 取消文件系统的共享。参见第14章“NFS网络文件系统”。

2. 终止正在访问文件系统的所有进程

如果想要终止正在访问文件系统的所有进程，首先应使用下列命令，找出当前正在访问文件系统的所有进程，以便了解需要终止运行哪些进程。

```
# fuser -c [ -u ] mount-point
```

其中，“-c”选项表示显示当前正在访问指定安装点（文件系统）中任何文件的所有进程。“-u”选项意味着在列出的每个进程ID后面给出与进程相关的注册用户名。“mount-point”表示需要检查的安装点（文件系统）。

然后，可以使用下列命令，终止运行当前正在访问文件系统的所有进程：

```
# fuser -c -k mount-point
```

在执行上述命令时，一个SIGKILL信号将会发送到当前正在使用指定文件系统的每个进程。

注意，在事先没有通知用户时，通常不应强行终止用户进程。

最后，可以使用下列命令，验证当前是否还有任何进程仍在访问文件系统：

```
# fuser -c mount-point
```

下列例子说明了怎样停止一个正在访问/export/home文件系统的进程4006：

```
# fuser -c /export/home
/export/home: 4006
# fuser -c -k /export/home
/export/home: 4006
# fuser -c /export/home
/export/home:
#
```

3. 卸载文件系统

除了“/”文件系统，以及单独的/usr或/var文件系统之外，为了卸载其他非虚拟的文件系统，首先应确保已经完成文件系统卸载前的准备工作，然后可以使用umount命令卸载文件系统。注意，仅当关闭系统时才能卸载“/”文件系统，以及单独的/usr等文件系统，否则系统无法正常运行。

下面的例子说明了怎样卸载一个安装在/export/home目录处的本地文件系统：

```
# umount /export/home
#
```

下面的例子说明了怎样卸载一个位于磁盘分区7中的文件系统：

```
# umount /dev/dsk/c0t0d0s7
#
```

下面的例子说明了怎样卸载/etc/vfstab文件中列举的，其“mount at boot”字段为yes的所有文件系统：

```
# umountall
#
```

注意：上述命令并不能卸载当前正处于工作状态的文件系统。在紧急情况下，可以使用“umount -f”命令强制卸载一个尚在工作的文件系统。除非不得已，通常不建议采取这种卸载方式，因为这种强制卸载有可能会造成已打开的文件丢失数据。下面的例子说明了怎样强制卸载位于安装点/mnt处的文件系统：

```
# umount -f /mnt
#
```

17.4 确定文件系统的类型

我们知道，利用虚拟文件系统，UNIX系统能够同时支持多个文件系统类型，不同的文件系统能够同时安装在系统中。对于用户而言，使用不同的文件系统并没有任何不同，可以采用统一的方法访问文件系统及其中的文件。但是，为了安装和维护一个文件系统，首先需要知道设备分区的文件系统类型。在确定一个设备分区的文件系统类型时，可以查阅/etc/vfstab文件，

检测相应设备名对应的“FS type”字段，即可确定一个设备分区的文件系统类型。对于未安装的设备，还可以利用fstyp命令，检测设备分区的文件系统类型。

fstyp命令用于确定安装或未安装的文件系统类型。UNIX系统提供了若干文件系统类型检测模块，当输入fstyp命令之后，系统将会采用探索法，利用每个文件系统检测模块确定给定的设备的文件系统类型。fstyp命令的语法格式如下：

```
fstyp [-v] device
```

其中，“-v”选项表示输出文件系统超级块的有关信息。device为设备分区的文件名，如/dev/rdisk/c0t2d0s6。

例如，使用下列命令可以确定磁盘分区/dev/dsk/c1t3d0s6为UFS文件系统。

```
# fstyp /dev/dsk/c1t3d0s6
ufs
#
```

如果增加“-v”选项，fstyp命令不仅能够确定文件系统的类型，还可以给出文件系统的超级块信息：

```
# fstyp -v /dev/dsk/c1t3d0s6
ufs
magic 11954 format dynamic time Fri Mar 17 18:44:42 2006
sblkno 16 cblkno 24 iblkno 32 dbkno 784
sbsize 2048 cgsiz 8192 cgoffset 216 cgmask 0xffffffffe0
ncg 1404 size 71410080 blocks 70331770
bsize 8192 shift 13 mask 0xfffffe000
fsiz 1024 shift 10 mask 0xfffffc000
frag 8 shift 3 fsbtodb 1
minfree 1% maxbpg 2048 optim time
maxcontig 128 rotelay 0ms rps 167
csaddr 784 cssiz 22528 shift 9 mask 0xfffffe00
ntrak 24 nsect 424 spc 10176 ncyl 14035
cpg 10 bpg 6360 fpg 50880 ipg 6016
nindir 2048 inopb 64 nspf 2
nbfree 8791469 ndir 2 nifree 8446460 nffree 9
cgrotor 0 fmod 0 ronly 0 logbno 0
fs_reclaim is not set
file system state is valid, fsclean is 1
.....
```

上述信息的第一行表示指定的磁盘分区是一个UFS文件系统。与使用mkfs命令创建文件系统有关的其他信息如表17-5所示。根据fstyp命令的输出信息，用户可以对文件系统有一个大概的了解。另外，也可以据此对创建的文件系统进行验证。参见第16章“文件系统内部组织”。

表17-5 fstyp命令输出的部分数据

字	段值	简单说明
sblkno	16	文件系统中超级块的地址（数据块号16）
cblkno	24	柱面组信息块的偏移地址
iblkno	32	信息节点区第一个信息节点的偏移地址

(续表)

字	段值	简单说明
dblkno	784	数据存储区第一个数据块的偏移地址
sbsize	2048	超级块的实际大小
cgsize	8192	柱面组的大小
ncg	1404	柱面组的数量
size	71410080	文件系统中1KB数据块的数量 ($\text{spc} \times \text{ncyl} / 2 = 71410080$)
blocks	70331770	文件系统数据存储区中1KB数据块的数量
bsize	8192	逻辑数据块的大小
fsize	1024	逻辑数据片的大小
frag	8	每个逻辑数据块包含的逻辑数据片数量
minfree	1%	最少必须保留的空闲数据块的百分比
maxbpg	2048	每个柱面组中的最大数据块数量
optim	time	创建文件系统时采用的优化原则
maxcontig	128	分配给一个文件的最大连续数据块的数量
rps	167	以秒为单位的磁盘转速
csaddr	784	柱面组汇总信息区中第一个数据块的地址
cssize	22528	柱面组汇总信息区的大小
ntrak	24	每个柱面拥有的磁道数量
nsect	424	每个磁道拥有的扇区数量
spc	10176	每个柱面拥有的扇区数量 ($\text{nsect} \times \text{ntrak} = 10176$)
ncyl	14035	文件系统拥有的柱面数 ($\text{spc} \times \text{ncyl} = 142820160$)
cpg	10	每个柱面组中的柱面数量
bpg	6360	每个柱面组中的数据块数量
fpg	50880	每个柱面组中的数据片数量
ipg	6016	每个柱面组中的信息节点数量
nindir	2048	间接地址存储块的数量
nbfree	8791469	空闲数据块的数量
ndir	2	现有目录的数量
nifree	8446460	空闲信息节点的数量
nffree	9	空闲数据片的数量
fmod	0	超级块修改标志
ronly	0	文件系统是否以只读方式安装的标志

17.5 检测与修复文件系统

本节以UFS文件系统为例，说明怎样利用fsck命令实现文件系统的检测与修复。fsck命令的主要功能是检测文件系统的完整性，对文件系统出现的问题尝试予以适当的修复。如果未加“-y”或“-n”等选项，在实施修复之前，fsck通常会提示用户确认建议的处理动作。fsck命令的语法格式简写如下：

```
fsck [-F fstype] [-m] [-V] [device]
fsck [-F fstype] [-nNyY] [-V] [-o fs-specific-options] [device]
```

其中，**device**表示文件系统的字符特殊文件名。表17-6给出了部分常用的选项及其说明。

表17-6 fsck命令

选项	简单说明
-F	用于指定文件系统的类型，如UFS等
-m	检测文件系统是否能够安装
-y	对于所有的修复请求，均以“yes”作为响应
-n	对于所有的修复请求，均以“no”作为响应
-o b=n	使用指定的备份超级块作为主超级块，修复文件系统
-o f	忽略超级块的状态标志，强制检测文件系统
-o p	以非交互方式检测文件系统，修复部分简单的问题。一旦发现严重的错误，需要用户干预时，立即结束文件系统的检测与修复。采用这个选项，可以并发地同时检测多个文件系统

17.5.1 何时需要检测文件系统

1. 文件系统受损的外部因素

UFS文件系统主要依赖于超级块，以跟踪、管理和维护其中的信息节点与数据块。当系统内存中的超级块与磁盘文件系统中的超级块没有适当地同步时，就会导致磁盘上的超级块与文件系统中的实际数据不一致。此时就需要修复文件系统，否则无法正常安装。

当由于下列原因而没有正常终止操作系统时，即可造成超级块与其维护的实际数据不一致，从而破坏文件系统：

- 电源故障，包括突然断电以及人为因素造成的电源故障；
- 强行关机，即在未正常停止UNIX系统之前即强行断开电源；
- 硬件故障，如磁盘出现坏块、控制器固件或内部电源出现故障等；
- 因UNIX系统内核故障而引起的异常关机。

一旦文件系统受损，按照/etc/vfstab文件的设置，在引导过程中，系统将会自动执行fsck命令，对文件系统的完整性与一致性进行检测。在文件系统的检测过程中，fsck将会尝试修复受损的文件系统，把修复后的文件系统安装到指定的目录位置。如果文件系统损坏严重，fsck无法完全修复，可能造成系统无法正常启动。

在文件系统修复期间，如果发现已经分配但未引用的文件或目录，fsck命令将会把它们置于lost+found目录，同时以其信息节点号命名这些文件或目录。如果lost+found目录不存在，或其中没有足够的空间，fsck命令将会尝试创建lost+found目录，或扩充其容量。

