

Part 1

第一部分

Linux 命令行

本 部 分 内 容

- 第 1 章 初识 Linux shell
- 第 2 章 走进 shell
- 第 3 章 基本的 bash shell 命令
- 第 4 章 更多的 bash shell 命令
- 第 5 章 理解 shell
- 第 6 章 使用 Linux 环境变量
- 第 7 章 理解 Linux 文件权限
- 第 8 章 管理文件系统
- 第 9 章 安装软件程序
- 第 10 章 使用编辑器



本章内容

- ❑ 什么是Linux
- ❑ Linux内核的组成
- ❑ 探索Linux桌面
- ❑ 了解Linux发行版

在深入研究如何使用Linux命令行和shell之前，最好先了解一下什么是Linux、它的历史及运作方式。本章将带你逐步了解什么是Linux，并介绍命令行和shell在Linux整体架构中的位置。

1.1 什么是 Linux

如果你以前从未接触过Linux，可能就不清楚为什么会有这么多不同的Linux发行版。在查看Linux软件包时，你肯定被发行版、LiveCD和GNU之类的术语搞晕过。初次进入Linux世界会让人觉得不那么得心应手。在开始学习命令和脚本之前，本章将为你稍稍揭开Linux系统的神秘面纱。

首先，Linux可划分为以下四部分：

- ❑ Linux内核
- ❑ GNU工具
- ❑ 图形化桌面环境
- ❑ 应用软件

每一部分在Linux系统中各司其职。但就单个部分而言，其作用并不大。图1-1是一个基本结构框图，展示了各部分是如何协作起来构成整个Linux系统的。

本节将详细介绍这四部分，然后概述它们如何通过协作构成一个完整的Linux系统。

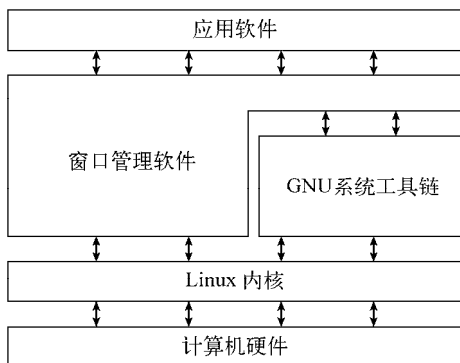


图1-1 Linux系统

1.1.1 深入探究 Linux 内核

Linux系统的核心是内核。内核控制着计算机系统上的所有硬件和软件，在必要时分配硬件，并根据需要执行软件。

如果你一直都在关注Linux世界，肯定听说过Linus Torvalds。Linus还在赫尔辛基大学上学时就开发了第一版Linux内核。起初他只是想仿造一款Unix系统而已，因为当时Unix操作系统在很多大学都很流行。

Linus完成了开发工作后，将Linux内核发布到了互联网社区，并征求改进意见。这个简单的举动引发了计算机操作系统领域内的一场革命。很快，Linus就收到了来自世界各地的学生和专业程序员的各种建议。

如果谁都可以修改内核程序代码，那么随之而来的将是彻底的混乱。为了简单起见，Linus担当起了所有改进建议的把关员。能否将建议代码并入内核完全取决于Linus。时至今日，这种概念依然在Linux内核代码开发过程中沿用，不同的是，现在是由一组开发人员来做这件事，而不再是Linus一个人。

内核主要负责以下四种功能：

- ❑ 系统内存管理
- ❑ 软件程序管理
- ❑ 硬件设备管理
- ❑ 文件系统管理

后面几节将会进一步探究以上每一种功能。

1. 系统内存管理

操作系统内核的主要功能之一就是内存管理。内核不仅管理服务器上的可用物理内存，还可以创建和管理虚拟内存（即实际并不存在的内存）。

内核通过硬盘上的存储空间来实现虚拟内存，这块区域称为交换空间（swap space）。内核不

断地在交换空间和实际的物理内存之间反复交换虚拟内存中的内容。这使得系统以为它拥有比物理内存更多的可用内存（如图1-2所示）。

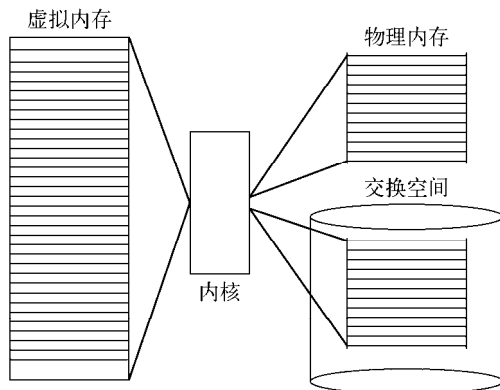


图1-2 Linux系统内存映射

内存存储单元按组划分成很多块，这些块称作页面（page）。内核将每个内存页面放在物理内存或交换空间。然后，内核会维护一个内存页面表，指明哪些页面位于物理内存内，哪些页面被换到了磁盘上。

内核会记录哪些内存页面正在使用中，并自动把一段时间未访问的内存页面复制到交换空间区域（称为换出，swapping out）——即使还有可用内存。当程序要访问一个已被换出的内存页面时，内核必须从物理内存换出另外一个内存页面给它让出空间，然后从交换空间换入请求的内存页面。显然，这个过程要花费时间，拖慢运行中的进程。只要Linux系统在运行，为运行中的程序换出内存页面的过程就不会停歇。

2. 软件程序管理

Linux操作系统将运行中的程序称为进程。进程可以在前台运行，将输出显示在屏幕上，也可以在后台运行，隐藏到幕后。内核控制着Linux系统如何管理运行在系统上的所有进程。

内核创建了第一个进程（称为init进程）来启动系统上所有其他进程。当内核启动时，它会将init进程加载到虚拟内存中。内核在启动任何其他进程时，都会在虚拟内存中给新进程分配一块专有区域来存储该进程用到的数据和代码。

一些Linux发行版使用一个表来管理在系统开机时要自动启动的进程。在Linux系统上，这个表通常位于专门文件/etc/inittab中。

另外一些系统（比如现在流行的Ubuntu Linux发行版）则采用/etc/init.d目录，将开机时启动或停止某个应用的脚本放在这个目录下。这些脚本通过/etc/rcX.d目录下的入口（entry）^①启动，这里的X代表运行级（run level）。

^① 这些入口实际上是到/etc/init.d目录中启动脚本的符号链接。——译者注（后文若无特殊说明，脚注均为“译者注”。）

Linux操作系统的init系统采用了运行级。运行级决定了init进程运行/etc/inittab文件或/etc/rcX.d目录中定义好的某些特定类型的进程。Linux操作系统有5个启动运行级。

运行级为1时，只启动基本的系统进程以及一个控制台终端进程。我们称之为单用户模式。单用户模式通常用来在系统有问题时进行紧急的文件系统维护。显然，在这种模式下，仅有一个入（通常是系统管理员）能登录到系统上操作数据。

标准的启动运行级是3。在这个运行级上，大多数应用软件，比如网络支持程序，都会启动。另一个Linux中常见的运行级是5。在这个运行级上系统会启动图形化的X Window系统，允许用户通过图形化桌面窗口登录系统。

Linux系统可以通过调整启动运行级来控制整个系统的功能。通过将运行级从3调整成5，系统就可以从基于控制台的系统变成更先进的图形化X Window系统。

在第4章，你将会学习如何使用ps命令查看当前运行在Linux系统上的进程。

3. 硬件设备管理

内核的另一职责是管理硬件设备。任何Linux系统需要与之通信的设备，都需要在内核代码中加入其驱动程序代码。驱动程序代码相当于应用程序和硬件设备的中间人，允许内核与设备之间交换数据。在Linux内核中有两种方法用于插入设备驱动代码：

- ❑ 编译进内核的设备驱动代码
- ❑ 可插入内核的设备驱动模块

以前，插入设备驱动代码的唯一途径是重新编译内核。每次给系统添加新设备，都要重新编译一遍内核代码。随着Linux内核支持的硬件设备越来越多，这个过程变得越来越低效。不过好在Linux开发人员设计出了一种更好的将驱动代码插入运行中的内核的方法。

开发人员提出了内核模块的概念。它允许将驱动代码插入到运行中的内核而无需重新编译内核。同时，当设备不再使用时也可将内核模块从内核中移走。这种方式极大地简化和扩展了硬件设备在Linux上的使用。

Linux系统将硬件设备当成特殊的文件，称为设备文件。设备文件有3种分类：

- ❑ 字符型设备文件
- ❑ 块设备文件
- ❑ 网络设备文件

字符型设备文件是指处理数据时每次只能处理一个字符的设备。大多数类型的调制解调器和终端都是作为字符型设备文件创建的。块设备文件是指处理数据时每次能处理大块数据的设备，比如硬盘。

网络设备文件是指采用数据包发送和接收数据的设备，包括各种网卡和一个特殊的回环设备。这个回环设备允许Linux系统使用常见的网络编程协议同自身通信。

Linux为系统上的每个设备都创建一种称为节点的特殊文件。与设备的所有通信都通过设备节点完成。每个节点都有唯一的数值对供Linux内核标识它。数值对包括一个主设备号和一个次设备号。类似的设备被划分到同样的主设备号下。次设备号用于标识主设备组下的某个特定设备。

4. 文件系统管理

不同于其他一些操作系统，Linux内核支持通过不同类型的文件系统从硬盘中读写数据。除了自有的诸多文件系统外，Linux还支持从其他操作系统（比如Microsoft Windows）采用的文件系统中读写数据。内核必须在编译时就加入对所有可能用到的文件系统的支持。表1-1列出了Linux系统用来读写数据的标准文件系统。

表1-1 Linux文件系统

文件系统	描 述
ext	Linux扩展文件系统，最早的Linux文件系统
ext2	第二扩展文件系统，在ext的基础上提供了更多的功能
ext3	第三扩展文件系统，支持日志功能
ext4	第四扩展文件系统，支持高级日志功能
hpfs	OS/2高性能文件系统
jfs	IBM日志文件系统
iso9660	ISO 9660文件系统（CD-ROM）
minix	MINIX文件系统
msdos	微软的FAT16
ncp	Netware文件系统
nfs	网络文件系统
ntfs	支持Microsoft NT文件系统
proc	访问系统信息
ReiserFS	高级Linux文件系统，能提供更好的性能和硬盘恢复功能
smb	支持网络访问的Samba SMB文件系统
sysv	较早期的Unix文件系统
ufs	BSD文件系统
umsdos	建立在msdos上的类Unix文件系统
vfat	Windows 95文件系统（FAT32）
XFS	高性能64位日志文件系统

Linux服务器所访问的所有硬盘都必须格式化成表1-1所列文件系统类型中的一种。

Linux内核采用虚拟文件系统（Virtual File System，VFS）作为和每个文件系统交互的接口。这为Linux内核同任何类型文件系统通信提供了一个标准接口。当每个文件系统都被挂载和使用时，VFS将信息都缓存在内存中。

1.1.2 GNU 工具

除了由内核控制硬件设备外，操作系统还需要工具来执行一些标准功能，比如控制文件和程序。Linus在创建Linux系统内核时，并没有可用的系统工具。然而他很幸运，就在开发Linux内核的同时，有一群人正在互联网上共同努力，模仿Unix操作系统开发一系列标准的计算机系

统工具。

GNU组织（GNU是GNU's Not Unix的缩写）开发了一套完整的Unix工具，但没有可以运行它们的内核系统。这些工具是在名为开源软件（open source software, OSS）的软件理念下开发的。

开源软件理念允许程序员开发软件，并将其免费发布。任何人都可以使用、修改该软件，或将该软件集成进自己的系统，无需支付任何授权费用。将Linux的内核和GNU操作系统工具整合起来，就产生了一款完整的、功能丰富的免费操作系统。

尽管通常将Linux内核和GNU工具的结合体称为Linux，但你也可能会在互联网上看到一些Linux纯粹主义者将其称为GNU/Linux系统，藉此向GNU组织所作的贡献致敬。

1. 核心GNU工具

GNU项目的主旨在于为Unix系统管理员设计出一套类似于Unix的环境。这个目标促使该项目移植了很多常见的Unix系统命令行工具。供Linux系统使用的这组核心工具被称为coreutils（core utilities）软件包。

GNU coreutils软件包由三部分构成：

- ❑ 用以处理文件的工具
- ❑ 用以操作文本的工具
- ❑ 用以管理进程的工具

这三组主要工具中的每一组都包含一些对Linux系统管理员和程序员至关重要的工具。本书将详细介绍GNU coreutils软件包中包含的所有工具。

2. shell

GNU/Linux shell是一种特殊的交互式工具。它为用户提供了启动程序、管理文件系统中的文件以及运行在Linux系统上的进程的途径。shell的核心是命令行提示符。命令行提示符是shell负责交互的部分。它允许你输入文本命令，然后解释命令，并在内核中执行。

shell包含了一组内部命令，用这些命令可以完成诸如复制文件、移动文件、重命名文件、显示和终止系统中正运行的程序等操作。shell也允许你在命令行提示符中输入程序的名称，它会将程序名传递给内核以启动它。

