Linux 系统管理员手册

0.6.1 版

Lars Wirzenius

liw@iki.fi

翻译：赵炯

gohigh@sh163.net

(gohigh@shtdu.edu.cn)

www.plinux.org

www.oldlinux.org

Linux 系统管理者手册：0.6.1 版
Lars Wirzenius 著

Linux 系统新手的系统管理概述。

翻译者 赵炯 gohigh@sh163.net

致谢

这里用于将来的致谢部分

原程序和预格式化的其他版本

本书的原程序及其它可读格式可以在 internet 上通过 anonymous FTP 在 Linux 文档计划的主页http://sunsite.unc.edu/LDP找到，或者从本书的主页http://www.iki.fi/liw/linux/sag/上
得到。本书起码还有 Postscript、Tex 及.DVI 等格式。

目录

第1章 绪论

“刚开始，本文是无格式的，自由的；以及在少量地方是空洞的。在作者的指尖不停地在键盘上移动的时候，作者说道，这里写些字吧，于是这里就有了文字。”

本手册，Linux 系统管理者手册，描述了使用 Linux 的系统管理方面。它适合于那些对系统管理一无所知的人（如在“这是什么？”），但起码已经掌握了常规使用的基本方面的人。本手册也没有告诉你如何安装 Linux；这部分的描述在安装与初学文档中。参见如下有关 Linux 文档的更多信息。

系统管理是一个人用于维持一个计算机系统在可用状态所需要的一切。它包括诸如备份文件（以及恢复文件）、安装新程序、为用户创建帐号（以及删除它们如果不再需要它们的话）、确定文件系统是否崩溃、等等。如果将计算机比作为一间屋子，系统管理则可称为维护，并可包括清理、修补坏了的窗户，以及其他诸如此类的事情。系统管理并没有称作维护，因为那也太简单了。[1]

本手册的结构是这样的，许多章节都可以单独的使用，所以，如果你需要有关比如说是备份的信息，你可以只阅读有关备份的哪个章节。这使得本书可以成为一本参考手册。然而，本手册最初仅仅是一本指南以及一个参考手册而已。

本手册并不适用于完全独立地使用。许多其它的 Linux 文档对系统管理者也同样的重要。总之，一个系统管理者就是一个拥有特权和职责的用户。一个非常重要的资源就是手册页（manual pages）,当遇到不熟悉的命令时应总是要参考它的。

虽然本手册是针对 Linux 的，它对基于 UNIX 的其它操作系统也同样是有用的。不幸的是，现在有太多的 UNIX 的不同的版本的存在，尤其在系统管理方面，想面面俱到是徒劳的。由于 Linux 开发的自然规则，甚至想包揽 Linux 的所有方面也是件困难的事。

并不存在 Linux 官方发行版，所以不同的人有不同的设置，许多人有自己建立的设置。尽管本书不是针对任何一个特定的发行版的，但我却只使用 Debian GNU/Linux 系统。如果可能的话，我会指出不同点，以及解释其中的变换处。

我已尝试着描述事情是如何工作的，而不仅仅是对每个任务列出“五个简单步骤”。这意味着并不是每个人都需要所有这些信息资料的，这些部分都作了标记并且如果你使用的是一个预设置的系统话，可以跳过这些部分。然而，阅读所有的信息将有助于你理解系统并且更易于使用和管理系统。

正如所有其它的与 Linux 相关的开发，这个工作的完成是基于自愿者的原则的：我写了这本书因为我觉得这是个乐趣，因为我觉得应该写这本书。然而，正如所有的自愿者的工作一样，我能够付出多大的努力来做这项工作是有个限度的，同样也在于我的知识和经验的多少。也就是说本手册并没有如此地好，如同一个收入盛丰的奇才来写作并且化上数年来完善它。当然，我想这样是非常好的事，但是不必的。

特别地，我在一点上走了捷径，那就是我没有十分彻底的包罗所有方面，这些方面早已很好地包含于其它一些有用的手册中。特别是关于程序方面的文档，诸如有关使用 mkfs 的详细资料等。我只是就本手册的需要描述了程序的用途和目的。有关更详细的信息，我请可爱的读者参考这些其它的手册。通常，所有参考的文档都是完整 Linux 文档集的一部分。

虽然我已使这本手册尽可能地好，我还是真诚地希望能从你这里听到使这本手册变的更好的任何建议。咒骂、实际错误、写作新方面的建议、章节的重写、各种 UNIX 版本是如何工作的的有关信息，我都感兴趣。我的联系信息在http://www.iki.fi/liw/mail-to-lasu.html上有。

在写作本书的过程中，有许多人帮助了我，直接地或是间接地。我要特别地感谢 Matt Welsh 的灵感和 LDP 的领导能力，Andy Oram 的使我又继续工作的有价值的反馈信息，Olaf Kirch 的鼓励我写作，Yggdrasil 的 Adam Richter 让我认识到其他人对本手册的兴趣。

Stephen Tweedie, H.~Peter Anvin, R\'emy Card, Theodore Ts'o, 以及 Stephen Tweedie 让我借助于他们的工作（因此使得本书看上去更厚了更有吸引力了）：xia 与 ext2 文件系统的比较、设备列表以及 ext2 文件系统的描述。这些已不再是本书的部分了。我对此非常感激，并且对于早期版本有时缺乏适当的著明出处感到非常抱歉。

另外，我要感谢 Mark Komarinski 他在 1993 年给了我他的资料以及 Linux 刊物中的许多系统管理专栏资料。这些资料是非常有价值的和具有启发性的。

还有许多人给了我很多有用的建议。我的小小的文件不允许我在这里列出所有人的名字，但其中一些人，按照字母的顺序，是 Paul Caprioli, Ales Cepek, Marie France Declerfayt, Dave Dobson, Olaf Flebbe, Helmut Geyer, Larry Greenfield 以及他的父亲、Stephen Harris, Jyrki Havia, Jim Haynes, York Lam, Timothy Andrew Lister, Jim Lynch, Michael J. Micek, Jacob Navia, Dan Poirier, Daniel Quinlan, Jouni K Seppthen, Philippe Steindl, G.B.\ Stotte. 对于我所遗忘的人我表示抱歉。

META 需要加上印刷上的惯例和 LDP 的广告如下。

Linux 文档计划

Linux 文档计划，简称 LDP，是一个松散的在一起工作的作者、校对者和编辑团体，旨在为 Linux 操作系统提供完整的文档。该计划的总协调人是 Greg Hankins.

该手册是 LDP 发布的手册集中的一本，该集中包括 Linux 用户手册、系统管理者手册、网络管理者手册以及核心黑客手册。这些手册以 source 格式、.dvi 格式以及 postscript output 格式在 sunsite.unc.edu 的 FTP 的/pub/Linux/docs/LDP 上均有。

我们鼓励任何有写作和编辑兴趣的任何人加入我们来改进 Linux 文档。如果你能使用 internet e-mail，你可以用gregh@sunsite.unc.edu与 Greg Hankins 联系。

注释

[1] 有些人确实是这样来称呼它的，但那是因为他们没有看过这本手册，真遗憾。

第2章 Linux 系统的概述

“上帝查看着他所创造的一切，并且看上去都很好” （起源 1：31）

这章给出了 Linux 系统的一个概述。首先，描述了所提供的主要功能和服务。其次，概要地叙述了实现这些服务的程序。本章的目的是给出对系统作为一个整体的理解，各个部分的详细描述在其他章节中。

2.1 操作系统的各个部分

一个 UNIX 操作系统由一个内核以及一些系统程序组成。当然还有一些用于工作的应用程序。内核是操作系统的心脏。[①] 它维护着磁盘上文件的轨迹，启动程序并且并行地运行着它们，给各种进程分配内存以及其他的资源，从网络上接收和发送数据包等等。内核自己做的事很少，但它提供了建立所有服务所需的工具。它也防止了任何人直接对硬件的访问，迫使任何人使用它所提供的工具。这样，内核为各个用户之间提供了一些保护。内核所提供的工具是通过系统调用（system calls）来使用的；有关更多的信息请参见 manual page 第二部分。

系统程序利用内核所提供的工具来实现一个操作系统所需的各种服务。系统程序，以及所有其他程序，运行于“内核之上”，也即所谓的用户模式（user mode）。系统程序和应用程序之间的区别在于它们专注的方面：应用程序主要用于将有用的事情做好（或用于游戏，如果它正好是一个游戏程序），而系统程序是用来使系统正常地工作。一个字处理程序是一个应用程序；而 telnet 是一个系统程序。这些区别常常是模糊不清的，然而，这仅仅对于强迫分类来说是重要的。

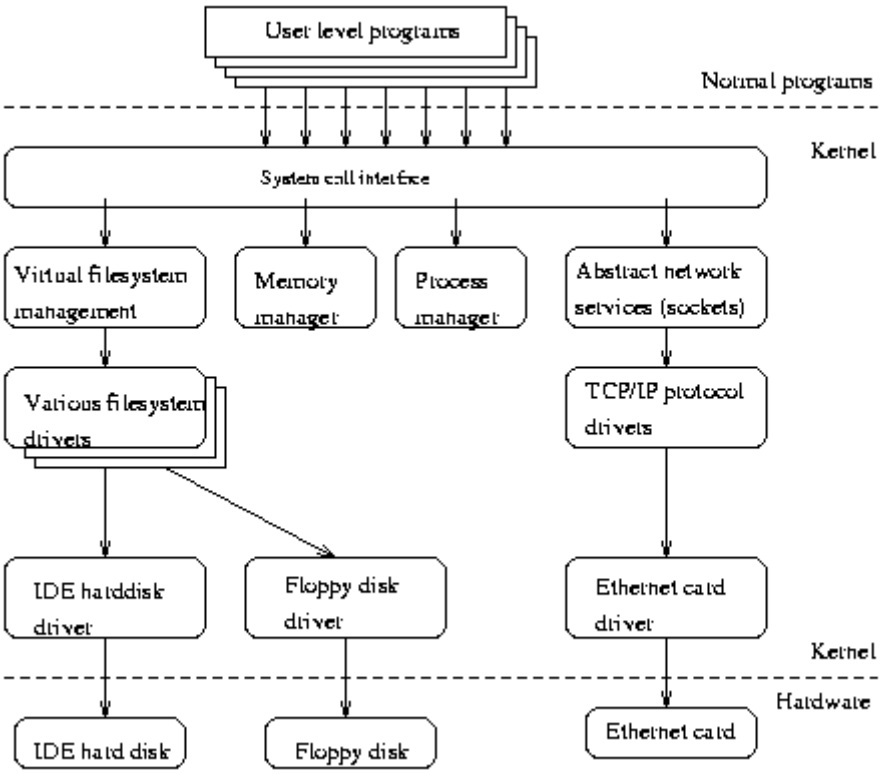
一个操作系统也可包含一些编译程序以及相应的库程序（在 Linux 下由指 GCC 以及 C 库程序），尽管不是所有的编程语言必需成为操作系统的一部分。文档，以及有时甚至一些游戏，也能算是操作系统的一部分。传统上来说，操作系统是由它的安装磁带或磁盘上的内容来定义的；对于 Linux 来讲，就不是那么清楚了，因为它散布在世界上很多的 FTP 站点上。

2.2 内核中的重要部分

Linux 内核由几个重要的部分组成：进程管理、内存管理、硬件设备驱动程序、文件系统驱动程序、网络管理以及各种其它部分。图 2-1 显示出了其中的一些。

图 2-1 Linux 内核中非常重要的部分中的一些

[①]实际上，它常被错误地当成自身就是操作系统，但并不是这样的。一个操作系统要比一个平凡的内核提供更多的服务。



也许内核中最重要的部分（没有它们就不能工作）是内存管理以及进程管理。内存管理用来给进程分配内存区域和分配交换空间区域，内核的部分，以及用于高速缓冲。进程管理用于创建进程，以及通过在处理器上交换活动进程来实现多任务的功能。

在最低层，内核包含每种它支持的硬件的设备驱动程序。由于世界充满了各种各样不同种类的硬件，硬件设备驱动程序的数量是庞大的。有许多在一些方面相似的硬件仅仅在软件是如何控制它们方面是有所区别的。这种相似性使得支持相同操作的驱动程序通用类成为可能；类的每个成员对于内核的其余部分来说有着同样的接口，但在实现它们方面有所不同。例如，所有的磁盘驱动程序对于内核的其它部分来说看上去是相同的，也即，它们都有如‘初始化驱动器’、‘读扇区 N’、‘写扇区 N’的操作。

有些内核提供的软件服务本身就有着相似的属性，并且因此可以抽象成类。例如，各种网络协议被抽象成一种编程接口，BSD 套接字库。另一个例子是虚拟文件系统（virtual filesystem）(VFS)层将其文件系统操作从其实现中抽象出来。每一种文件系统类型提供了各自文件系统操作的实现。当某个实体试尝使用一个文件系统时，该请求通过 VFS，发送到相应文件系统驱动程序中。

2.3 UNIX系统的主要服务工作

本节描述了一些非常重要的 UNIX 服务，但并不是很详细的。它们将在其它章节中更彻底地讨论。

2.3.1 初始化 Init

UNIX 系统中的一个非常重要的服务是由 init 提供的。Init 是每个 UNIX 系统第一个启

动的进程，作为内核引导所做的最后一件事。当 init 启动后，它通过执行各种启动事务来继续引导进程（检查并监视文件系统，启动后台程序 daemons,等等）。

Init 所做事情的精确列表依靠它是以何种方式；有几种方式可供选择。Init 通常提供单用户模式的方式,在该方式中除了用户 root 能在控制台使用一个 shell 外其它任何人都不能登录；常用的模式称为多用户模式(multiuser mode)。有些方式将此概括为运行层（run levels）；单用户和多用户模式被当作两种运行层，并且还可以有其它的层，例如，在控制台上运行 X。

在正常的操作下，init 确定 getty 正在工作着（用以允许用户登录），并且收取孤立进程（父辈进程已结束的进程；在 UNIX 中所有的进程必须属于单棵进程树，所以孤立进程必须被收取）。

当系统关闭时，init 负责杀死所有其它的进程，卸载所有的文件系统以及停止处理器的工作，以及任何它被配置成要做的工作。

2.3.2 从终端登录

从终端登录（通过串行线）以及从控制台登录（当没有运行 X 时）是有 getty 程序提供的。Init 为每个终端启动一个独立的 getty 实例以允许登录操作。Getty 读入用户名并且运行 login 程序，login 程序读入口令。如果用户名以及口令正确的话，login 就运行 shell 程序。当 shell 程序终止时，也即，用户退出时，或者 login 程序因用户名以及口令不匹配而终止时，init 注意到这点并启动一个新的 getty 实例。内核是不知道 login 的，这都是由系统程序来处理的。

2.3.3 系统日志 Syslog

内核及许多系统程序会产生出错、警告以及其他一些消息。以后甚至更晚些时候来观察这些消息常常是很重要的，因此这些消息应被写入一个文件中去。做这个事情的程序是 syslog。根据写入者或消息的重要性，它可以配置成对消息进行分类并写入不同的文件中去。例如，内核消息常常导入到一个独立的文件中，因为内核消息常常是非常重要的，并用来确定问题的所在。

2.3.4 定期命令执行：cron 和 at

用户和系统管理员常常需要定时地执行某些命令。例如，系统管理员想要运行一个命令来清理临时文件目录（/tmp 以及 /var/tmp）中的旧文件，以防止磁盘过满，因为并非所有程序在其后都会正确地清理。

Cron 服务即用来做此事的。每个用户都有一个 crontab 文件，在这个文件中他列出了他想执行的命令和这些命令被执行的时间。Cron 后台程序（守护程序）会在指定的时间启动命令。

At 服务与 cron 相类似，但它只执行一次：命令在指定时间被执行，但并不会重复执行。

2.3.5 图形用户界面

UNIX 和 Linux 并没有把用户界面（接口）集成到内核中去；而是由用户层程序来实现的。这对文本模式和图形模式环境是一样的。

这种安排使得系统非常灵活，但也有缺点，为每个程序编制一个不同的用户界面是非常简单的，这使得系统更难于学习。

Linux 中启初使用的图形环境称作 X 窗口系统（简称 X）。X 也并没有实现一个用户接口；它只实现了一个窗口系统， 也即，实现图形用户接口的一组工具。在 X 上实现的三种非常流行的用户接口形式是 Athena、Motif 以及 Open Look。

2.3.6 网络

连网是连接两台或多台计算机以使得它们能互相通信的一种活动。连接与通信的实际方法稍嫌复杂，但其最终结果却是非常有用的。

UNIX 操作系统有许多的连网特征。许多基本服务（文件系统、打印服务、备份等等）能够在网络上完成。这能使得系统管理简单化，因为这允许集中管理而仍然有着微机化以及分布计算的益处，比如低成本化和很好的容错性能。

然而本书仅仅对连网粗率一述；详细信息请参见 **Linux 网络管理员手册**，包括网络是如何工作的基本描述。

2.3.7 网络登录

网络登录的工作原理与通常的登录稍有区别。每个终端都有一独立的串行线，通过它即可进行登录操作。而对于每个通过网络登录的人来讲，存在着一个独立的虚拟网络连接，并且这些连接数是不限的。[1] 因此针对于每个虚拟连接而运行一个独立的 getty 是不可能的。通过网络来登录有几种不同的方法，telnet 和 rlogin 是 TCP/IP 网络中主要的方法。

与一群 gettys 相反，每种网络登录只有一个守护程序（telnet 和 rlogin 有各自的守护程序）听取所有的登录请求。当它识别到一个时，它就启动它的一个新的实例来处理这单个请求；原始实例将继续听取其它的登录请求。新的实例的工作原理同 getty 相类似。

2.3.8 网络文件系统

连网服务所带来的好处之一是通过*网络文件系统*共享文件。常用的一种称为网络文件系统，或 NFS，是由 Sun 公司研制开发的。

使用网络文件系统，一台机器上的程序所作的任何文件操作将通过网络送达另一台计算机。这使得这个程序误以为其它计算机上的所有文件好象是在该程序运行的计算机上。这使得信息的共享尤其得简单，因为它无须对程序作任何的修改。

2.3.9 邮件

电子邮件通常是经由计算机进行通信最重要的方法。一封电子信件使用一定的格式存

贮于一个文件中，并且由一个特定的邮件程序来发送和读取信件。

每个用户有一个接收邮件箱（一个特定格式的文件），所有的新邮件都存于此。当某个人发送了一个邮件，邮件程序就会定位到接收者的邮件箱并且将这封信添加到邮件箱文件中。如果接受者的邮件箱在另一台机器上，这封信就会发送到另一台机器上，该机器将会把这封信递送到最适合的邮件箱中去。

邮件系统由许多程序组成。到本地或远程邮件箱的邮件的递送是由一个程序来完成的（邮件传递代理或称 MTA，例如，sendmail 或 smail），而用户所使用的程序却是各种各样的（邮件用户代理或称 MUA，例如：pine 或 elm）。邮件箱通常存储于/var/spool/mail 中。

2.3.10 打印 Printing

在某个时刻只有一个用户能使用一台打印机，但是不在用户中共享打印机是不经济的。因此打印机被软件以打印队列的方式管理起来：所有的打印作业被放置于一个打印队列中并且当打印机完成了它的当前打印任务，下一个打印作业就会被自动地送达打印机。这就将用户从组织打印机队列以及争取打印机控制权的活动中解脱了出来。[2]

打印队列软件也能将打印输出到磁盘上，也即，当作业在队列中时文本被保存在一个文件中。这使得一个应用程序能够快速地输出打印作业到打印队列软件中；应用程序无须等待作业实际打印完才继续执行。这真是方便，因为这允许一个程序打印出一个版本，并且无须等待它打印出来而又制作出一个完全的修正新版本。

2.3.11 文件系统布局

文件系统可分成许多部分；通常和一个根（root）文件系统一起有/bin、/lib、/etc 以及一些其它目录；一个/usr 文件系统内含程序和不变的数据；一个/var 文件系统内含变化着的数据（如 log 文件）；以及一个包含每个人的个人文件的/home 文件系统。依赖于硬件的配置以及系统管理员的决策，文件系统的分布可以是不同的；它甚至全都合在一个文件系统中。

第三章稍详细地描述了文件系统的规划；Linux 文件系统标准更详细地描述了这些。

注释

[1] 其实，起码是可以有许多的。网络带宽仍然是不足的资源，通过一个网络连接的并发登录数仍然受限于一些实际的上限。

[2] 实际上，它们在打印机上形成了一个新的队列，等待它们被打印出来，因为没人使得队列软件能够确实地知道什么时候一个人的打印输出是确实已经完成了。这在办公室社会关系内确实是一个进步。

第3章 目录树概述

“两天之后，Pooh 出现了，交叉着双腿做着，摇摆着双腿，在那里，在他身旁，是四罐蜂蜜。。。”（A.A.\ Milne）

本章描述了基于 FSSTND 文件系统标准的标准 Linux 目录树的重要部分。略述了针对不同的目的将目录树分割成独立的（分离的）文件系统的常用方法并且给出了详细分割的动机。对分割的其它方法也进行了讨论。

3.1 背景

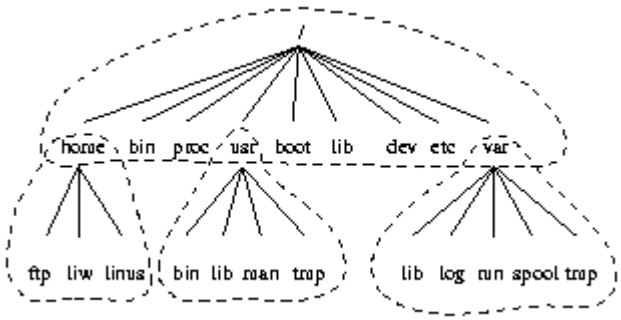
这一章松散地基于 Linux 文件系统标准，FSSTND，版本 1.2（见参考书目），它试图对如何在 Linux 系统中组织目录树设置一个标准。这样一个标准是有好处的，这将更容易地为 Linux 写出或移植软件，并且管理 Linux 机器，因为每件事都在它通常的地方。在这标准之后并没有权威来迫使每个人去遵守它，但是它以得到了绝大多数 Linux 发行的支持，如果不是所有的话。没有充分的理由而不遵守 FSSTND 并不是一个好主意。FSSTND 试图遵守 UNIX 传统以及当前的趋势，使得 Linux 系统对于有着其它 UNIX 系统经验的人也是熟悉的，反之也然。

本章并没有 FSSTND 来的更详细。一个系统管理员也应该通读 FSSTND 以期有完整的理解。

本章并没有详细解释所有的文件。目的不是描述每一个文件，而是从文件系统的观点给出一个系统概述。对于每个文件的更多的信息参见本手册的别的章节或 manual pages。

整个目录树是可以分割成更小的部分，每一部分可放在它自己的磁盘上或分区内，以利于适合磁盘大小的限制以及利于备份和其它一些系统管理。主要的目录部分有 root、/usr、/var 以及/home 文件系统（见图 3-1）。每个部分有不同的目的。目录树被设计成能够在 Linux 机器的网络中很好的工作，这些 Linux 机器可以在一个只读的设备上共享文件系统的某些部分（例如，一个 CD-ROM），或在一个 NFS 的网络上。