2. UFS文件系统受损的内部原理

文件操作是最常见的系统处理活动，在系统运行的每个工作日中，都可能会创建、修改或删除大量的文件。每当修改文件时，操作系统都会执行一系列文件系统的更新操作。这些更新信息如果能够及时地全部写到磁盘上，就不会破坏文件系统的完整性。通常，磁盘数据的更新是异步处理的，当用户进程访问文件系统，如写文件数据时，这些数据首先被复制到内存缓冲区中。此时，系统允许用户进程继续写数据，即使先前的数据尚未真正写到磁盘中。

因此，在任何给定的时刻，磁盘文件系统的状态总是滞后于内存超级块中表示的文件系统。当缓冲区需要分配做其他用处，或系统内核自动运行fsflush守护进程（通常以30秒为时间间隔），把缓冲区中的数据实际写到磁盘文件系统时，磁盘上的文件系统才能真正等同于内存超级块表示的文件系统。如果系统停止运行的前一刻未把内存中的超级块数据写到磁盘上，磁盘上的文件系统就会出现不一致的状态，造成文件系统受损。

3. 文件系统状态标志

超级块中有一个记录文件系统状态的标志字段，fsck命令使用这一标志字段确定文件系统的状态，决定是否需要检测文件系统的完整性。表17-7给出了状态标志字段的可能取值及其说明。

表17-7 文件系统状态标志值

状态标志值	简单说明
FSACTIVE	表示已安装文件系统的内存超级块数据已经修改。此时，如果系统意外停机，可能会丢失用户数据或元数据
FSBAD	表示文件系统包含不一致的文件系统数据
FSCLEAN	表示未安装的文件系统是完好的、未受损的
FSLOG	表示文件系统已经启用了日志功能。如果启用了此标志，文件系统只能具有FSLOG或FSBAD两种状态标志之一。如果没有启用日志功能，文件系统可以具有FSACTIVE、FSSTABLE或FSCLEAN三种状态标志之一
FSSTABLE	表示已安装的文件系统是空闲的。此时如果意外停机，不会丢失任何用户数据或元数据

17.5.2 文件系统检测的内容

在检测UFS文件系统时，fsck命令将会针对文件系统的超级块、柱面组信息块、信息节点、间接地址块和数据块等重要组成部分，检测其完整性。注意，fsck并不验证文件本身的数据内容。

1. 超级块

超级块（包括柱面组信息块）是文件系统的核心，其中存有文件和目录的关键数据和汇总信息，文件数据的每次改动都需要更新超级块和柱面组信息块，因此，超级块和柱面组信息块是文件系统中更新最频繁的部分，通常也最容易受到损坏。

所谓文件系统受损，主要是指超级块（包括柱面组信息块）中的数据受损。每当改动文件系统的信息节点或数据块时，都要修改超级块和柱面组信息块。如果CPU停止运行之前执行的最后一个命令不是sync命令，几乎肯定会损坏超级块。

超级块的完整性检测主要包括以下四个方面：

- 文件系统的大小；
- 信息节点的数量；
- 空闲数据块计数；
- 空闲信息节点计数。

(1) 检测文件系统和信息节点表的大小

文件系统的大小和信息节点区的大小均属于静态数据，一旦创建了文件系统之后，这些数

据是不会改变的。因此，**fsck**命令实际上并没有办法严格检测这种静态数据；因为这些数据在创建文件系统时就已确定。但是，**fsck**命令可以按照一种合理的推断检测这些静态数据。原则上说，文件系统的大小必须大于超级块和信息节点区占用的数据块数，信息节点的数量必须小于文件系统允许的最大值。信息节点含有除文件名和信息节点号之外的所有文件属性信息，文件系统的大小和其他统计数据是**fsck**命令最关注的重要信息。针对文件系统的所有检测都要求这些数字必须是正确的。如果检测到主超级块的静态数据有误，**fsck**命令将会要求操作员指定备份超级块的位置。

(2) 空闲数据块检测

空闲数据块是由柱面组信息块中的位图维护的，**fsck**命令将会检测文件未占用且标记为空闲的所有空闲数据块。在计数了所有空闲数据块之后，**fsck**命令还会检测并确定空闲数据块的数量加上信息节点声明已占用的数据块数量是否等于文件系统全部数据块的总量。如果发现柱面组信息块位图有任何问题，**fsck**命令将会重建柱面组信息块位图，剔除已分配的数据块。

超级块的汇总信息中含有文件系统中所有空闲数据块的统计计数，**fsck**命令将会对此计数与从文件系统中发现的实际空闲数据块总数进行比较。如果两个数字不一致，**fsck**命令将会以实际的空闲数据块计数替换超级块中的统计数字。

(3) 空闲信息节点检测

超级块中的汇总信息也包括文件系统中所有空闲信息节点的统计计数，**fsck**命令也会对此计数与文件系统中发现的实际空闲信息节点总数进行比较，如果两个数字不一致，**fsck**命令将会用实际的空闲信息节点数量替换超级块中的统计数字。

2. 信息节点

对信息节点的完整性检测从信息节点表的2号信息节点（0号和1号保留）位置开始，顺序检测每个信息节点的完整性。信息节点的完整性检测包括以下几方面内容：

- 类型与状态；
- 链接计数；
- 重复的数据块；
- 无效的数据块；
- 信息节点中的数据块计数。

(1) 类型与状态

信息节点中的模式（**mode**）字段描述了文件的类型及信息节点的状态。每个信息节点可以处于下列三种状态之一：

- 信息节点已经分配或占用（其中的模式字段为非0）；
- 信息节点尚未分配或空闲（其中的模式字段为0）；
- 信息节点部分分配（信息节点中必须具备的字段不完整）。

创建文件系统时，系统将会预留固定数量的信息节点，且仅当需要时才会分配这些信息节点。已分配的信息节点指向相应的文件，未分配的信息节点没有指向任何文件，因此其中应是空的。而部分分配的信息节点意味着其中的信息是不正确的。出现这种状态的一种可能原因是硬件故障导致垃圾数据写入信息节点中。对此情况，**fsck**命令应当采取的唯一正确动作就是清除这样的信息节点。

(2) 链接计数检测

每个信息节点均包含一个链接到当前信息节点的文件或目录项计数。**fsck**命令将会从根目录“/”开始考察整个目录结构，计算每个信息节点的实际链接计数，以验证每个信息节点的链接计数是否正确。信息节点中的链接计数与**fsck**命令确定的实际链接计数之间的差异可以归结为下列三种情况之一：

- 信息节点中的链接计数不为0，但实际计数为0。如果信息节点对应的文件或目录项并不存在，就会出现此类差异。在此情况下，**fsck**命令将会把没有归属的（断开链接的）文件置于lost+found目录中。

- 信息节点中的链接计数不为0，实际计数也不为0，但两个计数并不相同。如果已经增加或删除了某个目录项，但信息节点尚未更新，就会出现此种差异。在这种情况下，**fsck**命令将会使用实际的链接计数替换信息节点中的链接计数。

- 信息节点中的链接计数为0，但实际的链接计数不为0。在此情况下，**fsck**命令将会把信息节点中的链接计数改为实际链接计数。

（3）重复数据块检测

在文件系统中，每个数据块只能分配给一个信息节点，或者位于空闲数据存储块区。如果信息节点引用的某个数据块同时又归属于另外一个信息节点或空闲数据存储块区，可以认为此数据块是重复的。

每个信息节点中的地址数组均直接或间接地指向归属于信息节点的所有数据块。因为间接地址块也属于信息节点，如果间接地址块的归属出现问题，将会直接影响拥有间接地址块的信息节点。

fsck命令将会对归属于信息节点的每个数据块与已分配数据块位图进行比较。如果其他信息节点也声明拥有其中的某个数据块号，**fsck**命令将会把该数据块号置于重复数据块表中。否则，更新已分配数据块位图，使之包括相应的数据块号。

如果发现重复的数据块，**fsck**命令将会对信息节点位图进行第二次扫描，以便找出声明拥有每个重复数据块的其他信息节点。在此情况下，**fsck**命令无法肯定究竟那一个信息节点有误，因此，**fsck**命令将会提示用户选择究竟保留哪一个信息节点，应该清除哪一个信息节点。注意，如果信息节点中出现大量的重复数据块，通常都是由于间接地址块没有正确写入文件系统引起的。

（4）无效数据块检测

在UFS文件系统中，数据块从0开始顺序编号。超级块中的文件系统大小字段确定了最大的数据块编号。编号超出这个范围的数据块均可看做无效数据块，或简称坏块。

fsck命令将会检测信息节点声明拥有的每一个数据块号，确定其编号是否大于文件系统中第一个数据块号且小于或等于最后一个数据块号。如果数据块号超出了范围，即可认为这是一个无效的数据块。

导致信息节点中出现无效数据块的原因有可能是因为间接地址块没有写入文件系统造成的。对此，**fsck**命令将会提示用户清除这样的信息节点。

（5）信息节点中的数据块计数检测

每个信息节点均包含一个数据块引用计数。实际的数据块计数应是已分配的数据块和间接地址块之和。**fsck**命令将会计算数据块的数量，然后与信息节点声明拥有的数据块数进行比较。如果信息节点中的数据块计数有误，**fsck**命令将会提示用户予以修正。

每个信息节点也包含一个64位的表示文件大小的字段，这个字段给出了相应文件中的字节数。利用这个字段中的字节数可以计算出相应文件占用多少个数据块，然后再用这个数值与信息节点声明拥有的实际数据块数进行比较，即可得出信息节点中的数据块计数是否有误。

(6) 信息节点的连接性

每个已分配的信息节点都应存在于文件系统上的某个目录中。如果某个信息节点没有任何引用关系，即任何目录中都没有引用这个信息节点，即可认为此信息节点已经游离于文件系统，因而称做“未引用的”信息节点。

3. 目录

目录与普通文件的区别主要依据信息节点中的模式（mode）字段。目录数据块含有一系列目录项。目录数据块的完整性检测主要包括以下三个方面：

- 目录项中的信息节点号指向一个尚未分配的信息节点；
- 目录项中的信息节点号大于文件系统中信息节点的数量；
- “.”和“..”目录项的信息节点号有误；
- 游离的目录。

(1) 尚未分配的信息节点检测。

如果目录数据块中的信息节点号指向一个尚未分配的信息节点，fsck命令将会删除相应的文件或目录项。如果包含新目录项的目录数据块已经修改并写到磁盘文件系统，但相应的信息节点却没有写到文件系统，就会出现此类问题，这种情况通常是由于意外关机造成的。