你也可以将多个shell命令放入文件中作为程序执行。这些文件被称作shell脚本。你在命令行上执行的任何命令都可放进一个shell脚本中作为一组命令执行。这为创建那种需要把几个命令放在一起工作的工具提供了便利。

在Linux系统上，通常有好几种Linux shell可用。不同的shell有不同的特性，有些更利于创建脚本，有些则更利于管理进程。所有Linux发行版默认的shell都是bash shell。bash shell由GNU项目开发，被当作标准Unix shell——Bourne shell（以创建者的名字命名）的替代品。bash shell的名称就是针对Bourne shell的拼写所玩的一个文字游戏，称为Bourne again shell。

除了bash shell，本书还将介绍其他几种常见的shell。表1-2列出了Linux中常见的几种不同shell。

表1-2 Linux shell

shell	描 述
ash	一种运行在内存受限环境中简单的轻量级shell，但与bash shell完全兼容
korn	一种与Bourne shell兼容的编程shell，但支持如关联数组和浮点运算等一些高级的编程特性
tcsh	一种将C语言中的一些元素引入到shell脚本中的shell
zsh	一种结合了bash、tcsh和korn的特性，同时提供高级编程特性、共享历史文件和主题化提示符的高级shell

大多数Linux发行版包含多个shell,但它们通常会采用其中一个作为默认shell。如果你的Linux发行版包含多个shell，就请尽情尝试不同的shell，看看哪个能满足你的需要。

1.1.3 Linux 桌面环境

在Linux的早期（20世纪90年代初期），能用的只有一个简单的Linux操作系统文本界面。这个文本界面允许系统管理员运行程序，控制程序的执行，以及在系统中移动文件。

随着Microsoft Windows的普及，电脑用户已经不再满足于对着老式的文本界面工作了。这推动了OSS社区的更多开发活动，Linux图形化桌面环境应运而生。

完成工作的方式不止一种，Linux一直以来都以此而闻名。在图形化桌面上更是如此。Linux有各种图形化桌面可供选择。后面几节将会介绍其中一些比较流行的桌面。

1. X Window系统

有两个基本要素决定了视频环境：显卡和显示器。要在电脑上显示绚丽的画面，Linux软件就得知道如何与这两者互通。X Window软件是图形显示的核心部分。

X Window软件是直接和PC上的显卡及显示器打交道的底层程序。它控制着Linux程序如何在电脑上显示出漂亮的窗口和图形。

Linux并非唯一使用X Window的操作系统，它针对不同操作系统的版本。在Linux世界里，能够实现X Window的软件包可不止一种。

其中最流行的软件包是X.org。它提供了X Window系统的开源实现，支持当前市面上的很多新显卡。

另外两个X Window软件包也日渐流行。Fedora Linux发行版采用了试验性的Wayland软件；Ubuntu Linux发行版开发出了Mir显示服务器，用于其桌面环境。

在首次安装Linux发行版时，它会检测显卡和显示器，然后创建一个含有必要信息的X Window配置文件。在安装过程中，你可能会注意到安装程序会检测一次显示器，以此来确定所支持的视频模式。有时这会造成显示器黑屏几秒。由于现在有多种不同类型的显卡和显示器，这个过程可能会需要一段时间来完成。

核心的X Window软件可以产生图形化显示环境，但仅此而已。虽然对于运行独立应用这已经足够，但在日常PC使用中却并不是那么有用。它没有桌面环境供用户操作文件或是开启程序。

为此，你需要一个建立在X Window系统软件之上的桌面环境。

2. KDE桌面

KDE（K Desktop Environment，K桌面环境）最初于1996年作为开源项目发布。它会生成一个类似于Microsoft Windows的图形化桌面环境。如果你是Windows用户，KDE就集成了所有你熟悉的功能。图1-3展示了运行在openSuSE Linux发行版上的KDE 4桌面。



图1-3 openSuSE Linux系统上的KDE 4桌面

KDE桌面允许你把应用程序图标和文件图标放置在桌面的特定位置上。单击应用程序图标，Linux系统就会运行该应用程序。单击文件图标，KDE桌面就会确定使用哪种应用程序来处理该文件。

桌面底部的横条称为面板，由以下四部分构成。

- ❑ **KDE菜单：**和Windows的开始菜单非常类似，KDE菜单包含了启动已安装程序的链接。
- ❑ **程序快捷方式：**在面板上有直接从面板启动程序的快速链接。
- ❑ **任务栏：**任务栏显示着当前桌面正运行的程序的图标。
- ❑ **小应用程序：**面板上还有一些特殊小应用程序的图标，这些图标常常会根据小应用程序的状态发生变化。

所有的面板功能都和你在Windows上看到的类似。除了桌面功能，KDE项目还开发了大量的可运行在KDE环境中的应用程序。

3. GNOME桌面

GNOME (the GNU Network Object Model Environment, GNU网络对象模型环境) 是另一个流行的Linux桌面环境。GNOME于1999年首次发布, 现已成为许多Linux发行版默认的桌面环境 (不过用得最多的是Red Hat Linux)。

尽管GNOME决定不再沿用Microsoft Windows的标准观感 (look-and-feel), 但它还是集成了许多Windows用户习惯的功能:

- ❑ 一块放置图标的桌面区域
- ❑ 两个面板区域
- ❑ 拖放功能

图1-4展示了CentOS Linux发行版采用的标准GNOME桌面。

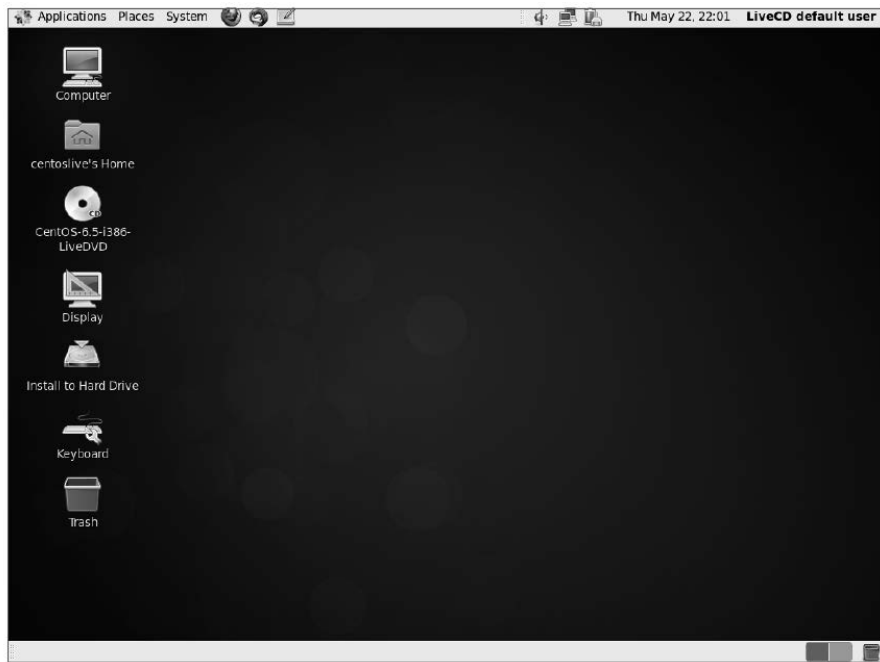


图1-4 CentOS Linux系统上的GNOME桌面

GNOME开发人员不甘示弱于KDE, 也开发了一批集成进GNOME桌面的图形化程序。

4. Unity桌面

如果你用的是Ubuntu Linux发行版, 你会注意到它与KDE和GNOME桌面环境有些不一样。准确来说, 这是因为负责开发Ubuntu的公司决定采用自己的一套叫作Unity的Linux桌面环境。

Unity桌面得名于该项目的目标——为工作站、平板电脑以及移动设备提供一致的桌面体验。不管你是在工作站还是在手机上使用Ubuntu, Unity桌面的使用方式都是一样的。图1-5展示了Ubuntu 14.04 LTS中的Unity桌面。

这些图形化桌面环境并不如KDE或GNOME桌面一样绚丽，但却提供了恰到好处的基本图形化功能。图1-6展示了Puppy Linux antiX发行版所采用的JWM桌面的外观。



图1-6 Puppy Linux发行版所采用的JWM桌面

如果你用的是老旧PC，尝试一下基于上述某个桌面环境的Linux发行版，看看怎么样，可能会有惊喜哦。

1.2 Linux 发行版

到此为止，你已经了解了构成完整Linux系统所需要的4个关键部件，那你可能在考虑要怎样才能把它们组成一个Linux系统。幸运的是，已经有人为你做好这些了。

我们将完整的Linux系统包称为发行版。有很多不同的Linux发行版来满足可能存在的各种运算需求。大多数发行版是为某个特定用户群定制的，比如商业用户、多媒体爱好者、软件开发人员或者普通家庭用户。每个定制的发行版都包含了支持特定功能所需的各种软件包，比如为多媒体爱好者准备的音频和视频编辑软件，为软件开发人员准备的编译器和集成开发环境（IDE）。

不同的Linux发行版通常归类为3种：

- ❑ 完整的核心Linux发行版
- ❑ 特定用途的发行版
- ❑ LiveCD测试发行版

后面几节将会探讨这些不同类型的Linux发行版，然后展示每种类型中一些Linux发行版示例。

1.2.1 核心 Linux 发行版

核心Linux发行版含有内核、一个或多个图形化桌面环境以及预编译好的几乎所有能见到的Linux应用。它提供了一站式的完整Linux安装。表1-4列出了一些较流行的核心Linux发行版。

表1-4 核心Linux发行版

发 行 版	描 述
Slackware	最早的Linux发行版中的一员，在Linux极客中比较流行
Red Hat	主要用于Internet服务器的商业发行版
Fedora	从Red Hat分离出的家用发行版
Gentoo	为高级Linux用户设计的发行版，仅包含Linux源代码
openSUSE	用于商用和家用的发行版
Debian	在Linux专家和商用Linux产品中流行的发行版

在Linux的早期，发行版是作为一叠软盘发布的。你必须下载多组文件，然后将其复制到软盘上。通常要用20张或更多的软盘来创建一个完整的发行版！毋庸多言，这是个痛苦的过程。

现今，家用电脑基本都有内置的CD和DVD光驱，Linux发行版也就用一组CD光盘或单张DVD光盘来发布。这大大简化了Linux的安装过程。

然而当新手在安装核心Linux发行版时，仍然经常遇到各种各样的问题。为了照顾到Linux用户的所有使用情景，单个发行版必须包含很多应用软件。从高端的Internet数据库服务器到常见的游戏，可谓应用尽有。鉴于Linux上可用应用程序的数量，一个完整的发行版通常至少要4张CD。

尽管发行版中的大量可选配置对Linux极客来说是好事，但对新手来说就是一场噩梦。多数发行版会在安装过程中询问一系列问题，以决定哪些应用要默认加载、PC上连接了哪些硬件以及怎样配置硬件设备。新手经常会被这些问题困扰，因此，他们经常是要么加载了过多的程序，要么没有加载够，到后来才发现计算机并没有按照他们预想的方式工作。

对新手来说，幸运的是，安装Linux还有更简便的方法。

1.2.2 特定用途的 Linux 发行版

Linux发行版的一个新子群已经出现了。它们通常基于某个主流发行版，但仅包含主流发行版中一小部分用于某种特定用途的应用程序。

除了提供特定软件外（比如仅为商业用户提供的办公应用），定制化发行版还尝试通过自动检测和自动配置常见硬件来帮助新手安装Linux。这使得Linux的安装过程轻松愉悦了许多。

表1-5列出了一些特定用途的Linux发行版以及它们的专长。

这只是特定用途的Linux发行版中的一小部分而已。像这样的发行版足有上百款，而且在互联网上还不断有新的成员加入。不管你的专长是什么，你都能找到一款为你量身定做的Linux发行版。

表1-5 特定用途的Linux发行版

发 行 版	描 述
CentOS	一款基于Red Hat企业版Linux源代码构建的免费发行版
Ubuntu	一款用于学校和家庭的免费发行版
PCLinuxOS	一款用于家庭和办公的免费发行版
Mint	一款用于家庭娱乐的免费发行版
dyne:bolic	一款用于音频和MIDI应用的免费发行版
Puppy Linux	一款适用于老旧PC的小型免费发行版

许多特定用途的Linux发行版都是基于Debian Linux。它们使用和Debian一样的安装文件，但仅打包了完整Debian系统中的一小部分。

1.2.3 Linux LiveCD

Linux世界中一个相对较新的现象是可引导的Linux CD发行版的出现。它无需安装就可以看到Linux系统是什么样的。多数现代PC都能从CD启动，而不是必须从标准硬盘启动。基于这点，一些Linux发行版创建了含有Linux样本系统（称为Linux LiveCD）的可引导CD。由于单张CD容量的限制，这个样本并非完整的Linux系统，不过令人惊喜的是，你可以自己加入各种软件。结果就是，你可以通过CD来启动PC，并且无需在硬盘安装任何东西就能运行Linux发行版。

这是一个不弄乱PC就体验各种Linux发行版的绝妙方法。只需插入CD就能引导了！所有的Linux软件都将直接从CD上运行。你可以从互联网上下载各种Linux LiveCD，刻录，然后体验。