图 3-1. 一个 Unix 目录树的部分。虚线指示出分区的限制。



目录树不同部分的规则的描述如下。

。对于每台机器来说，根（root）文件系统是确定的（它通常是存储于一个本地磁盘上，尽管它可以是一个内存虚拟盘或是个网络驱动器）并且包含有引导系统启动以及将系统引导至一个能够加载其它文件系统的状态所必需的文件。此时，根（root）文件系统的内容对于单用户状态已经足够。它也将含有用于修复坏系统的工具，以及从备份中恢复丢失的文件的工具。

。/usr 文件系统包含所有的命令、库文件、manual pages 以及正常操作时所需的其他一些非改变文件。/usr 中所有的文件都应该是通用的而非特定地用于某台给定的机器。在正常操作中，所有的文件均不能被改变。这使得这些文件能够在网络上共享，这样就能够做到节约磁盘空间而有成本效益（/usr 中文件通常要占用几百兆的空间），并且能够更容易地管理（仅仅主 /usr 在升级一个应用程序是需要改变，而非各台机器）。甚至如果文件系统是在一个本地磁盘上，它可以以只读方式加载上来，以减少在系统崩溃时文件系统的毁坏的机会。

。/var 文件系统包含有会改变的文件，如 spool 目录（用于 mail、news、printers 等等）、log 文件、格式化的 manual pages 以及临时文件。传统上，/var 内的任何东西都可以从/usr 中找到，但那样的话，就不可能将/usr 以只读方式加载了。

。/home 文件系统包含用户的主目录，也即，系统上的所有实际数据。分离 home 目录到它自己的目录树或文件系统使得备份更容易；而其它部分常不必备份，或者说不常备份（它们很少改变的）。一个大的/home 也可以分割成几个文件系统，这需要在/home 下加上一个额外的命名层，例如，/home/students 以及/home/staff。

尽管上面将不同部分都称为文件系统，并不需要它们实质上是在分离的文件系统上。它们可以容易地被放置进一单个的文件系统中如果系统是一个小的单用户系统并且用户想使得事情简单化。目录树也可以以不同的方式划分成文件系统，这依赖于磁盘有多大，以及对于各种目的空间是如何分配的。然而，重要的部分是按照标准来命名；尽管我们可以把/var 和/usr 放在同一个分区上，/usr/lib/libc.a 和/var/log/messages 的命名必须可以工作，例如通过移动/var 到/usr/var，并且作一个从/var 到/usr/var 的符号连接。

Unix 系统文件结构以目的的不同来组合（groups）文件，也即，所有的命令都在一个地方，所有的数据文件在另一个地方，所有的文档在第三个地方，以次类推。另一种方法是根据文件所属程序来组合文件，也即，所有 Emacs 文件都将存于一个目录中，所有 TeX 存于另一个目录，依次类推。第二种方法所带来的问题是难于共享文件（程序目录常常包含有静态的、共享的、会改变的以及非共享的文件），并且有时甚至难于找到文件（例如，manual pages 分散于许多地方，使得 manual page 程序寻找这些文件变成了维护上的噩梦）。

3.2 根（root）文件系统

根文件系统一般应设置的小一些，因为它包含着重要的文件并且小一些的、不常修改的文件系统就有一个更好的机会不被毁坏掉。一个毁坏的根文件系统通常意味着系统将不能启动了，除非使用特殊方法（例如，从软盘启动），所以你不会去冒这个险的。

根目录通常不包含任何其他文件，除了可能会有些系统的标准启动引导用的影像文件，常称为/vmlinuz。所有其它文件都在根文件系统的子目录中：

/bin

在启动引导期间所需的命令，也可以被普通用户使用（大概在启动引导以后）。

/sbin

如同/bin，但该目录中的命令不是给普通用户使用的，尽管他们在需要和允许时可能可以使用。

/etc

该目录中存储了与机器有关的配置文件。

/root

用户 root 的主目录。

/lib

根文件系统中的程序所需的共享库文件。

/lib/modules

可加载的内核模块，尤其是那些当从灾难中恢复启动引导系统时所需的模块文件（例如，网络以及文件系统驱动程序）。

/dev

设备文件。

/tmp

临时文件。启动后程序运行时应使用/var/tmp，而非/tmp，因为前者可能在有更大的空间的磁盘上。

/boot

用于系统启动引导加载程序。例如，LILO。内核影像文件常常是放置在此的，而不是直接放在根目录下。如果有许多内核影像文件，该目录很容易变得很大，因此此时将其置于一个分离独立的文件系统更好些。另一个理由就是要确保内核影像文件被存于 IDE 磁盘的 1024 柱面以内。

/mnt

系统管理员为临时加载的加载点。程序不会自动地加载到/mnt 的。/mnt 可以再划分成子目录（例如，/mnt/dosa 可以是使用 MS-DOS 文件系统的软驱，/mnt/exta 可以是一个 ext2 文件系统）。

/proc，/usr，/var，/home

其它文件系统的加载点。

3.3 /etc目录

/etc 目录包含有许多的文件。有些文件将在下面给予讨论。对于其他一些文件，你应该确定它们是属于哪个程序的并且阅读所属程序的 manual page。许多网络配置文件也包含于/etc 目录中，它们在***网络管理员手册***中给予了讨论。

/etc/rc 或 /etc/rc.d 或 /etc/rc?.d

在系统启动时或当改变了运行层时所要运行的描述（scripts）文件或者是描述文件的目录。更多信息请参见有关 init 的章节。

/etc/passwd

用户数据库，所含的域有用户名、实际名称、该用户主目录、加密过的口令以及有关各个用户的其它一些信息。该文件的格式在/man{passwd} manual page 中有记载。

/etc/fdprm

软盘参数表。描述了不同软盘的格式样式。由 setfdprm 程序使用。有关信息请参阅 setfdprm 的 manual page。

/etc/fstab

列出了在系统启动时由 mount –a 命令自动加载的文件系统（在/etc/rc 或者同等的启动文件中）。在 Linux 中，该文件也包含由 swapon –a 自动使用的的交换区的信息。有关更详细的信息，请参阅第四章中加载与卸载小节以及 mount 的 manual page。

/etc/group

与/etc/passwd 相类似，但描述了组的信息而非用户信息。详细信息请参见 group 的 manual page。

/etc/inittab

init 的配置文件。

/etc/issue

getty 输出的在 login 提示之前的信息。通常含有一小段叙述或是欢迎使用系统的消息。内容由系统管理员决定。

/etc/magic

file 的配置文件。包含了基于 file 猜测出的文件类型的各种文件格式的描述。

/etc/motd

当日消息，在成功登录后将自动输出（显示）。其内容由系统管理员决定。常用于给所有用户送信息，比如计划中的关机时间警告消息。

/etc/mtab

当前加载的文件系统的列表。最初是由启动引导描述文件所设置，并由 mount 命令自动地更新。当需要一个已加载文件系统列表时被使用，例如，在 df 命令中使用。

/etc/shadow

安装了影子（shadow）口令软件的系统中的影子口令文件。影子口令将加密的口令从/etc/passwd 中移入/etc/shadow 中；后者只有 root 用户才能读取。这使得破解口令变的极其困难。

/etc/login.defs

login 命令的配置文件。

/etc/printcap

与/etc/termcap 相类似，但用于打印机。具有不同的语法。

/etc/profile，/etc/csh.login，/etc/csh.cshrc

Bourne 或 C shell 在登录或启动时所执行的文件。系统管理员可以使用它为所有用户设置全局设置值。对于各个 shell 参见各自的 manual page。

/etc/securetty

确定安全终端，也即，确定能以 root 登录的终端。典型地，只有虚拟终端被列入，因此从调制解调器或网络侵入系统以获得超级用户的权限几乎是不可能的事（至少是非常困难的）。

/etc/shells

列出可信赖的 shell。Chsh 命令允许用户改变他们的登录 shell 为 shells 中所列出的 shell 之一。一台机器的提供 FTP 服务的服务器进程 ftpd,将检查用户的 shell 是否在/etc/shells 中被列出。除非用户的 shell 被列出，否则系统将阻止该用户的登录。

/etc/termcap

终端性能数据库。描述了各种终端可被哪些“escape 序列”所控制。取代那些直接输出的只能工作于一种特殊牌子的终端上的逃逸序列（escape sequence）的程序已经写出，该程序在/etc/termcap 中查询正确的序列来工作。其结果就是许多程序能够在绝大多数的终端上使用。详细信息参见 termcap、curs_termcap 以及 terminfo 的 manual pages。

3.4 /dev目录

/dev 目录包含所有设备的特殊设备文件。设备文件是以特定规则命名的文件；它们在设备列表（见 XXX）中进行了描述。设备文件是在系统安装时创建的，以后是由/dev/MAKEDEV 描述文件来创建。/dev/MAKEDEV.local 是由系统管理员写的描述文件，用以建立仅是本地的设备文件或连接文件（也即，那些不是标准 MAKEDEV 中的部分，如有些非标准设备驱动程序的设备文件）。

3.5 /usr文件系统

/ usr 文件系统通常很大，因为所有的程序都安装于此。/usr 中的所有文件通常来自 Linux 发行版；本地安装的程序以及其它程序都放在/usr/local 下。这使得从发行的一个新版中更新系统成为可能，甚至完全更新至新版而无须再次安装所有的程序。下面列出了/usr 中的一些子目录（有些不很重要的子目录这里没有列出，详细信息参见 FSSTND）。

/usr/x11R6

X 窗口系统，所有文件。为了简化 X 的开发与安装，X 的文件并没有集成到系统的其余部分中去。在/usr/x11R6 下有个目录树，与/usr 下的相类似。

/usr/X386

与/usr/X11R6 类似，但是是 X11 版本 5。

/usr/bin

几乎所有的用户命令。有些命令在/bin 或/usr/local/bin 之中。

/usr/sbin

系统管理命令，是根文件系统不需要的，例如，绝大多数的服务器程序。

/usr/man，/usr/info，/usr/doc

随机手册（manual pages）,GNU 信息文档以及各种其它的文档资料。

/usr/include

C 编程语言的头文件。为了一致性这本应位于/usr/lib 之下，但是传统的习惯势力一直支持这样的命名。

/usr/lib

程序和子系统的不变数据文件，包括一些场所配置文件。名字 lib 来源于 library;最初的编程子程序库是存于/usr/lib 中。

/usr/local

这个地方用于存放本地安装的软件和其它一些文件。

3.6 /var文件系统

/var 中包含着数据文件，当系统正常运行时，这些数据文件是变化的。对于每个系统来说它是特定的，也即，不能在网络上与其它计算机共享这些数据文件。

/var/catman

用于格式化过的 man pages 的高速缓冲。Manual pages 的原始资料通常是存放在/usr/man/man*中的；有些 manual pages 是以预格式化的版本形式给出的，存储于/usr/man/chat*中。其它一些 manual pages 在第一次使用时需要进行格式化处理；格式化过的版本就被存于/var/man 目录中，因此下一个人阅读同样的 page 时就不用等待格式化过程了。(/var/catman 的内容如同临时目录一样常常被清理的。)

/var/lib

当系统正常运行时改动过的文件。

/var/local

安装于/usr/local 中的程序的可变的数据（也即，由系统管理员安装的程序）。注意，即使是本地安装的程序也应该使用/var 的其它目录，例如，/var/lock。

/var/lock

上锁文件（锁定的文件）。许多程序遵循一个惯例，在/var/lock 目录中建立上锁文件，

用以指出它们正在使用一个特殊的设备或文件。其它的程序将会注意到这一点并会避免再使用这个特殊的设备或文件。

/var/log

各种程序的操作记录文件，特别是 login（/var/log/wtmp，该文件中记录了所有的对系统的登录和退出）以及 syslog（/var/log/messages，这里存储了内核和系统程序所有的消息）。/var/log 中的文件经常会变得很大，需要定期进行清理。

/var/run

含有有关系统信息的文件，这些信息在系统下一次启动之前一直有效。例如，/var/run/utmp 中含有当前登录的人的信息。

/var/spool

用于邮件、新闻、打印队列以及其它排队工作的目录。每个不同的 spool 在/var/spool 下面都有自己的子目录，例如，用户邮件箱在/var/spool/mail 之中。

/var/tmp

大的临时文件或者需要比在/tmp 中更长期存在的临时文件。（尽管系统管理员可能不允许很陈旧的文件存在于/var/tmp 中。）

3.7 /proc文件系统

/proc 文件系统是一个幻影文件系统，它在磁盘上并不存在。相反地，内核在内存中创建了它。它用于提供有关系统的信息（最初是关于处理器的，因此有了此名称）。一些重要的文件和目录在下面给出了说明。在 manual page 的 proc 中有更详细的说明。

/proc/1

是一个有关进程 1 信息的目录。每一个进程在/proc 下面都有一个以其进程号为名称的目录。

/proc/cpuinfo

有关处理器的信息，如它的类型、制造日期、型号以及性能。

/proc/devices

配置进当前运行内核的设备驱动程序的列表。

/proc/dma

显示当前哪个 DMA 通道正在被使用。

/proc/filesystems

配置进内核的文件系统。

/proc/interrupts

显示哪些中断被使用以及它们使用了多少次。

/proc/ioports

显示此时那些端口正被使用着。

/proc/kcore

是系统物理内存的一个映像。它的大小与你的物理内存完全一样，但实际上并不真正占据那样多的内存；他是在程序访问它时瞬时产生的。（记住：除非你将其复制到别的地方去，否则它在/proc 下不占用任何磁盘空间。）

/proc/kmsg

内核输出的消息。这些消息同样也被送到了 syslog 中。

/proc/ksyms

内核的符号表。

/proc/loadavg

系统的“平均负荷”；是三个毫无意义的指示值，指示出此时系统有多少工作要做。

/proc/meminfo

物理内存和交换区使用情况的信息。

/proc/modules

此时哪些内核模块被加载。

/proc/net

网络协议的状态信息。

/proc/self

一个进程目录的符号连接，该进程属于在/proc 中查看信息的程序的。当两个进程在观察/proc 时，它们取得不同的连接。这主要是为了方便程序观察它们自己的进程目录。

/proc/stat

有关系统的静态参数，如系统自启动以来的页故障（page faults）数。

/proc/uptime

系统启动的时间。

/proc/version

内核版本。

可以注意到，虽然以上文件基本上是易读的文本文件，这些文件的格式有时可能还是难于理解。有许多命令就只是读入上述文件并格式化成利于理解的格式。例如，free 程序读入/proc/meminfo 信息并将字节数转换成千字节（同时也加入了少量的信息）。

第4章 磁盘以及其他存储界质的使用

“在一个清晰的磁盘上你可以始终进行搜寻”

当你安装或者升级你的系统时，你需要在你的磁盘上做大量的工作。你必须在你的磁盘上创建文件系统使得文件能够存储在上面并且为你的系统的不同部分保留空间。

这一章叙述了所有这些初始的工作。通常，一旦你安装好系统，你就不必再从头开始这个工作，除非使用了软盘。如果你需要新增一个磁盘或想调整你的磁盘使用情况，你将需要再次阅读本章内容。

管理磁盘的基本任务有：

。格式化你的磁盘。要做各方面的事情来准备使用磁盘，如检查坏扇区。（如今对于许多硬盘来说已不需要格式化了。）

。如果你想将硬盘用于几种用途并且不想它们之间有任何的干扰，就需要对硬盘进行分区。分区的一个理由就是在同一个硬盘上安装不同的操作系统。另一个理由是将用户文件与系统文件分隔开来，这样做能简化备份并且能防止系统文件的毁坏。

。在每个硬盘上或分区上制作适当类型的文件系统。只有创建了文件系统，磁盘才对 Linux 可用；在上面文件才可被建立和访问。

。根据需要，自动地或手工地加载各种文件系统以形成单个树结构。（手工加载的文件系统通常也需要手工卸载下来。）

第五章包含了有关虚拟内存以及磁盘缓冲的信息，在使用磁盘时你同样要知道。

4.1 两类设备

UNIX，以及 Linux，能够识别两种不同的设备：随机访问块设备（如磁盘）以及字符设备（如磁带和串行线路），其中有些设备是连续设备，有些是随机访问的设备。在文件系统中每种支持的设备都以设备文件的形式表示。当你读写一个设备文件时，数据就会从它所代表的设备中取进或写出。这样就不需要特殊的程序（也不需要特别的编程方法，如获取中断或轮流查询一条串行线路）来访问设备了；例如，要打印一个文件，只要键入：

$ cat filename > /dev/lp1

$

文件的内容就会在打印机上打印出来了（当然，文件的内容打印机应能识别）。然而，因为几个人同时将他们的文件输出到打印机不是个好主意，人们一般使用特殊的程序来打印他们的文件（通常使用 lpr）。这个程序使得在同一时刻只有一个文件被打印出，并在打印机一旦打印完前一个文件就能自动地将文件发送到打印机。实际上，人们一点都不需要关心设备文件的。

由于在文件系统中设备是以文件的形式表示出的（在/dev 目录中），使用命令 ls 就很容易看到有那些设备文件存在。在 ls –l 的输出中，第一栏含有文件的类型和权限。例如，查看我的系统上的串行设备：

$ ls -l /dev/cua0

crw-rw-rw- 1 root uucp 5, 64 Nov 30 1993 /dev/cua0

$

第一栏的第一个字符，也即，上面 crw-rw-rw-中的‘c’，告诉一个见多识广的用户该文件的类型，在这个例子中是一个字符形设备。对于普通文件，第一个字符是‘-’，目录是‘d’，块设备是‘b’；详细信息请参见 ls 的 man page。

可以看出，即使设备本身并没有被安装，所有的设备文件依然是存在的。所以，你有一个/dev/sda 文件，并不意味着你就真的有一个 SCSI 硬盘。有了所有设备文件后，就使得安装简单化，很容易再增加新硬件（不再需要对新硬件找出正确的参数并为它创建设备文件了）。

4.2 硬盘

这个部分介绍有关于硬盘的术语。如果你已经知道这些术语和概念，你可以跳过本小节。

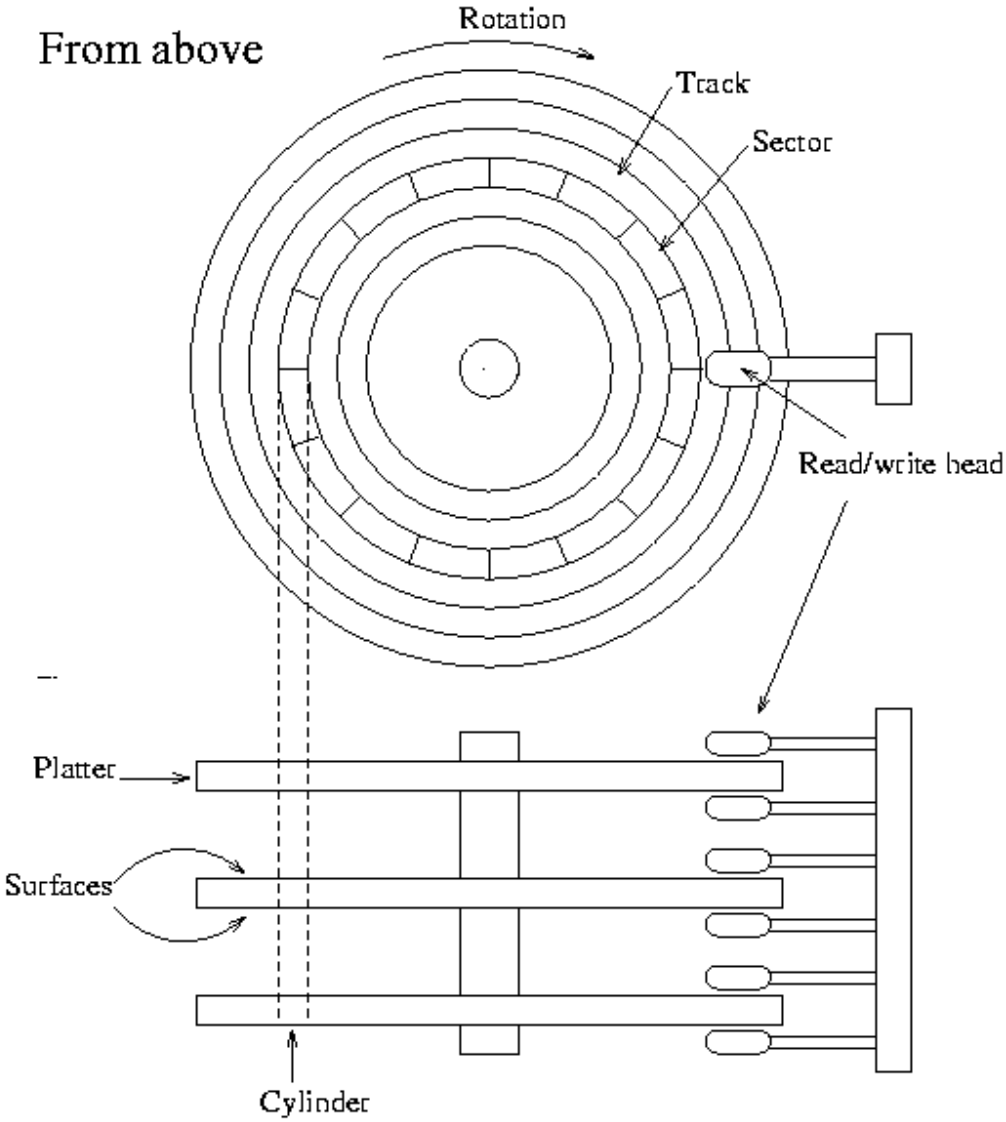
图 4-1 是硬盘内重要部分的示意图。一个硬盘是由一个或多个圆盘组成，[1] 这些盘的单面或双面上覆盖着用于记录数据的一层磁性物质。在每个面上，都有一个读写头用于读取或改变记录着的数据。这些圆盘绕着同一个轴旋转；典型的旋转速度是每分钟 3600 转，而高性能的硬盘有更高的转速。读写头沿着圆盘的半径方向移动；这个移动加上圆盘的旋转使得读写头能访问到圆盘表面的所有部分。

处理器（CPU）与实际磁盘通过磁盘控制器进行通信。这使得计算机的其余部分不需要知道如何使用驱动器，因为不同硬盘类型的控制器能够使用相同的接口与计算机的其它部分通信。因此，计算机只需说“咳，硬盘，给我想要的数据”，而不是又长又复杂的电子信号来移动读写头到适当的方位并等待正确的位置位于读写头下以及做所有其它烦琐而又必需的事情。（实际上，到控制器的接口仍然很复杂，但相比之下已简单多了。）控制器也能做些其它的事情，例如缓冲或坏扇区自动替换等。

以上就是一个人要对硬件有所理解的全部了。当然还有许多其它的工作，比如马达旋转圆盘并移动读写头，以及控制机械部分动作的电子线路，但这与理解硬盘工作原理几乎是无关的。

圆盘表面通常划分成许多的同心圆环，称作磁道（tracks）,并且这些圆环再被分为扇区（sectors）。这种分割是用于在硬盘上指定位置以及为文件分配硬盘空间的。为了在硬盘上找到指定的地方，人们可以说“面 3、磁道 5、扇区 7”。通常，所有磁道上的扇区数是相同的，但有些硬盘在在外磁道上放置了更多的扇区（所有扇区的物理面积大小是一样的，因此在稍长的外磁道上可以放置更多的扇区）。典型地，一个扇区能容纳 512 字节的数据。硬盘本身不能处理比一个扇区更少的数据量。