(2) 无效信息节点检测。

如果目录项中的信息节点号超出了信息节点区的范围，fsck命令将会删除相应的目录项。当由于某种原因而把垃圾数据写入目录数据块时，就会出现无效的信息节点。

(3) 不正确的“.”和“..”目录项检测。

“.”目录项的信息节点号必须位于目录数据块中的第一个目录项，且该目录的信息节点号必须指向自身。也就是说，其数值必须等于当前目录的信息节点号。

“..”目录项的信息节点号必须位于目录数据块中的第二个目录项，且此信息节点号必须等于父目录或自身（如果是根目录“/”）的信息节点号。

如果“.”和“..”目录项的信息节点号有误，fsck命令将会使用正确的数值予以替换。如果目录存在多个硬链接，fsck命令将会把第一个发现的硬链接看做“..”应当指向的真正父目录。在此情况下，fsck命令会建议用户删除其他名字。

(4) 游离的目录。

fsck命令还会检测整个文件系统的连接关系。如果发现某个目录与文件系统没有任何连接关系，fsck命令将会把相应的目录置于文件系统的lost+found目录中。当由于某种原因而仅仅把信息节点写入文件系统，却未及时把相应的目录数据块写入文件系统时，就会出现此情况。

4. 间接地址块

间接地址块也归信息节点拥有。因此，间接地址块的问题也会影响相应的信息节点。fsck检测的间接地址块问题主要包括以下两个方面：

- 多个信息节点声明拥有同一间接地址块；
- 间接地址块的编号超出文件系统的范围。

5. 数据块

信息节点可以直接或间接引用三种数据块：

- 普通数据块；
- 符号链接数据块；
- 目录数据块。

其中，从属于文件的普通数据块含有文件的实际数据内容。**fsck**命令不会尝试检测普通文件数据块中数据内容的有效性。符号链接数据块含有符号链接文件中实际储存的路径名。目录数据块含有目录中的所有目录项。**fsck**命令只能检测目录数据块的有效性。

17.5.3 交互检测与修复UFS文件系统

遇到下列情况时，用户也许需要交互地检测文件系统：

- 文件系统无法安装；
- 文件系统在使用过程中报错。

当文件系统在使用过程中报错时，错误信息可能会出现在控制台上，或写到系统的错误信息日志文件中。严重时还会引起系统瘫痪。例如，系统的错误信息日志文件`/var/adm/messages`也许会包含类似如下的错误信息（其中的**hostname**是系统的主机名）：

```
Sep 5 13:42:40 hostname ufs: [ID 879645 kern.notice] NOTICE: /: unexpected
free inode 630916, run fsck(1M)
```

在运行**fsck**命令检测与修复UFS文件系统之前，应记住下列注意事项：

- 在使用**fsck**命令检测文件系统期间，相应的文件系统应处于非工作状态。否则，当内存超级块定时更新磁盘超级块时，会导致文件系统不断发生变化，致使**fsck**无法准确地检测文件系统，尤其严重的是，超级块的更新与**fsck**的修复将会相互影响，造成超级块数据的混乱，因此极有可能损害甚或进一步破坏文件系统，严重时甚至会危及系统的正常运行。
- 在使用**fsck**命令检测文件系统之前，应卸下相应的文件系统。这将确保文件系统的数据结构处于完好状态。唯一的例外是“/”文件系统（如果`/usr`是一个单独的文件系统，也包括`/usr`文件系统），因为“/”等文件系统必须安装且处于工作状态，否则无法运行**fsck**命令。
- 如确实需要修复“/”等文件系统，应设法让“/”等文件系统处于非安装或非工作状态。
- 此外，只有超级用户才能使用**fsck**命令检测与修复文件系统。因此，在检测和修复文件系统时，必须使用**root**或**su**命令注册为超级用户。

1. 怎样检测和修复“/”等文件系统

当需要检测和修复“/”等文件系统时，应确保系统处于固件、安全或系统维护等单用户模式，之前还应查询`/etc/vfstab`文件，记住“/”等文件系统的设备文件名。建议的处理步骤如下。

(1) 利用CD/DVD等安装介质引导系统，中途退出安装程序；把系统引导至系统维护模式，或将系统引导至安全模式，确保“/”等文件系统没有文件访问活动。例如，在启动Solaris系统时可以选择“Solaris failsafe”，将系统引导至安全模式。

(2) 利用**fsck**命令检测与修复“/”等文件系统。注意，如果磁盘设备发生变动，文件系统的设备文件名也可能会发生变化，此时可以检查“**fsck -n**”命令的输出信息，其中的“Last

Mounted on ...”表示文件系统之前的安装点（通常也是文件系统的名字），前一行即相应的设备文件名。在下面的例子中，“/”文件系统的设备文件名为/dev/dsk/c0t0d0s0:

```
# fsck -n /dev/rdisk/c0t0d0s0
** /dev/rdisk/c0t0d0s0 (NO WRITE)
** Last Mounted on /
.....
# fsck -F ufs /dev/rdisk/c0t0d0s0
** /dev/rdisk/c0t0d0s0
** Last Mounted on /
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
.....
```

(3) 根据fsck命令的提示，校正和修复文件系统。详细内容请参见17.5.7节。

(4) 如果必要或出现下列信息时，可再次运行fsck命令，重复检测文件系统：

```
FILE SYSTEM STATE NOT SET TO OKAY 或
FILE SYSTEM MODIFIED
```

常规情况下，执行一次fsck命令也许无法完全修复所有的错误，但如果多次运行fsck命令仍然不能解决问题，则需要参考17.5.6节中的说明。

(5) 安装已修复的文件系统，检查lost+found目录中是否存在任何文件。fsck命令收集且置入lost+found目录中的文件以其信息节点号命名。如果存在，可以使用file等命令确定文件的类型，再用grep、cat或od等命令检查其内容，重新命名这些文件，并把它们移至适当的目录。最后，删除或备份lost+found目录中剩下的暂时无法辨别和处理的文件或目录，以免不必要地占用该目录的目录项空间。

(6) 使用init或shutdown等命令，把系统重新引导至多用户工作模式。

2. 怎样检测和修复其他文件系统

除了“/”文件系统之外，检测和修复其他用户文件系统的步骤如下。

(1) 卸下文件系统，确保不存在文件系统访问活动。然后使用安装点目录或相应的设备文件名作为fsck命令的参数，检测与修复文件系统。例如：

```
# umount /export/home
# fsck -F ufs /dev/rdisk/c0t0d0s7
** /dev/dsk/c0t0d0s7
** Last Mounted on /export/home
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
.....
```

(2) 修正fsck命令报告的错误，详细内容请参见17.5.7节。

(3) 如果必要或出现下列信息时，可再次运行fsck命令，重复检测文件系统：

```
FILE SYSTEM STATE NOT SET TO OKAY 或
FILE SYSTEM MODIFIED
```

常规情况下，执行一次fsck命令也许无法完全修正所有的错误，但如果多次运行fsck命令仍然不能修复所有的问题，则需要参考17.5.6节中的说明。

(4) 安装已修复的文件系统，检查lost+found目录中是否存在任何文件或目录。

(5) 同样，如果存在fsck命令收集的文件，可以使用file等命令确定文件的类型，再使用grep、cat或od等命令检查其内容，在重新命名文件之后，将这些文件移至适当的目录。最后，删除或备份lost+found目录中剩下的暂时无法处理的文件或目录，以免占用lost+found目录的存储空间。

下面的例子说明了怎样检测和修复相应于/dev/rdisk/c0t0d0s6设备文件的文件系统，校正不正确的数据块计数（假定之前已经卸载了文件系统）：

```
# fsck -F ufs /dev/rdisk/c0t0d0s6
** Phase 1 - Check Block and Sizes
INCORRECT BLOCK COUNT I=2529 (6 should be 2)
CORRECT? y
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Cylinder Groups
929 files, 8928 used, 2851 free (75 frags, 347 blocks, 0.6% fragmentation)
/dev/rdisk/c0t0d0s6 FILE SYSTEM STATE SET TO OKAY
***** FILE SYSTEM WAS MODIFIED *****
```

17.5.4 自动检测与修复UFS文件系统

利用“-o p”或“-y”选项，可以任由fsck命令自动检测与修复文件系统。前者的优点是能够并发地同时检测多个文件系统，缺点是如果遇到需要操作员干预的问题，将会立即终止fsck命令的运行。

利用“-o p”选项运行fsck命令时，fsck不会检查文件系统的清除标志（FSCLEAN），因而总是能够执行一次文件系统的全面检测。但请注意，此时执行的检测动作只是交互运行fsck命令的所有处理动作的一个子集（即其中的一部分）。

下面的例子说明了怎样利用“-o p”选项，自动地检测/export/home文件系统：

```
# fsck -F ufs -o p /export/home
** Phase 1 - Check Block and Sizes
** Phase 2 - Check Pathnames
.....
```

采用交互方式检测文件系统时，也可以使用“-y”选项运行fsck命令，实现文件系统的自动检测。“-y”选项表示，对于fsck程序提出的每一个修复建议，总是以“yes”给予确认。例如：

```
# fsck -F ufs -y /export/home
** Phase 1 - Check Block and Sizes
** Phase 2 - Check Pathnames
.....
```

17.5.5 恢复严重受损的超级块

上一章曾经讲过，UFS文件系统存有多多个超级块副本（备份超级块），当fsck命令检测到文件系统的主超级块已严重受损时，可以采用“fsck -o b=n”命令，利用其中的任何一个超级块副本替换主超级块（如果文件系统中的所有超级块均已损坏，则无法恢复，只能重建文件

系统，再利用备份数据恢复系统）。恢复步骤如下。

(1) 使用下列**newfs**命令显示超级块中的数据（注意，“-N”选项可用于输出之前使用**newfs**命令创建文件系统时用做超级块副本的数据块号，如果忽略了这个选项，将会毁灭文件系统中的所有数据，而代之重建一个空的文件系统）：

```
# newfs -N /dev/rdisk/device-name
```

(2) 使用下列**fsck**命令，同时给出超级块副本，使之替换受损的主超级块，以便使用指定的超级块副本恢复主超级块（注意，不管文件系统的规模如何，几乎总是可以使用32号数据块作为超级块的副本，或者用“**newfs -N**”命令给出的其他超级块副本）：

```
# fsck -F ufs -o b=block-number /dev/rdisk/device-name
```