表1-6列出了一些可用的流行Linux LiveCD。

表1-6 Linux LiveCD发行版

发 行 版	描 述
Knoppix	来自德国的一款Linux发行版，也是最早的LiveCD Linux
PCLinuxOS	一款成熟的LiveCD形式的Linux发行版
Ubuntu	为多种语言设计的世界级Linux项目
Slax	基于Slackware Linux的一款LiveCD Linux
Puppy Linux	为老旧PC设计的一款全功能Linux

你能在这张表中看到熟悉的面孔。许多特定用途的Linux发行版都有对应的Linux LiveCD版本。一些Linux LiveCD发行版，比如Ubuntu，允许直接从LiveCD安装整个发行版。这使你可以从CD引导启动，先体验一下此Linux发行版，如果喜欢的话，再把它安装到硬盘上。这个功能极其方便易用。

就像所有美好的事物一样，Linux LiveCD也有一些不足之处。由于要从CD上访问所有东西，应用程序会运行得更慢，而如果再搭配上陈旧缓慢的PC和光驱，那更是慢上加慢。还有，由于无法向CD写入数据，对Linux系统作的任何修改都会在重启后失效。

不过,有一些Linux LiveCD的改进帮助解决了上述一些问题。这些改进包括:

- ❑ 能将CD上的Linux系统文件复制到内存中;
- ❑ 能将系统文件复制到硬盘上;
- ❑ 能在U盘上存储系统设置;
- ❑ 能在U盘上存储用户设置。

一些Linux LiveCD,如Puppy Linux,只包含最少数量的Linux系统文件。当CD引导启动时,LiveCD的启动脚本直接把它们复制到内存中。这允许在Linux启动后立即把CD从光驱中取走。这不仅提高了程序运行速度(因为程序从内存中运行时更快),而且还空出了CD光驱,供你用Puppy Linux自带的软件转录音频CD或播放视频DVD。

其他Linux LiveCD用另外的方法,同样允许你在启动后将CD从光驱中拿走。这种方法是将核心Linux文件作为一个文件复制到Windows硬盘上。待CD启动后,系统会寻找那个文件,并从中读取系统文件。dyne:bolic Linux LiveCD采用的就是这种技术,我们称之为对接。当然,你必须在从CD引导启动之前把系统文件复制到硬盘里。

一种非常流行的技术就是用常见的U盘(也称为闪存或闪盘)来存储Linux LiveCD会话数据。几乎每个Linux LiveCD都能识别插入的U盘(即使是在Windows下格式化的)并从U盘上读写文件。这允许你启动Linux LiveCD,使用Linux应用来创建文件,再将这些文件存储在U盘上,然后用Windows应用(或者在另外一台电脑上)访问这些文件。这该有多酷!

1.3 小结

本章探讨了Linux系统及其基本工作原理。Linux内核是系统的核心,控制着内存、程序和硬件之间的交互。GNU工具也是Linux系统中的一个重要部分。本书关注的焦点Linux shell是GNU核心工具集中的一部分。本章还讨论了Linux系统中的最后一个组件:Linux桌面环境。随着时间推移,一切都发生了改变。现今的Linux可以支持多种图形化桌面环境。

本章还探讨了各种Linux发行版。Linux发行版就是把Linux系统的各个不同部分汇集起来组成一个易于安装的包。Linux发行版有囊括各种软件的成熟的Linux发行版,也有只包含针对某种特定功能软件包的特定用途发行版。Linux LiveCD则是一种无需将Linux安装到硬盘就能体验Linux的发行版。

下一章将开始了解启动命令行和shell脚本编程体验所需的基本知识。你将了解如何从绚丽的图形化桌面环境获得Linux shell工具。就目前而言,这绝非易事。

本章内容

- ❑ 访问命令行
- ❑ 通过Linux控制台终端访问CLI
- ❑ 通过图形化终端仿真器访问CLI
- ❑ 使用GNOME终端仿真器
- ❑ 使用Konsole终端仿真器
- ❑ 使用xterm终端仿真器

在Linux早期，可以用来工作的只有shell。那时，系统管理员、程序员和系统用户都端坐在Linux控制台终端前，输入shell命令，查看文本输出。如今，伴随着图形化桌面环境的应用，想在系统中找到shell提示符来输入命令都变得困难起来。本章讨论了如何进入命令行环境，带你逐步了解可能会在各种Linux发行版中碰到的终端仿真软件包。

2.1 进入命令行

在图形化桌面出现之前，与Unix系统进行交互的唯一方式就是借助由shell所提供的文本命令行界面（command line interface，CLI）。CLI只能接受文本输入，也只能显示出文本和基本的图形输出。

由于这些限制，输出设备并不需要多华丽。通常只需要一个简单的哑终端就可以使用Unix系统。所谓的哑终端无非就是利用通信电缆（一般是一条多线束的串行电缆）连接到Unix系统上的一台显示器和一个键盘。这种简单的组合可以轻松地输入文本数据，并查看文本输出结果。

如你所知，如今的Linux环境相较以前已经发生了巨大变化。所有的Linux发行版都配备了某种类型的图形化桌面环境。但是，如果想输入shell命令，仍旧需要使用文本显示来访问shell的CLI。于是现在的问题就归结为一点：有时还真是不容易在Linux发行版上找到进入CLI的方法。

2.1.1 控制台终端

进入CLI的一种方法是让Linux系统退出图形化桌面模式，进入文本模式。这样在显示器上就只有一个简单的shell CLI，跟图形化桌面出现以前一样。这种模式称作Linux控制台，因为它仿真了早期的硬接线控制台终端，而且是一种同Linux系统交互的直接接口。

Linux系统启动后，它会自动创建出一些虚拟控制台。虚拟控制台是运行在Linux系统内存中的终端会话。无需在计算机上连接多个哑终端，大多数Linux发行版会启动5~6个（有时会更多）虚拟控制台，你在一台计算机的显示器和键盘上就可以访问它们。

2.1.2 图形化终端

除了虚拟化终端控制台，还可以使用Linux图形化桌面环境中的终端仿真包。终端仿真包会在一个桌面图形化窗口中模拟控制台终端的使用。图2-1展示了一个运行在Linux图形化桌面环境中的终端仿真器。

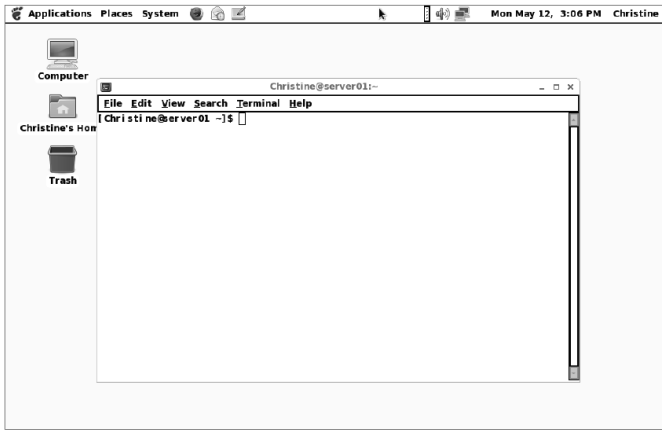


图2-1 运行在Linux桌面上的终端仿真器

图形化终端仿真只负责Linux图形化体验的一部分。完整的体验效果需要借助多个组件来实现，其中就包括图形化终端仿真软件（称为客户端）。表2-1展示了Linux图形化桌面环境的不同组成部分。

表2-1 图形界面的组成

名 称	例 子	描 述
客户端	图形化终端仿真器，桌面环境，网络浏览器	请求图形化服务的应用
显示服务器	Mir, Wayland Compositor, Xserver	负责管理显示（屏幕）和输入设备（键盘、鼠标、触摸屏）
窗口管理器	Compiz, Metacity, Kwin	为窗口加入边框，提供窗口移动和管理功能
部件库	Athenal（Xaw），X Intrinsic	为桌面环境中的客户端添加菜单以及外观项

要想在桌面中使用命令行，关键在于图形化终端仿真器。可以把图形化终端仿真器看作GUI中（in the GUI）的CLI终端，将虚拟控制台终端看作GUI以外（outside the GUI）的CLI终端。理解各种终端及其特性能够提高你的命令行体验。

2.2 通过 Linux 控制台终端访问 CLI

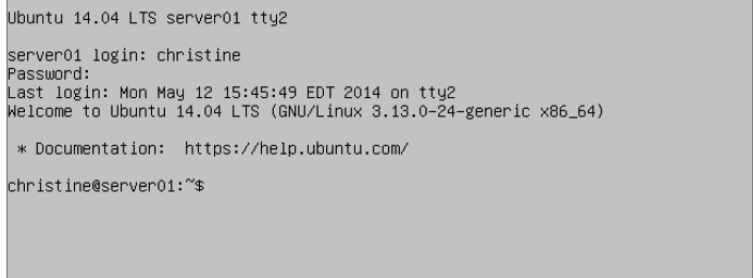
在Linux的早期，在启动系统时你只会在显示器上看到一个登录提示符，除此之外就没别的了。之前说过，这就是Linux控制台。它是唯一可以为系统输入命令的地方。

尽管在启动时会创建多个虚拟控制台，但很多Linux发行版在完成启动过程之后会切换到图形化环境。这为用户提供了图形化登录以及桌面体验。这样一来，就只能通过手动方式来访问虚拟控制台了。

在大多数Linux发行版中，你可以使用简单的按键组合来访问某个Linux虚拟控制台。通常必须按下Ctrl+Alt组合键，然后按功能键（F1~F7）进入要使用的虚拟控制台。功能键F1生成虚拟控制台1，F2键生成虚拟控制台2，F3键生成虚拟控制台3，F4键生成虚拟控制台4，依次类推。

说明 Linux发行版通常使用Ctrl+Alt组合键配合F1或F7来进入图形界面。Ubuntu使用F7，而RHEL则使用F1。最好还是测试一下自己所使用的发行版是如何进入图形界面的。

文本模式的虚拟控制台采用全屏的方式显示文本登录界面。图2-2展示了一个虚拟控制台的文本登录界面。



```
Ubuntu 14.04 LTS server01 tty2
server01 login: christine
Password:
Last login: Mon May 12 15:45:49 EDT 2014 on tty2
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

christine@server01:~$
```

图2-2 Linux虚拟控制台登录界面

注意，在图2-2中第一行文本的最后有一个词tty2。这个词中的2表明这是虚拟控制台2，可以通过Ctrl+Alt+F2组合键进入。tty代表电传打字机（teletypewriter）。这是一个古老的名词，指的是一台用于发送消息的机器。

说明 不是所有的Linux发行版都会在登录界面上显示虚拟控制台的tty号。

在login:提示符后输入用户ID, 然后再在Password:提示符后输入密码, 就可以进入控制台终端了。如果你之前从来没有用过这种方式登录, 那要注意在这里输入密码和在图形环境中输入不太一样。在图形环境中, 输入密码的时候会看到点号或星号, 但是在虚拟控制台中, 输入密码的时候什么都不会显示。

登入虚拟控制台之后, 你就进入了Linux CLI。记住, 在Linux虚拟控制台中是无法运行任何图形化程序的。

一旦登录完成, 你可以保持此次登录的活动状态, 然后在不中断活动会话的同时切换到另一个虚拟控制台。你可以在所有虚拟控制台之间切换, 拥有多个活动会话。在使用CLI时, 这个特性为你提供了巨大的灵活性。

还有一些灵活性涉及虚拟控制台的外观。尽管虚拟控制台只是文本模式的控制台终端, 但你可以修改文字和背景色。

比如可将终端的背景色设置成白色、文本设置成黑色, 这样可让眼睛轻松些。登录之后, 有好几种方法可实现这样的修改。其中一种方法是输入命令setterm -inversescreen on, 然后按回车键, 如图2-3所示。注意, 在途中我们使用选项on启用了inversescreen特性。也可以使用选项off关闭该特性。

```
CentOS release 6.5 (Final)
Kernel 2.6.32-431.17.1.el6.x86_64 on an x86_64
server01 login: Christine
Password:
Last login: Mon May 19 15:31:33 on tty2
[Christine@server01 ~]$
[Christine@server01 ~]$ setterm -inversescreen on
[Christine@server01 ~]$ _
```

图2-3 启用了inversescreen的Linux虚拟控制台

另一种方法是连着输入两条命令。输入setterm -background white, 然后按回车键, 接着输入setterm -foreground black, 再按回车键。要注意, 因为先修改的是终端的背景色, 所以可能会很难看清接下来输入的命令。

在上面的命令中, 你不用像inversescreen那样去启用或关闭什么特性。共有8种颜色可供选择, 分别是black、red、green、yellow、blue、magenta、cyan和white (这种颜色在有些发行版中看起来像灰色)。你可以赋予纯文本模式的控制台终端富有创意的外观效果。表2-2展示了setterm命令的一些选项, 可以用于增进控制台终端的可读性, 或改善外观。