图 4-1.硬盘原理图



以同样的方法，每个面也被分成磁道（及扇区）。这意味着当一个面的读写头位于一个磁道上时，其它面上的读写头也位于相应的磁道上。所有这些相关磁道总合起来就叫作一个柱面（cylinder）。从一个磁道（柱面）移动到另一个磁道（柱面）是需要时间的，所以常把要一起访问的数据放在一起（如，一个文件）一个柱面上，这样就不用移动读写头来读取所有的数据，因此可以提高性能。并不是总能够象这样来放置文件的；在磁盘上几个地方存储数据的文件被称为有碎片的（fragmented）。

面数（或磁头数，这两个是一回事）、柱面数以及扇区数有各种大小；它们的数值的说明被称为硬盘的规格参数（geometry）。规格参数通常存储在一个特殊的、由电池供电的称作 CMOS RAM 的内存处，在操作系统启动或驱动程序初始化期间，操作系统能够从它那里获得这些参数。

不幸的是，BIOS [2] 有一个设计限制，这使得它不能在 CMOS RAM 中指定大于 1024 个磁道的数值，对于一个大容量硬盘来说这个数也太小了。为了克服这个缺点，硬盘控制器会曲解规格参数，并且把计算机给出的地址（addresses）转换成符合实际情况的数值。例如，一个硬盘可能有 8 个头、2048 个磁道以及每磁道有 35 个扇区。 [3] 其控制器可以对计算机说谎，并声称该硬盘有 16 个头、1024 个磁道以及每个磁道有 35 个扇区，这样在

磁道数上就没有超过限制，并且通过将磁头数减半以及将磁道数加倍来转换计算机给出的地址值。实际的算术运算可能很复杂，因为所给出的参数并不会象这里的一样好（但是再次重申，这些细节与理解原理是无关的）。这个转换曲解了操作系统对于磁盘是如何组织的看法，因此使用所有数据在同一柱面的窍门来提高性能是不切实际的。

这个转换仅对 IDE 硬盘是个问题。SCSI 硬盘使用了一个连续（顺序）扇区号（也即，控制器将顺序扇区号翻译成磁头、柱面以及扇区三个数），以及一个完全不同的 CPU 与控制器交流的方式，因此它们是不存在这类问题的。然而，请注意，计算机也同样不能知道 SCSI 硬盘的实际规格参数。

既然 Linux 不会知道一个硬盘的实际规格参数，它的文件系统甚至也不强求将文件存储于同一个柱面上。而是试着给文件分配顺序编号的扇区，这几乎总能给出同样的性能。再有，控制器上的高速缓冲、控制器的自动提前获取功能使问题变得更加复杂化。

每个硬盘都以一个独立的设备文件来表示。（通常）可以有 2 到 4 个 IDE 硬盘。它们分别表示为/dev/hda，/dev/hdb，/dev/hdc，/dev/hdd。SCSI 硬盘表示为/dev/sda，/dev/sdb，/dev/sdc，等等。对于其它类型的硬盘也有相类似的命名规则；更多的信息请参见 XXX（设备列表）。注意，硬盘的设备文件可以访问整个硬盘，而不管各分区的（这在下面讨论），如果你不小心就很容易搞糟各分区以及其中的数据的。硬盘的设备文件通常仅用来访问主引导记录的（这也将在下面讨论）。

4.3 软盘

软盘是由单面或双面覆盖着磁性物质的软膜组成，如同硬盘一样。软盘本身不带有读写头，读写头是包含在驱动器中的。一张软盘与硬盘中的一个圆盘相对应，但是是可取走的并且一个驱动器可以访问不同的软盘，而硬盘却是不可分割的一个整体。

与硬盘一样，一张软盘也分成磁道和扇区（并且一张软盘的两面相应的磁道组成一个柱面），但是与硬盘比起来，它们的数量很少。

一个软盘驱动器可以使用几种不同类型的盘片；例如，一个 3.5 寸驱动器可以使用 720kB 以及 1.44MB 的磁盘。因为驱动器的操作有所不同并且操作系统必须知道磁盘容量的大小，所以软盘驱动器有许多设备文件，每个驱动器以及磁盘类型组合成一个设备文件。因此，/dev/fd0H1440 表示第一个软驱（fd0），它必须是一个 3.5 英寸驱动器，使用一个 3.5 英寸、1440kB（1440）大小的高密度盘片(H)，也即，一张标准的 3.5 英寸 HD 软盘。有关软盘驱动器命名规则的更多信息，请参见 XXX（设备列表）。

然而，软盘驱动器的命名是复杂的，因此 Linux 有一种特别的软盘设备类型，它能够自动地检测出驱动器中盘片的类型。其工作原理是使用不同软盘类型试读新插入软盘的第一个扇区直到它发现正确的类型为止。这自然需要软盘首先是格式化过的。这些自动设备称为/dev/fd0，/dev/fd1，等等。

自动设备为访问一张盘片所使用的参数也可以用程序\cmd{setfdprm}来设置。在你需要用到不符合任何常规软盘容量大小的磁盘时，这是非常有用的，例如，如果磁盘的扇区数是与众不同的，或者由于某些原因自动检测失败并且适当的设备文件丢失。

除了许多标准软盘以外，Linux 还能够处理许多非标准软盘。其中有些需要特殊的格式化程序。现在我们将略过这种磁盘类型，不过同时你可以查看/etc/fdprm 文件。它详细说明了 setfdprm 能识别的设置。

操作系统必须知道在软盘驱动器中的盘片是什么时候给换掉的，例如，为了避免使用前张盘片的缓冲的数据。不幸的是，用于此的信号线常常是断掉的，更糟的是，当从 MS-DOS 中使用驱动器时，这不是总是受到注意的。在使用软盘时，如果你碰上了怪异的问题，这

可能就是原因所在。唯一的修正方法是维修软盘驱动器。

4.4 CD-ROM

CD-ROM 驱动器使用一光读取的、塑胶覆盖的盘片。信息以小“孔”的形式记录在盘片[4]的表面，这些小空沿着一螺旋线从盘片中间延伸至盘片边沿。驱动器引导一激光束沿着螺旋线来读取盘片上的信息。当激光束照到一个孔时，激光沿着一个方向被反射出来；当激光束照到光滑的表面时，它被反射到另一个方向去了。这很容易进行编码并用来存储信息。剩下的事很容易，仅仅是机械问题了。

与硬盘相比，CD-ROM 的速度很慢。尽管一个典型的硬盘的平均寻道时间不到 15 毫秒，一个快速 CD-ROM 驱动器要用十分之一秒的时间来寻道。实际的数据传输速率还是相当高的，能达到每秒几百千字节。慢速意味着 CD-ROM 是不能令人满意地代替硬盘（有些 Linux 发行版在 CD-ROM 上提供‘活’文件系统，以至于不必将文件拷贝到硬盘上，使得安装更容易，节省了大量硬盘空间），尽管仍有这个可能。对于安装新的软件，使用 CD-ROM 是非常好的，因为在安装期间速度并不是最主要的。

有几种方式用来安排 CD-ROM 上的数据。最流行的方式是按照国际标准 ISO 9660。这个标准说明了一个最小文件系统，它甚至比 MS-DOS 所用的还要粗略。从另一方面来讲，它是如此地小，以至于任何操作系统都可以将其映射入自己的系统中。

对于标准 UNIX 的使用，ISO 9660 是不可用的，所以已经开发出了标准的扩展版，称为 Rock Ridge 扩展。Rock Ridge 允许更长的文件名、符号连接、以及许多其它的好处，使得 CD-ROM 看上去多少象一个临时 UNIX 文件系统。更佳的是，一个 Rock Ridge 文件系统仍然是一个有效的 ISO 9660 文件系统，使得它对非 UNIX 系统同样有用。Linux 支持 ISO 9660 以及 Rock Ridge 扩展这两者；扩展是自动识别和使用的。

然而，文件系统只是战役的一半。许多 CD-ROM 包含有需要特殊程序才能访问的数据，并且大多数这类程序不能在 Linux 下运行（很可能除非在 dosemu 下—Linux 的 MS-DOS 模拟器）。

CD-ROM 驱动器是通过相应的设备文件来访问的。有几种方法将 CD-ROM 连接至计算机：通过 SCSI、通过声卡、或通过 EIDE。所需的硬件叙述已超出了本书范围，但是连接的类型决定了设备文件。详细信息参见 XXX（设备列表）。

4.5 磁带

磁带驱动器使用磁带，同用于乐曲的盒式磁带类似[5]。磁带是顺序连续的，这意味着要想到达它的指定部分，你必须经过这之间的所有部分。一张盘片可以随机地访问，也即，你可以直接跳到盘片的任何地方。磁带的串行访问特征使得它们很慢。

从另一方面讲，磁带制造廉价，因为它们不需要快速。它们也可以做得很长，因此可以容纳大量的数据。这使得磁带非常适合于存档和备份这样的事，这都不需要高速的，但却有低价和大容量的好处。

4.6 格式化

格式化（formatting）就是在磁性介质上写上记号，用于标志出磁道和扇区。在格式化之前，磁盘表面的磁信号是杂乱无章的。当格式化后，沿着磁道的轨迹方向带来了一定的次序，并且磁道被分成扇区。实际的细节并不完全如此，但那我们就不管了。重要的是磁盘没有格式化就不能用。

在这，这个术语有些混淆：在 MS-DOS 中，格式化这个词也同时用于表示创建文件系统的过程（这将在下面讨论）。在其中，两个过程常常合在了一起，尤其是对软盘。当需要加以区别时，实际的格式化被称作低级格式化（*low-level-formatting*），而创建文件系统被称作高级格式化（*high-level formatting*）。在 UNIX 环境中，这两者分别称作格式化和创建文件系统，这也是在本书中的称呼方法。

对于 IDE 和一些 SCSI 硬盘来说，格式化早已在出厂前就做好了，不需要重复再做；因此大多数人无需关心它。实际上，对硬盘进行格式化有可能导致硬盘工作不正常，例如，硬盘有可能需要非常特殊的方式进行格式化，以获得自动坏扇区替换的功能。

不管怎么说，需要格式化或能够格式化的磁盘常需要一个特殊的程序，因为不同的硬盘内的格式化逻辑接口是不一样的。格式化用的程序通常或者是在控制器的 BIOS 中，或者是以 MS-DOS 程序的形式提供；这两种方式均不适用于 Linux。

在格式化磁盘时很可能碰到坏点，称作坏块（*bad blocks*）或坏扇区（*bad sectors*）。有时这是由驱动器自身来处理的，但即使是这样，如果碰到很多坏的地方，就需要想办法避免使用磁盘坏的部分。处理这样情况的过程是建在文件系统中的；如何增加这方面的信息将在下面叙述。另一种方法是创建只包含磁盘坏区的分区；如果坏区较大的话，这倒是个不错的主意，因为文件系统对付大块坏区有可能出问题。

软盘是用 **fdformat** 进行格式化的。软盘设备文件将作为参数给出。例如，下面的命令将在第一个软驱中格式化一张高密度、3.5 英寸的软盘：

```
$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
$
```

注意，如果你想要使用一个自动识别的设备（例如，/dev/fd0），你必须首先用 **setfdprm** 来设置设备的参数。要达到以上同样的目的，就得象下面这样做：

```
$ setfdprm /dev/fd0 1440/1440
$ fdformat /dev/fd0
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... done
$
```

选择符合软盘类型的设备文件来做这事通常是更为方便的。注意，超容量格式化是不明智的。

Fdformat 也会对软盘进行验证工作，也即，检查坏块。它将对坏块试着读写几次（你通常可以听到，驱动器的噪声有很大的不同）。如果软盘只是边缘有些损坏（由于磁头脏，有些错误是伪信号），**fdformat** 是不会抱怨的，但是实际错误将取消验证过程。内核将为所发现的每个 I/O 错误打印出日志消息；这消息将在控制台上显示出来或者，如果使用了 **syslog**，会打印到/usr/log/message 文件中。Fdformat 自己不会说出出错的地方（人们通常并不关心，因为软盘太便宜了，坏了就丢了算了）。

$ fdformat /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
Verifying ... read: Unknown error
$

badblocks 命令能用于查找任何磁盘或分区上的坏块（包括软盘）。它并不格式化磁盘，所以它甚至可以用来检查文件系统。下面的例子是检查一张含有两个坏块的 3.5 英寸软盘。

$ badblocks /dev/fd0H1440 1440
718
719
$

badblocks 输出它找到的坏块的块号。许多文件系统可以避免这些坏块。在创建文件系统时会初始化一张已知坏块的列表，并在以后可以对其进行修改。最初寻找坏块的工作可以用命令 **mkfs** 来做（它是用于初始化文件系统的），但以后要用 **badblocks** 且新的坏块要用 **fsck** 加进去。我们将在以后讨论\cmd{mkfs}和 **fsck**。

许多现代硬盘会自动地注意到坏块，并且会使用特殊预留着的好块来替代它们。这对操作系统来说是看不见的。如果你对它是如何工作的感到好奇的话，这个特性会在磁盘的手册中有。即使是这样的硬盘，如果坏块太多的话，硬盘还是不能用的。

4.7 分区

一个硬盘可以分成几个区（*partitions*）。每个分区就如同一个独立的硬盘。这个想法是，如果你有一个硬盘，并且想要在上面安装两个操作系统，你可以将这个硬盘分成两个分区。每个操作系统自由地使用各自的分区而不触及另一个。这样两个操作系统就可以和平地共处于同一个硬盘中。如果没有分区的话，你就得为每个操作系统购买一个硬盘。

软盘是不分区的。这并没有什么技术上的原因，只是因为软盘的容量太小了，因此分区极少使用。CD-ROM 盘也常常是不分区的，因为它作为一个大磁盘使用很容易。很少有需要几个操作系统放在一张光盘上的。

主引导记录(MBR)、引导扇区以及分区表

硬盘是如何分区的信息存储于硬盘的第一个扇区中（也即，第一张盘片的第一个磁道的第一个扇区）。这第一个扇区就是硬盘的主引导记录（*master boot record MBR*）；当机器首次启动时，BIOS 会读入这个扇区。主引导记录中含有一个小程序，用于读分区表、检查哪个分区是活动分区（也即，标志为可启动的），并且读入那个分区的第一个扇区，分区的启动扇区（*boot sector*）(MBR 也是一个启动扇区，但它有一个特殊的地位，因此有一个特殊的名字)。这个启动扇区也包括一个小程序，用于读入当前分区操作系统的第一部分（假设它是可启动的），然后开始运行整个操作系统。

分区的方案没有建于硬件中，或者甚至 BIOS 中。这是许多操作系统都遵守的惯例。但并不是所有的操作系统都是按惯例做的，这是个例外。有些操作系统支持分区，但只在硬盘上占用一个分区，并在此分区上使用自己的内部分区方法。后面这种分区类型能与其

它的操作系统和平共处（包括 Linux），并且不需要任何特殊的方法。但是不支持分区的操作系统不能在同一个硬盘上与其它操作系统共处。

作为安全预防，将分区表写在一张纸上是个好主意，这样一旦发生破坏，你不会丢失所有的文件。（坏分区可以用 **fdisk** 修复）。相关的信息可以用命令 **fdisk –l** 给出：

$ fdisk -l /dev/hda

Disk /dev/hda: 15 heads, 57 sectors, 790 cylinders
Units = cylinders of 855 * 512 bytes

Device Boot	Begin	Start	End	Blocks	Id	System
/dev/hda1	1	1	24	10231+	82	Linux swap
/dev/hda2	25	25	48	10260	83	Linux native
/dev/hda3	49	49	408	153900	83	Linux native
/dev/hda4	409	409	790	163305	5	Extended
/dev/hda5	409	409	744	143611+	83	Linux native
/dev/hda6	745	745	790	19636+	83	Linux native

$

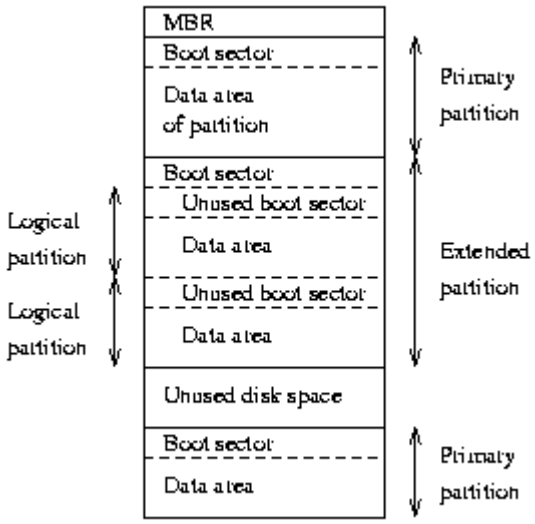
扩展与逻辑分区

最初的 PC 上硬盘的分区方案只允许有 4 个分区。这在实际使用中很快就变得太小了，部分原因是有些人需要多于 4 个操作系统（Linux、MS-DOS、OS/2、Minix、FreeBSD、NetBSD、或者 Windows/NT，等等），而主要原因是有时一个操作系统拥有几个分区是个很好的主意。例如，对 Linux 来说交换空间通常最好是放置在它自己的分区内，而不是放在主 Linux 分区中，这是考虑到速度的因素（见下面）。

为克服这个设计问题，发明了扩展分区（*extended partitions*）。这个诀窍允许主分区再分成子分区。如此划分的主分区就是扩展分区（*extended partitions*）；子分区是逻辑分区（*logical partitions*）。它们的表现同主[6]分区一样，但是以不同的方式创建的。不存在速度方面的差别。

一个硬盘的分区结构看上去如同图 4-2 中所示。硬盘被分成三个主分区，第二个主分区再被分成两个逻辑分区。硬盘还有些部分根本没有分区。整个硬盘以及各个主分区都有一个引导扇区。

图 4-2. 一个简单的硬盘分区



分区类型

分区表(MBR 中的以及扩展分区中的)中的每个分区含有一个字节的分区类型识别码。是用于识别使用分区的操作系统，或用于其他用途。目的是为了防止两个操作系统偶然使用了同一个分区。然而，实际上操作系统并不关心分区类型字节；例如，Linux 根本不关心它是什么。更糟的是，有些操作系统错误地使用了它；例如，至少在 DR-DOS 的某些版本中忽略了该字节中非常重要的位，而其它却没有。

并没有标准化机构来指定每个字节值是什么含义，但是某些公认的值的含义见表 4-1 所示。Linux 的 **fdisk** 程序也有同样的列表。

表 4-1。分区类型（取自 Linux fdisk 程序）。

0	Empty	40	Venix 80286	94	Amoeba BBT
1	DOS 12-bit FAT	51	Novell?	a5	BSD/386
2	XENIX root	52	Microport	b7	BSDI fs
3	XENIX usr	63	GNU HURD	b8	BSDI swap
4	DOS 16-bit f <32M	64	Novell	c7	Syrinx
5	Extended	75	PC/IX	db	CP/M
6	DOS 16-bit >32M	80	Old MINIX	e1	DOS access
7	OS/2 HPFS	81	Linux/MINIX	e3	DOS R/O
8	AIX	82	Linux swap	f2	DOS secondary
9	AIX bootable	83	Linux native	Ff	BBT
a	OS/2 Boot Manag	93	Amoeba		

对硬盘进行分区

有许多程序用于创建和删除分区。大多数操作系统都有自己的，最好要用操作系统自己的程序，以防它可能做些其它程序不能做特别的事。这些程序大多数都称为 **fdisk**，包括 Linux 系统也是这样称呼的，或者有些相应改变。使用 Linux 的 fdisk 的详细信息见 man page。Cfdisk 命令与 fdisk 相类似，但有一个更好的（全屏幕）用户界面。

当使用 IDE 硬盘时，启动引导分区（含有启动内核映像文件的分区）必须完全位于头 1024 柱面以内。这是因为硬盘在启动时是通过 BIOS 使用的（在系统进入保护模式之前），

而 BIOS 不能处理大于 1024 的柱面。有时候很可能用到只有部分分区在 1024 柱面内的启动分区，但只要所有通过 BIOS 来读取的文件位于头 1024 个柱面内就行。由于这种安排非常困难，所以这样做是很不利的；你不可能知道什么时候一个内核的更新或者硬盘的整理碎片活动将导致系统不可启动。因此，要确信你的启动分区完全在头 1024 个柱面以内。

实际上，有些新版的 BIOS 以及 IDE 硬盘能够处理大于 1024 柱面的情况。如果你有这样的系统，你可以不管这个问题；如果你不能非常的确信，就将启动所用的文件放在 1024 个柱面以内。

每个分区应该有偶数个扇区值，因为 Linux 系统使用一个 1 千字节的块大小，也即，两个扇区。一个奇数的扇区值将导致最后一个扇区没有用上。虽然这不会产生任何问题，但是却很不顺眼，并且有些版本的 **fdisk** 会对此提出警告。

更改分区的大小通常需要首先备份该分区上的所有你想保存的数据（最好是备份整个硬盘，看情况定），删除这个分区，再创建新分区，接着再恢复所有的数据到新分区上。如果分区是增大的，你可能同时需要调整相邻分区的大小（以及备份和恢复）。

MS-DOS 有个程序，叫 **fips**，可以用来调整 MS-DOS 分区的大小而无需进行备份和恢复操作。但对于其它系统这仍然是必须的。

设备文件与分区

每个分区以及扩展分区都有自己的设备文件。这些文件的命名规则是在硬盘名的后面加上分区号，1-4 号是主分区（不管那里有多少个主分区）以及 5-8 是逻辑分区号（不管它是属于那个主分区的）。例如，/dev/hda1 是第一个 IDE 硬盘上的第一个主分区，/dev/sdb7 是第二个 SCSI 硬盘上的第三个扩展分区。XXX（设备列表）中的设备列表给出了更多的信息。

4.8 文件系统

什么是文件系统？

文件系统（*filesystem*）是操作系统用以明了磁盘或分区上的文件的一种方法以及数据结构；也即，磁盘上文件组织的方法。这个词也用于指一个用于存储文件的分区或磁盘，或者是指给定文件系统的类型。因此，某人可以说“我有两个文件系统”意思是说他有两个存储文件的分区，或者某人说“扩展文件系统”，意思是说文件系统的类型。

区别对待盘片或分区以及在其上的文件系统是非常重要的。很少程序（很理智地讲，包括创建文件系统的程序）是在磁盘或分区的原始扇区上直接操作的；如果其上已存在一个文件系统，将会被毁坏或受到严重地破坏。大多数程序是在文件系统上操作的，因此不能在无文件系统的分区上工作（或在类型不对的分区上操作）。

在一个分区或磁盘能被用作一个文件系统之前，它需要被初始化过，并且簿记数据结构需已被写入磁盘。这个过程称为制作一个文件系统（*making a filesystem*）。

大部分 UNIX 文件系统类型有一个相似的通常结构,尽管在精确的细节方面变化很大。主要概念有超级块（*superblock*）、i 节点（*inode*）、数据块（*data block*）、目录块（*directory block*）、以及间接块（*indirection block*）。超级块包含整个文件系统的信息，如文件系统的大小（这里的实际信息依赖于是何种文件系统）。一个 i 节点包括一个文件的所有信息，除了它的名字。名字是和 i 节点号一起存储于目录中的。一个目录项是由文件名以及表示相应文件的 i 节点号组成。i 节点还包括存储文件中数据的数据块的数量。在 i 节点中只有容