下面的例子说明了怎样使用超级块副本5264，修复主超级块受损的文件系统：

```
# newfs -N /dev/rdisk/c0t3d0s7
/dev/rdisk/c0t3d0s7: 163944 sectors in 506 cylinders of 9 tracks, 36 sectors
83.9MB in 32 cyl groups (16 c/g, 2.65MB/g, 1216 i/g)
super-block backups (for fsck -b #) at:
32, 5264, 10496, 15728, 20960, 26192, 31424, 36656, 41888,
47120, 52352, 57584, 62816, 68048, 73280, 78512, 82976, 88208,
93440, 98672, 103904, 109136, 114368, 119600, 124832, 130064, 135296,
140528, 145760, 150992, 156224, 161456,
# fsck -F ufs -o b=5264 /dev/rdisk/c0t3d0s7
Alternate superblock location: 5264.
** /dev/rdisk/c0t3d0s7
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
36 files, 867 used, 75712 free (16 frags, 9462 blocks, 0.0% fragmentation)
/dev/rdisk/c0t3d0s7 FILE SYSTEM STATE SET TO OKAY
***** FILE SYSTEM WAS MODIFIED *****
#
```

17.5.6 解决fsck命令无法修复的UFS文件系统问题

fsck命令采用多次扫描分析的阶段处理方式，后期扫描分析阶段修正的错误可能会暴露出需要在早期扫描分析阶段才能检测处理的其他问题。因此，有时需要多次地重复运行**fsck**命令，直至**fsck**命令不再报错。这种做法能够确保发现和修复所有的错误。**fsck**命令不会自己连续运行直至排除所有错误，因此，用户需要手工地重复输入命令。

用户应当关注**fsck**命令显示的各种信息，这些信息有助于修正文件系统错误。例如，假定有一条信息指出某个目录已经损坏，如果删除该目录，**fsck**命令修复文件系统的工作可能很快就成功地结束了。

如果**fsck**命令仍然不能修复文件系统，用户可以尝试使用**find**、**ff**、**clri**和**ncheck**命令，找出和修正文件系统错误。最坏的情况下也许需要重建文件系统，从备份介质中恢复用户数据。

如果文件系统无法完全修复，但能以只读方式安装文件系统，可以尝试使用**cp**、**tar**或**cpio**

命令从文件系统中取出全部或部分数据。

如果是由于磁盘故障导致文件系统出错，也许需要重新格式化磁盘或划分磁盘分区，然后再重建文件系统，利用备份介质恢复其中的用户数据。

如果是硬件错误，通常会重现甚至重复地多次显示相同的错误信息。`format`等命令会尝试剔除磁盘上的坏块。然而，如果磁盘受损严重，问题可能会持续存在，即使重新格式化之后也是如此。

17.5.7 fsck的阶段处理方式

通常，当系统因异常停机等原因，导致最新的文件系统变化没有及时更新到磁盘时，在重新引导系统的过程中，`fsck`命令将会以非交互的方式，自动检测并修复文件系统。但这种修复只能解决基本的问题，一旦遇到严重的情况，`fsck`不会进一步尝试自行修复，而是报告出错信息，然后停止运行。

当用户以交互方式运行`fsck`命令时，`fsck`将会报告检测到的所有错误，并自行解决自己能够修复的简单问题。对于比较严重的错误，`fsck`命令将会给出简明的错误信息，同时提请用户做出选择。当使用“-y”或“-n”选项运行`fsck`命令时，用户的选择总是按照预定的“yes”或“no”响应`fsck`命令每一个提问。

在这种文件系统的修复过程中，某些校正动作可能会导致数据的丢失。数据丢失的数量和严重程度可以从`fsck`命令的输出信息中做出判定。

`fsck`命令是一个采用多次（或多阶段）扫描方式进行分析处理的文件系统检测程序。在完成初始化之后，`fsck`命令将分5个阶段，分别检测数据块和文件的大小、路径名、连接性、引用计数以及空闲数据块和信息节点位图，采取必要的纠正动作，或请求用户做出选择：

- 初始化阶段；
- 第一阶段：检测数据块和文件的大小；
- 第二阶段：检测文件目录的路径名；
- 第三阶段：检测目录的连接性；
- 第四阶段：检测引用计数；
- 第五阶段：检测柱面组信息块位图（UFS文件系统版的`fsck`命令）。

下面是以交互方式运行`fsck`命令，并成功地结束后给出的信息：

```
# fsck -F ufs /dev/rdisk/c0t0d0s7
** /dev/rdisk/c0t0d0s7
** Last Mounted on /export/home
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
2 files, 9 used, 2833540 free (20 frags, 354190 blocks, 0.0% fragmentation)
#
```

1. 初始化阶段

在初始化阶段，`fsck`首先执行命令的行语法检查。在正式开始执行文件系统检测之前，`fsck`

将设置工作表，打开必要的文件。

如果存在，这个阶段的错误信息主要涉及用户提供的命令行选项是否正确，运行fsck命令时申请分配内存，打开文件的请求是否能够得到满足，文件的状态及大小检测能否通过，创建文件是否成功。上述初始化阶段的任何错误都会中止以非交互形式运行的fsck命令。

2. 第一阶段：检测文件的数据块计数和大小

这个阶段主要检测信息节点表，检测信息节点的组成部分（如文件类型、数据块计数、文件的大小以及信息节点结构的格式）是否有误，报告并修复下列错误：

- 检测信息节点的文件类型；
- 设置零连接计数表；
- 考察信息节点引用的数据块是否存在无效或重复情况；
- 检测信息节点中的文件或目录的大小；
- 检测信息节点的格式。

除了文件或目录大小有误，数据块数量与文件或目录的大小不一致，信息节点的格式或内容不完整，以及文件类型有误等错误情况之外，这个阶段出现的其他错误将会终止fsck自动检测文件系统。

表17-8给出了第一阶段可能出现的一些错误信息及其解释。

表17-8 第一阶段可能出现的错误信息

错误信息	简单说明以及建议的处理动作
block BAD I=inode	表示分配给相应信息节点的数据块是一个坏块
block DUP I=inode	表示分配给相应信息节点的数据块也分配给了其他信息节点，或与空闲数据块位图中的某个数据块号重复
BAD MODE: MAKE IT A FILE?	说明信息节点中的文件类型和访问权限信息受损。回答“y”将允许fsck把文件的类型设置为普通文件，把访问权限设置为744
EXCESSIVE BAD BLOCKS I=inode (CONTINUE?)	表示分配给相应信息节点的数据块坏块太多。回答“y”将会继续检测，但需用户关注这个信息节点号，以便之后利用ncheck或ff等命令调查相应的文件
EXCESSIVE DUP BLOCKS I=inode (CONTINUE?)	说明分配给相应信息节点的数据块存在太多的重复情况。回答“y”将会继续检测，但需用户关注这个信息节点号
INCORRECT BLOCK COUNT I=inode (X should be Y) (CORRECT?)	说明信息节点表示的文件或目录大小有误，fsck发现信息节点中的数据块计数X不正确，经分析，实际计数应为Y。为了使用Y替换信息节点中的数据块计数X，可在CORRECT提示下输入“y”，使fsck调整数据块计数。否则输入“n”
PARTIALLY ALLOCATED INODE I=inode (CLEAR?)	表示信息节点的格式（或内容）不完整，如访问权限字段为空，但地址字段确有数据等。回答“y”可释放相应的信息节点，否则输入“n”
PARTIALLY TRUNCATED INODE I=inode (SALVAGE?)	表示信息节点的文件大小字段小于实际分配的数量。此时可输入“y”予以校正，或输入“n”忽略之
UNKNOWN FILE TYPE I=inode (CLEAR?)	如果文件的类型有误，即并非常规的普通文件、块特殊文件、字符特殊文件、目录文件、管道文件，以及符号连接文件等，将会出现此错误信息。回答“y”将会清除相应的信息节点；回答“n”可为用户保留进一步检查问题的时间

3. 重复执行第一段（Phase 1B）的分析处理

如果在第一阶段发现重复的数据块，fsck将会重新检查信息节点表，找出其中也包含同一数据块的信息节点。如果fsck发现重复的数据块，将会显示下列错误信息：

```
block DUP I=inode
```

这一错误信息说明，信息节点inode中的一个数据块号block与另一个信息节点包含的数据块号是重复的。这个错误将导致在第二阶段的分析处理期间出现BAD/DUP错误。而数据块交叉重叠的错误正是第一阶段要考察的问题，因此，当发现重复的数据块时，fsck需要重新执行第一阶段的分析处理，找出先前曾经拥有同一数据块的信息节点。也就是说，在此情况下，fsck将会再次（甚至多次）执行第一阶段的检测任务，回头再分析处理重复的数据块。

4. 第二阶段：检测文件目录的路径名

在这个阶段中，fsck将会检查并分析文件系统中的所有目录，检测其中的目录项是否指向未分配的信息节点、坏的信息节点，同时也检测root的信息节点，删除前一阶段发现的含无效信息节点号的目录项，报告并修复下列错误：

- 根（/）目录信息节点中的模式（即访问权限）和状态是否正确；
- 目录信息节点中的地址指针是否超出了文件系统的范围；
- 目录项中的信息节点号是否有效；
- 目录的完整性。