表2-2 用于设置前景色和背景色的setterm选项

选 项	参 数	描 述
-background	black、red、green、yellow、blue、magenta、cyan或white	将终端的背景色改为指定颜色
-foreground	black、red、green、yellow、blue、magenta、cyan或white	将终端的前景色改为指定颜色
-inversescreen	on或off	交换背景色和前景色
-reset	无	将终端外观恢复成默认设置并清屏
-store	无	将终端当前的前景色和背景色设置成-reset选项的值

如果不涉及GUI，虚拟控制台终端访问CLI自然是不错的选择。但有时候需要一边访问CLI，一边运行图形化程序。使用终端仿真软件包可以解决这个问题，这也是在GUI中访问shell CLI的一种流行的方式。接下来的部分将介绍能够提供图形化终端仿真的常见软件包。

2.3 通过图形化终端仿真访问 CLI

相较于虚拟化控制台终端，图形化桌面环境提供了更多访问CLI的方式。在图形化环境下，有大量可用的图形化终端仿真器。每个软件包都有各自独特的特性及选项。表2-3列举出了一些流行的图形化终端仿真器软件包及其网址。

表2-3 流行的图形化终端仿真器软件包

名 称	网 址
Eterm	http://www.eterm.org
Final Term	http://finalterm.org
GNOME Terminal	https://help.gnome.org/users/gnome-terminal/stable
Guake	https://github.com/Guake/guake
Konsole Terminal	http://konsole.kde.org
LillyTerm	http://lilyterm.luna.com.tw/index.html
LXTerminal	http://wiki.lxde.org/en/LXTerminal
mrxvt	https://code.google.com/p/mrxvt
ROXTerm	http://roxterm.sourceforge.net
rxvt	http://sourceforge.net/projects/rxvt
rxvt-unicode	http://software.schmorp.de/pkg/rxvt-unicode
Sakura	https://launchpad.net/sakura
st	http://st.suckless.org
Terminator	https://launchpad.net/terminator
Terminology	http://www.enlightenment.org/p.php?p=about/terminology
tilda	http://tilda.sourceforge.net/tildaabout.php
UXterm	http://manpages.ubuntu.com/manpages/gutsy/man1/uxterm.1.html
Wterm	http://sourceforge.net/projects/wterm

(续)

名 称	网 址
xterm	http://invisible-island.net/xterm
Xfce4 Terminal	http://docs.xfce.org/apps/terminal/start
Yakuake	http://extragear.kde.org/apps/yakuake

2

尽管可用的图形化终端仿真器软件包不少，但本章只重点关注其中常用的三个。它们分别是 GNOME Terminal、Konsole Terminal和xterm，通常都会默认安装在Linux发行版中。

2.4 使用 GNOME Terminal 仿真器

GNOME Terminal是GNOME桌面环境的默认终端仿真器。很多发行版，如RHEL、Fedora和CentOS，默认采用的都是GNOME桌面环境，因此GNOME Terminal自然也就是默认配备了。不过其他一些桌面环境，比如Ubuntu Unity，也采用GNOME Terminal作为默认的终端仿真软件包。它使用起来非常简单，是Linux新手的不错选择。这部分将带你学习如何访问、配置和使用GNOME终端仿真器。

2.4.1 访问 GNOME Terminal

每个图形化桌面环境都有不同的方式访问GNOME终端仿真器。本节讲述了如何在GNOME、Unity和KDE桌面环境中访问GNOME Terminal。

说明 如果你使用的桌面环境并没有在表2-3中列出，那你就得逐个查看桌面环境中的各种菜单来找到GNOME终端仿真器。它在菜单中通常叫作Terminal。

在GNOME桌面环境中，访问GNOME Terminal非常直截了当。找到左上角的菜单，点击Applications，从下拉菜单中选择System Tools，点击Terminal。如果写成简写法的话，这一系列操作就像这样：Applications ⇨ System Tools ⇨ Terminal。

图2-1就是一张GNOME Terminal的图片。它展示了在CentOS发行版的GNOME桌面环境中访问GNOME Terminal。

在Unity桌面环境中，访问GNOME终端得费点事。最简单的方法是Dash ⇨ Search，然后输入Terminal。GNOME终端会作为一个名为Terminal的应用程序显示在Dash区域。点击对应的图标就可以打开GNOME终端仿真器了。

窍门 在一些Linux发行版的桌面环境中，例如Ubuntu的Unity，可以使用快捷键Ctrl+Alt+T快速访问GNOME终端。

在KDE桌面环境中，默认的仿真器是Konsole终端仿真器。必须通过菜单才能访问。找到屏幕左下角名为Kickoff Application Launcher的图标，然后依次点击Application ⇌ Utilities ⇌ Terminal。

在大多数桌面环境中，可以创建一个启动器（launcher）访问GNOME Terminal。启动器是桌面上的一个图标，可以利用它启动一个选定的应用程序。这是个很棒的特性，可以让你在桌面环境中快速访问终端仿真器。如果不想使用快捷键或是你的桌面环境中无法使用快捷键，这个特性就尤为有用。

例如，在GNOME桌面环境中，要创建一个启动器的话，可以在桌面中间单击右键，在出现的下拉菜单中选择Select Create Launcher...，然后会打开一个名为Create Launcher的窗口。在Type字段中选择Application。在Name字段中输入图标的名称。在Command字段中输入gnome-terminal。点击Ok，保存为新的启动器。一个带有指定名称图标的启动器就出现在了桌面上。双击就可以打开GNOME终端仿真器了。

说明 在Command字段中输入gnome-terminal时，输入的实际上是用来启动GNOME终端仿真器的shell命令。在第3章中会学到如何为gnome-terminal这类命令加入特定的命令行选项来获得特殊的配置，以及如何查看可用的选项。

在GNOME终端仿真器应用中，菜单提供了多种配置选项，应用本身也包含了很多可用的快捷键。了解这些选项能够增进GNOME Terminal CLI的使用体验。

2.4.2 菜单栏

GNOME Terminal的菜单栏包含了配置选项和定制选项，可以通过它们使你的GNOME Terminal符合自己的使用习惯。接下来的几张表格简要地描述了菜单栏中各种配置选项以及对应的快捷键。

说明 在阅读书中所描述的这些GNOME Terminal菜单选项时，要注意的是，这和你所使用的Linux发行版的GNOME Terminal的菜单选项可能会略有不同。因为一些Linux发行版采用的GNOME Terminal的版本比较旧。

表2-4展示了GNOME Terminal的File菜单下的配置选项。File菜单中包含了可用于创建和管理所有CLI终端会话的菜单项。

表2-4 File菜单

名 称	快 捷 键	描 述
Open Terminal	Shift+Ctrl+N	在新的GNOME Terminal窗口中启动一个新的shell会话
Open Tab	Shift+Ctrl+T	在现有的GNOME Terminal窗口的新标签中启动一个新的shell会话

(续)

名 称	快 捷 键	描 述
New Profile	无	定制会话并将其保存为配置文件 (profile)，以备随后再次使用
Save Contents	无	将回滚缓冲区 (scrollback buffer) 中的内容保存到文本文件中
Close Tab	Shift+Ctrl+W	关闭当前标签中的会话
Close Window	Shift+Ctrl+Q	关闭当前的GNOME Terminal会话

注意，和在网络浏览器中一样，你可以在GNOME Terminal会话中打开新的标签来启动一个全新的CLI会话。每个标签中的会话均被视为独立的CLI会话。

窍门 并不是非得点击菜单项才能进入File菜单中的选项。大多数选项可以通过在会话区域中点击右键找到。

表2-5所展示的Edit菜单中的菜单项用于处理标签内的文本内容。可以使用鼠标在会话窗口中的任意位置复制、粘贴文本。

表2-5 Edit菜单

名 称	快 捷 键	描 述
Copy	Shift+Ctrl+C	将所选的文本复制到GNOME的剪贴板中
Paste	Shift+Ctrl+V	将GNOME剪贴板中的文本粘贴到会话中
Paste Filenames		粘贴已复制的文件名和对应的路径
Select All	无	选中回滚缓冲区中的全部输出
Profiles	无	添加、删除或修改GNOME Terminal的配置文件
Keyboard Shortcuts	无	创建快捷键来快速访问GNOME Terminal的各种特性
Profile Preferences	无	编辑当前会话的配置文件

Paste Filenames菜单项只有在最新版的GNOME Terminal中才能找到，因此在你的系统中可能会看不到。

表2-6所展示的View菜单中包含用于控制CLI会话窗口外观的菜单项。这些选项能够为视力有缺陷的用户带来帮助。

表2-6 View菜单

名 称	快 捷 键	描 述
Show Menubar	无	打开/关闭菜单栏
Full Screen	F11	打开/关闭终端窗口全桌面显示模式
Zoom In	Ctrl++	逐步增大窗口显示字号
Zoom Out	Ctrl+-	逐步减小窗口显示字号
Normal Size	Ctrl+0	恢复默认字号

要注意的是，如果关闭了菜单栏显示，会话的菜单栏就会消失。不过你可以在任何一个终端

会话窗口中点击右键，然后选择Show Menubar，轻而易举地找回菜单栏。

表2-7所展示的Search菜单中的菜单项用于在终端会话中进行简单的搜索。这些搜索类似于在网络浏览器或字处理软件中进行的操作。

表2-7 Search菜单

名 称	快 捷 键	描 述
Find	Shift+Ctrl+F	打开Find窗口，提供待搜索文本的搜索选项
Find Next	Shift+Ctrl+H	从终端会话的当前位置开始向前搜索指定文本
Find Previous	Shift+Ctrl+G	从终端会话的当前位置开始向后搜索指定文本

表2-8所展示的Terminal菜单中的菜单项用于控制终端仿真会话的特性。这些菜单项并没有对应的快捷键。

表2-8 Terminal菜单

名 称	描 述
Change Profile	切换到新的配置文件
Set Title	修改标签会话的标题
Set Character Encoding	选择用于发送和显示字符的字符集
Reset	发送终端会话重置控制码
Reset and Clear	发送终端会话重置控制码并清除终端会话显示
Window Size List	列出可用于调整当前终端窗口大小的列表

Reset选项非常有用。某天，你可能不小心让终端会话显示了一堆杂乱无章的字符和符号。这时候根本识别不出什么文本信息。这通常是因为在屏幕上显示了非文本文件。可以通过选择Reset或Reset and Clear让屏幕恢复正常。

表2-9所展示的Tabs菜单中的菜单项用于控制标签的位置以及活动标签的选择。这个菜单只有在打开多个标签会话时才会出现。

表2-9 Tabs菜单

名 称	快 捷 键	描 述
Next Tab	Ctrl+PageDown	使下一个标签成为活动标签
Previous Tab	Ctrl+PageUp	使上一个标签成为活动标签
Move Tab Left	Shift+Ctrl+PageUp	将当前标签移动到前一个标签的前面
Move Tab Right	Shift+Ctrl+PageDown	将当前标签移动到下一个标签的后面
Detach Tab	无	删除该标签并使用该标签会话启动一个新的GNOME Terminal窗口
Tab List	无	列出当前正在运行的标签（选择一个标签，转入对应的会话）
Terminal List	无	列出当前正在运行的终端（选择一个终端，转入对应的会话。当打开多个窗口会话的时候才会出现该菜单项）

最后，Help菜单包含了两个菜单项。Contents提供了一份完整的GNOME Terminal手册，可供你研究GNOME Terminal的各个菜单项和特性。About菜单项可以告诉你当前运行的GNOME

Terminal的版本。

除了GNOME终端仿真软件包，另一个常用的软件包是Konsole Terminal。两者在很多方面类似。不过两者间存在的差异还是让我们很有必要单独开辟一节来讲解的。

2.5 使用 Konsole Terminal 仿真器

KDE桌面项目拥有自己的终端仿真软件包：Konsole Terminal。Konsole软件包具备基本的终端仿真特性，另外还包含了一些更高级的图形应用程序功能。本节描述了Konsole Terminal的特性及其用法。

2.5.1 访问 Konsole Terminal

Konsole Terminal是KDE桌面环境的默认终端仿真器，可以通过KDE环境的菜单系统轻而易举地访问到。在其他桌面环境中，访问Konsole Terminal就要麻烦一点了。

在KDE桌面环境中，可以通过点击屏幕左下角名为Kickoff Application Launcher的图标来访问Konsole Terminal。然后点击Applications ⇨ System ⇨ Terminal (Konsole)。

说明 你可能会在KDE菜单环境中看到两个终端菜单项。如果是这样的话，下方包含文字Konsole的Terminal菜单项就是Konsole终端。

在GNOME桌面环境中，通常并没有默认安装Konsole终端。如果已经安装过的话，你可以通过GNOME的菜单系统进行访问。在屏幕左上角点击Applications ⇨ System Tools ⇨ Konsole。

说明 你的系统中可能并没有安装Konsole终端仿真软件包。如果想安装的话，请阅读第9章来学习如何在命令行中安装软件。

如果在Unity桌面环境中安装了Konsole，可以通过Dash ⇨ Search，然后输入Konsole进行访问。Konsole Terminal会作为一个名为Konsole的应用程序显示在Dash区域。点击对应的图标打开Konsole终端仿真器。