纳几块数据块的空间，然而，如果需要更多的数据块，会动态地为指向数据块的指针分配更多的空间。这些动态分配的块就是间接块；这个名字说明为了找到所需的数据块，必须首先在间接块中查到数据块的号码。

UNIX 文件系统通常允许在一个文件中创建一个孔（*hole*）（这是用 lseek 来做的；请查看 manual page），这意味着，文件系统只是假装在文件的一个特别的地方仅有零个字节，但并没有为文件保留磁盘扇区（这就是说，这个文件将使用更少的磁盘空间）。对于小的二进制文件、Linux 共享库文件、一些数据库以及其它一些特殊情况，这事常有的事。（孔是通过在间接块或 i 节点中存入当作数据块地址的特殊值来实现的。这个特殊的地址表示没有为文件的那个部分分配数据块，因此，文件中就有了一个孔。）

孔是比较有用的。在作者的系统中，简单的测试表明通过孔的使用在大约 200MB 使用的磁盘空间里能节约 4MB 的空间。而且，在这个系统中只有相应很少的程序以及并没有数据库文件。

各种文件系统

Linux 支持几种类型的文件系统。其中重要的几种有：

minix

最老的、认为也是最可靠的，但性能上是很有限的（有时没有标志，文件名最多 30 个字符）并且容量也是有限的（每个这样的文件系统最多 64MB）。

xia

一个 minix 文件系统的修正版本，降低了对文件名长度以及文件系统大小的限制，但并没有引进新的特性。它并不流行，但据报道工作的很好。

ext2

非常有特色的出自于 Linux 的文件系统，也是当今最流行的文件系统。它被设计成易于升级的，这样文件系统代码的新版本就不需要重建已存在的文件系统。

ext

是 ext2 的老版本，不是向上兼容的。已不再在安装中使用了，并且大多数人都已转换到了 ext2。

另外，还支持一些外来的文件系统，使得与其他操作系统交换文件变得更容易了。这些外来的文件系统工作起来如同本地的一样，只是它们缺乏一些通常的 UNIX 的特征，或者有着奇特的限制或其它古怪的地方。

msdos

与 MS-DOS 的（以及 OS/2 和 Windows NT）FAT 文件系统兼容。

usmdos

是在 Linux 下对 msdos 文件系统的扩展，支持长文件名、所有者、权限、连接以及设备文件。这使得一个常规的 msdos 文件系统能够想 Linux 的文件系统一样的使用，因此对 Linux 来说也就无需一个独立的分区了。

Iso9660

标准的 CD-ROM 文件系统；自动支持允许更长文件名的 Rock Ridge 扩展文件系统。它是对标准 CD-ROM 文件系统的扩展。

nfs

一个网络文件系统，允许在许多计算机之间共享一个文件系统，易于这些计算机对文件的访问。

hpfs

OS/2 的文件系统。

sysv

SystemV/386,以及 Xenix 文件系统。

要依具体情况来选择使用文件系统。如果兼容性或其它原因使得需要使用一个非原始的文件系统，那就必须使用其中的一个。如果能自由地选用文件系统的话，那么选用 ext2 也许是最明知的，因为它有所有的特征并且没有性能上的下降。

还有一种 proc 文件系统，通常作为/proc 目录来访问，它其实完全不是一个真正的文件系统，尽管看起来它很象。proc 文件系统使得访问某些内核的数据结构变得很容易，如进程列表（因此还有名字）。它使得这些数据结构看起来很象是一个文件系统，并且能够使用所有通常的文件工具来操作处理。例如，要取得所有进程的一个列表，可以使用命令：

$ ls -l /proc

```
total 0
dr-xr-xr-x    4 root     root            0 Jan 31 20:37 1
dr-xr-xr-x    4 liw      users           0 Jan 31 20:37 63
dr-xr-xr-x    4 liw      users           0 Jan 31 20:37 94
dr-xr-xr-x    4 liw      users           0 Jan 31 20:37 95
dr-xr-xr-x    4 root     users           0 Jan 31 20:37 98
dr-xr-xr-x    4 liw      users           0 Jan 31 20:37 99
-r--r--r--    1 root     root            0 Jan 31 20:37 devices
-r--r--r--    1 root     root            0 Jan 31 20:37 dma
-r--r--r--    1 root     root            0 Jan 31 20:37 filesystems
-r--r--r--    1 root     root            0 Jan 31 20:37 interrupts
-r--------    1 root     root      8654848 Jan 31 20:37 kcore
-r--r--r--    1 root     root            0 Jan 31 11:50 kmsg
-r--r--r--    1 root     root            0 Jan 31 20:37 ksyms
-r--r--r--    1 root     root            0 Jan 31 11:51 loadavg
-r--r--r--    1 root     root            0 Jan 31 20:37 meminfo
-r--r--r--    1 root     root            0 Jan 31 20:37 modules
dr-xr-xr-x    2 root     root            0 Jan 31 20:37 net
dr-xr-xr-x    4 root     root            0 Jan 31 20:37 self
-r--r--r--    1 root     root            0 Jan 31 20:37 stat
-r--r--r--    1 root     root            0 Jan 31 20:37 uptime
```

-r--r--r-- 1 root root 0 Jan 31 20:37 version
$

（尽管这里有些特别的文件是与进程无关的。上面这个例子已简化过）

请注意，尽管是这样，它仍被称为文件系统，虽然 proc 文件系统的任何部分都与磁盘无关。它仅存在于内核的想象中。无论何时，当有人想查看 proc 文件系统的任何部分时，内核就会使得该部分好象的确是存在于某个地方的，尽管是这样，它并不是个文件系统。因此，尽管它有一个多兆字节的/proc/kcore 文件，它其实不占用任何磁盘空间。

4.8.1 应该使用那一种文件系统？

使用同时许多不同文件系统是很少有的。目前，ext2fs 是最流行的一个，也可能是最明智的选择了。根据薄记结构的性能、速度、（感知的）可靠性、兼容性以及各种其他原因，选用其他的文件系统是可取的。这需要进行逐步分析才能决定。

4.8.2 创建一个文件系统

文件系统是用 **mkfs** 命令创建的，也即，初始化的。事实上，每种文件系统类型都有自己独特的创建程序。**mkfs** 只是一个前端程序，它根据所选的文件系统类型来运行适当的程序。类型是用-t fstype 选项来选择的。

由 **mkfs** 调用的程序有些略微不同的命令行界面。共有的以及最重要的选项总结如下；详细信息参见 manual pages。

-t fstype
选择文件系统的类型。

-c
查找坏块从而初始化坏块列表。

-l filename
从文件中读取坏块列表。

为了在软盘上创建一个 ext2 文件系统，必须执行以下命令：

$ fdformat -n /dev/fd0H1440
Double-sided, 80 tracks, 18 sec/track. Total capacity 1440 kB.
Formatting ... done
$ badblocks /dev/fd0H1440 1440 $>$ bad-blocks
$ mkfs -t ext2 -l bad-blocks /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user

```
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

首先，格式化软盘（-n 选项指出无需验证，也即，不进行坏块检查）。然后使用 badblocks 命令来查找坏块，并将输出重定向到一个文件中，bad-blocks。最后，创建文件系统，并用 badblocks 找到的坏块来初始化坏块列表。

也可以在 mkfs 中使用-c 选项来替代 badblocks 命令以及它的输出文件。下面的例子就是这样做的。

```
$ mkfs -t ext2 -c /dev/fd0H1440
mke2fs 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
360 inodes, 1440 blocks
72 blocks (5.00%) reserved for the super user
First data block=1
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1 block group
8192 blocks per group, 8192 fragments per group
360 inodes per group

Checking for bad blocks (read-only test): done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
$
```

使用-c 选项比分开来使用 badblocks 命令更为方便，但是 badblocks 用于检查已有的文件系统是必须的。

在硬盘上或分区上创建文件系统的过程是和在软盘上一样的，只是无需格式化操作。

4.8.3 加载与卸载

在使用一个文件系统时，首先必须加载它。然后操作系统会做一系列的薄记作业以确保每件事都工作正常。由于 UNIX 中的所有文件都在一棵单目录树中，加载操作使得新文件系统的内容看上去象是已加载文件系统子目录中的内容。

例如，图 4-3 显示了三个独立的文件系统，每一个都有自己的根目录。当后两个文件系统被分别加载到第一个文件系统的/home 以及/usr 下面时，我们可以获得一棵单目录树，如图 4-4 中所示。

图 4-3.三个独立的文件系统。

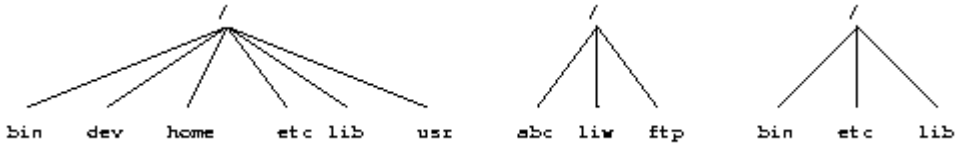
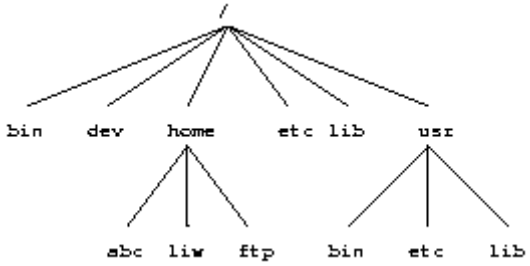


图 4-4./home 以及/usr 已被加载。



可以按以下方法进行加载：

$ mount /dev/hda2 /home
$ mount /dev/hda3 /usr
$

mount 命令带有两个参数。第一个是与包含该文件系统的磁盘或分区相关的设备文件。第二个是目录，在该目录下面文件系统将被加载。在执行完上面的命令后，两个文件系统的内容看上去分别象是/home 以及/usr 目录中的内容。于是，可以说“/dev/hda2 加载到了/home”,对/usr 也是类似的。为了看各个文件系统的内容，只要观察它所加载到的目录的内容，就好象这个目录是别的目录一样。请注意设备文件/dev/hda2 和被加载目录/home 之间的区别。设备文件给出了访问磁盘上原始数据的能力，而被加载目录给出了访问磁盘上文件的能力。被加载的目录称为加载点（*mount point*）。

Linux 支持许多文件系统类型。**mount** 命令会试图猜测要加载的文件系统的类型。你也可以使用-t fstype 选项来直接指定类型；这有时是必须的，因为 mount 的启发式工作方式并不总是有效的。例如，为了加载一张 MS-DOS 软盘，你可以用下面的命令：

$ mount –t msdos /dev/fd0 /floppy
$

被加载到的目录并不需要是空的，尽管这个目录必须存在。然后当文件系统加载上来以后，该目录中原来的任何文件就都不可访问了。（但任何已被打开的文件将继续可以访问。从其它目录来的有硬连接的文件也可以用那些硬连接名来访问。）这样做并无害处，甚至很有用处。例如，有些人将/tmp 和/var/tmp 同步起来，使/tmp 成为到/var/tmp 的一个符号连接。当系统启动时，在/var 文件系统被加载之前，根文件系统上的/var/tmp 目录先被使用。当/var 被加载后，它将使得根文件系统上的/var/tmp 目录不可访问。如果/var/tmp 在根文件系统上不存在，那么在加载/var 之前就不能使用临时文件了。

如果你不会往文件系统上写入任何东西，那么可以使用带-r 选项的 mount 命令来做一

个只读加载(*readonly mount*)。这将使得内核停止任何往文件系统上写的企图，并且不会更新 i 节点内的文件访问时间。对于只读界质，只读加载方式是必须的，例如，CD-ROM。

细心的读者可能已注意到一个微小的逻辑问题。由于第一个文件系统（被称为根文件系统 *root filesystem*，因为它包含根目录）显然不会加载到其它的文件系统上，那么它是如何被加载的呢？好吧，答案是它是被魔术般地加载上去的。[7] 在启动时根文件系统被魔术般地加载上去，并且我们可以始终认为它是已加载的。如果根文件系统不能被加载，系统就启动不了。作为根文件系统被魔术加载的文件系统的名字或是已被编译进内核，或是使用 LILO 或 **rdev** 设置的。

通常，根文件系统首先以只读方式加载。然后，启动描述文件将运行 **fsck** 来验证它的有效性，并且如果没有问题的话，再重加载成可写的。**fsck** 不能对一个已加载的文件运行，因为当 **fsck** 正在运行时，任何对文件系统的改变都会带来麻烦。由于根文件系统在被验证时是已只读方式加载的，所以 **fsck** 可以不用担心地修复任何问题，重加载操作将会回写文件系统保存在内存中的任何原数据。

在许多系统中，还有其它一些文件系统应该在系统启动时被自动加载。这些文件系统是在/etc/fstab 文件中指定的；文件的详细格式请参见 **fstab** 的 man page。当另加的文件系统被加载时要依靠许多的参数，并且可以由系统管理员来配置；见第六章。

当一个文件系统不再需要被加载时，可以使用 **umount** 命令来卸载。[8] **umount** 有一个参数：设备文件名或是加载点。例如，要卸载上个例子的目录，可以使用命令

```
$ umount /dev/hda2
$ umount /usr
$
```

如何使用这个命令的更详细的信息请参见 man page。要特别强调，一定要卸载一个被加载的软盘，而不能仅仅弹出驱动器中的软盘完事！这是由于磁盘缓冲，在卸载之前数据并不需要马上就写如软盘，所以过早地从驱动器中取走软盘有可能导致软盘的内容乱掉。如果你只是从软盘中读取信息，这还没问题，但如果是写的话，甚至是偶然地，其结果将是灾难性的。

加载与卸载需要超级用户权限，也即，只有 root 用户能做。原因是如果允许任何用户能够在任何目录上加载软盘的话，那么就很容易创建一张伪装成/bin/sh 的特落依木马程序软盘，或任何其它经常被使用的程序。但是，常常需要允许用户能够使用软盘，有几种方法可用：

。给用户 root 的口令。这显然是极不安全的，但是最简单的解决方法。如果不作安全方面的考虑，这是种方法很好，对于不连网的、个人系统，常常是这样做的。

。使用如 **sudo** 这样的程序来允许用户使用加载。这仍然是不安全的，但不用直接给出超级用户的权限。[9]

。让用户使用 mtools 命令，这是一个维护 MS-DOS 文件系统的工具包，而无需加载它。这在只使用 MS-DOS 软盘时可以工作的很好，但是显得很笨拙。

。在/etc/fstab 中列出软盘设备文件以及允许它们加载的加载点等参数。

以上最后一种方式可以通过往\fn{/etc/fstab}中加入一行来实现，如下所示：

```
/dev/fd0        /floppy     msdos   user,noauto     0       0
```

各列为：要加载的设备文件、加载的目录、文件系统类型、选项、备份频率（用于 **dump**），以及 fsck 通行号（指定在启动时文件系统应被检查的顺序；0 意味着不用检查）。

Noauto 选项阻止当系统启动时自动加载它（也即，它阻止 **mount –a** 加载它）。用户选项允许任何用户来加载这个文件系统，并且，由于安全的原因，不允许从这个已加载的文件系统中执行程序（常规的或 setuid）以及解释执行设备文件。除了这，任何用户可以用以下命令，加载一张 msdos 文件系统软盘：

```
$ mount /floppy
$
```

软盘能够（当然也是需要的）用相应的命令\cmd{umount}来卸载。

如果你想提供对几种软盘文件系统类型的访问，你需要给出几个加载点。各个加载点的设置可以是不同的。例如，要访问 MS-DOS 以及 ext2 的软盘，你可以在/etc/fstab 中加入以下两行：

```
/dev/fd0      /dosfloppy      msdos     user,noauto    0    0
/dev/fd0      /ext2floppy     ext2      user,noauto    0    0
```

对于 MS-DOS 文件系统（不仅仅是软盘）来说，你可能想用 uid、gid 以及 umask 文件系统选项来限制对它的访问，详细叙述参见 mount 的 manual page。如果你不小心，加载了一个任何人都至少有读访问权限的 MS-DOS 文件系统，是不明智的。

4.8.4 使用 fsck 检查文件系统的完整性

文件系统是种复杂的东西，正因为如此，它们常会含有错误。一个文件系统的正确性和有效性可以用 **fsck** 命令来检查。可以用它来修复所发现的小问题，并且在有不可修复的问题时提醒用户注意。幸运的是，实现文件系统的代码能够很有效地进行调试，所以很少真正存在问题的，如果有问题通常也是因为掉电、硬件出错或者操作出错；例如，没有正常关闭系统等。

大多数系统都设置成在启动时自动执行 **fsck**，所以在系统使用前任何错误都可以检测出（并希望也被纠正过来）。使用一个已毁坏的文件系统会使得情况变得更糟：如果数据结构被破坏了，使用这个文件系统会造成更大的破坏，导致更多的数据丢失。然而，在一个大的文件系统上运行 fsck 需要耗费一段时间，而且如果系统是正常关闭的那么几乎是不可能发生错误的，此时可以使用几种技巧来来跳过对文件系统的检查。第一种是，如果存在文件/etc/fastboot，就不会对文件系统作检查。第二种方法是，ext2 文件系统在它的超级块内有一个特殊的标记来指出文件系统在前一次加载后是否正常地卸载。如果标志指出文件系统是正常卸载的（假设正常的卸载就表示文件系统没问题），那么 e2fsck（**fsck** 的 ext2 文件系统版）就不会对文件系统作检查。/etc/fastboot 技巧能否工作依赖于系统的启动描述文件，但是每当使用 **e2fsck** 命令时就有 ext2 技巧起作用。而且必须明确地 **e2fsck** 命令不带选项。（详细信息参见 **e2fsck** 的 man page。）

只有在启动时自动加载的文件系统才会被自动地检测。手工使用 **fsck** 命令来对其它文件系统进行检测，例如，软盘。

如果 **fsck** 发现了不可修复的问题，你要么需要有通常文件系统是如何工作的深入的知识以及知道坏文件系统的类型，要么有一个好备份。后一种方法是容易（尽管有时也是乏

味的）做到的。前者有时需要朋友、Linux 新闻组以及邮件列表或其他的支持源来做到，如果你不知道怎么办的话。有关这方面的事我真想告诉你更多些，但我的教育和经验的匮乏阻碍了我。Theodore T'so 的 **debugfs** 程序应该是很有用的。

fsck 必须对未加载的文件系统运行，绝对不要对已加载的文件系统运行（除了在启动期间对只读的根文件系统）。这是因为它访问磁盘的原始数据，因此能够在操作系统不知情的情况下修改文件系统。这样如果操作系统搞混的话就有可能带来问题。

4.8.5 使用 badblocks 检查磁盘错误

周期性地检查磁盘坏块是个好主意。这是用 **badblocks** 命令来做的。它输出一张它找到的所有坏块号的列表。这张列表可以送至 fsck 命令并记录在文件系统数据结构中使得操作系统不会用这些坏块来存储数据。下面的例子显示出了如何能做到这点。

```
$ badblocks /dev/fd0H1440 1440 > bad-blocks
$ fsck -t ext2 -l bad-blocks /dev/fd0H1440
Parallelizing fsck version 0.5a (5-Apr-94)
e2fsck 0.5a, 5-Apr-94 for EXT2 FS 0.5, 94/03/10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Check reference counts.
Pass 5: Checking group summary information.

/dev/fd0H1440: ***** FILE SYSTEM WAS MODIFIED *****
/dev/fd0H1440: 11/360 files, 63/1440 blocks
$
```

如果 badblocks 报告一个坏块已被使用，**e2fsck** 将会试着将该块上的数据移到另外的地方去。如果该数据块真的坏得一点也不能用了，而不仅只是边缘有些坏，那么所属的文件就破坏了。

4.8.6 整理磁盘碎片

当一个文件被写入磁盘，它不可能总被写进连续的块中。一个文件如果没有被存储于连续的块中，就称为是有碎片的(*fragmented*)。读有碎片的文件要更长的时间，因为磁盘的读写头要移动更多次。应该尽量避免碎片，尽管在一个有很好的预读缓冲的系统中不是个问题。

ext2 文件系统通过把在一个文件中的所有块都集中放在一起（即使它们不能存储于顺序的扇区中）试图使碎片尽可能地少。Ext2 文件系统总是有效地分配离文件中其它块最近的自由块给该文件。所以，对于 ext2 来说，很少需要担心碎片的事。有一个整理 ext2 文件系统碎片的程序，在参考书目中参见 XXX（ext2-defrag）。

有许多 MS-DOS 整理碎片的程序，用来在文件系统中移动块以除去碎片。对于其它文件系统，整理碎片的工作需要通过备份文件系统，然后重建该文件系统，并从备份中恢复

文件来做到除去碎片。对于所有文件系统来说在做整理碎片工作之前做文件系统的备份是个好主意，因为在整理碎片的过程中有许多事可能出差错。

4.8.7 文件系统的通用工具

有许多其它工具对管理文件系统也是很有用的。**df** 用于显示一个或多个文件系统上的空磁盘空间；**du** 用于显示一个目录以及其内所有文件所占用的磁盘空间。这些工具可用于查出浪费磁盘空间的地方。

Sync 使得缓冲中未写的块（见第五章中的缓冲一节）写入磁盘。很少需要手工做这事；后台进程 **update** 会自动地做这事。在碰到大灾难是它还是很有用的，例如，如果 **update** 或它的协助进程 **bdflush** 死去了，或者如果你必须立即关闭电源而等不及 **update** 来执行。

4.8.8 ext2 文件系统的通用工具

除了文件系统的创建程序(**mke2fs**)以及检查程序（**e2fsck**）可以直接使用或通过文件系统类型独立的前端使用外，ext2 文件系统还有其它一些有用工具。

tune2fs 用于调整文件系统参数。一些感兴趣的参数有：

。最大加载记数值。当文件系统被加载了太多次时，**e2fsck** 会迫使进行检查操作，即使完整标志已被设置。对于一个用于开发或测试的系统来说，将该数值降低是个好主意。

。检查之间的最长时间。**e2fsck** 也能确定两次检查之间等待的最长时间，即使完整标志已被设置，并且文件系统并没有不时地被加载。然而，这是可以使其不起作用的。

。为 root 保留的块数。Ext2 为 root 保留了一些块，以用于如果文件系统数据满了，在不删除任何东西的情况下，仍能做系统管理的工作。保留值的大小缺省值为百分之 5，这对于绝大多数硬盘来说并不浪费。然而，对于软盘来说，是无需保留任何块的。

更多信息请参见 **tune2fs** 的 manual page。

dumpe2fs 显示一个 ext2 文件系统的信息，绝大多数信息来自超级块。图 4-5 显示出了一个简单的输出结果。其中有些信息是技术上的并且需要对文件系统是如何工作的有个了解（见附录 XXX ext2fspaper），但是大多数信息即使对非管理员来说也是容易理解的。

图 4-5.dumpe2fs 的简单输出。

```
dumpe2fs 0.5b, 11-Mar-95 for EXT2 FS 0.5a, 94/10/23
Filesystem magic number:  0xEF53
Filesystem state:         clean
Errors behavior:          Continue
Inode count:              360
Block count:              1440
Reserved block count:     72
Free blocks:              1133
Free inodes:              326
First block:              1
Block size:               1024
```

```
Fragment size:                1024
Blocks per group:             8192
Fragments per group:          8192
Inodes per group:             360
Last mount time:              Tue Aug    8 01:52:52 1995
Last write time:              Tue Aug    8 01:53:28 1995
Mount count:                  3
Maximum mount count:          20
Last checked:                 Tue Aug    8 01:06:31 1995
Check interval:               0
Reserved blocks uid:          0 (user root)
Reserved blocks gid:          0 (group root)