表17-9给出了第二阶段可能出现的一些错误信息及其解释。

表17-9 第二阶段可能出现的错误信息

错误信息	简单说明以及建议的处理动作
BAD INODE NUMBER FOR '.' I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime DIR=fname (FIX?)	表示“.”目录项的信息节点号与当前目录项的信息节点号不匹配。回答“y”时fsck将会调整“.”，使其信息节点号与当前目录保持一致
BAD INODE NUMBER FOR '..' I =inode OWNER=UID MODE=fmode SIZE= fsize MTIME=modtime DIR=fname (FIX?)	表示“..”目录项的信息节点号与其父目录项的信息节点不匹配。回答“y”时fsck将会调整“..”，使信息节点号与其父目录一致
DIRECTORY TOO SHORT I=inode OWNER =UID MODE=fmode SIZE=fsize MTIME= modtime DIR=fname	目录中的目录项小于规定的最小目录数量（即一个目录最少应包含当前目录“.”和父目录“..”两个目录项）。回答“y”可使fsck设置一个最小值，但并不能真正校正此问题，只能保证这是一个合法的目录。因此，事后还需要做进一步的检查
DIRECTORY fname LENGTH size NOT MULTIPLE bsize (ADJUST?)	目录的大小并不等于数据块（4096或8192）的倍数。回答“y”允许fsck适当调整目录的大小。同样，这并不能真正解决问题，只能保证这是一个合法的目录。因此，事后还需要做进一步的检查
DIRECTORY CORRUPTED I=inode OWNER= UID MODE=fmode SIZE=fsize MTIME= modtime DIR=fname (SALVAGE?)	信息节点表示的文件类型为目录，但其并不具有目录的内部数据格式。如果回答“y”，fsck将会在尝试修复目录时错误地删除其中的文件项。除非了解此问题及其后果，否则应回答“n”
EXTRA '.' ENTRY I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime DIR=fname (FIX?)	表示目录中存在多个“.”目录项。回答“y”允许fsck删除多余的“.”目录项

(续表)

错误信息	简单说明以及建议的处理动作
EXTRA '..' ENTRY I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime DIR=fname (FIX?)	表示目录中存在多个“..”目录项。回答“y”允许fsck删除多余的“..”目录项
I OUT OF RANGE I=inode NAME=fname (REMOVE?)	信息节点号超出了信息节点表的范围。此时建议回答“y”，以删除目录中引用的错误信息节点
MISSING '.' I=inode OWNER=UID MODE= fmode SIZE=fsize MTIME=modtime DIR= fname (FIX?)	表示目录中缺失第一个目录项（即当前目录“.”）。回答“y”允许fsck重新创建一个“.”目录项
MISSING '.' I=inode OWNER=UID MODE= fmode SIZE=fsize MTIME=modtime DIR=fname CANNOT FIX, FIRST ENTRY IN DIRECTORY CONTAINS fname	表示目录中缺失第一个目录项（即当前目录“.”），但fsck无法校正此问题。解决的办法是先安装文件系统，把第一个目录项移至其他目录，然后再重新运行fsck
MISSING '.' I=inode OWNER=UID MODE= fmode SIZE=fsize MTIME=modtime DIR= fname CANNOT FIX, INSUFFICIENT SPACE TO ADD '.'	表示目录中缺失第一个目录项（即当前目录“.”），但由于文件系统没有剩余的空间，fsck也无法校正这个问题。解决的办法是先安装文件系统，尽可能删除一些无用的文件，然后再重新运行fsck
NAME TOO LONG pathname	fsck发现文件的路径名太长。通常情况下，这表示存在循环的符号链接。此时应回答“y”，令fsck删除此循环链接
ROOT INODE UNALLOCATED. (ALLOCATE?)	说明根目录的信息节点（即2号信息节点）并未分配。此时，fsck会赋予2号信息节点一个默认值。在第三阶段，fsck将会把根目录中的子目录和文件重新连接到lost+found目录中
ROOT INODE NOT DIRECTORY (FIX?)	说明根目录信息节点（即2号信息节点）中的文件类型并非目录。如果回答“y”，fsck将会把其文件类型校正为目录
UNALLOCATED I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime {FILE DIR}=fname (REMOVE?)	说明目录或文件指向一个未分配的信息节点，同时给出文件或目录的属主、模式（即访问权限）、大小，以及最后一次修改时间。为了删除此目录或文件，可在“REMOVE”提示下输入“y”。否则输入“n”，以忽略此错误信息
ZERO LENGTH DIRECTORY I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime DIR=fname (REMOVE?)	说明目录的大小为0。实际上，每个目录至少应包含两个目录项（“.”和“..”）。回答“y”将会删除相应的目录

注意：上述与“.”目录项有关的问题，也同样会出现在“..”目录项身上。修正的方法也一样。

5. 第三阶段：检测信息节点与目录的连接性

在这个阶段，fsck将会根据第二阶段对目录的考察，重点检测信息节点表中是否存在没有目录归属的信息节点，因为每一个分配的信息节点至少都应有一个目录归属。该阶段报告并修复下列错误：

- 没有目录归属的信息节点；
- lost+found目录是否存在，其目录项位置是否已全部占用。

表17-10给出了第三阶段可能出现的一些错误信息及其解释。

表17-10 第三阶段可能出现的错误信息

错误信息	简单说明以及建议的处理动作
DIR I=inode CONNECTED. PARENT WAS I=inode	这只是一个说明信息，表示先前的lost+found目录连接操作已经成功
DIRECTORY fname LENGTH fsize NOT MULTIPLE OF bsize (ADJUST?)	给定目录的大小并不等于数据块的倍数。为了适当调整目录的大小，可在“ADJUST”提示下输入“y”，否则输入“n”
NO SPACE LEFT IN /lost+found (EXPAND?)	文件系统的/lost+found目录没有可用的空间，已无法再创建或增加文件。为了扩展该目录，可在“EXPAND”提示下输入“y”
NO lost+found DIRECTORY (CREATE?)	文件系统根目录中不存在lost+found子目录。为了创建该目录，可在“CREATE”提示下输入“y”
UNREF DIR I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime (RECONNECT?)	说明信息节点没有目录归属。经过详细考察文件系统，fsck发现了一个信息节点没有连接到任何目录中，同时给出了其属主、模式字段（即访问权限）、大小，以及最后一次修改时间。为了把此信息节点连接到lost+found目录中，以便进一步考察，可在“RECONNECT”提示下输入“y”（否则输入“n”）。如果连接成功，将会显示一个“CONNECTED”信息。否则将会输出一个表示lost+found目录错误的信息

6. 第四阶段：检测引用计数

在这个阶段，fsck将会检测连接计数，检测空闲信息节点计数。根据第一、第二和第三阶段的检测结果，处理第三阶段发现的没有目录归属的文件，处理第一阶段发现的坏块或重复数据块的问题。报告并修复下列错误：

- 是否存在游离（未引用）的文件或目录；
- lost+found目录是否存在，其目录项位置是否已全部占用；
- 文件、目录、符号链接文件以及特殊文件的链接计数是否正确；
- 文件和目录中是否存在无效或重复的数据块；
- 空闲信息节点的统计计数是否正确。

这个阶段发现的所有错误（除了lost+found目录下没有空间之外）都是可以校正的（包括自动检测方式）。

表17-11给出了第四阶段可能出现的一些错误信息及其解释。

表17-11 第四阶段可能出现的错误信息

错误信息	简单说明以及建议的处理动作
BAD/DUP {FILE DIR} I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime (CLEAR?)	说明文件或目录出现了无效或重复的数据块，同时也给出了文件或目录的信息节点号、属主、模式字段（即访问权限）、大小，以及最后一次修改时间。为了校正这一错误，可在“CLEAR”提示下输入“y”。否则输入“n”
FREE INODE COUNT WRONG IN SUPERBLK	在计数信息节点表中未分配信息节点的数量，并与超级块中的记录进行比较时，fsck发现统计数据有误。回答“y”将允许fsck调整超级块相应字段的值
LINK COUNT {FILE DIR} I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime COUNT=amt1 SHOULD BE amt2 (ADJUST?)	在检索了引用相关信息节点号的所有目录结构之后，再与信息节点中的链接计数字段进行比较，发现两者不符。回答“y”将会按照fsck计算的数值，调整信息节点中的链接计数字段

(续表)

错误信息	简单说明以及建议的处理动作
NO SPACE LEFT IN / lost+found (EXPAND?)	文件系统根目录中的lost+found目录已满, 无法再创建或增加其他文件。为了扩大lost+found目录, 可输入“y”
UNREF {FILE DIR} I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime (RECONNECT?)	说明信息节点没有目录归属。回答“y”将会把相应的信息节点重新连接到lost+found目录, 使之引用此信息节点
UNREF {FILE DIR} I=inode OWNER=UID MODE=fmode SIZE=fsize MTIME=modtime (CLEAR?)	如果先前在重新连接提示下回答了“n”, fsck将会请求清除相应的信息节点

在下面的例子中, fsck命令发现/export/home文件系统中一个目录的链接计数出现了错误:

```
# fsck -F ufs /dev/rdisk/c0d0s7
**/dev/rdisk/c0d0s7
** Currently Mounted on /export/home
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
LINK COUNT DIR I=4 OWNER=root MODE=40700
SIZE=4096 MTIME=Nov 11 1:56 2005 COUNT 2 SHOULD BE 3
ADJUST? y
```

7. 第五阶段: 检测柱面组信息块位图

在这个阶段, fsck主要检查空闲数据块区, 检测坏块、重复的数据块、缺失的数据块、空闲和已用数据块计数、空闲和已用信息节点以及相应的位图, 报告并修复下列错误:

- 已用信息节点位图是否遗漏了已分配的信息节点;
- 空闲数据块位图是否遗漏了空闲的数据块;
- 已用信息节点位图中是否存在空闲的信息节点;
- 空闲数据块的统计计数是否正确;
- 已用信息节点的统计计数是否正确。

表17-12给出了第五阶段可能出现的一些错误信息及其解释。

表17-12 第五阶段可能出现的错误信息

错误信息	简单说明以及建议的处理动作
BLK(S) MISSING IN BIT MAPS (SALVAGE?)	说明柱面组信息块位图中漏记了某些空闲数据块。为了校正这一错误, 需要重新组织空闲数据块位图, 此时, 可在“SALVAGE?”提示后输入“y”。否则输入“n”, 以忽略此错误情况
FREE BLK COUNT WRONG IN SUPERBLOCK	在计数未分配的空闲数据块, 并与超级块中的相应值进行比较时, fsck发现两者不符, 回答“y”将允许fsck进行适当的调整
SUMMARY INFORMATION BAD (SALVAGE?)	说明汇总信息不正确, 需根据fsck的检测结果进行更新。为此, 可在“SALVAGE”提示下输入“y”