图2-4展示了在CentOS Linux发行版的KDE桌面环境中访问Konsole Terminal。

记住，在大多数桌面环境中，可以创建一个启动器来访问如Konsole Terminal这样的应用程序。需要用于启动器启动Konsole终端仿真器的命令是`konsole`。另外，如果已经安装过Konsole Terminal的话，可以在其他的终端模拟器中输入`konsole`，然后按回车键来启动。

和GNOME Terminal类似，Konsole Terminal也通过菜单提供了一些配置选项和快捷键。接下来将会逐一讲述这些选项。

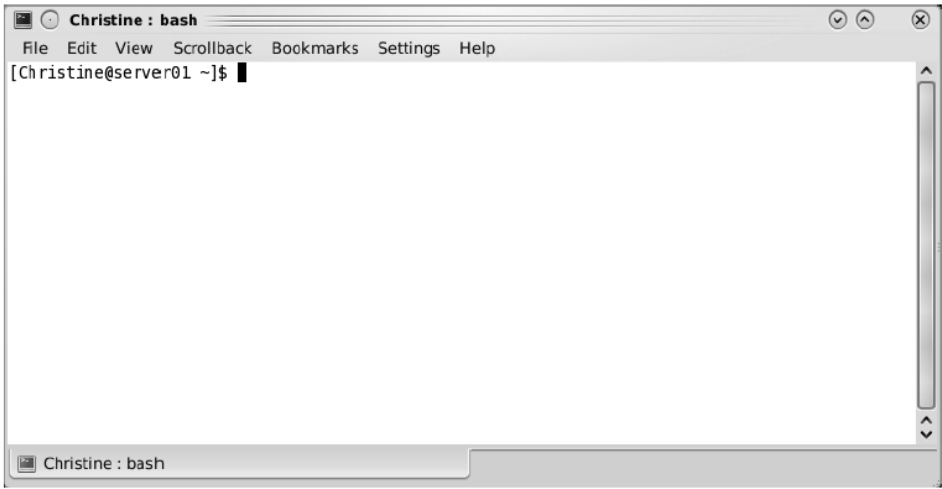


图2-4 Konsole Terminal

2.5.2 菜单栏

Konsole Terminal的菜单栏包含了查看和更改终端仿真会话特性所需的配置及定制化选项。下面的几张表格简要描述了菜单选项及其快捷键。

窍门 在活动会话区域中点击右键时，Konsole Terminal会弹出一个简单的菜单。一些菜单项可以在这个非常方便的菜单中找到。

表2-10中所展示的File菜单提供了可用于在当前窗口或新窗口中打开新标签的选项。

表2-10 File菜单

名 称	快 捷 键	描 述
New Tab	Ctrl+Shift+N	在现有的Konsole Terminal窗口的新标签中启动一个新的shell会话
New Window	Ctrl+Shift+M	在新的Konsole Terminal窗口中启动一个新的shell会话
shell	无	打开采用默认配置文件的shell
Open Browser Here	无	打开默认的文件浏览器应用
Close Tab	Ctrl+Shift+W	关闭当前标签中的会话
Quit	Ctrl+Shift+Q	退出Konsole Terminal仿真应用

在首次启动Konsole Terminal时，菜单中唯一列出的配置文件就是shell。随着越来越多的配置文件被创建及保存，它们的名字都会出现在菜单中。

说明 在阅读书中所描述的Konsole Terminal菜单项时，要注意的是，这可能会和你使用的Linux发行版中的Konsole Terminal有所不同。因为一些Linux发行版中采用的Konsole Terminal仿真软件包的版本比较旧。

2

表2-11中所展示的Edit菜单提供了可用于处理会话中的文本内容的选项。除此之外，可以管理标签名称的选项也在此列。

表2-11 Edit菜单

名 称	快 捷 键	描 述
Copy	Ctrl+Shift+C	将选择的文本复制到Konsole的剪贴板中
Paste	Ctrl+Shift+V	将Konsole剪贴板中的文本粘贴到会话中
Rename Tab	Ctrl+Alt+S	修改标签会话的标题
Copy Input To	无	开始/停止将会话输入复制到所选的其他会话中
Clear Display	无	清除终端会话中的内容
Clear & Reset	无	清除终端会话中的内容并发送终端会话重置控制码

Konsole有一种很好的方法来跟踪每个标签会话中正在进行的活动。你可以使用Rename Tab菜单项对标签进行命名，使其符合当前执行的任务。这可以帮助我们知道那些打开的标签究竟是干什么的。

表2-12所展示的View菜单中的菜单项用于控制Konsole Terminal窗口中单个会话的视图。除此之外，可监视终端会话活动的选项也在此列。

表2-12 View菜单

名 称	快 捷 键	描 述
Split View	无	控制显示在Konsole Terminal窗口中的多个标签会话
Detach View	Ctrl+Shift+H	删除一个标签会话并使用该标签中的会话启动一个新的Konsole Terminal窗口
Show Menu Bar	无	打开/关闭菜单栏
Full Screen Mode	Ctrl+Shift+F11	打开/关闭终端窗口的全屏模式
Monitor for Silence	Ctrl+Shift+I	打开/关闭无活动标签（tab silence）的特殊消息
Monitor for Activity	Ctrl+Shift+A	打开/关闭活动标签（tab activity）的特殊消息
Character Encoding	无	选择用于发送和显示字符的字符集
Increase Text Size	Ctrl++	逐步增大窗口显示字号
Decrease Text Size	Ctrl+-	逐步减小窗口显示字号

菜单项Monitor for Silence用于指明无活动标签。如果在当前标签会话内超过10秒钟没有出现新的文本内容，那该标签就成了无活动标签。这允许你在等待应用程序输出时切换到另一个标签。

由菜单项Monitor for Activity所打开的活动标签功能会在标签会话中出现新的文本内容时发出一条消息。这一选项能让你注意到应用程序产生了新的输出。

Konsole为每个标签保存了一个叫作回滚缓冲区的历史记录。这个历史记录中包含了已经不在当前终端可视区域中的文本内容。默认的是在回滚缓冲区内保存最近的1000行文本。表2-13所展示的Scrollback菜单中的菜单项可用于查看该缓冲区。

表2-13 Scrollback菜单

名 称	快 捷 键	描 述
Search Output	Ctrl+Shift+F	打开Konsole Terminal窗口底部的Find窗口，提供回滚文本搜索选项
Find Next	F3	在回滚缓冲区历史记录中查找下一个匹配的文本
Find Previous	Shift+F3	在回滚缓冲区历史记录中查找上一个匹配的文本
Save Output	无	将回滚缓冲区中的内容保存在一个文本文件或HTML文件中
Scrollback Options	无	打开Scrollback Options窗口来配置回滚缓冲区选项
Clear Scrollback	无	删除回滚缓冲区中的内容
Clear Scrollback & Reset	Ctrl+Shift+X	删除回滚缓冲区中的内容并重置终端窗口

你也可以使用窗口可视区域中的滚动条向后翻看回滚缓冲区中的内容。另外，也可以使用Shift+UpArrow逐行向后翻看，或是使用Shift+PageUp逐页（24行）向后翻看。

表2-14中所展示的Bookmarks菜单中的菜单项可用于管理Konsole Terminal窗口中的书签。书签能够保存活动会话的目录位置，让你随后可以在相同会话或新的会话中轻松返回之前的位置。

表2-14 Bookmark菜单

名 称	快 捷 键	描 述
Add Bookmark	Ctrl+Shift+B	在当前目录位置上创建新的书签
Bookmark Tabs as Folder	无	为当前所有的终端标签会话创建一个新的书签
New Bookmark Folder	无	创建新的书签文件夹
Edit Bookmarks	无	编辑已有的书签

表2-15所展示的Settings菜单中的菜单项可用于定制和管理配置文件。另外，你还可以为当前的标签会话再添加些许功能。这些菜单项并没有对应的快捷键。

表2-15 Settings菜单

名 称	描 述
Change Profile	将所选的配置文件应用于当前标签
Edit Current Profile	打开Edit Profile窗口，提供配置文件配置选项
Manage Profiles	打开Manage Profile窗口，提供配置文件管理选项
Configure Shortcuts	创建Konsole Terminal命令快捷键
Configure Notifications	创建定制化的Konsole Terminal方案及会话

Configure Notifications项允许将会话中发生的特定事件与不同的行为关联起来。当出现某个

事件时，就会触发指定的行为（或一系列行为）。

表2-16中所展示的Help菜单中的菜单项给出了完整的Konsole手册（如果你的Linux发行版中已经安装了KDE手册）以及标准的About Konsole对话框。

2

表2-16 Help菜单

名 称	快 捷 键	描 述
Konsole Handbook	无	包含了完整的Konsole手册
What's This?	Shift+F1	包含了终端部件的帮助信息
Report Bug	无	打开Submit Bug Report（提交bug报告）表单
Switch Application Language	无	打开Switch Application's Language（切换应用程序语言）表单
About Konsole	无	显示当前Konsole Terminal的版本
About KDE		显示当前KDE桌面环境的版本

有一份相当全面的文档可以帮助你使用Konsole终端仿真器软件包。除此之外，在你碰到程序故障的时候，还可以使用Bug Report表单向Konsole Terminal开发人员提交问题。

相较于另一个流行的软件包xterm，Konsole终端仿真器软件包算是年轻一代了。在下一节中，我们将探望一下“老古董”xterm。

2.6 使用 xterm 终端仿真器

最古老也是最基础的终端仿真软件包是xterm。xterm软件包在X Window出现之前就有了，通常默认包含在发行版中。

尽管xterm是功能完善的仿真软件包，但是它并不需要太多的资源（如内存）来运行。正因为如此，在专门为老旧硬件设计的Linux发行版中，xterm非常流行。有些图形化桌面环境就用它作为默认终端仿真器软件包。

xterm软件包尽管没有提供太多炫目的特性，但是却把一件事做到了极致：它能够仿真旧式终端，如DEC公司的VT102、VT220以及Tektronix 4014终端。对于VT102和VT220终端，xterm甚至能够仿真VT序列色彩控制码，让你可以在脚本中使用色彩。

说明 DEC VT102及VT220盛行于20世纪80年代和90年代初期，用于连接Unix系统的哑文本终端。VT102/VT220不仅能显示文本，还能够使用块模式图形显示基本的图形结构。由于在很多商业环境中这种终端访问方式仍在使用，因而使得VT102/VT220仿真依然流行。

图2-5展示了运行在图形化Linux桌面中的xterm。可以看出，它非常朴素。
如今得花点心思才能把xterm终端仿真器找出来。它常常并没有被包含在桌面环境的菜单中。



图2-5 xterm终端

2.6.1 访问 xterm

在Ubuntu的Unity桌面中，xterm是默认安装的。可以通过Dash ⇄ Search，然后输入xterm进行访问。xterm会作为一个名为XTerm的应用出现在Dash区域。点击对应的图标就可以打开xterm终端仿真器。

说明 在Ubuntu中搜索xterm时，你可能会看到另一个叫作UXTerm的终端。这只不过是加入了Unicode支持的xterm仿真器软件包而已。

GNOME和KDE桌面环境中并没有默认安装xterm。你得先安装它（可以参阅第9章安装软件包）。安装完成之后，你必须从另一个终端仿真器中启动xterm。打开一个终端仿真器进入CLI，输入xterm并按回车键。记住，也可以创建桌面启动器来启动xterm。

xterm包让你可以使用命令行参数设置自己的特性。下面的内容将讨论这些特性以及如何进行修改。

2.6.2 命令行参数

xterm的命令行参数非常多。你可以控制大量的特性来对终端仿真实施定制，例如允许或禁止某种VT仿真。

说明 xterm包含数量众多的配置选项，在此无法一一列举。在bash手册中有大量的文档可供参考。第3章中会讲到如何阅读bash手册。另外，xterm开发团队也在其网站上提供了很好的帮助：<http://invisible-island.net/xterm/>。

可以通过向xterm命令加入参数来调用某些配置选项。例如，要想让xterm仿真DEC VT100终端，可以输入命令xterm -ti vt100，然后按回车键。表2-17给出了一些可以配合xterm终端仿真器使用的参数。

表2-17 xterm命令行参数

参 数	描 述
-bg <i>color</i>	指定终端背景色
-fb <i>font</i>	指定粗体文本所使用的字体
-fg <i>color</i>	指定文本颜色
-fn <i>font</i>	指定文本字体
-fw <i>font</i>	指定宽文本字体
-lf <i>filename</i>	指定用于屏幕日志的文件名
-ms <i>color</i>	指定文本光标颜色
-name <i>name</i>	指定标题栏中的应用程序名称
-ti <i>terminal</i>	指定要仿真的终端类型

一些xterm命令行参数使用加号(+)或减号(-)来指明如何设置某种特性。加号表示启用某种特性，减号表示关闭某种特性。不过反过来也行。加号可以表示禁止某种特性，减号可以表示允许某种特性，例如在使用bc参数的时候。表2-18中列出了可以使用+/-命令行参数设置的一些常用特性。