Group 0:
  Block bitmap at 3, Inode bitmap at 4, Inode table at 5
  1133 free blocks, 326 free inodes, 2 directories
  Free blocks: 307-1439
  Free inodes: 35-360
```

debugfs 是文件系统的调试器(debugger)。它允许对磁盘上的文件系统的数据结构进行直接的访问，因此能用来维修文件系统已破坏但 fsck 不能自动修复的磁盘。它也因为用于恢复被删除的文件而著称。然而，debugfs 极需要你理解你所做的；一次误解可能破坏你的全部数据。

dump 以及 **restore** 能用于备份一个 ext2 文件系统。它们是传统 UNIX 备份工具的 ext2 的特定版本。有关备份的更多信息请参见第 10 章。

4.9 无文件系统的磁盘

不是所有的磁盘或分区被用作文件系统的。例如，交换分区上是没有文件系统的。许多软盘是以模拟磁带驱动器的方式使用的，所以一个 tar 文件或其它文件是直接写在原始盘片上的，而非文件系统中。Linux 启动盘片不含文件系统，只含有原始内核。

不使用文件系统有可以使得磁盘用途更多的好处，因为一个文件系统总有些薄记工作要做。这也使得磁盘更容易兼容于其它系统：例如，tar 文件的格式在所有的系统上都是一样的，而文件系统在大多数系统上是不同的。你会很快地习惯于没有文件系统的的磁盘如果你需要的话。可启动的 Linux 软盘也不必有个文件系统，尽管常常是这样的。

使用原始磁盘的一个理由是做它们的映像文件。例如，如果磁盘含有部分损坏的文件系统，在进行修复它之前做一个完全一样的拷贝是个好主意，从此，你可以再次更多的损坏的部分。这样做的一种方法是使用 **dd**：

```
$ dd if=/dev/fd0H1440 of=floppy-image
2880+0 records in
2880+0 records out
$ dd if=floppy-image of=/dev/fd0H1440
```

```
2880+0 records in
2880+0 records out
$
```

第一个 dd 作了一个软盘的完全一样的拷贝映像文件 floppy-image，第二个 dd 将映像文件写入软盘。（在第二个命令之前，用户大概已换过磁盘。否则的话这个命令对就很可能没一点用了。）

4.10 分配磁盘空间

4.10.1 分区的方案

以尽可能好的方法对一个磁盘进行分区并不是件容易的事。更糟糕的是没有普遍正确的方法来进行分区的；会有很多因素要加以考虑。

传统的方法是有一个（相对来说）较小的根（root）文件系统，它包含/bin、/etc、/dev、/lib、/tmp 以及其它一些用于启动与运行系统的信息。如此，根文件系统（在它自己的分区或磁盘上）是系统启动所需的一切。理由是，如果根文件系统较小且不是工作过重的话，当系统垮掉时它也不会轻易地毁坏，并且你会由此发现修复任何由系统垮掉而造成的问题是非常容易的。然后为目录/usr、用户主目录（常在/home 下）、以及交换空间创建独立的分区或使用独立的磁盘。在自己分区中的独立的用户主目录（含有用户文件）使得备份更容易，因为通常不需要备份程序（它们驻留在/usr 下）。在一个网络环境中，在几个机器之间共享/usr 也是可能的（例如，通过使用 NFS），由此减少了几十兆或几百兆乘上机器数量的总的磁盘空间容量。

有许多分区所带来的问题是把总的自由磁盘空间分成了许多小块。现今，当磁盘以及（有希望地）操作系统越来越可靠，许多人更喜欢只有一个含有所有文件的分区。另一方面来讲，备份（以及恢复）一个小的分区是很容易的。

对于一个小容量硬盘（假设你不作内核开发）来讲，最佳的方法很可能是只有一个分区。对于大硬盘来说，有几个大的分区通常是比较好的，只是万一确实出了问题。（注意，这里的‘小’与‘大’是相对来说的；对磁盘空间的需求决定了极限是什么。）

如果你有几个硬盘，你可能希望根文件系统（包括/usr）在一个硬盘上，而用户主目录在另一个上面。

用不同的分区方案作些试验是个好主意(随着时间的过去,而不仅是当第一次安装时)。这需要一些工作量，因为原则上需要你进行几次系统的安装，但这是能够确定你做得正确的唯一方法。

4.10.2 空间需求

你所要安装的 Linux 版本会给出一些针对不同的配置需要多少磁盘空间的指示。分开安装的程序也同样会给出容量的要求。这将帮助你计划硬盘空间的使用，但是你应该为将来所需准备并保留一些额外的空间。

你为用户文件准备的空间大小依赖于你的用户想要作些什么工作。大多数用户会为他们的文件要求尽可能多的硬盘空间。但是，他们所满意的容量是有很大不同的。有些用户只做一些轻微的字处理，只要有几兆字节就会满足，其他做繁重的图象处理的用户将需要

几个 GB 的硬盘空间。

顺便说一句，当比较以千字节或兆字节给出的文件的大小与以兆字节给出的磁盘空间的大小时，需要知道这两者的大小单位略有些不同。有些磁盘生产厂商喜欢称千字节为 1000 字节、兆字节为 1000 千字节，而除他们以外的计算机界都使用 1024 作为因子。因此，我的 345MB 硬盘其实是一个 330MB 的硬盘。[10]

4.10.3 硬盘空间分配实例

我以前有一个 109MB 的硬盘。现在我使用一个 330MB 的硬盘。我将解释怎样以及为什么我这样对这两个硬盘分区。

当我的需求以及我所使用的操作系统改变时，我的 109MB 硬盘被用许多方法分过区；这里我将解释两种典型的分区方案。首先，我习惯于将 MS-DOS 与 Linux 放在一起。为了这，我需要约 20MB 的硬盘空间，或是足够于存放 MS-DOS 系统、一个 C 编译器、一些其它的工具、和我正编制的程序以及足够的空闲磁盘空间即可。对于 Linux，我有一个 10MB 的交换区，剩下的 79MB 作成单个分区用于存放我在 Linux 下的所有文件。我试验过为 root、/usr 和/home 分别开设独立的分区，但每个分区总没有足够的空闲磁盘空间。

当我不再需要 MS-DOS 后，我对硬盘重新分了区，于是我就有了一个 12MB 的交换区以及又一次把剩余的空间作成了一个单独的文件系统。

我的 330MB 硬盘被分成了如下几个区：

5MB	Root 文件系统
10MB	交换分区
180MB	\fn{/usr}文件系统
120MB	\fn{/home}文件系统
15MB	临时使用分区

临时分区用于那些需要自己的分区的事情，例如，试用不同的 Linux 发行版本或者用于比较文件系统的速度。当再也需要这个分区时，就将它用作交换空间（我喜欢同时开着许多 windows 窗口）。

4.10.4 为 Linux 增加更多的磁盘空间

为 Linux 增加更多的磁盘空间是很容易的，至少是在硬件都安装妥当之后（硬件的安装不属于本书的讨论范围）。根据需要你可以先进行格式化，然后照上述创建分区和文件系统，并在/etc/fstab 文件中加入适当的命令行，使得它可以自动地被加载。

4.10.5 节省磁盘空间的一些技巧

节省磁盘空间的最好的技巧是不安装不需要的程序。大多数 Linux 发行版都有一个只安装部分所含的软件包的选项，通过分析你的需求，你可能会注意到大部分软件包你都是不需要的。这会节省许多的磁盘空间，因为其中有些程序是非常大的。甚至在你的确需要一个特定的软件包或程序时，你也不需要全都安装它。例如，有些随机文档是不必要的，还有些 GNU Emacs 的 Elisp 文件、有些 X11 的字库文件以及一些编程用的库文件都是不需

要的。

如果你不能卸下软件包，你可以用压缩的办法。诸如 **gzip** 或 **zip** 的压缩程序能够压缩（或解压）单个文件或一组文件。**gzexe** 系统能够不可见地[透明地]为用户压缩以及解压执行程序（未使用的程序被压缩，当得到使用时就解压）。正在试验中的 DouBle 系统能够压缩整个文件系统中的所有文件，而对于使用该文件系统的程序是不可见的[或说是透明的]。（如果你对诸如 MS-DOS 下的 Stacker 程序熟悉的话，它们的原理是一样的。）

注释

[1] 圆盘是用坚硬的物质做成的，例如，铝合金，硬盘也就从此得名。

[2] BIOS 是一些存储于 ROM 芯片中的内建的软件。除了其它的功能外，它管理启动最初的步骤。

[3] 这些数据是完全虚构的。

[4] 也就是说，盘片内的表面，在塑胶覆盖里面的金属盘。

[5] 当然也是完全不同的。

[6] 不合逻辑吗？

[7] 有关更多的信息，参见内核原代码或内核黑客手册。

[8] 它当然是非加载的，但是 n 在 70 年代神秘地消失了，并且再也没出现。如果你发现了它，请告知贝尔实验室，NJ。

[9] 为了用户的利益，需要好好思考几秒钟。

[10]Sic transit discus mundi。

第5章 内存管理

“Minnet, jag har tappat mitt minne, 鋓 jag svensk eller finne, kommer inte ih 鑫...” (Bosse 譁terberg)

本章描述了 Linux 内存管理的特性，也即，虚拟内存和磁盘缓冲。叙述了系统管理员需要考虑到的内存管理的目的、工作原理以及其他一些事情。

5.1 什么是虚拟内存？

Linux 支持虚拟内存（*virtual memory*），虚拟内存是指使用磁盘当作 RAM 的扩展，这样可用的内存的大小就相应地增大了。内核会将暂时不用的内存块的内容写到硬盘上，这样一来，这块内存就可用于其它目的。当需要用到原始的内容时，它们被重新读入内存。这些操作对用户来说是完全透明的；Linux 下运行的程序只是看到有大量的内存可供使用而并没有注意到时不时它们的一部分是驻留在硬盘上的。当然，读写硬盘要比直接使用真实内存慢得多（要慢数千倍），所以程序就不会象一直在内存中运行的那样快。用作虚拟内存的硬盘部分被称为交换空间（*swap space*）。

Linux 能够使用文件系统中的一个常规文件或一个独立的分区作为交换空间。交换分区要快一些，但是很容易改变交换文件的大小（也就无需重分区整个硬盘，并且可以从临时分区中安装任何东西）。当你知道你需要多大的交换空间时，你应该使用交换分区，但是如果你不能确定的话，你可以首先使用一个交换文件，然后使用一阵子系统，你就可以感觉到要有多大的交换空间，此时，当你能够确信它的大小时就创建一个交换分区。

你应该知道，Linux 允许同时使用几个交换分区以及/或者交换文件。这意味着如果你只是偶尔地另外需要一个交换空间时，你可以在当时设置一个额外的交换文件，而不是一直分配这个交换空间。

操作系统术语注释：计算机科学常常将交换[swapping](将整个进程写到交换空间)与页面调度[paging]（在某个时刻，仅仅固定大小的几千字节写到交换空间内）加以区别。页面调度通常更有效，这也是 Linux 的做法，但是传统的 Linux 术语却指的是交换。[1]

5.2 创建交换空间

一个交换文件是一个普通的文件；对内核来说一点也不特殊。对内核有关系的是它不能有孔，并且它是用 **mkswap** 来准备的。而且，它必须驻留在一个本地硬盘上，它不能由于实现的原因而驻留在一个通过 NFS 加载的文件系统中。

关于孔是重要的。交换文件保留了磁盘空间，以至于内核能够快速地交换出页面而无需做分配磁盘扇区给文件时所要做的一些事。内核仅仅是使用早已分配给交换文件的任何扇区而已。因为文件中的一个孔意味着没有磁盘扇区分配（给该文件的孔的相应部分），对内核来说就不能使用这类有孔的文件。

创建无孔的交换文件的一个好方法是通过下列命令：

```
$ dd if=/dev/zero of=/extra-swap bs=1024 count=1024
1024+0 records in
1024+0 records out
$
```

上面/extra-swap 是交换文件的名字，大小由 count=后面的数值给出。大小最好是 4 的倍数，因为内核写出的内存页面（*memory pages*）大小是 4 千字节。如果大小不是 4 的倍数，最后几千字节就用不上了。

一个交换分区也并没有什么特别的。你可以象创建其它分区一样地创建它；唯一的区别在于它是作为一个原始的分区使用的，也即，它不包括任何的文件系统。将交换分区标记为类型 82（Linux 交换分区）是个好主意；这将使得分区的列表更清楚，尽管对内核来说并不是一定要这样的。

在创建了一个交换文件或一个交换分区以后，你必须在它的开头部分写上一个签名；这个签名中包括了一些由内核使用的管理信息。这是用\cmd{mkswap}命令来做到的，用法如下：

```
$ mkswap /extra-swap 1024
Setting up swapspace, size = 1044480 bytes
$
```

请注意此时交换空间还没有被使用：它已存在，但内核还没有用它作为虚拟内存。

你必须非常小心地使用 **mkswap**，因为它不检查这个文件或分区是否已被别人使用。你可以非常容易地使用 **mkswap** 来覆盖重要的文件以及分区！幸运的是，仅仅在安装系统时，你才需要使用 **mkswap**。

Linux 内存管理程序限制每个交换空间最大约为 127MB（由于各种技术上的原因，实际的限制大小为(4096-10) * 8 * 4096 = 133890048$ 字节，或 127.6875 兆字节）。然而，你可以同时使用多至 16 个交换空间，总容量几乎达 2GB。[2]

5.3 交换空间的使用

一个已初始化的交换空间是使用命令 **swapon** 投入正式使用的。该命令告诉内核这个交换空间可以被使用了。到交换空间的路径是作为参数给出的，所以，开始在一个临时交换文件上使用交换的命令如下：

```
$ swapon /extra-swap
$
```

通过把交换空间列入/etc/fstab 文件中就能被自动地使用了。

```
/dev/hda8       none       swap       sw       0       0
/swapfile       none       swap       sw       0       0
```

启动描述文件会执行命令 **swapon –a**，这个命令会启动列于/etc/fstab 中的所有交换空间。因此，**swapon** 命令通常仅用于需要有外加的交换空间时。

你可以用 **free** 命令监视交换空间的使用情况。它将给出已使用了多少的交换空间。

```
$ free
             total       used       free     shared    buffers
Mem:         15152      14896        256      12404       2528
-/+ buffers:             12368       2784
Swap:        32452       6684      25768
$
```

输出的第一行（Mem:）显示出物理内存的使用情况。总和(total)列中并没有显示出被内核使用的内存，它通常将近一兆字节。已用列(used column)显示出已用内存的总和（第二行没有把缓冲算进来）。空闲列（free column）显示了所有未被使用的空闲内存。共享列（shared column）显示出了被几个进程共享的内存的大小；共享的内存越多，情况就越好。缓存列（buffer column）显示出了当前磁盘缓存的大小。已缓冲列（cached column）显示出了已使用的缓存的大小。

最后一行（Swap:）显示出了与交换空间相应的信息。如果这一行的数值都是零，表示你的交换空间没有被击活。

也可通过用 **top** 命令来获得同样的信息，或者使用 proc 文件系统中的文件 /proc/meminfo 。通常要取得指定交换空间的使用情况是困难的。

可以使用命令 **swapoff** 来移去一个交换空间。通常没有必要这样做，但临时交换空间除外。一般，在交换空间中的页面首先被换入内存；如果此时没有足够的物理内存来容纳它们又将被交换出来（到其他的交换空间中）。如果没有足够的虚拟内存来容纳所有这些页面，Linux 就会波动而不正常；但经过一段较长的时间 Linux 会恢复，但此时系统已不可用了。在移去一个交换空间之前，你应该检查（例如，用 **free**）是否有足够的空闲内存。

任何由 **swapon** –a 而自动被使用的所有交换空间都能够用 **swapoff –a** 命令移去；该命令参考/etc/fstab 文件来确定移去什么。任何手工设置使用的交换空间将始终可以被使用。

有时，尽管有许多的空闲内存，仍然会有许多的交换空间正被使用。这是有可能发生的，例如如果在某一时刻有进行交换的必要，但后来一个占用很多物理内存的大进程结束并释放内存时。被交换出的数据并不会自动地交换进内存，除非有这个需要时。此时物理内存会在一段时间内保持空闲状态。对此并没有什么可担心的，但是知道了是怎么一回事我们也就放心了。

5.4 与其它操作系统共享交换空间

许多操作系统使用了虚拟内存的方法。因为它们仅在运行时才需要交换空间，以即决不会在同一时间使用交换空间，因此，除了当前正在运行的操作系统的交换空间，其它的就是一种浪费。所以让它们共享一个交换空间将会更有效率。这是可能的，但需要有一定的了解。在 HOWTO 技巧文档中含有如何实现这种做法的一些建议。

5.5 分配交换空间

有些人会对你说需要用物理内存的两倍容量来分配交换空间，但这是不对的。下面是合适的做法：

。估计你的总内存需求。这是某一时刻你所需要的最大的内存容量，也就是在同一时

刻你想运行的所有程序所需内存的总和。通过同时运行所有的程序你可以做到这一点。

例如，如果你要运行 X，你将给它分配大约 8MB 内存，gcc 需要几兆字节（有些文件要求异呼寻常的大量的内存量，多至几十兆字节，但通常约 4 兆字节应该够了），等等。内核本身要用大约 1 兆字节、普通的 shell 以及其它一些工具可能需要几百千字节（就说总和要 1 兆字节吧）。并不需要进行精确的计算，粗率的估计也就足够了，但你必须考虑到最坏的情况。

注意，如果会有几个人同时使用这个系统，他们都将消耗内存。然而，如果两个人同时运行一个程序，内存消耗的总量并不是翻倍，因为代码页以及共享的库只存在一份。

Free 以及 ps 命令对估计所需的内存容量是很有帮助的。

。对第一步中的估计放宽一些。这是因为对程序在内存中占用多少的估计通常是不准的，因为你很可能忘掉几个你要运行的程序，以及，确信你还要有一些多余的空间用于以防万一。这需几兆字节就够了。（多分配总比少分配交换空间要好，但并不需要过分这样以至于使用整个硬盘，因为不用的交换空间是浪费的空间；参见后面的有关增加交换空间。）同样，因为处理数值更好做，你可以将容量值加大到整数兆字节。

。基于上面的计算，你就知道了你将需要总和为多少的内存。所以，为了分配交换空间，你仅需从所需总内存量中减去实际物理内存的容量，你就知道了你需要多少的交换空间。（在某些 UNIX 版本中，你还需要为物理内存的映像分配空间，所以第二步中算出的总量正是你所需要的交换空间的容量，而无需再做上述中的减法运算了。）

。如果你计算出的交换空间容量远远大于你的物理内存（大于两倍以上），你通常需要再买些内存来，否则的话，系统的性能将非常低。

有几个交换空间是个好主意，即使计算指出你一个都不需要。Linux 系统常常动不动就使用交换空间，以保持尽可能多的空闲物理内存。即使并没有什么事情需要内存，Linux 也会交换出暂时不用的内存页面。这可以避免等待交换所需的时间：当磁盘闲着，就可以提前做好交换。

可以将交换空间分散在几个硬盘之上。针对相关磁盘的速度以及对磁盘的访问模式，这样做可以提高性能。你可能想实验几个方案，但是你要认识到这些实验常常是非常困难的。不要相信其中一个方案比另一个好的说法，因为并不总是这样的。

5.6 高速缓冲

与访问（真正的）的内存相比，磁盘[3]的读写是很慢的。另外，在相应较短的时间内多次读磁盘同样的部分也是常有的事。例如，某人也许首先阅读了一段 e-mail 消息，然后为了答复又将这段消息读入编辑器中，然后又在将这个消息拷贝到文件夹中时，使得邮件程序又一次读入它。或者考虑一下在一个有着许多用户的系统中 ls 命令会被使用多少次。通过将信息从磁盘上仅读入一次并将其存于内存中，除了第一次读以外，可以加快所有其它读的速度。这叫作磁盘缓冲（*disk buffering*），被用作此目的的内存称为高速缓冲（buffer cache）。

不幸的是，由于内存是一种有限而又不充足的资源，高速缓冲不可能做的很大（它不可能包容要用到的所有数据）。当缓冲充满了数据时，其中最长时间不用的数据将被舍弃以腾出内存空间用于新的数据。

对写磁盘操作来说磁盘缓冲技术同样有效。一方面，被写入磁盘的数据常常会很快地又被读出（例如，原代码文件被保存到一个文件中，又被编译器读入），所以将要被写的数据放入缓冲中是个好主意。另一方面，通过将数据放入缓冲中，而不是将其立刻写入磁盘，程序可以加快运行的速度。以后，写的操作可以在后台完成，而不会拖延程序的执行。

大多数操作系统都有高速缓冲（尽管可能称呼不同），但是并不是都遵守上面的原理。有些是直接写（*write-through*）：数据将被立刻写入磁盘（当然，数据也被放入缓存中）。如果写操作是在以后做的，那么该缓存被称为后台写（*write-back*）。后台写比直接写更有效，但也容易出错：如果机器崩溃，或者突然掉电，或者是软盘在缓冲中等待写的数据被写入软盘之前被从驱动器中取走，缓冲中改变过的数据就被丢失了。如果仍未被写入的数据含有重要的薄记信息，这甚至可能意味着文件系统（如果有的话）已不完整。

由于上述原因，在使用适当的关闭过程之前，绝对不要关掉电源（见第六章），不要在卸载（如果已被加载）之前将软盘从驱动器中取出来，也不要在任何正在使用软盘的程序指示出完成了软盘操作并且软盘灯熄灭之前将软盘取出来。**sync** 命令倾空（*flushes*）缓冲，也即，强迫所有未被写的数据写入磁盘，可用以确定所有的写操作都已完成。在传统的 UNIX 系统中，有一个叫做 **update** 的程序运行于后台，每隔 30 秒做一次 **sync** 操作，因此通常无需手工使用 **sync** 命令了。Linux 另外有一个后台程序，**bdflush**，这个程序执行更频繁的但不是全面的同步操作，以避免有时 **sync** 的大量磁盘 I/O 操作所带来的磁盘的突然冻结。

在 Linux 中，**bdflush** 是由 **update** 启动的。通常没有理由来担心此事，但如果由于某些原因 **bdflush** 进程死掉了，内核会对此作出警告，此时你就要手工地启动它了（**/sbin/update**）。

缓存（cache）实际并不是缓冲文件的，而是缓冲块的，块是磁盘 I/O 操作的最小单元（在 Linux 中，它们通常是 1KB）。这样，目录、超级块、其它文件系统的薄记数据以及非文件系统的磁盘数据都可以被缓冲了。

缓冲的效力主要是由它的大小决定的。缓冲大小太小的话等于没用：它只能容纳一点数据，因此在被重用时，所有缓冲的数据都将被倾空。实际的大小依赖于数据读写的频次、相同数据被访问的频率。只有用实验的方法才能知道。

如果缓存有固定的大小，那么缓存太大了也不好，因为这会使得空闲的内存太小而导致进行交换操作（这同样是慢的）。为了最有效地使用实际内存，Linux 自动地使用所有空闲的内存作为高速缓冲，当程序需要更多的内存时，它也会自动地减小缓冲的大小。

在 Linux 中，你不需要为使用缓冲做任何事情，它是完全自动处理的。除了上面所提到的有关按照适当的步骤来关机和取出软盘，你不用担心它。

注释

[1] 因此常常没有必要地令一些计算机专家感到恼怒。

[2] 这里十亿字节，那里十亿字节，很快我们就开始讨论有关实际内存了。

[3] 除了 RAM 磁盘，这是个明显的理由。

第6章 启动与关闭

让我行动起来
哦…你必须如此…你必须如此
永远、永远永远不要停止
让它动起来
哦…让它动起来，永远、永远永远不要停止
你使一个成年人哭泣，
你使一个成年人哭泣
（滚石乐队）

本节解释了当 Linux 系统启动以及关闭时，做了些什么，以及为什么要适当地操作。如果不按照适当的步骤来做，文件有可能毁坏或丢失。

6.1 启动与关闭概述

打开一个计算机系统并把它的操作系统装入[1]内存中的过程称为引导（*booting*）。这个名字来自于计算机用引导程序将自己启动起来，但是这个过程本身实际上是比较复杂的。

在引导过程其间，计算机首先装入一段称为引导装入程序（*bootstrap loader*）的很短的一块代码，接下来，这块代码装入操作系统代码并运行之。引导装入程序通常存储于硬盘或软盘上的固定位置处。分成这两个过程的原因是，操作系统是一个大又复杂的程序，但是计算机装入的第一段程序必须很短小（几百个字节吧），以避免使得固件毫无必要地过分复杂。

不同类型的计算机的引导过程是不同的。对于 PC 来说，计算机（它的 BIOS）读入软盘或硬盘的第一个扇区（称为引导扇区 *boot sector*）。引导装入程序就包含在这个扇区中。它再将操作系统从磁盘（或其它的地方）的某处装入到内存中去。

在 Linux 被装入内存后，它就初始化各种硬件以及设备驱动程序，然后运行 **init**。**init** 程序再启动其它进程，以允许用户登录和做工作。这部分的细节将在下面加以讨论。

为了关闭一个 Linux 系统，首先所有的进程都被告知进行结束操作（这使得它们能够关闭任何文件以及做其它必要的操作，以使得各件事情有条有理），然后文件系统和交换区被卸载下来，最后一条说明可以关闭电源的消息显示在控制台屏幕上。如果没有按照正确的步骤来做，很有可能发生可怕的事情；最重要的是，文件系统高速缓冲可能没被刷新，这意味着高速缓冲中的数据被丢失并且磁盘上的文件系统变得不完整了，因此文件系统很有可能变得不可用了。

6.2 引导过程详述

你可以从软盘或硬盘上来引导 Linux 系统。安装与使用手册中的安装部分（XXX 引用）告诉你如何安装 Linux 系统，所以你可以用想用的方式来引导系统。

当引导一个 PC 系统时，BIOS 会做各种测试操作以检查一切是否正常，[1] 然后将开

始正真的引导过程。它将选择一个磁盘驱动器（典型地，如果有软盘插入在驱动器中，就首先选择软盘驱动器，否则就选择计算机中安装着的第一个硬盘；然而，选择的顺序通常是可以配置的）并且将读入它的第一个扇区。这个扇区称为引导扇区（*boot sector*）；对于硬盘来说，它也被称为主引导记录（*master boot record*），因为一个硬盘可以包含有几个分区，每个分区都有它自己的引导扇区。

引导扇区中含有一个短小的程序（小得可以纳入一个扇区中）它的任务是将实际的操作系统从硬盘上读入内存并启动操作系统。当从软盘上引导一个 Linux 系统时，引导扇区中含有一些代码，这些代码将开始的几百个数据块（当然依赖于实际的内核的大小）读入到内存预定的地方。在 Linux 引导软盘上，不含有文件系统，内核只是存储在连续的扇区中，因此这样可以简化引导的过程。然而，从含有文件系统的软盘上引导一个系统也是可能的，即通过使用 LILO，Linux 装入程序（LInux LOader）。

当从硬盘上引导时，主引导记录中的代码将检查分区表（该表同样在主引导记录中）、确定活动分区（标记为可以引导的扇区）、从该分区读入引导扇区并开始执行该引导扇区中的代码。该分区的引导扇区中的代码做与软盘的引导扇区同样的事情：从该分区中将内核读入内存中并启动它。然而，详细过程有所不同，因为，只含有内核映像的独立分区通常是无益的，所以该分区的引导扇区的代码不能只是按顺序去读磁盘，它必须寻找到文件系统放置内核的扇区。有几种方法来处理这些问题，但最常用的方法是使用 LILO。（然而具体如何使用与本讨论无关；详细信息参见 LILO 的文档；叙述的很完整的。）

当使用 LILO 进行引导过程时，通常会直接引导缺省的内核并启动它。配置 LILO，使之可以引导几个内核中的一个或者甚至是不同与 Linux 的其它操作系统，这也是可能的，并且，让用户在引导期间选择引导哪个内核或哪种操作系统也是可能的。LILO 可以配置成为在引导时刻如果按下 **alt**、**shift** 或者 **ctrl** 键时（当 LILO 被装入内存），LILO 会提示要引导除了缺省内核的那一种。另外，LILO 可以配置成为有缺省提问时间的状态，当缺省时间一到就引导缺省内核。

使用 LILO，在内核或操作系统名称之后给出一内核命令行参数（*kernel command line argument*）也是可能的。

从软盘引导或从硬盘引导各有各的优点，但是一般来说从硬盘上来引导更好一些，因为这避免了使用软盘的争论。这样做也更快。但是，安装系统从硬盘引导可能更麻烦一点，所以许多人会首先从软盘来引导系统，然后，当系统安装好并且运行正常之后，再安装 LILO 并从硬盘引导启动。

在不管用什么方法，Linux 内核被读入内存以后，就会大概地发生以下事情：

。Linux 内核是压缩安装的，所以首先它要自解压。内核映像文件的开头部分含有一段小程序来做解压工作。

。如果你有一块 Linux 能识别出的 super-VGA 卡并且它有几种特殊的文本模式（比如 100 列、40 行），Linux 会提问你要用哪一种文本模式。在编译内核期间，可以预设一种显示模式，就不会再有这种提问出现了。使用 LILO 或 **rdev** 也同样能做到。

。在这之后，内核会检查有些什么其它的硬件（硬盘、软驱、网络适配器等等），并会适当地配置其中一些硬件的设备驱动文件；在内核做这些工作时，会输出有关检查结果的信息。例如，当我引导系统时，就有如下信息：