下面的例子说明/export/home文件系统超级块中的空闲数据块计数出现了错误:

```
# fsck -F ufs /dev/rdisk/c0d0s7
** /dev/rdisk/c0d0s7
** Currently Mounted on /export/home
** Phase 1-Check Blocks and sizes
** Phase 2-Check Pathnames
** Phase 3-Check Connectivity
** Phase 4-Check Reference Counts
** Phase 5-Check Cyl groups
FREE BLK COUNT (S) WRONG IN SUPERBLK
SALVAGE? Y
```

8. 检测汇总阶段

一旦完成文件系统的检测，fsck命令将会在最后给出一个分析处理的总结报告，即给出下列文件系统使用情况的汇总信息:

```
# files, # used, # free (# frags, # blocks, % fragmentation)
```

汇总信息的前三个数据表示检测的文件系统有多少个文件，使用了多少逻辑数据片，还有多少逻辑数据片是空闲的。圆括号中的数据是对空闲数据片的进一步分解与表述，其中依次为空闲数据片的数量、空闲数据块的数量，以及空闲数据片占全部数据片的百分比。

- # files 当前已经使用的信息节点数量。
- # used 当前已经使用的数据片数量。
- # free 全部空闲数据片的总和。
- # frags 其中空闲的纯数据片的数量。
- # blocks 其中用于提供数据片的空闲数据块的数量。
- % fragmentation 数据片碎化比率（空闲的纯数据片×100/文件系统中所有数据片）。

在这个阶段，如果所有问题都得到了解决，fsck将会请求用户确认是否需要把超级块中的文件系统标志设置为OKAY。回答“yes”后fsck才会设置这一标志，否则文件系统仍然无法安装，即使文件系统已经修复。

表17-13给出了fsck最后处理阶段可能出现的一些错误信息及其解释。

表17-13 汇总阶段可能出现的错误信息

错误信息	简单说明以及建议的处理动作
***** FILE SYSTEM WAS MODIFIED *****	表示fsck已经修改了文件系统。如果修复的文件系统是“/”文件系统，系统会自动地重新引导。如果是其他文件系统，且文件系统已经安装，用户需要卸载文件系统，并再次运行fsck命令。否则，内存中的超级块有可能会覆盖fsck已做的检测与修复工作
filesysname FILE SYSTEM STATE SET TO OKAY	表示文件系统已标记为完好状态（FSSTABLE）。如果是“/”之外的其他文件系统，也可以使用“fsck -m”命令再次确定文件系统是否还需要检测
filesysname FILE SYSTEM STATE NOT SET TO OKAY	表示文件系统没有标记为完好状态（FSSTABLE），说明文件系统还有问题没有得到修复。此时应再次检测文件系统，也可以使用“fsck -m”命令确定文件系统是否确实还需要检测

例如，以交互方式运行fsck命令，且成功地结束之后，fsck将会给出类似如下的信息：

```
# fsck -F ufs /dev/rdisk/c0d0s7
** /dev/rdisk/c0d0s7
** Last Mounted on /export/home
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
2 files, 9 used, 2833540 free (20 frags, 354190 blocks, 0.0% fragmentation)
#
```

上述汇总信息表示，/dev/rdisk/c0d0s7对应的文件系统 (/export/home)-中共有两个文件，占用了9个数据片，尚有2833540个空闲数据片。在这些空闲数据片中，只有20个是真正的数据片，还有354190个是可用于提供数据片的空闲数据块（354190个数据块×8个数据片/数据块 + 20个数据片=2 833 540个数据片）。数据片碎化比率（空闲的纯数据片×100/文件系统中的所有数据片）为零。

17.5.8 利用其他工具修复文件系统

1. 利用ncheck命令恢复丢失的数据

ncheck命令用于显示指定设备文件（即文件系统）中的文件名及其对应的信息节点号。其语法格式简写如下：

```
ncheck [-F FSType] [-V] -i i-list -s [-o m] [device]
```

其中，“-i”选项用于指定信息节点号，多个信息节点号之间必须加逗号分隔符且不能有额外的空格。“-s”选项说明仅输出特殊文件及设置了setuid标志位的文件。这个选项可用于检测系统中是否存在有违安全策略的文件。示例如下：

```
# ncheck -i 129 /dev/dsk/c0d0s7
/dev/dsk/c0d0s7:
129      /gqxing/ftplog
#
```

注意：除非检测的设备文件名对应的文件系统为“/”，否则，ncheck命令的输出信息给出的文件名并非真正的绝对路径名。如果检测的设备文件名对应的文件系统为/export/home，则上述文件的真正文件名应是/export/home/gqxing/ftplog。

此外，可以使用“-s”选项检测系统中是否存在有违安全策略的文件。例如，下列命令给出的文件均为设置了setuid标志位的文件，这些文件在执行期间均具有超级用户的特权。

```
# ncheck -s /dev/dsk/c0d0s0
/dev/dsk/c0d0s0:
337      /usr/bin/at
338      /usr/bin/atq
339      /usr/bin/atrm
366      /usr/bin/crontab
444      /usr/bin/login
464      /usr/bin/passwd
494      /usr/bin/su
```

```
514      /usr/bin/write
918      /usr/sbin/traceroute
928      /usr/sbin/wall
.....
```

根据位于lost+found目录中的信息节点号，利用ncheck命令即可找出其对应的文件。因而便可以检测相应的文件是否丢失数据，如果确实如此，可以把以信息节点号命名的文件中的内容加到正确的文件中。

2. 利用ff命令恢复丢失的数据

ff命令用于列出文件系统中的文件名及其统计信息。ff命令的语法格式如下：

```
ff [-F FSType] [-V] [generic_opts] [-o specific_opts] device
```

ff命令的主要目的是输出指定设备文件系统中的文件及其信息节点。如果再使用其他选项，还可以输出更多的文件属性信息，如文件的大小或属主。另外，ff命令提供的一些选项可用于限定ff仅仅输出特定的文件信息。例如，“-i”选项可用于限定输出指定信息节点对应的文件信息。

表17-14给出的是大多数文件系统通常均支持的普通命令选项。

表17-14 ff命令的部分选项

选项	简单说明
-I	禁止在文件名的后面输出其信息节点号
-l	对于具有多个链接的文件，生成所有的文件路径名
-p prefix	把指定的前缀加到生成的每个路径名前面，构成真正的绝对路径名。默认值为“.”
-s	禁止在文件名的后面输出文件的大小（以字节为单位）
-u	在文件名的后面输出文件的属主
-a -n	选择输出在指定的天数之内访问过的文件
-m -n	选择输出在指定的天数之内曾经更新或新创建的文件
-c -n	选择输出在指定的天数之内状态信息发生变化的文件
-n file	选择输出比给定文件的修改时间更近的文件
-i inode-list	仅输出与给定信息节点有关的文件。如果同时给出多个信息节点，信息节点号之间必须加逗号“,”分隔符，且中间不能有空格
-o	用于指定UFS文件系统特定的选项，其中包括： <ul style="list-style-type: none">• a: 输出“.”和“..”目录• m: 输出文件的模式字段，其中包括访问权限等信息，这个选项必须与“-i inode-list”选项一起使用• s: 仅输出特殊文件和设置了setuid标志的文件

例如，下列命令用于显示信息节点号为444的文件，同时输出其访问权限等信息：

```
# ff -F ufs -i 444 -o m /dev/dsk/c0d0s0
/dev/dsk/c0d0s0:
mode 104555 uid 0      gid 2      ino 444      /usr/bin/login
#
```

3. 利用clri命令清除重复或无效的信息节点

clri命令主要用于清除信息节点。其语法格式简写如下：

```
clri [-F FSType] [-V] device i-number
```

其中，**device**是文件系统的设备文件名。**i-number**是准备删除的信息节点。

clri命令的主要功能是把数值0写入指定文件系统的信息节点，从而释放指定的信息节点，使之成为可以重新分配的空闲信息节点。执行clri命令之后，相应文件的任何数据块也将随之丢失。这一事实可在运行fsck命令时得到确认。例如：

```
# clri /dev/rdisk/c0d0s7 130
clearing 130
# fsck /dev/rdisk/c0d0s7
** /dev/rdisk/c0d0s7
** Last Mounted on /export/home
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
UNALLOCATED I=130 OWNER=root MODE=0
SIZE=0 MTIME=Jan 1 08:00 1970
NAME=/its/du3

REMOVE? y

** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
FREE BLK COUNT(S) WRONG IN SUPERBLK
SALVAGE? y

136 files, 1083 used, 10076753 free (33 frags, 1259590 blocks, 0.0% fragmenta-
tion)

***** FILE SYSTEM WAS MODIFIED *****
#
```

这个命令的主要目的是删除由于某种原因而一时没有目录归属的文件。当使用clri命令删除这样的文件信息节点时，应当仔细跟踪所有文件系统上的文件指针，并予以删除。否则，当重新分配这个信息节点时，原有的信息节点也将指向新建的文件，使得两个文件名均使用同一个信息节点。此时，如果删除原有的文件指针，又会危及新建的文件，致使新的文件指向一个未分配的信息节点——这样便会陷入一个怪圈。

参 考 文 献

- [1] Berny Goodheart, James Cox. The Magic Garden Explained, The Internals of UNIX System V Release 4[M]. Prentice Hall, 1994.
- [2] Maurice J. Bach. The Design of the UNIX Operating System[M]. Prentice Hall, 1986.
- [3] Kenneth Rosen. UNIX: The Complete Reference, Second Edition[M]. McGraw-Hill, 2007.
- [4] Douglass E. Comer, David L. Stevens. Internetworking with TCP/IP[M]. Prentice Hall, 1995.
- [5] Craig Hunt. TCP/IP Network Administration. O'Reilly & Associates[M], 1994.
- [6] W. Richard Stevens. Advanced Programming in the UNIX Environment[M]. Addison-Wesley, 1992.
- [7] Mendel Cooper. Advanced Bash-Scripting Guide (<http://download.csdn.net/source/949704>).
- [8] AT&T UNIX System V Release 4.X Documentations.
- [9] Sun Microsystem Solaris 10 Documentations.
- [10] IBM AIX 6.1 Documentations.
- [11] HP HP-UX 11i Documentations.