表2-18 xterm +/-命令行参数

参 数	描 述
ah	启用/禁止文本光标高亮
aw	启用/禁止文本行自动环绕
bc	启用/禁止文本光标闪烁
cm	启用/禁止识别ANSI色彩更改控制码
fullscreen	启用/禁止全屏模式
j	启用/禁止跳跃式滚动
l	启用/禁止将屏幕数据记录进日志文件
mb	启用/禁止边缘响铃
rv	启用/禁止图像反转
t	启用/禁止Tektronix模式

要注意，不是所有的xterm实现都支持这些命令行参数。你可以在xterm启动后，使用-help参数来确定你所使用的xterm实现支持哪些参数。

现在你已经了解了三种终端仿真器软件包，重要的问题是：哪个是最好的终端仿真器。对于这个问题，并没有权威的答案。要使用哪个仿真器软件包取决于你的个人需求。不过，能有这么多选择总是好事。

2.7 小结

为了着手学习Linux命令行命令，得先能访问命令行。在图形化界面的世界里，有时会费点周折。本章讨论了能够获得Linux命令行的一些不同的界面。

首先，我们讲解了通过虚拟控制台终端（不涉及GUI的终端）和通过图形化终端仿真软件包（GUI中的终端）访问CLI时的不同。简要对比了两种访问方式之间的差别。

接下来，我们详细探究了通过虚拟控制台终端访问CLI，包括像更改背景色这类控制台终端配置选项。

在学习了虚拟控制台终端之后，本章还讲述了利用图形化终端仿真器访问CLI，其中主要涉及三种终端仿真器：GNOME Terminal、Konsole Terminal以及xterm。

本章还讨论了GNOME桌面项目的GNOME终端仿真软件包。GNOME Terminal通常默认安装在GNOME桌面环境中。藉由菜单以及快捷键，它可以很方便地设置多种终端特性。

然后讨论了KDE桌面项目的Konsole终端仿真软件包。Konsole Terminal通常默认安装在KDE桌面环境中。它提供了诸多漂亮的特性，例如能够监测到空闲的终端。

最后讲到的是xterm终端仿真器软件包。xterm是Linux中第一个可用的终端仿真器。它能够仿真旧式终端硬件，如VT和Tektronix终端。

下一章将开始接触Linux命令行。你将从中学习到Linux文件系统导航以及创建、删除、处理文件所需的命令。

本章内容

- ❑ 使用shell
- ❑ bash手册
- ❑ 浏览文件系统
- ❑ 文件和目录列表
- ❑ 管理文件和目录
- ❑ 查看文件内容

大多数Linux发行版的默认shell都是GNU bash shell^①。本章将介绍bash shell的一些基本特性，例如bash手册、tab键自动补全以及显示文件内容，带你逐步了解怎样用bash shell提供的基本命令来操作Linux文件和目录。如果你已经熟悉了Linux环境中的这些基本操作，可以直接跳过本章，从第4章开始了解更多的高级命令。

3.1 启动 shell

GNU bash shell能提供对Linux系统的交互式访问。它是作为普通程序运行的，通常是在用户登录终端时启动。登录时系统启动的shell依赖于用户账户的配置。

/etc/passwd文件包含了所有系统用户账户列表以及每个用户的基本配置信息。以下是从/etc/passwd文件中取出的样例条目：

```
christine:x:501:501:Christine Bresnahan:/home/christine:/bin/bash
```

每个条目有七个字段，字段之间用冒号分隔。系统使用字段中的数据来赋予用户账户某些特定特性。其中的大多数条目将在第7章有更加详细的介绍。现在先将注意力放在最后一个字段上，该字段指定了用户使用的shell程序。

^① 在6.10之后的大部分Ubuntu版本上，默认的shell是dash。

说明 尽管本书的重点放在了GNU bash shell，但是也会谈及其他一些shell。第23章中讲解了如何使用如dash和tcsh之类的shell。

在前面的/etc/passwd样例条目中，用户christine使用/bin/bash作为自己的默认shell程序。这意味着当christine登录Linux系统后，bash shell会自动启动。

尽管bash shell会在登录时自动启动，但是，是否会出现shell命令行界面（CLI）则依赖于所使用的登录方式。如果采用虚拟控制台终端登录，CLI提示符会自动出现，你可以输入shell命令。但如果是通过图形化桌面环境登录Linux系统，你就需要启动一个图形化终端仿真器来访问shell CLI提示符。

3.2 shell 提示符

一旦启动了终端仿真软件包或者登录Linux虚拟控制台，你就会看到shell CLI提示符。提示符就是进入shell世界的大门，是你输入shell命令的地方。

默认bash shell提示符是美元符号（\$），这个符号表明shell在等待用户输入。不同的Linux发行版采用不同格式的提示符。在Ubuntu Linux系统上，shell提示符看起来是这样的：

```
christine@server01:~$
```

在CentOS系统上是这样的：

```
[christine@server01 ~]$
```

除了作为shell的入口，提示符还能够提供其他的辅助信息。在上面的两个例子中，提示符中显示了当前用户ID名christine。另外还包括系统名server01。在本章的后续部分，你会学习到更多可以在提示符中显示的内容。

窍门 如果你还是CLI新手，请记住，在输入shell命令之后，需要按回车键才能让shell执行你输入的命令。

shell提示符并非一成不变。你可根据自己的需要改变它。第6章讲到了如何修改shell CLI提示符。

可以把shell CLI提示符想象成一名助手，它帮助你使用Linux系统，给你有益的提示，告诉你什么时候shell可以接受新的命令。shell中另一个大有帮助的东西是bash手册。

3.3 bash 手册

大多数Linux发行版自带用以查找shell命令及其他GNU工具信息的在线手册。熟悉手册对使用各种Linux工具大有裨益，尤其是在你要弄清各种命令行参数的时候。

man命令用来访问存储在Linux系统上的手册页面。在想要查找的工具的名称前面输入man命令，就可以找到那个工具相应的手册条目。图3-1展示了查找xterm命令的手册页面的例子。输入命令man xterm就可以进入该页面。

XTERM(1)X Window SystemXTERM(1)

NAME

xterm - terminal emulator for X

SYNOPSIS

xterm [-toolkitoption ...] [-option ...] [shell]

DESCRIPTION

The xterm program is a terminal emulator for the X Window System. It provides DEC VT102/VT220 and selected features from higher-level terminals such as VT320/VT420/VT520 (VTxxx). It also provides Tektronix 4014 emulation for programs that cannot use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), xterm will use the facilities to notify programs running in the window whenever it is resized.

The VTxxx and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height/width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor. The active window can be chosen through escape sequences, the "VT Options"

Manual page xterm(1) line 1 (press h for help or q to quit)

图3-1 xterm命令的手册页面

注意图3-1中xterm命令的DESCRIPTION段落。这些段落排列的并不紧密，字里行间全是技术行话。bash手册并不是按部就班的学习指南，而是作为快速参考来使用的。

窍门 如果你是新接触bash shell，可能一开始会觉得手册页并不太有用。但是，如果养成了阅读手册的习惯，尤其是阅读第一段或是DESCRIPTION部分的前两段，最终你会学到各种技术行话，手册页也会变得越来越有用。

当使用man命令查看命令手册页的时候，这些手册页是由分页程序（pager）来显示的。分页程序是一种实用工具，能够逐页显示文本。可以通过点击空格键进行翻页，或是使用回车键逐行查看。另外还可以使用箭头键向前向后滚动手册页的内容（假设你用的终端仿真软件包支持箭头键功能）。

读完了手册页，可以点击q键退出。退出手册页之后，你会重新获得shell CLI提示符，这表示shell正在等待接受下一条命令。

窍门 bash手册甚至包含了一份有关其自身的参考信息。输入man man来查看与手册页相关的手册页。

手册页将与命令相关的信息分成了不同的节。每一节惯用的命名标准如表3-1所示。

表3-1 Linux手册页惯用的节名

节	描 述
Name	显示命令名和一段简短的描述
Synopsis	命令的语法
Confi guration	命令配置信息
Description	命令的一般性描述
Options	命令选项描述
Exit Status	命令的退出状态指示
Return Value	命令的返回值
Errors	命令的错误消息
Environment	描述所使用的环境变量
Files	命令用到的文件
Versions	命令的版本信息
Conforming To	命名所遵从的标准
Notes	其他有帮助的资料
Bugs	提供提交bug的途径
Example	展示命令的用法
Authors	命令开发人员的信息
Copyright	命令源代码的版权状况
See Also	与该命令类型的其他命令

并不是每一个命令的手册页都包含表3-1中列出的所有节。还有一些命令的节名并没有在上面的节名惯用标准中列出。

窍门 如果不记得命令名怎么办？可以使用关键字搜索手册页。语法是：`man -k 关键字`。例如，要查找与终端相关的命令，可以输入`man -k terminal`。

除了对节按照惯例进行命名,手册页还有对应的内容区域。每个内容区域都分配了一个数字,从1开始,一直到9,如表3-2所示。

表3-2 Linux手册页的内容区域

区 域 号	所涵盖的内容
1	可执行程序或shell命令
2	系统调用
3	库调用
4	特殊文件
5	文件格式与约定

(续)

区 域 号	所涵盖的内容
6	游戏
7	概览、约定及杂项
8	超级用户和系统管理员命令
9	内核例程

man工具通常提供的是命令所对应的最低编号的内容。例如，在图3-1中，我们输入的是命令man xterm，请注意，在现实内容的左上角和右上角，单词XTERM后的括号中有一个数字：(1)。这表示所显示的手册页来自内容区域1（可执行程序或shell命令）。

一个命令偶尔会在多个内容区域都有对应的手册页。比如说，有个叫作hostname的命令。手册页中既包括该命令的相关信息，也包括对系统主机名的概述。要想查看所需要的页面，可以输入man section# topic。对手册页中的第1部分而言，就是输入man 1 hostname。对于手册页中的第7部分，就是输入man 7 hostname。

你也可以只看各部分内容的简介：输入man 1 intro阅读第1部分，输入man 2 intro阅读第2部分，输入man 3 intro阅读第3部分，等等。

手册页不是唯一的参考资料。还有另一种叫作info页面的信息。可以输入info info来了解info页面的相关内容。

另外，大多数命令都可以接受-help或--help选项。例如你可以输入hostname -help来查看帮助。关于帮助的更多信息，可以输入help help。（看出这里面的门道没？）

显然有不少有用的资源可供参考。不过，很多基本的shell概念还是需要详细的解释。在下一节中，我们要讲讲如何浏览Linux文件系统。

3.4 浏览文件系统

当登录系统并获得shell命令提示符后，你通常位于自己的主目录中。一般情况下，你会想去逛逛主目录之外的其他地方。本节将告诉你如何使用shell命令来实现这个目标。在开始前，先了解一下Linux文件系统，为下一步作铺垫。

3.4.1 Linux 文件系统

如果你刚接触Linux系统，可能就很难弄清楚Linux如何引用文件和目录，对已经习惯Microsoft Windows操作系统方式的人来说更是如此。在继续探索Linux系统之前，先了解一下它的布局是有帮助的。

你将注意到的第一个不同点是，Linux在路径名中不使用驱动器盘符。在Windows中，PC上安装的物理驱动器决定了文件的路径名。Windows会为每个物理磁盘驱动器分配一个盘符，每个驱动器都会有自己的目录结构，以便访问存储其中的文件。

举个例子，在Windows中经常看到这样的文件路径：

```
c:\Users\Rich\Documents\test.doc
```

这种Windows文件路径表明了文件test.doc究竟位于哪个磁盘分区中。如果你将test.doc保存在闪存上，该闪存由J来标识，那么文件的路径就是J:\test.doc。该路径表明文件位于J盘的根目录下。

Linux则采用了一种不同的方式。Linux将文件存储在单个目录结构中，这个目录被称为虚拟目录（virtual directory）。虚拟目录将安装在PC上的所有存储设备的文件路径纳入单个目录结构中。

Linux虚拟目录结构只包含一个称为根（root）目录的基础目录。根目录下的目录和文件会按照访问它们的目录路径一一列出，这点跟Windows类似。

窍门 你将会发现Linux使用正斜线（/）而不是反斜线（\）在文件路径中划分目录。在Linux中，反斜线用来标识转义字符，要是用在文件路径中的话会导致各种各样的问题。如果你之前用的是Windows环境，就需要一点时间来适应。

在Linux中，你会看到下面这种路径：

```
/home/Rich/Documents/test.doc
```

这表明文件test.doc位于Documents目录，Documents又位于rich目录中，rich则在home目录中。要注意的是，路径本身并没有提供任何有关文件究竟存放在哪个物理磁盘上的信息。

Linux虚拟目录中比较复杂的部分是它如何协调管理各个存储设备。在Linux PC上安装的第一块硬盘称为根驱动器。根驱动器包含了虚拟目录的核心，其他目录都是在那里开始构建的。

Linux会在根驱动器上创建一些特别的目录，我们称之为挂载点（mount point）。挂载点是虚拟目录中用于分配额外存储设备的目录。虚拟目录会让文件和目录出现在这些挂载点目录中，然而实际上它们却存储在另外一个驱动器中。