```
LILO boot:
Loading linux.
Console: colour EGA+ 80x25, 8 virtual consoles
Serial driver version 3.94 with no serial options enabled
```

```
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
lp_init: lp1 exists (0), using polling driver
Memory: 7332k/8192k available (300k kernel code, 384k reserved, 176k data)
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
Loopback device init
Warning WD8013 board not found at i/o = 280.
Math coprocessor using irq13 error reporting.
Partition check:
  hda: hda1 hda2 hda3
VFS: Mounted root (ext filesystem).
Linux version 0.99.pl9-1 (root@haven) 05/01/93 14:12:20
```

在不同的系统上，上面显示的文字会有所不同。这与所使用的硬件、所用的 Linux 版本以及与它是如何配置的有关。

。然后，内核将试图加载根文件系统。根文件系统被加载的地方，在内核编译的时候可以配置，也可以在任何时候用 **rdev** 或 LILO 来加以指定。文件系统的类型是自动检测出的。如果根文件系统的加载失败了，例如，因为你忘了将相应的文件系统的驱动程序包含入内核，内核将不知所措而使系统停机（不管怎样，此时内核没别的可做）。

根文件系统是以只读方式加载的（这可以用设置地点的同样方式来设置）。这使得文件系统被加载后也能够进行检查；对以读写方式加载的文件系统进行检查不是个好主意。

。在这以后，内核在后台（这将总是进程 1 号）启动程序 **init**（位于/sbin/init）。**Init** 做各种各样的启动琐事。它所做的确切的事要视它是如何配置的而定；更多信息参见第七章（至今还未写）。至少它会启动一些基本的后台程序。

。然后，**init** 程序转换到多用户模式，并且为虚拟终端以及串行线路启动一个 **getty** 程序。**getty** 这个程序用于让人通过虚拟终端以及串行终端登录系统。**init** 也可能启动其它一些程序，这要看它是如何配置而定。

。这之后，引导过程就完成了，并且系统启动并正常运行了。

6.3 关闭系统详述

按照正确的过程来关闭 Linux 系统是很重要的。如果你没有这样做，你的文件系统很可能会变得毫无价值并且文件也会变得混乱。这是因为 Linux 有一个磁盘缓冲，它不会将数据立刻写入磁盘的，而是在一定的间隔时间。这极大地改进了性能，但也意味着如果你随意地关闭电源，此时缓冲中可能保存着许多数据并且磁盘上的数据会不完全而不能成为一个完整运行的文件系统（因为只有一些数据被写入硬盘）。

不能仅仅扳动电源开关的另一个理由是，在一个多任务系统中，后台可能正运行着许多程序，随意地关闭电源损失会非常惨重的。通过进行适当的关机过程，你可以确信所有的后台进程都能保存它们的数据。

正确地关闭 Linux 系统的命令是 **shutdown**。它通常有两种方法使用。

如果你正在运行只有你一个用户的系统，使用 **shutdown** 的通常方法是关闭所有正在运行的程序，退出所有的虚拟终端，以 root 在一个虚拟终端上登录（或者如果你已是 root 的话就不要退出了，但你要转到 root 的主目录中或 root 目录中，以避免不卸载的问题），

然后执行命令 **shutdown –h now**（如果你要延迟关闭系统，则用+号和一个代表分钟的数字来替代 now，尽管你在单用户系统上不需要这样做）。

另一种方法是，如果你的系统上现有许多用户，则使用命令 **shutdown -h +time message**，这里 time 是系统停止的延迟时间，message 系统为什么要关闭的简短解释消息。

```
# shutdown -h +10 'We will install a new disk.  System should
> be back on-line in three hours.'
#
```

这将警告所有用户系统将在十分钟内关闭，并且他们最好马上退出以防止数据的丢失。警告信息将显示在所有登录的用户的终端上，包括所有 **xterms**：

```
Broadcast message from root (ttyp0) Wed Aug  2 01:03:25 1995...

We will install a new disk.  System should
be back on-line in three hours.
The system is going DOWN for system halt in 10 minutes !!
```

[
从 root(ttyp0)播出消息，Wed Aug 2 01:03:25 1995