反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：（010）88254396；（010）88258888

传 真：（010）88254397

E-mail: dbqq@phei.com.cn

通信地址：北京市万寿路173信箱

电子工业出版社总编办公室

[G e n e r a l I n f o r m a t i o n]

书名= U N I X从入门到精通

作者= 邢国庆，庞俊华，陈智建编著

页数= 4 6 6

出版社= 北京市：电子工业出版社

出版日期= 2 0 1 0 . 0 3

S S号= 1 2 4 9 2 6 7 7

D X号= 0 0 0 0 0 6 8 6 9 3 2 1

URL= 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 8 6 9 3 2 1 & d = C B D 0 7 9 2 F 8 0 E E B 8 0 0 2
4 2 3 4 6 F E 9 3 2 B F 0 B C

第 1 章	U N I X 概述与安装	
	1 . 1 U N I X 早期发展过程概述	
	1 . 1 . 1 U N I X 的缘起	
	1 . 1 . 2 U N I X 的交替发展	
	1 . 1 . 3 U N I X 的战国时代	
	1 . 1 . 4 策略与标准之争	
	1 . 1 . 5 U N I X 的黑暗时期	
	1 . 1 . 6 A T & T U N I X S y s t e m V R e l e a s e	
4 . 0	1 . 1 . 7 后 U N I X 时代	
	1 . 2 U N I X 的层次组织结构	
	1 . 3 U N I X 的逻辑组织结构	
	1 . 3 . 1 进程管理子系统	
	1 . 3 . 2 内存管理子系统	
	1 . 3 . 3 文件管理子系统	
	1 . 3 . 4 I / O 管理子系统	
	1 . 3 . 5 硬件系统	
	1 . 4 安装 S o l a r i s 操作系统	
	1 . 4 . 1 硬件要求	
	1 . 4 . 2 安装步骤	
第 2 章	命令行基础知识	
	2 . 1 命令行结构	
	2 . 2 后台进程	
	2 . 3 标准输入、标准输出与标准错误输出	
	2 . 4 输入输出重定向	
	2 . 5 管道	
	2 . 6 元字符与文件名生成	
	2 . 7 转义与引用	
	2 . 8 命令历史	
	2 . 8 . 1 f c 命令	
	2 . 8 . 2 h i s t o r y 命令	
	2 . 8 . 3 重复执行先前的命令	
	2 . 8 . 4 编辑并执行校正后的命令	
	2 . 8 . 5 命令行补充	
	2 . 9 命令别名	
	2 . 1 0 作业控制	
	2 . 1 1 会话记录	
	2 . 1 2 使用 m a n 命令查询系统参考手册	
第 3 章	文件系统基础知识	
	3 . 1 文件系统的层次结构	
	3 . 1 . 1 树形层次结构	
	3 . 1 . 2 路径名	
	3 . 2 文件系统的组织结构	
	3 . 3 文件的类型	
	3 . 3 . 1 普通文件	
	3 . 3 . 2 目录文件	
	3 . 3 . 3 特殊文件	
	3 . 3 . 4 链接文件	
	3 . 3 . 5 符号链接文件	
	3 . 3 . 6 管道文件	

	3 . 4 文件的安全保护机制
	3 . 4 . 1 显示文件的访问权限
	3 . 4 . 2 修改文件的访问权限
	3 . 4 . 3 设置文件的访问权限
	3 . 4 . 4 其他访问权限设置
第 4 章	文件和目录操作
	4 . 1 创建文件
	4 . 2 显示文件列表
	4 . 2 . 1 使用 l s 命令显示文件列表
	4 . 2 . 2 利用通配符显示文件
	4 . 2 . 3 显示隐藏文件
	4 . 2 . 4 递归显示目录与文件
	4 . 3 显示文件内容
	4 . 3 . 1 使用 c a t 命令显示文件
	4 . 3 . 2 使用 m o r e 命令分页显示文件
	4 . 3 . 3 使用 h e a d 命令显示文件前几行内容
	4 . 3 . 4 使用 t a i l 命令显示文件最后几行内容
	4 . 4 复制文件
	4 . 5 移动文件
	4 . 6 删除文件
	4 . 7 显示当前工作目录
	4 . 8 改换目录
	4 . 9 创建目录
	4 . 1 0 移动目录
	4 . 1 1 复制目录
	4 . 1 2 删除目录
	4 . 1 3 比较文件之间的差别
	4 . 1 3 . 1 使用 d i f f 命令比较两个文件
	4 . 1 3 . 2 使用 d i f f 3 命令比较三个文件
	4 . 1 4 从系统中检索文件
	4 . 1 4 . 1 简单检索
	4 . 1 4 . 2 使用逻辑运算符
	4 . 1 4 . 3 利用 f i n d 命令本身实现其他处理功能
	4 . 1 4 . 4 利用管道实现其他处理功能
	4 . 1 5 检索文件内容
	4 . 1 5 . 1 利用 g r e p 检索文件内容
	4 . 1 5 . 2 过滤其他命令的输出数据
	4 . 1 5 . 3 使用 g r e p 检索多个文件
	4 . 1 5 . 4 检索不包含特定字符串的文本行
	4 . 1 5 . 5 在 g r e p 中使用正则表达式
	4 . 1 5 . 6 检索元字符本身
	4 . 1 5 . 7 在命令行中使用引号
	4 . 1 6 排序
第 5 章	编辑文件
	5 . 1 启动 v i 编辑器
	5 . 1 . 1 创建文件
	5 . 1 . 2 状态行
	5 . 2 v i 编辑器的两种工作模式
	5 . 2 . 1 输入模式
	5 . 2 . 2 命令模式
	5 . 3 保存编辑的文件并退出 v i
	5 . 4 v i 编辑器的基本命令
	5 . 4 . 1 移动光标位置
	5 . 4 . 2 输入文本

	5 . 4 . 3 修改与替换文本
	5 . 4 . 4 撤销先前的修改
	5 . 4 . 5 删除文本
	5 . 4 . 6 复制、删除与粘贴文本
	5 . 4 . 7 重复执行指定次数的命令
	5 . 5 使用 e x 命令
	5 . 5 . 1 显示行号
	5 . 5 . 2 多行复制
	5 . 5 . 3 移动文本行
	5 . 5 . 4 删除文本行
	5 . 6 检索与替换
	5 . 6 . 1 检索字符串
	5 . 6 . 2 模式检索
	5 . 6 . 3 替换字符串
	5 . 7 编辑多个文件
	5 . 7 . 1 编辑多个文件
	5 . 7 . 2 合并文件与合并文本行
	5 . 8 定制 v i 编辑器的运行环境
	5 . 8 . 1 临时设定 v i 的运行环境
	5 . 8 . 2 永久性地定制 v i 的运行环境
	5 . 9 其他特殊说明
	5 . 9 . 1 删除或替换特殊字符
	5 . 9 . 2 在编辑期间运行 U N I X 命令
	5 . 1 0 v i 编辑器命令总结
第 6 章	S h e l l 基础知识
	6 . 1 s h e l l 与 S h e l l 脚本
	6 . 1 . 1 为什么需要 S h e l l 编程
	6 . 1 . 2 什么是 S h e l l 脚本
	6 . 1 . 3 运行 S h e l l 脚本
	6 . 1 . 4 退出与出口状态
	6 . 1 . 5 调用适当的 S h e l l 解释程序
	6 . 1 . 6 位置参数
	6 . 2 变量与变量替换
	6 . 2 . 1 变量分类
	6 . 2 . 2 变量赋值
	6 . 2 . 3 内部变量
	6 . 2 . 4 变量引用与替换
	6 . 2 . 5 变量的间接引用
	6 . 2 . 6 特殊的变量替换
	6 . 2 . 7 变量声明与类型定义
	6 . 3 命令与命令替换
	6 . 3 . 1 S h e l l 内部命令
	6 . 3 . 2 部分命令介绍
	6 . 3 . 3 命令替换
	6 . 4 t e s t 语句
	6 . 4 . 1 文件测试运算符
	6 . 4 . 2 字符串测试运算符
	6 . 4 . 3 整数值测试运算符
	6 . 4 . 4 逻辑运算符
	6 . 5 命令行的解释执行过程
	6 . 5 . 1 读取命令行
	6 . 5 . 2 命令历史替换
	6 . 5 . 3 命令别名替换
	6 . 5 . 4 花括号扩展

	6 . 5 . 5 波浪号替换
	6 . 5 . 6 I / O重定向
	6 . 5 . 7 变量替换
	6 . 5 . 8 算术运算结果替换
	6 . 5 . 9 命令替换
	6 . 5 . 1 0 单词解析
	6 . 5 . 1 1 文件名生成
	6 . 5 . 1 2 引用字符处理
	6 . 5 . 1 3 进程替换
	6 . 5 . 1 4 环境处理
	6 . 5 . 1 5 执行命令
	6 . 5 . 1 6 跟踪执行过程
	6 . 5 . 1 7 实例验证
第 7 章	S h e l l 高级编程
	7 . 1 i f 条件语句
	7 . 1 . 1 i f 语句的表现形式
	7 . 1 . 2 嵌套的 i f - t h e n 条件测试
	7 . 2 c a s e 分支语句
	7 . 3 f o r 循环语句
	7 . 4 w h i l e 循环语句
	7 . 5 u n t i l 循环语句
	7 . 6 s e l e c t 循环语句
	7 . 7 嵌套的循环
	7 . 8 循环控制与辅助编程命令
	7 . 8 . 1 b r e a k 和 c o n t i n u e 命令
	7 . 8 . 2 t r u e 命令
	7 . 8 . 3 s l e e p 命令
	7 . 8 . 4 s h i f t 命令
	7 . 8 . 5 g e t o p t 命令
	7 . 8 . 6 g e t o p t s 命令
	7 . 9 循环语句的 I / O 重定向
	7 . 9 . 1 w h i l e 循环的 I / O 重定向
	7 . 9 . 2 u n t i l 循环的 I / O 重定向
	7 . 9 . 3 f o r 循环的 I / O 重定向
	7 . 1 0 h e r e 文档
	7 . 1 1 S h e l l 函数
	7 . 1 2 逻辑与和逻辑或并列结构
	7 . 1 2 . 1 逻辑与命令并列结构
	7 . 1 2 . 2 逻辑或命令并列结构
	7 . 1 3 S h e l l 数组
	7 . 1 4 信号的捕捉与处理
	7 . 1 5 其他 S h e l l 课题
	7 . 1 5 . 1 子 S h e l l
	7 . 1 5 . 2 S h e l l 脚本的调试
	7 . 1 5 . 3 系统性能考虑
第 8 章	进程管理
	8 . 1 p s 命令概述
	8 . 2 查询进程及其状态信息
	8 . 2 . 1 查询当前活动的进程
	8 . 2 . 2 查询系统中所有的进程
	8 . 2 . 3 显示进程的重要状态信息
	8 . 2 . 4 显示进程的详细状态信息
	8 . 3 监控进程及系统资源
	8 . 4 终止进程的运行