通常系统文件会存储在根驱动器中，而用户文件则存储在另一驱动器中，如图3-2所示。

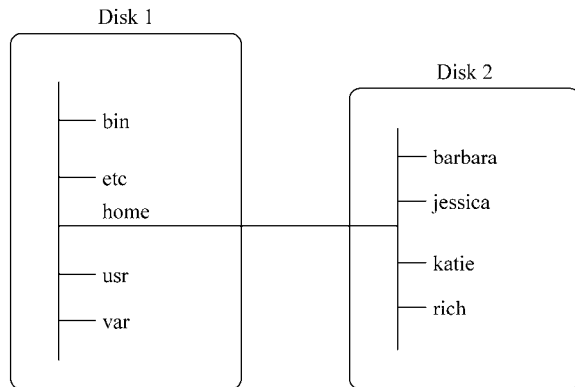


图3-2 Linux文件结构

图3-2展示了计算机中的两块硬盘。一块硬盘和虚拟目录的根目录（由正斜线/表示）关联起来。剩下的硬盘就可以挂载到虚拟目录结构中的任何地方。在这个例子中，第二块硬盘被挂载到了/home位置，用户目录都位于这个位置。

Linux文件系统结构是从Unix文件结构演进过来的。在Linux文件系统中，通用的目录名用于表示一些常见的功能。表3-3列出了一些较常见的Linux顶层虚拟目录名及其内容。

表3-3 常见Linux目录名称

目 录	用 途
/	虚拟目录的根目录。通常不会在这里存储文件
/bin	二进制目录，存放许多用户级的GNU工具
/boot	启动目录，存放启动文件
/dev	设备目录，Linux在这里创建设备节点
/etc	系统配置文件目录
/home	主目录，Linux在这里创建用户目录
/lib	库目录，存放系统和应用程序的库文件
/media	媒体目录，可移动媒体设备的常用挂载点
/mnt	挂载目录，另一个可移动媒体设备的常用挂载点
/opt	可选目录，常用于存放第三方软件包和数据文件
/proc	进程目录，存放现有硬件及当前进程的相关信息
/root	root用户的主目录
/sbin	系统二进制目录，存放许多GNU管理员级工具
/run	运行目录，存放系统运作时的运行时数据
/srv	服务目录，存放本地服务的相关文件
/sys	系统目录，存放系统硬件信息的相关文件
/tmp	临时目录，可以在该目录中创建和删除临时工作文件
/usr	用户二进制目录，大量用户级的GNU工具和数据文件都存储在这里
/var	可变目录，用以存放经常变化的文件，比如日志文件

常见的目录名均基于文件系统层级标准（filesystem hierarchy standard，FHS）。很多Linux发行版都遵循了FHS。这样一来，你就能够在任何兼容FHS的Linux系统中轻而易举地查找文件。

说明 FHS偶尔会进行更新。你可能会发现有些Linux发行版仍在使用旧的FHS标准，而另外一些则只实现了部分当前标准。要想保持与FHS标准同步，请访问其官方主页：<http://www.pathname.com/fhs>。

在登录系统并获得一个shell CLI提示符后，会话将从主目录开始。主目录是分配给用户账户的一个特有目录。用户账户在创建之后，系统通常会为其分配一个特有的目录（参见第7章）。

可以使用图形界面在虚拟目录中跳转。要想在CLI提示符下切换虚拟目录，需要使用cd命令。

3.4.2 遍历目录

在Linux文件系统上，可以使用切换目录命令`cd`将shell会话切换到另一个目录。`cd`命令的格式非常简单：

```
cd destination
```

`cd`命令可接受单个参数`destination`，用以指定想切换到的目录名。如果没有为`cd`命令指定目标路径，它将切换到用户主目录。

`destination`参数可以用两种方式表示：一种是使用绝对文件路径，另一种是使用相对文件路径。

接下来将分别阐述这两种方法。这两者之间的不同对于理解文件系统遍历非常重要。

1. 绝对文件路径

用户可在虚拟目录中采用绝对文件路径引用目录名。绝对文件路径定义了虚拟目录结构中该目录的确切位置，以虚拟目录的根目录开始，相当于目录的全名。

绝对文件路径总是以正斜线（/）作为起始，指明虚拟文件系统的根目录。因此，如果要指向`usr`目录所包含的`bin`目录下的用户二进制文件，可以使用如下绝对文件路径：

```
/usr/bin
```

使用绝对文件路径可以清晰表明用户想切换到的确切位置。要用绝对文件路径来切换到文件系统中的某个特定位置，只需在`cd`命令后指定全路径名：

```
christine@server01:~$ cd /usr/bin
christine@server01:/usr/bin$
```

注意，在上面的例子中，提示符中一开始有一个波浪号（~）。在切换到另一个目录之后，这个波浪号被`/usr/bin`替代了。CLI提示符正是用它来帮助你跟踪当前所在虚拟目录结构中的位置。波浪号表明shell会话位于你的主目录中。在切换出主目录之后，如果提示符已经进行了相关配置的话，绝对文件路径就会显示在提示符中。

说明 如果你的shell CLI提示符中并没有显示shell会话的当前位置，那是因为它并没有进行相关的配置。如果你希望修改CLI提示符的话，第6章会告诉你如何更改配置。

如果没有配置好提示符来显示当前shell会话的绝对文件路径，也可以使用shell命令来显示所处的位置。`pwd`命令可以显示出shell会话的当前目录，这个目录被称为当前工作目录。`pwd`命令的用法如下：

```
christine@server01:/usr/bin$ pwd
/usr/bin
christine@server01:/usr/bin$
```

窍门 在切换到新的当前工作目录时使用`pwd`命令，是很好的习惯。因为很多shell命令都是在当前工作目录中操作的，在发出命令之前，你应该始终确保自己处在正确的目录之中。

可以使用绝对文件路径切换到Linux虚拟目录结构中的任何一级：

```
christine@server01:/usr/bin$ cd /var/log
christine@server01:/var/log$
christine@server01:/var/log$ pwd
/var/log
christine@server01:/var/log$
```

还可以从Linux虚拟目录中的任何一级跳回主目录：

```
christine@server01:/var/log$ cd
christine@server01:~$
christine@server01:~$ pwd
/home/christine
christine@server01:~$
```

但是，如果你只是在自己的主目录中工作，经常使用绝对文件路径的话未免太过冗长。例如，若已经位于目录`/home/christine`，再输入下面这样的命令切换到`Documents`目录就有些繁琐了：

```
cd /home/christine/Documents
```

幸好还有一种简单的解决方法。

2. 相对文件路径

相对文件路径允许用户指定一个基于当前位置的目标文件路径。相对文件路径不以代表根目录的正斜线（/）开头，而是以目录名（如果用户准备切换到当前工作目录下的一个目录）或是一个特殊字符开始。假如你位于`home`目录中，并希望切换到`Documents`子目录，那你可以使用`cd`命令加上一个相对文件路径：

```
christine@server01:~$ pwd
/home/christine
christine@server01:~$
christine@server01:~$ cd Documents
christine@server01:~/Documents$ pwd
/home/christine/Documents
christine@server01:~/Documents$
```

上面的例子并没有使用正斜线（/），而是采用了相对文件路径将当前工作目录从`/home/christine`改为`/home/christine/Documents`，大大减少了输入内容。

另外，此例中还要注意的，如果提示符经过配置可以显示出当前工作目录，它就会一直显示波浪号。这表明当前工作目录位于用户`home`目录之下。

窍门 如果你刚接触命令行和Linux目录结构，建议暂时先坚持使用绝对文件路径。等熟悉了目录布局之后，再使用相对文件路径。

可以在任何包含子目录的目录中使用带有相对文件路径的`cd`命令。也可以使用一个特殊字符来表示相对目录位置。

有两个特殊字符可用于相对文件路径中：

❑ 单点符 (`.`)，表示当前目录；

❑ 双点符 (`..`)，表示当前目录的父目录。

你可以使用单点符，不过对`cd`命令来说，这没有什么意义。在本章后面你会看到另一个命令如何有效地在相对文件路径中使用单点符。

双点符在目录层级中移动时非常便利。如果你处在在主目录下的`Documents`目录中，需要切换到主目录下的`Downloads`目录，可以这么做：

```
christine@server01:~/Documents$ pwd
/home/christine/Documents
christine@server01:~/Documents$ cd ../Downloads
christine@server01:~/Downloads$ pwd
/home/christine/Downloads
christine@server01:~/Downloads$
```

双点符先将用户带到上一级目录，也就是用户的主目录，然后`/Downloads`这部分再将用户带到下一级目录，即`Downloads`目录。必要时用户也可用多个双点符来向上切换目录。假如现在位于主目录中 (`/home/christine`)，想切换到`/etc`目录，可以输入如下命令：

```
christine@server01:~$ cd ../../etc
christine@server01:/etc$ pwd
/etc
christine@server01:/etc$
```

当然，在上面这种情况下，采用相对路径其实比采用绝对路径输入的字符更多，用绝对路径的话，用户只需输入`/etc`。因此，只在必要的时候才使用相对文件路径。

说明 在shell CLI提示符中加入足够的信息非常方便，本节正是这么做的。不过出于清晰性的考虑，在书中余下的例子里，我们只使用一个简单的`$`提示符。

既然你已经知道如何遍历文件系统和验证当前工作目录，那就可以开始探索各种目录中究竟都有些什么东西了。下一节将学习如何查看目录中的文件。

3.5 文件和目录列表

要想知道系统中有哪些文件，可以使用列表命令 (`ls`)。本节将描述`ls`命令和可用来格式化其输出信息的选项。

3.5.1 基本列表功能

`ls`命令最基本的形式会显示当前目录下的文件和目录：

```
$ ls
Desktop      Downloads      Music      Pictures  Templates  Videos
Documents    examples.desktop  my_script  Public    test_file
$
```

注意，`ls`命令输出的列表是按字母排序的（按列排序而不是按行排序）。如果用户用的是支持彩色的终端仿真器，`ls`命令还可以用不同的颜色来区分不同类型的文件。`LS_COLORS`环境变量控制着这个功能。（第6章中会讲到环境变量。）不同的Linux发行版根据各自终端仿真器的能力设置这个环境变量。

如果没安装彩色终端仿真器，可用带`-F`参数的`ls`命令轻松区分文件和目录。使用`-F`参数可以得到如下输出：

```
$ ls -F
Desktop/    Downloads/      Music/      Pictures/  Templates/  Videos/
Documents/  examples.desktop  my_script*  Public/    test_file
$
```

`-F`参数在目录名后加了正斜线（/），以方便用户在输出中分辨它们。类似地，它会在可执行文件（比如上面的`my_script`文件）的后面加个星号，以使用户找出可在系统上运行的文件。

基本的`ls`命令在某种意义上有点容易让人误解。它显示了当前目录下的文件和目录，但并没有将全部都显示出来。Linux经常采用隐藏文件来保存配置信息。在Linux上，隐藏文件通常是文件名以点号开始的文件。这些文件并没有在默认的`ls`命令输出中显示出来，因此我们称其为隐藏文件。

要把隐藏文件和普通文件及目录一起显示出来，就得用到`-a`参数。下面是一个带有`-a`参数的`ls`命令的例子：

```
$ ls -a
.          .compiz      examples.desktop  Music      test_file
..         .config      .gconf            my_script  Videos
.bash_history Desktop      .gstreamer-0.10  Pictures   .Xauthority
.bash_logout .dmrc        .ICEauthority     .profile   .xsession-errors
.bashrc      Documents    .local            Public     .xsession-errors.old
.cache       Downloads    .mozilla          Templates
$
```

所有以点号开头的隐藏文件现在都显示出来了。注意，有三个以`.bash`开始的文件。它们是`bash` shell环境所使用的隐藏文件，在第6章会对其进行详细的讲解。

`-R`参数是`ls`命令可用的另一个参数，叫作递归选项。它列出了当前目录下包含的子目录中的文件。如果目录很多，这个输出就会很长。以下是`-R`参数输出的简单例子：

```
$ ls -F -R
.:
Desktop/    Downloads/      Music/      Pictures/  Templates/  Videos/
Documents/  examples.desktop  my_script*  Public/    test_file

./Desktop:

./Documents:
```

```

./Downloads:

./Music:
ILoveLinux.mp3*

./Pictures:

./Public:

./Templates:

./Videos:
$

```

注意，首先-R参数显示了当前目录下的内容，也就是之前例子中用户home目录下的那些文件。另外，它还显示出了用户home目录下所有子目录及其内容。只有Music子目录中包含了一个可执行文件ILoveLinux.mp3。