```
我们将安装一个新硬盘。系统将
在 3 小时后启动。
系统将在 10 分钟内关闭!!
```

]

警告信息在系统关闭之前会自动重复几遍，而且随着关闭时刻的迫近，警告信息播出的时间间隔越来越短。

当延迟时间到开始正真的关闭操作时，所有的文件系统（除了根文件系统）都被卸载，所有用户进程（如果还有人没退出）都被终止，后台程序也关闭了，所有文件系统被卸载，并且一般地所有的工作安排都关闭了。当所有这一切做完以后，**init** 打印出你可以关闭系统电源的提示。然后，也仅仅在此时，你才可以关闭机器的电源。

尽管在任何正常的机器上很少发生，有时，不能正常地关闭系统。例如，如果内核发生问题并且错误地执行指令，就会完全不接受任何新的命令，因此，想正常地关闭系统就很困难了，你此时所能做的就是期望系统没有遭到严重的损坏并把电源关掉。如果问题不严重的话（例如，有人砍了你的键盘），并且内核以及 **update** 程序仍能正常地工作的话，等待几分钟，给 **update** 一个机会来刷新缓冲，并仅在此之后关闭电源。

有些人喜欢使用三次 **sync** [3] 来关闭系统，等待磁盘 I/O 停止，然后关闭电源。如果没有程序运行着，那么这和使用 **shutdown** 是等同的。然而，这样做没有卸载任何文件系统并且可能会导致 ext2fs 的“干净文件系统”标志。三次使用 sync 来关闭系统的方法不值得推荐。

（如果你觉得奇怪话：使用三个 sync 是因为在 UNIX 的早期，当命令分开来键入时，对磁盘 I/O 通常会有足够的时间。）

6.4 重启动

重启动表示再次启动引导系统。做法是首先完全关闭系统，关闭电源，然后再打开电源。一个更简单的方法是请求 **shutdown** 来重启动系统，而不是仅仅停止系统。这是用带选项-r 的 **shutdown** 来做的，例如，执行命令 **shutdown –r now**。

当按下 ctrl-alt-del 组合键时，大多数 Linux 系统运行 **shutdown –r now** 来重启动系统。然而，ctrl-alt-del 组合键的功能是可以配置的，因此在一个多用户的机器上最好是在重启动前给出一些延迟时间。对于任何人都可以动的系统甚至可以将 ctrl-alt-del 配置成不起任何作用。

6.5 单用户模式

shutdown 命令也可以将系统转到单用户模式，在这种方式下除了 root 使用控制台，别人是不可以登录系统的。这对于系统正常运行时不能进行的系统管理任务是非常有用的。

6.6 紧急启动（引导）盘

并不是总能从计算机的硬盘上引导启动的。例如，如果你配置 LILO 有误，会使你的系统不可引导。针对这些情况，你需要另外一个总能工作的方法（在硬盘能工作时）。对于典型的 Pc 机来说，这是指从软盘驱动器上启动。

大多数 Linux 发行版本允许在安装时创建一个紧急引导盘（*emergency boot floppy*）。这样做是好有益处的。然而，有些这样的引导盘仅含有一个内核，并且假设使用发行的安装盘上的程序来修复任何碰到的问题。有时候有了这些程序还是不够的；例如，你可能需要从备份中恢复一些文件，但所用的备份软件在安装盘没有。

因此，有可能需要创建一张定制的 root 软盘。Graham Chapman 著的 Bootdisk HOWTO（XXX 引用）这方面的说明。当然，你必须保持你的紧急引导盘和 root 盘是最新的。

加载了 root 盘的软盘驱动器不能再用于其它用途。这对于只有一个软盘驱动器的人来说甚为不便。然而，如果你有足够的内存的话，你可以配置你的引导盘将 root 盘的内容装入 RAM 盘中（针对于此，引导盘上的内核需要进行特别的配置）。一旦 root 盘的内容被装入 RAM 盘中，软盘驱动器就可以加载其它磁盘了。

注释

[1] 在早期的计算机中，仅仅打开计算机电源是不够的，你还要手工地将操作系统装入内存。而这些新发明的事情会自己执行完所有这些过程。

[2] 这称为开机自检（*power on self test*），或简称 *POST*。

[3] sync 用于刷新高速缓冲。

第7章 初始化进程（init）

“Uuno on numero yksi ”（Slogan for a series of Finnish movies.）

本章描述 **init** 进程，它是内核启动的第一个用户级进程。**init** 有许多很重要的任务，比如象启动 **getty**（用于用户登录）、实现运行级别、以及处理孤立进程。本章解释了怎样配置 **init** 以及如何运用不同的运行级别。

7.1 第一步init

对于 Linux 系统的运行来说，**init** 程序是最基本的程序之一。但你仍可以大部分的忽律它。一个好的 Linux 发行版本通常随带有一个 **init** 的配置，这个配置适合于绝大多数系统的工作，在这样一些系统上不需要对 **init** 做任何事。通常，只有你在碰到诸如串行终端挂住了、拨入（不是拨出）调制解调器、或者你希望改变缺省的运行级别时你才需要关心 **init**。

当内核启动了自己之后（已被装入内存、已经开始运行、已经初始化了所有的设备驱动程序和数据结构等等），通过启动用户级程序 **init** 来完成引导进程的内核部分。因此，**init** 总是第一个进程（它的进程号总是 1）。

内核在几个位置上来查寻 **init**，这几个位置以前常用来放置 **init**，但是 **init** 的最适当的位置（在 Linux 系统上）是/sbin/init。如果内核没有找到 **init**，它就会试着运行/bin/sh，如果还是失败了，那么系统的启动就宣告失败了。

当 **init** 开始运行，它通过执行一些管理任务来结束引导进程，例如检查文件系统、清理/tmp、启动各种服务以及为每个终端和虚拟控制台启动 **getty**，在这些地方用户将登录系统（见第八章）。

在系统完全起来之后，**init** 为每个用户已退出的终端重启 **getty**（这样下一个用户就可以登录）。**init** 同样也收集孤立的进程：当一个进程启动了一个子进程并且在子进程之前终止了，这个子进程立刻成为 **init** 的子进程。对于各种技术方面的原因来说这是很重要的，知道这些也是有好处的，因为这便于理解进程列表和进程树图。[1] **init** 的变种很少。绝大多数 Linux 发行版本使用 **sysinit**（由 Miguel van Smoorenburg 著），它是基于 System V 的 **init** 设计。UNIX 的 BSD 版本有一个不同的 **init**。最主要的不同在于运行级别：System V 有而 BSD 没有（至少是传统上说）。这种区别并不是主要的。在此我们仅讨论 **sysvinit**。

7.2 配置init以启动getty：/etc/inittab文件

当 **init** 启动后，**init** 读取/etc/inittab 配置文件。当系统正在运行时，如果发出 HUP 信号，**init** 会重读它；[2] 这个特性就使得对 **init** 的配置文件作过的更改不需要再重新启动系统就能起作用了。

/etc/inittab 文件有点复杂。我们将从配置 **getty** 行的简单情况说起。etc/inittab 中的行由四个冒号限定的域组成：

id:runlevels:action:process

下面对各个域进行了描述。另外，/etc/inittab 可以包含空行以及以数字符号（'#'）开始的行；这些行均被忽略。

id

这确定文件中的一行。对于 **getty** 行来说，指定了它在其上运行的终端（设备文件名/dev/tty 后面的字符）。对于别的行来说，是没有意义的（除了有长度的限制），但它必须是唯一的。

7.3 runlevels

该行应考虑的运行级别。运行级别以单个数字给出，没有分隔符。（运行级别在下一节中讨论。）

action

对于该行应采取的动作，也即，respawn 再次运行下一个域中的命令，当它存在时，或者仅运行一次。

process

要运行的命令。

为了在第一个虚拟终端上（/dev/tty1）运行 **getty**、在所有的正规多用户运行级别中（2-5），应该写入下面这行：

1:2345:respawn:/sbin/getty 9600 tty1

第一个域指出这是对应于/dev/tty1 的行。第二个域说明它应用于运行级别 2，3，4 和 5。第三个域是说在命令退出之后，应被再次执行（因此，用户可以登录、退出并且再次登录）。最后一个域是在第一个虚拟终端上运行 **getty** 的命令。[3]

如果你需要给系统增加终端或者拨入调制解调器线路，你应该给/etc/inittab 增加更多的行，每一行对应一个终端或一条拨入线。详细信息，参见 **init**、inittab 以及 **getty** 的 manual page。

如果一个命令运行时失败了，并且 init 配置成重运行它，它会使用许多的系统资源：init 运行它、它失败了、init 再运行它、再次失败等等，没完没了。为了避免这样，init 将追踪一个命令重运行了多少次，并且如果重运行的频率太高，它将被延时五分钟后再运行。

一个运行级别（*run level*）是 init 以及整个系统的状态，它定义了能够提供什么系统服务。运行级别用数字来定义，见表 7-1。对于如何使用用户定义运行级别（2 到 5）没有一致的意见。有些系统管理员使用运行级别来定义哪个子系统工作，也即，X 是否能运行、网络是否能工作等等。其他人总是让所有子系统工作着或者单独地运行以及停止它们，而不改变它们的运行级别，因为运行级别对于控制他们的系统来说显得太粗率了。你必须自己决定，但是按照你的 Linux 发行版本的做法来做也许是最容易的了。

表 7-1. 运行级别数

0	终止系统

1	单用户模式（用于特别管理）
2-5	正常操作（用户定义）
6	重启动

运行级别通过如下行所示的行在/etc/inittab 中配置：

l2:2:wait:/etc/init.d/rc 2

第一个域是任意给的符号，第二个域指出是运行级别 2。第三个域说明当进入该运行级别时，**init** 应该运行第四个域中的命令一次，并且 **init** 应该等待它的结束。在进入运行级别 2 时，在需要时/etc/init.d/rc 命令运行或者停止服务。

第四个域中的命令做所有设置一个运行级别的艰巨工作。它启动还没有运行的服务，并且停止在新的运行级别中不应再运行的服务。确切的命令是什么以及运行级别是如何配置的，依赖于各个 Linux 发行版本。

当 **init** 开始运行时，它在/etc/inittab 中查寻一行，该行指定了缺省的运行级别：

id:2:initdefault:

通过给内核一个 single 或 emergency 命令行参数，你可以在 **init** 运行开始时转到一个非缺省的运行级别上。例如，内核命令行参数可以通过 LILO 给出。这使得你可以选择单用户模式（运行级别 1）。

当系统正在运行时，**telinit** 命令可以改变运行级别。当运行级别改变时，init 就运行/etc/inittab 中相应的命令。

7.3.1 /etc/inittab 中的特殊配置

/etc/inittab 有些特殊的特性，它允许 **init** 对特别的环境作出响应。这些与众不同的特性在第三个域中由关键字标出。一些例子如下：

powerwait

当系统电源失败时，允许 **init** 关闭系统。这里假设使用了 UPS 以及用于监视 UPS 和通知 init 电源失败的软件。

ctrlaltdel

当用户在控制台上按了 ctrl-alt-del 组合键时，允许 **init** 重新（启动）引导系统。注意，系统管理员能够配置对 ctrl-alt-del 组合键的响应为其它的什么，例如，忽略它，如果系统是在一个公共的环境中（或者开始 **nethack**。）

sysinit

当系统引导时要执行的命令。例如，这个命令通常是清理/tmp。

上面所列并不是全部。对于所有的关键字以及如何使用它们请参见 inittab 的 manual page。

7.4 启动（引导）进入单用户模式

一个很重要的运行级别是单用户模式（*single user mode*）(运行级别 1)，在这个模式中只有系统管理员在使用机器并且只有很少的系统服务在运行，如登录服务。对于一些管理任务来说单用户模式是必须的， [4] 如在/usr 分区上运行 **fsck**，因为这需要该分区没被加载，除非几乎所有的系统服务都被终止了，否则不可能会有这种情况。

通过 **telinit** 请求运行级别 1，一个运行着的系统可以转换到单用户模式。在启动时，可以通过在内核的命令行上给出 single 或 emergency 来进入单用户模式：内核同样也将命令行给 **init**，**init** 会理解那个单词并且不会使用缺省的运行级别。（内核命令行输入的方法依赖于系统是如何引导的。）

在加载文件系统之前，引导进入单用户模式有时是需要的，这样就可以手工运行 **fsck** 命令了，否则的话很可能损坏/usr 分区（在一个有问题的文件系统上的任何操作会更进一步地损坏它，所以 **fsck** 要尽早地运行）。

如果启动时 fsck 的自动检查失败了，启动描述文件 **init** 就会自动地进入单用户模式。这是试图避免系统使用一个文件系统，这个文件系统损坏的太严重以至于 fsck 都不能够自动地修复它。这样的毁坏情况是相当少的，通常是硬盘有问题或是在试验一个内核版本，但是有准备总比没有好。

作为一个安全措施，一个正确配置的系统应该在运行单用户模式的 shell 之前要求口令。否则的话，只要给 LILO 输入适当的一行参数就很容易地以 root 身份进入系统。（当然，如果由于文件系统的问题而使/etc/passwd 毁坏时，就不是这样了。如果是这样的话，你手头最好有张引导软盘。）

注释

[1] **init** 本身不允许终止。即使使用 SIGKILL 也不能终止 **init**。

[2] 例如，作为 root，使用命令 **kill –HUP 1**。

[3] 不同版本的 getty 运行起来不同。参见 manual page，并确定这是正确的 manual page。

[4] 它通常不应用于使用 **nethack**。

第8章 登录与退出

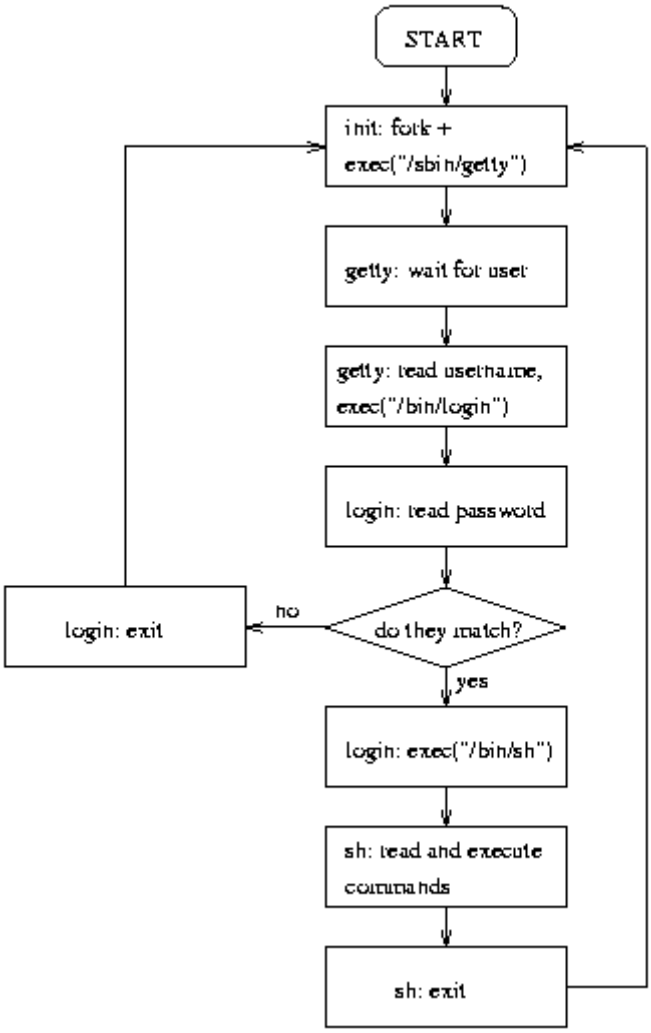
“我并不关心能接收象我这样的人作为会员的俱乐部。”（Groucho Marx）

本章描述了当一个用户登录或退出时会发生些什么。对于后台进程的各种交互作用、log 文件、配置文件等等也进行了一定的叙述。

8.1 从终端登录

图 8-1 显示出通过终端登录是如何进行的。首先，**init** 确信对于终端连接（或控制台）有一个对应的 **getty** 程序。**getty** 在终端上侦听并且等待用户通告他已准备好登录（这通常意味着用户必须键入些什么）。当它注意到一个用户，**getty** 输出一段欢迎消息（存储于 /etc/issue），提示输入用户名，并且最终运行 **login** 程序。**login** 将用户名作为参数，提示用户输入口令。如果这些都匹配的话，**login** 开始运行为该用户配置的 **shell**；否则的话，它就退出并终止该进程（也许是在给用户另一个输入用户名和口令的机会以后）。**init** 注意到这个进程终止以后，就开始为该终端运行一个新的 **getty** 程序。

图 8-1. 从终端登录：init、getty、login 以及 shell 的交互作用



注意唯一的新进程是由 **init** 创建的（使用 fork 系统调用）；**getty** 和 **login** 仅仅替换运行在进程中的程序（使用 exec 系统调用）。

对于串行线路需要一个独立的程序来检测一个用户，因为要知道一个终端已成为活动终端是比较复杂的（传统上是这样的）。**getty** 也同样会适应于连接的速度和其它的设置，这对于拨入连接尤其重要，因为对于各个拨入呼叫的参数常常是不一样的。

现在有几种 **getty** 和 **init** 的版本供使用，它们各有优缺点。好好弄懂你的系统上的版本以及其它的版本（你可以使用 Linux 软件 Map 来搜寻它们）是有益处的。如果你没有拨入连接，就无需关心 **getty**，但是 **init** 仍然是很重要的。

8.2 通过网络登录

在同一个网络中的两台计算机通常由单根电缆相连接。当它们在网络上互相通信时，各台计算机中参与通信的程序通过一个虚拟连接（*virtual connection*）连在了一起，一种想象中的电缆。就虚拟连接两端的程序而言，它们各自主宰着自己的电缆。然而，由于这是一种想象中的电缆而非真实的电缆，两台计算机的操作系统可以有几条虚拟连接，这些虚拟连接共享同一物理电缆。这样，仅使用单根电缆，几个程序就能同时通信而无需知道或关心其它的通信。这甚至使得几台计算机使用同一根电缆的情况成为可能；虚拟连接存在

于两台计算机之间，其它的计算机会忽略这些没有参与的连接。

那是很复杂的，是对现实的一种极其抽象的描述。然而，也许这已够理解为什么从网络登录与常规的登录有所不同的重要原因了。不同计算机上的两个程序想要进行通信时就会建立虚拟连接。由于从原理上讲，从网络中的任何一台计算机都可以登录进网络中的任何其它计算机中，就会存在大量的潜在的虚拟通信。由于这个原因，为每个潜在的登录运行一个 **getty** 是不现实的。

有一个单个进程 inetd（对应于 **getty**）用来处理所有的网络登录。当它注意到进来一个网络登录（也即，它注意到它有了一条到其它计算机的虚拟连接），它就启动一个新进程来处理那个登录。原始的进程仍然继续侦听新的登录请求。

让问题变得更为复杂的是，对于网络登录有不止一种的通信协议存在。最为重要的两种是 **telnet** 和 **rlogin**。除了登录以外，还可能建立许多其它的虚拟连接（用于 FTP、Gopher、HTTP 等其它网络服务）。针对各种类型的连接使用不同的进程将是低效的，所以取而代之的是只存在一个侦听进程，它可以识别出连接的类型并且能启动相应的程序来提供服务。这个单个侦听者称为\cmd{inetd}；详细信息请参见 *Linux 网络管理员手册*。

8.3 Login做些什么

login 程序用于验证用户（确信用户名和口令是匹配的）、通过设置串行线路的许可来为用户设置一个初始环境、开始运行 **shell**。

初始设置的一部分是用于输出/etc/motd 文件的内容（当天的短讯）以及检查电子邮件。这可以通过在用户主目录中建立一个称为.hushlogin 的文件来取消这些操作。

如果文件/etc/nologin 存在，登录的功能就被取消。通常，这个文件是由 **shutdown** 以及其它相应操作产生的。**login** 会检查这个文件，如果该文件存在的话就会拒绝登录，**login** 会在其退出前将该文件的内容显示在用户终端上。

login 在一个系统日志文件中记录所有失败的登录企图（通过 **syslog**）。它也记录所有的 root 登录。这些对于跟踪入侵者来说是很有用的。

当前已登录的人被列于文件/var/run/utmp 中。这个文件在系统重新启动或关闭之前一直是有效的；当系统引导完成后，这个文件是空的。该文件列出了每个用户和他使用的终端（或网络连接）以及其它一些有用信息。**who**、**w** 以及其它一些类似命令查询 utmp 来看谁已登录系统。

所有成功的登录都被记录进/var/log/wtmp。这个文件会越来越大，没有限制，所以有必要定期对它清理，例如通过一个以一星期为周期的 **cron** 作业。[1] **last** 命令用于浏览 wtmp 文件。

utmp 和 wtmp 两者均是二进制格式（见 utmp 的 manual page）；很不幸，不用专用程序不能查看到它们的内容。

8.4 X以及xdm

XXX X 通过 xdm 实现登录；同样也可用：xterm –ls

8.5 访问控制

传统上，用户数据库包含在/etc/passwd 文件中。有些系统使用影子口令（*shadow passwords*），并且将口令移入了**/etc/shadow**。有许多有共享帐号的计算机的站点使用 NIS

或一些其它方法来存储用户数据；它们通常可以自动地从某个中央位置将数据库拷贝到所有其它计算机中。

用户数据库不但包含有口令，而且还包括一些其它的有关用户的信息，比如他们的真实名称、主目录和登录 shells。这些其它信息需要是公用的，这样任何人都可以阅读它。因此口令是加密存储的。这的确有不足之处，能够访问加密了的口令的任何人可以使用各种口令工具来猜测它，而无需实际地试着登录计算机中。影子口令试图避免这个缺点，做法是将口令移入另一个文件，这个文件只有 root 用户可以读取（口令仍然是加密存储的）。然而，以后在一个不支持影子口令的系统上安装影子口令有可能很困难。

不管口令怎样，确信一个系统上的所有口令是否都健全是很重要的，也即，不容易被猜测出来。**crack** 程序可用来破解口令；它所能猜测出的任何口令都不算是好口令。虽然系统入侵者能够使用 **crack** 程序，系统管理员同样也可以使用它来避免不好的口令。**passwd** 程序也能够增强口令的不易猜测性；实际上更有效的是在于 CPU 周期上，因为破解口令需要大量的计算。

用户组数据库在/etc/group 文件中；对于有影子口令的系统，可能存在一个/etc/shadow.group 文件。

root 用户通常不能够在大多数终端或网络上登录，只能在列于/etc/securetty 文件中的终端上登录。这使得必须直接访问这些物理终端之一。然而，可以以任何其它用户在任何终端上登录，并使用 **su** 命令来成为 root 用户。

8.6 shell的启动

当一个交互式的登录 shell 启动时，它自动执行一个或多个预定义的文件。不同的 shells 执行不同的文件；详细信息参见各种 shell 的文档。

大多数 shells 首先运行一些全局文件，例如，Bourne shell（**/bin/sh**）以及它的派生者执行/etc/profile 文件；它还执行用户主目录中的.profile 文件。/etc/profile 允许系统管理员设置一个公共的用户环境，特别是除了一般的目录，可以通过设置 PATH 来包括当地命令目录。另外，如果需要的话，.profile 文件允许用户覆盖 profile 定义的缺省环境，以定制自己的使用环境以适合自己的口味。

注释

[1] 好的 Linux 发行版在箱外做这事。

第9章 管理用户帐号

“系统管理员与毒品销售商的相似之处在于：两者都是用 K 来度量，两者都有用户。”（老的，陈旧的计算机笑话。）

本章解释如何创建新用户帐号，如何修改这些帐号的属性，以及如何删除帐号。不同的 Linux 系统有不同的工具来做这些工作。

9.1 什么是帐号？

当一台计算机被许多人使用时，常常需要区分这些用户，例如，他们的私有文件保持私有。甚至，在某一时刻，计算机只能被单个用户使用时这也是很重要的，就如同许多微型计算机一样。 [1] 因此，每个用户都有一个唯一的用户名，这个用户名是用于登录的。

然而，对于用户来说除了名字还有许多其它特性。一个帐号（*account*）是指属于一个用户的所有文件、资源、以及信息。这个术语暗示如同在银行一样，并且在一个商业系统中每个帐号通常有些金钱与之有关，并且依赖于用户如何使用系统，那些金钱以不同的速率花光。例如，磁盘空间可能每兆字节每天有个价，处理时间可以每秒钟有个价。

9.2 创建一个用户

Linux 内核本身将用户仅仅看作是一些数字。每个用户都用唯一的一个数字加以识别，即用户 id（*user id*）或 *uid*，因为对于计算机来说处理数字比处理文字来的更快更容易。在内核外的一个独立的数据库为每个文字名称，即用户名（*username*），指派了一个用户 id。该数据库同样也包含一些另外的信息。

要创建一个用户，你需要给用户数据库增加有关该用户的信息，并为该用户建立一个登录目录（用户主目录）。还常常需要训练该用户，并为它设立适当的初始环境。

许多 Linux 发行版带有一个创建帐号的程序。现有几种这样的程序。两个可选的命令行命令是 **adduser** 和 **useradd**；同样也许有个 GUI 上的工具。不管是什么样的程序，其结果是很少需要进行手工作业。尽管如此，细节却是很多并且复杂而难以理解，这些程序使得任何事情看上去都很简单而直接。不过，在*手工创建用户帐号*一节中描述了如何手工做到这点。

/etc/passwd 以及其他信息文件

UNIX 系统中基本的用户数据库是字符型文件，/etc/passwd（称为口令文件），里面列出了所有有效用户的用户名以及它们相关的信息。每个用户名对应文件中的一行，并且用冒号分成七个域：

。用户名
。口令，以加密的方式。
。数字形式的用户 id。
。数字形式的组 id。

。帐号的全名或其它描述。
。登录（主）目录。
。登录 shell（登录时运行的程序）。

在 passwd 的 manual page 中有对格式更详细的解释。

系统上的任何用户都可以读取口令文件，例如，他们可以知道其他用户的名称。这同样意味着每个人也都能看到口令（第二个域）。口令文件对口令进行了加密，所以从理论上讲是没有问题的。然而，这个加密术是可以破的，特别是当口令取得不好时（也即，它很短或者它可以从字典中查到）。因此，将口令放在口令文件中是不太好的。

许多 Linux 系统有影子口令（*shadow passwords*）。这是存储口令的另一种方法：加密的口令被存储在一个独立的文件中，/etc/shadow，只有 root 用户能读取这个文件。/etc/passwd 文件的第二个域只含有一个特殊的标记。任何一个需要验证用户的程序需要设置 uid，由此可以访问影子口令文件了。只使用口令文件其它域的常规的程序是得不到口令的。[2]

选取用户和组的 id 数值

在许多系统上，用户和组的 id 数值是怎样的并无关系，但是如果你使用网络文件系统（NFS），你必须在所有的系统中有相同的 uid 和 gid。这是因为 NFS 也用 uid 的数值来确认用户的。如果你没有使用 NFS，你可以让你的帐号创建工具自动地选择它们。

如果你使用 NFS，你就必须为同步帐号信息创建一种机制。另一种方法是使用 NIS 系统（见 XXX 网络管理员手册）。

然而，你应该避免重复使用曾经有的 uid 值（以及用户名），因为 uid（或用户名）的新的拥有者有可能能够访问到原来的相同用户名的文件（或邮件、或其它的信息）。

初始化环境：/etc/skel

当为一个新用户创建了登录（主）目录时，就为用/etc/skel 中的文件对这个目录进行初始化工作。系统管理员能够在/etc/skel 中建立文件，这些文件可以为用户提供一个很好的缺省环境。例如，他可以建立一个/etc/skel/.profile 文件，在其中可以为一些编辑器设置成对新用户友好的 EDITOR 环境。

然而，通常最好尽量保持/etc/skel 越小越好，否则的话就很难更新已存在的用户的这些文件了。例如，如果友好编辑器的名称改变了，那么所有现存用户就不得不编辑修改他们的.profile 文件。虽然系统管理员可以用一个描述文件来作自动的修改，当这样做几乎肯定会搞乱一些人的.profile 文件。

只要可能，尽量将全局配置放入全局文件中，比如/etc/profile。这样就可以在不破坏用户自己的设置下做更新工作。

手工创建用户帐号

按以下步骤手工地创建一个新帐号：

。使用 **vipw** 编辑/etc/passwd 并为新帐号加入一新行。小心语法。*不要直接用一个编辑器编辑该文件！* **vipw** 会锁住这个文件，这样别的命令就不能同时更新它。你应该让口令域成‘*’，这样这个帐号就还不可以登录。
。类似地，如果你也想创建一个新组时，使用 **vigr** 来编辑/etc/group 文件。
。使用 **mkdir** 为该用户建立一个登录（主）目录。
。将/etc/skel 中的文件拷贝到这个新目录中。
。用 **chown** 和 **chmod** 修改所有权和权限。此时，-R 选项是非常有用的。不同的站点

的权限稍有不同，但是通常以下命令所做的是正确的：

```
cd /home/newusername
chown -R username.group .
chmod -R go=u,go-w .
chmod go= .
```

。用 passwd 设置口令。

当你在最后一步设置好口令以后，该帐号就可以被使用了。在所有的工作都做完之前你不应该设置口令，否则的话，很有可能当你仍在拷贝文件时该用户冒昧地登录了进来。

有时需要创建不被用户使用的哑帐号（dummy accounts）[3]。例如，为了设置一个匿名 FTP 服务器（这样任何人都可以从上面下载文件，而无需首先有一个帐号），必须创建一个叫做 ftp 的帐号。在这种情况下，不需要设置口令（上面的最后一步）。确实，最好不要，这样没人可以使用这个帐号，除非他们首先成为 root，而 root 可以变成任何用户的。

9.3 更改用户属性

有一些命令用于更改一个帐号的各种属性（也即，/etc/passwd 中的相关域）：

chfn
更改全名域。

chsh
更改登录的 shell。

passwd
改变口令。

超级用户可以使用这些命令来改变任何帐号的属性。普通用户只能更改自己帐号的属性。有时也需要不让普通用户使用这些命令（用 **chmod** 来改），例如在一个有许多新手的环境中。

其它的一些任务需要手工来做。例如，要改变用户名，你需要直接编辑/etc/passwd 文件（记住，用 **vipw**）。同样地，要将用户加入进一些组中或从一些组中删除，你需要编辑/etc/group 文件（用 **vigr**）。这些任务其实是不多的，然而，你要很小心地去做：例如，如果你改变了用户名，用户就再也收不到原来的 e-mail 了，除非你也建立了邮件的别名。[4]

9.4 删除一个用户

要删除一个用户，你首先要删除他的所有文件、邮箱、邮件别名、打印作业、cron 和 at 作业、以及所有该用户的其它一切。然后，从/etc/passwd 以及 /etc/group 中移去相关行（记住，从所有组中移去该用户名）。在开始进行删除工作时，最好首先禁用该帐号（见下面），以免在该帐号正被删除之际，用户正使用该帐号。

记住，用户可能在他的登录目录（他的主目录）之外还有文件。**find** 命令可用来找出它们：

find / -user username

然而，如果你的硬盘很大的话，上面这条命令将需要很长的时间。如果你加载上了网络磁盘，你就必须小心，不要搜索网络或其上的服务器。

有些 Linux 发行版带有做这个工作的专用命令；参见 **deluser** 或 **userdel**。不过，手工同样也能轻易地做到，而且使用命令可能做不彻底。

9.5 临时禁用一个用户

有时需要临时禁用一个帐号，而不是删除它。例如，用户可能没有付费，或者系统管理员可能怀疑有个解密高手已经取得了该帐号的口令。

最好的禁用一个帐号的方法是将它的 shell 换成一个特殊的程序，这个程序只是打印一条消息。这样，不管谁想登录进这个帐号，都将失败，并会知道为什么。这条消息可以告诉用户去与系统管理员联系以解决任何问题。

同样也可以将用户名或口令改成别的，但这样做的话这个用户会被搞懵了，不知道怎么回事。让用户感到困惑意味着会有更多的麻烦。[5]

建立特殊程序的一个简单方法是写一个‘tail scripts（描述文件）’：

```
#!/usr/bin/tail +2
This account has been closed due to a security breach.
Please call 555-1234 and wait for the men in black to arrive.
```

头两个字符（‘#!’）告诉内核该行的余下部分是命令，是用于解释执行这个文件的。在这种情况下的 tail 命令将在标准输出设备上输出除第一行以外的所有信息。

如果用户 billg 被怀疑有破坏安全的行为，系统管理员可能会象以下这样做：