- 8 . 5 调整进程的调度类别及优先级
 - 8 . 5 . 1 显示进程的调度类别与优先级
 - 8 . 5 . 2 按照指定的调度类别与优先级运行进程
 - 8 . 5 . 3 调整进程的调度类别与优先级
 - 8 . 5 . 4 设置实时进程的时间片
- 8 . 6 调整分时进程的优先级
 - 8 . 6 . 1 n i c e 命令
 - 8 . 6 . 2 r e n i c e 命令
 - 8 . 6 . 3 调整进程优先级的作用
- 8 . 7 定时运行系统任务和用户程序
 - 8 . 7 . 1 c r o n 守护进程的调度过程
 - 8 . 7 . 2 调度定时重复执行的任务
 - 8 . 7 . 3 提交一次性定时执行的任务
- 8 . 8 调度重复执行的任务
 - 8 . 8 . 1 c r o n t a b 的工作原理
 - 8 . 8 . 2 创建和编辑 c r o n t a b 文件
 - 8 . 8 . 3 显示 c r o n t a b 文件
 - 8 . 8 . 4 删除 c r o n t a b 文件
 - 8 . 8 . 5 c r o n t a b 命令的访问控制
 - 8 . 8 . 6 应用实例——数据库定时备份
- 8 . 9 调度一次性执行的作业
 - 8 . 9 . 1 提交 a t 作业
 - 8 . 9 . 2 显示 a t 作业及作业队列
 - 8 . 9 . 3 删除 a t 作业
 - 8 . 9 . 4 a t 命令的访问控制
 - 8 . 9 . 5 应用实例——系统定时关机

第 9 章 用户管理

- 9 . 1 增加与删除用户
 - 9 . 1 . 1 / e t c / p a s s w d 文件
 - 9 . 1 . 2 / e t c / s h a d o w 文件
 - 9 . 1 . 3 用户管理实例
- 9 . 2 定制用户的工作环境
 - 9 . 2 . 1 选择命令解释程序
 - 9 . 2 . 2 设置用户初始化文件
 - 9 . 2 . 3 定制 S h e l l 工作环境
- 9 . 3 增加与删除用户组
- 9 . 4 监控用户
 - 9 . 4 . 1 利用 w h o 命令查询系统中的用户
 - 9 . 4 . 2 利用 f i n g e r 命令查询系统中的用户
 - 9 . 4 . 3 利用 w 命令查询系统中的用户活动
 - 9 . 4 . 4 向注册用户发送消息
- 9 . 5 以不同用户的身份访问系统

第 1 0 章 软件包的制作与管理

- 1 0 . 1 软件包组成简介
 - 1 0 . 1 . 1 基本组成部分
 - 1 0 . 1 . 2 选用的信息文件
 - 1 0 . 1 . 3 选用的 S h e l l 脚本文件
- 1 0 . 2 软件包的相关文件和命令
 - 1 0 . 2 . 1 p k g i n f o 文件
 - 1 0 . 2 . 2 p r o t o t y p e 文件
 - 1 0 . 2 . 3 p k g m a p 文件
 - 1 0 . 2 . 4 c o p y r i g h t 文件
 - 1 0 . 2 . 5 d e p e n d 文件
 - 1 0 . 2 . 6 s p a c e 文件

	10.2.7	compver 文件
	10.2.8	软件包的相关工具
	10.3	制作软件包
	10.3.1	制作软件包的步骤
	10.3.2	创建 pkginfo 文件
	10.3.3	利用 pkgproto 命令创建 prototype 文件
	10.3.4	利用 pkgmk 命令制作软件包
	10.3.5	pkgtrans 命令
	10.4	安装软件包
	10.5	查询软件包
	10.6	检测软件包
	10.7	卸载软件包
第 11 章		磁盘空间管理
	11.1	查询磁盘空间信息
	11.1.1	常用的磁盘空间管理工具
	11.1.2	使用 df 命令检查存储空间的使用情况
	11.1.3	使用 du 命令检查存储空间占用情况
	11.1.4	使用 quota 命令查询每个用户占用的存储空间
	11.1.5	使用 find 命令找出超大容量的文件
	11.1.6	使用 find 命令找出长期闲置的文件
	11.1.7	使用 find 命令找出并删除 core 文件
	11.1.8	使用 ls 命令检测文件的大小
	11.2	采用标准工具备份与恢复数据
	11.2.1	利用 cpio 实现备份和恢复
	11.2.2	利用 tar 实现备份和恢复
	11.2.3	利用 dd 实现数据的复制
	11.3	文件系统限额管理
	11.3.1	限额概述
	11.3.2	设置限额
	11.3.3	限额的维护
第 12 章		TCP / IP 网络管理
	12.1	TCP / IP 简介
	12.1.1	TCP / IP 协议的层次结构
	12.1.2	TCP / IP 协议如何处理数据通信
	12.2	网络接口设置
	12.3	主机名字解析
	12.4	网络路由设置
	12.4.1	静态路由
	12.4.2	动态路由
	12.5	配置网络服务
	12.6	网络管理与维护
	12.6.1	使用 ifconfig 命令维护网络接口
	12.6.2	使用 netstat 命令监控网络状态
	12.6.3	使用 ping 命令测试远程主机的连通性
	12.6.4	使用 ftp 命令检测网络主机的传输性能
	12.6.5	使用 traceroute 命令跟踪路由信息
第 13 章		TCP / IP 网络应用
	13.1	OpenSSH
	13.1.1	sshd __config 配置文件
	13.1.2	ssh__config 配置文件
	13.1.3	使用 SSH 注册到远程系统
	13.1.4	使用 SSH 执行远程系统中的命令
	13.1.5	使用 SCP 替代 FTP
	13.1.6	使用 SFTP 替代 FTP

	1 3 . 1 . 7	S S H与S C P的无密码注册
	1 3 . 1 . 8	O p e n S S H的安全考虑
1 3 . 2	T e l n e t	远程系统注册
1 3 . 3	F T P	文件传输
	1 3 . 3 . 1	连接F T P服务器
	1 3 . 3 . 2	F T P应用
	1 3 . 3 . 3	F T P访问控制
	1 3 . 3 . 4	F T P自动注册
第 1 4 章	网络文件系统	
	1 4 . 1	N F S简述
	1 4 . 2	配置N F S服务器
	1 4 . 3	配置N F S客户系统
	1 4 . 3 . 1	安装远程文件系统
	1 4 . 3 . 2	设置 / e t c / v f s t a b 文件
	1 4 . 4	N F S自动安装
	1 4 . 4 . 1	主映射文件
	1 4 . 4 . 2	直接映射文件
	1 4 . 4 . 3	间接映射文件
第 1 5 章	系统启动与关机	
	1 5 . 1	磁盘分区与初始引导
	1 5 . 1 . 1	磁盘分区
	1 5 . 1 . 2	初始引导过程
	1 5 . 1 . 3	系统初始化
	1 5 . 2	i n i t 进程与系统生成
	1 5 . 2 . 1	运行级
	1 5 . 2 . 2	/ e t c / i n i t t a b 文件
	1 5 . 2 . 3	处理方式
	1 5 . 2 . 4	/ e t c / i n i t t a b 文件举例
	1 5 . 2 . 5	启动用户定义的应用程序
	1 5 . 3	用户注册过程
	1 5 . 3 . 1	用户注册的处理过程
	1 5 . 3 . 2	u t m p x 和 w t m p x 文件
	1 5 . 4	系统关机过程
	1 5 . 4 . 1	使用 s h u t d o w n 命令关闭系统
	1 5 . 4 . 2	使用 i n i t 命令关闭系统
	1 5 . 4 . 3	使用其他命令关机
	1 5 . 5	应用实例
第 1 6 章	文件系统内部组织	
	1 6 . 1	文件系统的组织结构
	1 6 . 2	超级块
	1 6 . 3	信息节点
	1 6 . 3 . 1	特权标志位
	1 6 . 3 . 2	数据块地址数组
	1 6 . 4	数据区与空闲数据存储块的组织
	1 6 . 5	信息节点的分配与释放
	1 6 . 6	数据块的分配与释放
	1 6 . 7	信息节点与目录和文件的关系
	1 6 . 8	U F S文件系统
	1 6 . 8 . 1	U F S文件系统的组织结构
	1 6 . 8 . 2	引导块
	1 6 . 8 . 3	超级块
	1 6 . 8 . 4	柱面组信息块
	1 6 . 8 . 5	信息节点区与信息节点
	1 6 . 8 . 6	数据块区

	1 6 . 8 . 7 数据块的分配与释放过程
	1 6 . 8 . 8 信息节点的分配与释放过程
第 1 7 章	文件系统管理
	1 7 . 1 创建文件系统
	1 7 . 1 . 1 使用m k f s 命令创建U F S文件系统
	1 7 . 1 . 2 使用n e w f s 命令创建文件系统
	1 7 . 2 使用l a b e l i t 命令命名文件系统
	1 7 . 3 安装、卸载文件系统
	1 7 . 3 . 1 安装文件系统
	1 7 . 3 . 2 / e t c / v f s t a b 文件
	1 7 . 3 . 3 安装文件系统
	1 7 . 3 . 4 卸载文件系统
	1 7 . 4 确定文件系统的类型
	1 7 . 5 检测与修复文件系统
	1 7 . 5 . 1 何时需要检测文件系统
	1 7 . 5 . 2 文件系统检测的内容
	1 7 . 5 . 3 交互检测与修复U F S文件系统
	1 7 . 5 . 4 自动检测与修复U F S文件系统
	1 7 . 5 . 5 恢复严重受损的超级块
	1 7 . 5 . 6 解决f s c k 命令无法修复的U F S文件系统问题
	1 7 . 5 . 7 f s c k 的阶段处理方式
	1 7 . 5 . 8 利用其他工具修复文件系统
参考文献	