```
# chsh -s /usr/local/lib/no-login/security billg
# su - tester
This account has been closed due to a security breach.
Please call 555-1234 and wait for the men in black to arrive.
#
```

当然，这里的 su 的目的是用于测试改变是否发挥作用了。

Tail scripts 应该存放在一个独立的目录中，这样，他们的名字不会妨碍普通用户的命令。

注释

[1] 如果我的姐姐能够看到我的情书，那是多么地尴尬啊。

[2] 的确，这说明口令文件含有一个用户的除了口令之外的所有信息。这真是发展的奇妙之处。

[3] 超现实的用户？

[4] 例如，由于结婚了，用户的名字可能会改变并且他想让他的用户名反映他的新名字。

[5] 如果你是个 BOFH，你会觉得很有趣。

第10章 备份

硬件不是绝对可靠的。
软件是绝对不可靠的。
人不是绝对可靠的。
自然是绝对可靠的。

本章解释了为什么要备份、如何备份、什么时候进行备份以及如何从备份中恢复数据。

10.1 备份的重要性

你的数据是有价值的。重建这些数据是要花费你许多的时间和精力的，而且那是需要花费金钱的或者至少是个人的忧伤和泪水；有时它是不可重建的，举例来说，如果它是某些试验的结果的话。由于这是一种投资，你应该保护它并且采取一定的措施来避免数据的丢失。

数据的丢失，主要有四个原因：硬件出错、软件有问题、人操作因素、或者是自然灾难。[1] 尽管现代的硬件越来越可靠，但本质上看来还是会坏的。最重要的存储数据的硬件是硬盘，它依赖于在充满电磁干扰的世界里微小的磁场未被触及的状况。现代的软件甚至并没有趋于更可靠；极其稳固可靠的程序是一个特例，而不是常有的。人的行为是非常不可靠的，他们或者会造成一个错误，或者会有目的地恶意地破坏数据。自然并不是邪恶的，但有时即使是在正常的状态下也会造成巨大的破坏。总的来说，所有的部分都能正常地工作是个小小的奇迹。

备份是保护投资的一种方法。通过对数据拥有几个拷贝，那么有一个被损坏的话也无所谓（代价只是从备份中恢复丢失的数据）。

正确地作好备份是很重要的。正如与实际世界相关的所有事情一样，备份迟早也会失效。要做好备份的部分工作是确信它是有效的、可用的；你不会想知道你的备份是不可用的。[2] 让人感到雪上加霜的是，当你正在做备份时数据突然坏掉了；如果你只有一个备份的话，它也许同样会损坏，给你留下的只是辛勤工作的烟尘。[3] 或者在你恢复数据时，发现有些重要数据忘记备份了，就象有个 15000 个用户站点上的用户数据库。最好的情况是，你的所有备份都能正常的工作，但是你所用的最后一个磁带驱动器所读的那类的磁带肯定是含有水分的。

当说到备份工作时，常要用到偏执狂这种工作态度。

10.2 选择备份的介质

有关备份的最重要的决定是备份介质的选择。你必须考虑到成本、可靠性、速度、实用性和可用性。

成本是重要的，因为你应该有所需数据的几个时间的多个备份存储着。所以便宜的介质常常是必须的。

可靠性是极其重要的，因为一个坏的备份能够使一个成年人叫爹喊娘。备份介质必须

能够在多年里保持数据。你所使用介质的方法影响其作为备份介质的可靠性。硬盘通常是非常可靠的，但如果硬盘是在要备份的同一台计算机上，那么作为备份介质它并不能说是最可靠的。

如果备份不用交互地进行，那么速度就不是非常重要的。只要无需监督，就不用关心备份是否要进行两个小时。从另一方面来说，如果备份一定要在计算机空闲时才能进行的话，那么速度就成为一个要考虑的问题了。

实用性显然是需要的，因为你不可能使用一种不存在的介质。不太明显的是即使在将来对备份介质的需求是否能满足，并且在其它的计算机上是否能用。否则的话，灾难之后你也许不可能恢复你的备份了。

可用性对于是否经常做备份是个大要素。越容易做备份越好。一种备份介质一定不要太难于使用。

典型的可供选择的办法是使用软盘和磁带。软盘非常便宜，还算可靠，不太快，到处都有，但不适用于备份大量的数据。磁带有便宜的也有稍贵一些的，相当可靠，相当快，基本上到处都有，并且，就磁带容量大小而言，十分适用的。

还有些其它的方法可选择。它们常常没有很好的实用性，但是如果这不是个问题的话，它要比其它的方法好。例如，磁光盘既有软盘（它们是随机访问的，使得恢复单个文件速度很快）的好处，也有磁带（能含有大量的数据）的优点。

10.3 选择备份工具

有许多工具可用于制作备份。传统的 UNIX 备份工具是 **tar**、**cpio**、以及 **dump**。另外，还有许多第三方的软件包（有免费的也有商业版的）可供使用。对备份介质的选择会影响对备份工具的选择。

tar 和 **cpio** 是相似的，而且从备份的观点来看，这两者几乎是相同的。这两者都能将文件存储于磁带上，以及从磁带上恢复文件。都几乎可以使用任何介质，因为内核的设备驱动程序处理低级的设备操作，并且对于用户级程序来说这些设备看上去都是一样的。有些 UNIX 版的 **tar** 和 **cpio** 在处理不寻常的文件时可能会碰上问题（符号连接、设备文件、有很长的路径名的文件等等），但是，Linux 版的就可以正确地处理所有这些文件。

dump 有些不同，它是直接读取文件系统而非通过文件系统来读取其中的文件。它也是特定为备份的用途而编写的；**tar** 和 **cpio** 其实是用于文件归档的，尽管它们也同样用于备份操作。

直接读取文件系统有某些优点。这使得它可以在不影响文件时间标签的情况下备份文件；而对于 **tar** 和 **cpio** 来讲，你就必须首先以只读方式来加载文件系统。如果所有的东西都要备份，那么直接读取文件系统也就更有效，因为这样可以减少磁头的运动。主要的缺点是特定的文件系统类型要求特定的备份程序；Linux 的 **dump** 程序只能对 ext2 文件系统进行操作。

dump 也同样支持备份级别（levels）（这将在下面讨论）；而对于 tar 和 cpio 来讲，这需要用其它的工具来实现了。

有关第三方备份工具的比较不在本书范围内。Linux 软件图（Linux Software Map,LSM）列出了许多着方面的自由软件。

10.4 简单备份

一个简单备份方案是指一次备份所有的东西，然后在上次备份以后，再备份所有作过

修改的东西。以上所指的第一个备份称为完全备份（*full backup*），接下来的一系列备份称为增量备份（*incremental backups*）。完全备份经常比增量备份要花更多的劳动，因为有很多的数据要写入磁带并且一个完全备份有可能一个磁带（或软盘）还不够。从增量备份中进行恢复要比从完全备份进行恢复需要多次和更多的工作量。恢复能够进行优化，这样自从前次的完全备份后，你总可以备份所有的东西；这样，备份可能需要多花上一点工作量，但是就不会再需要恢复比完全备份以及增量备份更多的数据量。

如果你需要每天都制作备份并且你有六盒磁带的话，你可以使用第一盒磁带做第一个完全备份（比如说，在星期五），用第二盒至第五盒做增量备份（星期一到星期四）。接下来用第六盒磁带做一个新的完全备份（第二个星期五），再用第 2 至 5 盒重做增量备份。直到你又有了一个完全备份之前，你无需重写磁带 1 上的备份，免得当你在做完全备份时发生什么问题。在磁带 6 上做过完全备份之后，可将磁带 1 保存在其它什么地方，这样当你的其它备份磁带在火灾中毁坏时，你起码还有些备份留下。当你需要制作下一个完全备份时，把磁带 1 拿出来做，而把磁带 6 保存在其它地方。

如果你有多于六盒的磁带，可以使用多余的磁带作完全备份。每次拿最早用的磁带做完全备份。这样你就能有从几个星期前到现在的几个完全备份，如果你想找出一个老文件或一个文件的老版本，但是现在这个文件已被删掉了的话，那就能派上用处了。

10.4.1 使用 tar 进行备份

完全备份能够很容易地使用 tar 来做：

```
# tar --create --file /dev/ftape /usr/src
tar: Removing leading / from absolute path names in the archive
#
```

上面的例子使用了 GNU 的 **tar** 版本以及它的长选项名。传统的 **tar** 版本只能使用单字符的选项。GNU 的版本也能够处理备份超过一个磁带或一张软盘的情况，当然也能够处理很长的路径；并不是所有传统的版本都能做到这些的。（Linux 只使用 GNU 的 **tar**。）

如果你的备份在一盒磁带上放不下，你就需要使用—multi-volume （-M）选项：

```
# tar -cMf /dev/fd0H1440 /usr/src
tar: Removing leading / from absolute path names in the archive
Prepare volume \#2 for /dev/fd0H1440 and hit return:
#
```

注意，在备份之前你应该先格式化你的软盘，或者在 **tar** 要求一张新软盘时用另一个窗口或虚拟终端来做格式化。

在做好了一个备份之后，你应该使用选项—compare（-d）来检查它是否正确：

```
# tar --compare --verbose -f /dev/ftape
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
....
```

#

忽略了对所做备份进行检查就意味着直到在你已经丢失原始数据之前，你不会注意到你所做的备份是否可用。

增量备份可以使用 tar 来做，使用—newer （-N）选项：

```
# tar --create --newer '8 Sep 1995' --file /dev/ftape /usr/src --verbose
tar: Removing leading / from absolute path names in the archive
usr/src/
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/modules/
usr/src/linux-1.2.10-includes/include/asm-generic/
usr/src/linux-1.2.10-includes/include/asm-i386/
usr/src/linux-1.2.10-includes/include/asm-mips/
usr/src/linux-1.2.10-includes/include/asm-alpha/
usr/src/linux-1.2.10-includes/include/asm-m68k/
usr/src/linux-1.2.10-includes/include/asm-sparc/
usr/src/patch-1.2.11.gz
#
```

不幸的是，当一个文件的 i 节点信息已经改变时，tar 不可能知道，例如，它的权限位已经改变或者当它的文件名已经改变时。这可以用 **find** 以及比较当前文件系统与以前备份的文件系统的文件列表的方法来应付。有关做这些事的描述文件以及程序可以在 Linux 的 ftp 站点上找到。

10.4.2 使用 tar 进行备份恢复

tar 的 –extract （-x）选项用于提取文件：

```
# tar --extract --same-permissions --verbose --file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#
```

你也可以通过在命令行上给出名字来只抽取指定的文件或目录（包括它的所有文件以及子目录）：

 # tar xpvf /dev/fd0H1440 usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
#

如果你只想看看备份卷上有些什么文件时可以使用—list（-t）选项：

 # tar --list --file /dev/fd0H1440
usr/src/
usr/src/linux
usr/src/linux-1.2.10-includes/
usr/src/linux-1.2.10-includes/include/
usr/src/linux-1.2.10-includes/include/linux/
usr/src/linux-1.2.10-includes/include/linux/hdreg.h
usr/src/linux-1.2.10-includes/include/linux/kernel.h
...
#

请注意，tar 顺序地读取备份卷，所以对于大的卷，它是很慢的。然而，当使用磁带驱动器或其它一些顺序介质时，不可能使用随机访问数据库技术的。

tar 不能恰当地处理已删除的文件。如果你需要从完全备份以及增量备份上恢复一个文件系统时，并且你已经在两次备份之间删除了一个文件，这个文件在你做完恢复的操作后又将存在。如果该文件含有敏感数据并且本应不该再存在的话，就可能成为一个大问题。

10.5 多级备份

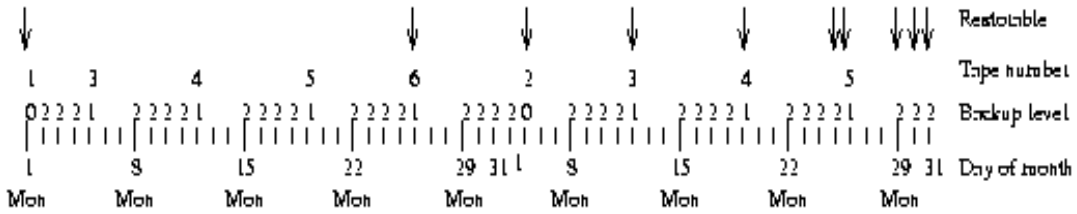
前节所述的简单备份方法常适用于个人使用或小站点使用，对于有重大责任的使用，采用多级备份方法将更为适用。

简单的方法有两个备份级别：完全备份和增量备份。这可以通用化到任何数量的级别。一个完全备份可看作是级别 0，增量备份的不同级就是备份级 1、2、3 等等。在每个增量备份级操作中，要备份从本级或前一级的前次备份以来所有已经改变过的东西。

这样做的目的是可以方便地有一个更长的备份记录史（*backup history*）。在上节的例子中，备份记录史可以追溯到上一个完全备份。若有更多的磁带，但每盒新磁带记录一星期，就能再延伸备份历史，不过这样可能太昂贵了点。较长的备份历史是有用的，因为被删除的或已毁坏的文件经常长期得不到注意。即使不是一个最新版的文件也总比没有的好。

图 10-1 显示出了每天使用那一个备份级别，以及哪些备份在月底可以用于恢复操作。

图 10-1. 多级备份方案的例子。



备份级也能用于使得文件系统的恢复工作时间达到最短。如果你有许多增量备份，并且这些增量备份的级别一个比一个高而不一样，那么当你需要重建整个文件系统时，你就需要恢复所有这些备份。然而，你可以使用级别不是顺序递增的备份级，这样就可以减少恢复操作时要用到的备份数量。

为了减少用于恢复操作时的已备份磁带数量,你可以对每个增量磁带使用更小的级别。然而，这样的话制作备份所用的时间就增加了（自从上次完全备份以来，每个备份要拷贝所有的东西）。一个更好的备份方案是在 dump 的 manual page 中提出的，并在表 XX 中进行了叙述（有效备份级别）。使用下面的连续备份级别：3，2，5，4，7，6，9，8，9 等等。这样可使得备份和恢复操作的时间都较少。你所需要的备份时间最多是抵两个工作日。一次恢复操作要用到的备份好的磁带的数量依赖于两次完全备份之间的时间长短，但数量要比在简单备份中用的要少。

表 10-1. 使用多备份级的有效备份方案。

磁带	级别	备份（天数）	恢复用到的备份磁带数
1	0	n/a	1
2	3	1	1, 2
3	2	2	1, 3
4	5	1	1, 2, 4
5	4	2	1, 2, 5
6	7	1	1, 2, 5, 6
7	6	2	1, 2, 5, 7
8	9	1	1, 2, 5, 7, 8
9	8	2	1, 2, 5, 7, 9
10	9	1	1, 2, 5, 7, 9, 10
11	9	1	1, 2, 5, 7, 9, 10, 11
. . .	9	1	1, 2, 5, 7, 9, 10, 11, . . .

一个好的备份方案能够减少所需的工作总量，但这也意味着要明了许多事情。你必须判定这样是否值得。

dump 有内建的对备份级的支持。而对于 **tar** 和 **cpio** 来说，则还需要用 shell 描述语言来实现。

10.6 需要备份些什么

你当然想尽可能多备份些东西。主要的例外是那些很容易重装的软件，[4] 但即使是这些软件，也可能含有一些配置文件，这些文件是备份操作的重要对象；以免从头开始重新配置所有这些软件。另一个例外是/proc 文件系统；因为它只含有内核总是自动产生的数据，所以没有必要备份它。尤其没有必要备份/proc/kcore 文件，因为它只是当前物理内存的一个映像；而且它也非常大。

可备份也可不备份的部分包括 news spool、log 文件、以及/var 中的许多数据。你必须决定什么数据对你来说是重要的。

显然要备份的数据包括用户文件（/home）以及系统配置文件（/etc，但很可能其它一些散布在整个文件系统中）。

10.7 压缩备份

备份会占用大量的介质空间，这些会花费很多钱。为了减少所需的空间，备份能够被压缩。这有几种方法来做到。有些程序支持内建的压缩功能；例如，GNU 的 **tar** 程序的—gzip（-z）选项在把数据写入备份介质前，将整个备份数据输送到 **gzip** 压缩程序进行压缩。

不幸的是，压缩的备份可能会带来麻烦。由于压缩操作的本性，如果一个比特错了，那么其余所有的压缩数据都将毫无用处。有些备份程序含有内建的纠错功能，但并没有处理大量错误数据的办法。这意味着如果备份使用 GNU **tar** 的方法压缩，整个输出是一个压缩好的单元（文件），那么只要有一丁点错，整个备份都玩完。备份必须可靠，因此压缩的办法并不是个好主意。

另一种方法是独立地压缩每个文件。这仍然意味着一个文件要是出错，那么该文件整个地就会坏掉了，但不伤及其它文件。这个文件可能是由于各种原因毁坏的，所以，这种办法要比一点也不压缩好一些。afio 程序（cpio 的一个变种）可用来做这个工作。

压缩操作需要花费一些时间，这可能使得备份程序不能尽可能快地将数据写到磁带驱动器中。[5] 这可以用输出缓冲来避免（如果备份程序编制的很好的话可以编成内部的缓冲，或者用其它的程序来做到），但即使这样，运行工作也并不一定足够好。应该只是在很慢的计算机上才会有这个问题。

注释

[1] 第五个原因是“其它一些原因”。
[2] 不要笑，这已经发生在一些人身上。
[3] 在那里，但又不是…
[4] 你必须决定怎样做对你来说容易些，有些人觉得从几打软盘上安装来的简单。
[5] 如果磁带驱动器没有得到足够快的数据，它就会暂停；这会使得备份更慢，而且对磁带和驱动器都不利。

第11章 保持时间

“时间是一种幻觉，午餐时间更是如此。” （Douglas Adams.）

本章解释了 Linux 系统是如何维护[或保持]时间的，以及需要做些什么来避免引起麻烦。通常，关于时间你用不着做什么事，但理解它是很有好处的。

11.1 时区（Time zones）

时间的度量主要是基于有规则的自然现象，比如行星的旋转导致了有白天和黑夜的交替周期。两个连续周期所用的总时间是个常数，但是白天以及黑夜的周期长度却是在变化。一个简单的常数例子是正午。

正午是一天中太阳离地面位置最高时的时刻。由于地球是圆的，[1]正午在不同的地方有不同的时刻。这致使有了*当地时间*（*local time*）的概念。人们用许多单位来度量时间，其中许多的度量单位都与自然现象相关，如正午。只要你待在同一个地方，就用不着关心有不同的当地时间。

在你需要与远地通信的时候，你就会注意到需要一个共同的时间。在现代时期，世界上的许多地方都在相互通信，于是，定义了一个度量时间的全球标准。这个时间称为*世界时（universal time）*（UT 或 UTC，以前以格林威治平均时间 Greenwich Mean Time 或 GMT，因为这在英格兰格林威治是当地时间）。当有不同当地时间的人进行通信时，他们就能用世界时来表达时间了。这样就不会混淆事情应在什么时候发生了。

每个当地时间都称为一个时区［time zone］。虽然地理上理应所有正午时刻相同的地方属于同一个时区，但是政治使它变的复杂。由于各种原因，许多国家使用夏令时（*daylight savings time*），他们调整时钟使工作时获得更多的日光，然后在冬天时把时钟调回。别的国家不这样做。那些这样做的国家，常常争执什么时候调整时钟，而且他们几乎年年改变规则。这使得时区的转换非常的不易。

时区最好是以地理位置命名或者是通过告知当地与世界时之差的方法。在美国和一些其他国家，当地时间都有一个名字以及三个字母的缩写。但是，缩写不是唯一的，并且除非与国家名连用，否则不宜使用。最好使用当地名称，如赫尔辛基，来谈论当地时间而不要用东欧时间（East European time）。因为不是所有的东欧国家都遵循同样的规律的。

Linux 有一个时区软件包，里面包括所有的现存时区，而且当规则改变时能够很容易更新。每个系统管理员所需做的就是选择合适的时区。同样，每个用户可以设置自己的时区；这显的很重要，因为许多使用计算机的人是通过 internet 网来自不同的国家。当你本地时区的夏令时规定改变时，一定至少要更新 Linux 系统中是那个部分。除了设置系统时区和更新时区数据文件以外，很少要为时间方面烦恼的。

11.2 硬件和软件时钟

个人计算机有一颗电池来驱动硬件时钟，这颗电池确保即使在计算机的其它部分没有电时时钟也能工作。硬件时钟可以从 BIOS 的设置屏幕上或从正在运行的操作系统中

进行设置。

Linux 内核从硬件时钟上直接取得时间。在引导期间，Linux 将自己的时钟设置成与硬件时钟同步。在这以后，两个时钟都独立地运行。Linux 维持着自己的时钟，因为读取硬件时钟是很慢的并且也比较复杂。

内核时钟总是显示世界时。这样，内核就根本不需要知道什么时区。这种简明性导致高可靠性并使得更新时区信息变得很容易。每一个进程独自处理时区的转换（使用属于时区软件包中的标准工具）。

硬件时钟可以是本地时间也可以是世界时。通常将硬件时钟设置成世界时更好一些，因为这样当夏令时开始或结束时你就不需要调整硬件时钟了（UTC 没有 DST）。不幸的是，有些 PC 操作系统，包括 MS-DOS、Windows、OS/2，假设硬件时钟显示的是本地时间。Linux 对于这两者都可以处理，但如果硬件时钟显示本地时间时，那么，在夏令时开始和结束时都需要调整它（否则，它将不会显示本地时间）。

11.3 显示和设置时间

在 Debian 系统中，系统时区由符号连接/etc/localtime 确定。这个连接指向描述本地时区的时区数据文件。时区数据文件存储于/usr/lib/zoneinfo 中。其它 Linux 发行版做法可能不同。

一个用户可以通过设置 TZ 环境变量来改变他的私有时区。如果 TZ 没有被设置，就使用系统时区。TZ 变量的语法在 tzset 的 manual page 中有描述。

date 命令用于显示当前日期和时间。[2] 例如：

```
$ date
Sun Jul 14 21:53:41 EET DST 1996
$
```

那个时间是星期日，7 月 14 日，1996 年，在大概晚上十点差十分钟，在称为“EET DST”的时区（可能是指东欧夏令时）。**date** 也可用来显示世界时：

```
$ date -u
Sun Jul 14 18:53:42 UTC 1996
Sun Jul 14 18:53:42 UTC 1996
$
```

date 也同样用于设置内核软件时钟：

```
# date 07142157
Sun Jul 14 21:57:00 EET DST 1996
# date
Sun Jul 14 21:57:02 EET DST 1996
#
```

详细资料参见 **date** 的 manual page；语法有些深奥而难懂。只有超级用户 root 能设置时间。虽然每个用户可以有他自己的时区，但时钟对每个人来说是一样的。

date 仅显示或设置软件时钟。而 **clock** 命令用于同步软件时钟和硬件时钟。它用于系统引导时，读取硬件时钟并设置软件时钟。如果你需要设置这两个时钟，首先要用 **date** 来设置软件时钟，然后用 **clock –w** 来设置硬件时钟。

clock 命令的-u 选项告诉系统硬件时钟是世界时。你*必须*正确地使用-u 选项，否则的话，你的计算机系统会对这个时间到底是什么感到困惑。

更改时钟要小心，Unix 系统的许多部分需要正确工作的时钟。例如，**cron** 后台程序周期性地运行命令。如果你更改了时钟值，那么它会对是否要运行那些命令感到困惑。在早期的 Unix 系统上，有人把时钟设快了二十年，这时 **cron** 就想要一次性地立刻运行所有二十年的周期命令。目前的 **cron** 版本能够正确地处理这个问题，但你仍需要小心。时间上的大的向前或向后跳跃要比小的或向前的跳跃危险的多。

11.4 当时钟不准时

Linux 软件时钟并不总是精确的。它是由 PC 硬件产生的周期性的*定时器中断（timer interrupt）*来工作的。如果系统运行了太多的进程，它就需要较长的时间来执行定时器中断程序，并且软件时钟就会漏掉一些中断。硬件时钟是独立运行的，通常比较精确。如果你经常重新引导你的计算机（对于大多数不是服务器的系统通常如此），它通常能保持较精确的时间。

如果你需要调整硬件时钟，最简单的办法是重新引导机器，进入 BIOS 的设置屏幕，并在那里来设置。这样做可以避免所有的改变系统时间可能带来的问题。如果不能从 BIOS 中来做的话，就用 **date** 以及 **clock** 来设置新时间（要按这个次序），并且如果系统的一些部分开始工作的很奇怪时，就要重新引导系统了。

一个连网的计算机（即使是通过 modem 连网的）能够通过比较自己的时间和网上其它计算机的时间来自动地检查自己的时钟。如果知道其它计算机有着非常精确的时间的话，那么大家都会有很精确的时间。这可以通过使用 **rdate** 和 **netdate** 命令来做到。这两个命令都会检测远程计算机上的时间（**netdate** 能够操作几个远程计算机），并且将本地计算机的时间设置成和远程的一样。通过周期性地运行这些命令，你的计算机就能够保持和远程计算机同样精确的时间了。

XXX 中有有关 NTP 智能的一些叙述。

注释

[1] 根据最近的调查。
[2] 要对 time 命令小心，它不显示当前时间。

第12章 词汇表（草案）

“The Librarian of the Unseen University had unilaterally decided to aid comprehension by producing an Orang-utan/Human Dictionary. He'd been working on it for three months. It wasn't easy. He'd got as far as `Oook.'” (Terry Pratchett, ``Men At Arms")

这是一个有关 Linux 系统管理概念词语定义的一张简明列表。

Ambition（野心、雄心）

有趣语句的写作，用于希望放入 Linux cookie 文件中。

application program（应用程序）

做有用事情的软件。使用应用程序是购买计算机的原因。同样参见系统程序、操作系统。

daemon（后台程序）

一个潜伏在后台的进程，通常不受主意，直到某事触发它动作。例如，\cmd{update}后台程序每三十秒左右执行一次来刷新高速缓冲，\cmd{sendmail}在有人发送邮件时就会触发执行。

file system（文件系统）

操作系统用于明了磁盘或分区上文件的方法和数据结构；磁盘上文件组织的方法。也用于存储文件或文件系统类型有关的分区或磁盘。

glossary（词汇表）

一个词组的列表以及对词组如何用法的解释。不要与词典混淆，词典同样是词组的列表以及对词组的解释。

kernel（内核）

操作系统的组成部分，用于实现与硬件的交互操作以及资源的共享。也参见系统程序。

operation system （操作系统）

在用户和他们运行的应用程序之间共享计算机系统资源的软件（处理器、内存、磁盘空间、网络带宽等等）。控制对系统的访问以提供安全机制。同样参见内核、系统程序、应用程序。

system call（系统调用）

内核提供给应用程序的服务，以及它们调用的方法。参见 manual pages 的第二部分。

system program（系统程序）

实现操作系统高层次功能的程序，也即，与硬件不是直接相关的程序。有时候需要特别优先权来运行（例如，分发电子邮件），但通常只是系统的一部分（例如，一个编译器）。

同样参见应用程序、内核、操作系统。