窍门 选项并一定要像例子中那样分开输入：`ls -F -R`。它们可以进行如下合并：`ls -FR`。

在上一个例子中，子目录中没再包含子目录。如果有更多的子目录，-R参数会继续进行遍历。正如你所看到的，如果目录结构很庞大，输出内容会变得很长。

3.5.2 显示长列表

在基本的输出列表中，ls命令并未输出太多每个文件的相关信息。要显示附加信息，另一个常用的参数是-l。-l参数会产生长列表格式的输出，包含了目录中每个文件的更多相关信息。

```

$ ls -l
total 48
drwxr-xr-x 2 christine christine 4096 Apr 22 20:37 Desktop
drwxr-xr-x 2 christine christine 4096 Apr 22 20:37 Documents
drwxr-xr-x 2 christine christine 4096 Apr 22 20:37 Downloads
-rw-r--r-- 1 christine christine 8980 Apr 22 13:36 examples.desktop
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fall
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fell
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fill
-rw-rw-r-- 1 christine christine 0 May 21 13:44 full
drwxr-xr-x 2 christine christine 4096 May 21 11:39 Music
-rw-rw-r-- 1 christine christine 0 May 21 13:25 my_file
-rw-rw-r-- 1 christine christine 0 May 21 13:25 my_script
-rwxr-w-r-- 1 christine christine 54 May 21 11:26 my_script
-rw-rw-r-- 1 christine christine 0 May 21 13:42 new_file
drwxr-xr-x 2 christine christine 4096 Apr 22 20:37 Pictures
drwxr-xr-x 2 christine christine 4096 Apr 22 20:37 Public
drwxr-xr-x 2 christine christine 4096 Apr 22 20:37 Templates
-rw-rw-r-- 1 christine christine 0 May 21 11:28 test_file
drwxr-xr-x 2 christine christine 4096 Apr 22 20:37 Videos
$

```

这种长列表格式的输出现在每一行中列出了单个文件或目录。除了文件名，输出中还有其他有用信息。输出的第一行显示了在目录中包含的总块数。在此之后，每一行都包含了关于文件（或目录）的下述信息：

- ❑ 文件类型，比如目录（d）、文件（-）、字符型文件（c）或块设备（b）；
- ❑ 文件的权限（参见第6章）；
- ❑ 文件的硬链接总数；
- ❑ 文件属主的用户名；
- ❑ 文件属组的组名；
- ❑ 文件的大小（以字节为单位）；
- ❑ 文件的上次修改时间；
- ❑ 文件名或目录名。

-l参数是一个强大的工具。有了它，你几乎可以看到系统上任何文件或目录的大部分信息。

在进行文件管理时，ls命令的很多参数都能派上用场。如果在shell提示符中输入man ls，就能看到可用来修改ls命令输出的参数有好几页。

别忘了可以将多个参数结合起来使用。你不时地会发现一些参数组合不仅能够显示出所需的内容，而且还容易记忆，例如ls -alF。

3.5.3 过滤输出列表

由前面的例子可知，默认情况下，ls命令会输出目录下的所有非隐藏文件。有时这个输出会显得过多，当你只需要查看单个少数文件信息时更是如此。

幸而ls命令还支持在命令行中定义过滤器。它会用过滤器来决定应该在输出中显示哪些文件或目录。

这个过滤器就是一个进行简单文本匹配的字符串。可以在要用的命令行参数之后添加这个过滤器：

```
$ ls -l my_script
-rwxrw-r-- 1 christine christine 54 May 21 11:26 my_script
$
```

当用户指定特定文件的名称作为过滤器时，ls命令只会显示该文件的信息。有时你可能不知道要找的那个文件的确切名称。ls命令能够识别标准通配符，并在过滤器中用它们进行模式匹配：

- ❑ 问号（?）代表一个字符；
- ❑ 星号（*）代表零个或多个字符。

问号可用于过滤器字符串中替代任意位置的单个字符。例如：

```
$ ls -l my_scr?pt
-rw-rw-r-- 1 christine christine 0 May 21 13:25 my_script
-rwxrw-r-- 1 christine christine 54 May 21 11:26 my_script
$
```

其中，过滤器`my_scr?pt`与目录中的两个文件匹配。类似地，星号可匹配零个或多个字符。

```
$ ls -l my*
-rw-rw-r-- 1 christine christine 0 May 21 13:25 my_file
-rw-rw-r-- 1 christine christine 0 May 21 13:25 my_scrapt
-rwxrw-r-- 1 christine christine 54 May 21 11:26 my_script
$
```

使用星号找到了三个名字以`my`开头的文件。和问号一样，你可以把星号放在过滤器中的任意位置。

```
$ ls -l my_s*t
-rw-rw-r-- 1 christine christine 0 May 21 13:25 my_scrapt
-rwxrw-r-- 1 christine christine 54 May 21 11:26 my_script
$
```

在过滤器中使用星号和问号被称为文件扩展匹配（file globbing），指的是使用通配符进行模式匹配的过程。通配符正式的名称叫作元字符通配符（metacharacter wildcards）。除了星号和问号之外，还有更多的元字符通配符可用于文件扩展匹配。可以使用中括号。

```
$ ls -l my_scr[ai]pt
-rw-rw-r-- 1 christine christine 0 May 21 13:25 my_scrapt
-rwxrw-r-- 1 christine christine 54 May 21 11:26 my_script
$
```

在这个例子中，我们使用了中括号以及在特定位置上可能出现的两种字符：`a`或`i`。中括号表示一个字符位置并给出多个可能的选择。可以像上面的例子那样将待选的字符列出来，也可以指定字符范围，例如字母范围`[a - i]`。

```
$ ls -l f[a-i]ll
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fall
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fell
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fill
$
```

另外，可以使用感叹号（`!`）将不需要的内容排除在外。

```
$ ls -l f[!a]ll
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fell
-rw-rw-r-- 1 christine christine 0 May 21 13:44 fill
-rw-rw-r-- 1 christine christine 0 May 21 13:44 full
$
```

在进行文件搜索时，文件扩展匹配是一个功能强大的特性。它也可以用于`ls`以外的其他shell命令。本章随后的部分会有更多相关的例子。

3.6 处理文件

shell提供了很多在Linux文件系统上操作文件的命令。本节将带你逐步了解文件处理所需要的一些基本的shell命令。

3.6.1 创建文件

你总会时不时地遇到要创建空文件的情况。例如，有时应用程序希望在它们写入数据之前，某个日志文件已经存在。这时，可用touch命令轻松创建空文件。

```
$ touch test_one
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:17 test_one
$
```

touch命令创建了你指定的新文件，并将你的用户名作为文件的属主。注意，文件的大小是零，因为touch命令只创建了一个空文件。

touch命令还可用来改变文件的修改时间。这个操作并不需要改变文件的内容。

```
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:17 test_one
$ touch test_one
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
$
```

test_one文件的修改时间现在已经从最初的时间14:17更新到了14:35。如果只想改变访问时间，可用-a参数。

```
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
$ touch -a test_one
$ ls -l test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
$ ls -l --time=atime test_one
-rw-rw-r-- 1 christine christine 0 May 21 14:55 test_one
$
```

在上面的例子中，要注意的是，如果只使用ls -l命令，并不会显示访问时间。这是因为默认显示的是修改时间。要想查看文件的访问时间，需要加入另外一个参数：--time=atime。有了这个参数，就能够显示出已经更改过的文件访问时间。

创建空文件和更改文件时间戳算不上你在Linux系统中的日常工作。不过复制文件可是在使用shell时经常要干的活儿。

3.6.2 复制文件

对系统管理员来说，在文件系统中将文件和目录从一个位置复制到另一个位置可谓家常便饭。cp命令可以完成这个任务。

在最基本的用法里，cp命令需要两个参数——源对象和目标对象：

```
cp source destination
```

当source和destination参数都是文件名时，cp命令将源文件复制成一个新文件，并且以destination命名。新文件就像全新的文件一样，有新的修改时间。

```
$ cp test_one test_two
$ ls -l test_*
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
-rw-rw-r-- 1 christine christine 0 May 21 15:15 test_two
$
```

新文件`test_two`和文件`test_one`的修改时间并不一样。如果目标文件已经存在，`cp`命令可能并不会提醒这一点。最好是加上`-i`选项，强制shell询问是否需要覆盖已有文件。

```
$ ls -l test_*
-rw-rw-r-- 1 christine christine 0 May 21 14:35 test_one
-rw-rw-r-- 1 christine christine 0 May 21 15:15 test_two
$
$ cp -i test_one test_two
cp: overwrite 'test_two'? n
$
```

如果不回答`y`，文件复制将不会继续。也可以将文件复制到现有目录中。

```
$ cp -i test_one /home/christine/Documents/
$
$ ls -l /home/christine/Documents
total 0
-rw-rw-r-- 1 christine christine 0 May 21 15:25 test_one
$
```

新文件现就在目录`Documents`中了，和源文件同名。

说明 之前的例子在目标目录名尾部加上了一个正斜线(`/`)，这表明`Documents`是目录而非文件。这有助于明确目的，而且在复制单个文件时非常重要。如果没有使用正斜线，子目录`/home/christine/Documents`又不存在，就会有麻烦。在这种情况下，试图将一个文件复制到`Documents`子目录反而会创建一个名为`Documents`的文件，连错误消息都不会显示！

上一个例子采用了绝对路径，不过也可以使用相对路径。

```
$ cp -i test_one Documents/
cp: overwrite 'Documents/test_one'? y
$
$ ls -l Documents
total 0
-rw-rw-r-- 1 christine christine 0 May 21 15:28 test_one
$
```

本章在前面介绍了特殊符号可以用在相对文件路径中。其中的单点符(`.`)就很适合用于`cp`命令。记住，单点符表示当前工作目录。如果需要将一个带有很长的源对象名的文件复制到当前工作目录中时，单点符能够简化该任务。

```
$ cp -i /etc/NetworkManager/NetworkManager.conf .
$
$ ls -l NetworkManager.conf
-rw-r--r-- 1 christine christine 76 May 21 15:55 NetworkManager.conf
$
```


想找到那个单点符可真是不容易！仔细看的话，你会发现它在第一行命令的末尾。如果你的源对象名很长，使用单点符要比输入完整的目标对象名省事得多。

窍门 `cp`命令的参数要比这里叙述的多得多。别忘了用`man cp`，你可以看到`cp`命令所有的可用参数。

`cp`命令的`-R`参数威力强大。可以用它在一条命令中递归地复制整个目录的内容。

```
$ ls -Fd *Scripts
Scripts/
$ ls -l Scripts/
total 25
-rwxrw-r-- 1 christine christine 929 Apr  2 08:23 file_mod.sh
-rwxrw-r-- 1 christine christine 254 Jan  2 14:18 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 Jan  2 13:42 SUID_search.sh
$
$ cp -R Scripts/ Mod_Scripts
$ ls -Fd *Scripts
Mod_Scripts/ Scripts/
$ ls -l Mod_Scripts
total 25
-rwxrw-r-- 1 christine christine 929 May 21 16:16 file_mod.sh
-rwxrw-r-- 1 christine christine 254 May 21 16:16 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 May 21 16:16 SUID_search.sh
$
```

在执行`cp -R`命令之前，目录`Mod_Scripts`并不存在。它是随着`cp -R`命令被创建的，整个`Scripts`目录中的内容都被复制到其中。注意，在新的`Mod_Scripts`目录中，所有的文件都有对应的新日期。`Mod_Scripts`目录现在已经成为了`Scripts`目录的完整副本。

说明 在上面的例子中，`ls`命令加入了`-Fd`选项。之前你已经见过`-F`选项了，不过`-d`选项可能还是第一次碰到。后者只列出目录本身的信息，不列出其中的内容。

也可以在`cp`命令中使用通配符。

```
$ cp *script Mod_Scripts/
$ ls -l Mod_Scripts
total 26
-rwxrw-r-- 1 christine christine 929 May 21 16:16 file_mod.sh
-rwxrw-r-- 1 christine christine 54  May 21 16:27 my_script
-rwxrw-r-- 1 christine christine 254 May 21 16:16 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 May 21 16:16 SUID_search.sh
$
```

该命令将所有以`script`结尾的文件复制到`Mod_Scripts`目录中。在这里，只需要复制一个文件：`my_script`。

在复制文件的时候，除了单点符和通配符之外，另一个shell特性也能派上用场。那就是制表键自动补全。

3.6.3 制表键自动补全

在使用命令行时，很容易输错命令、目录名或文件名。实际上，对长目录名或文件名来说，输错的几率还是蛮高的。

这正是制表键自动补全挺身而出的时候。制表键自动补全允许你在输入文件名或目录名时按一下制表键，让shell帮忙将内容补充完整。

```
$ ls really*
really_ridiculously_long_file_name
$
$ cp really_ridiculously_long_file_name Mod_Scripts/
ls -l Mod_Scripts
total 26
-rwxrw-r-- 1 christine christine 929 May 21 16:16 file_mod.sh
-rwxrw-r-- 1 christine christine 54 May 21 16:27 my_script
-rw-rw-r-- 1 christine christine 0 May 21 17:08
really_ridiculously_long_file_name
-rwxrw-r-- 1 christine christine 254 May 21 16:16 SGID_search.sh
-rwxrw-r-- 1 christine christine 243 May 21 16:16 SUID_search.sh
$
```

在上面的例子中，我们输入了命令`cp really`，然后按制表键，shell就将剩下的文件名自动补充完整了！当然了，目标目录还是得输入的，不过仍然可以利用命令补全来避免输入错误。

使用制表键自动补全的技巧在于要给shell足够的文件名信息，使其能够将需要文件同其他文件区分开。假如有另一个文件名也是以`really`开头，那么就算按了制表键，也无法完成文件名的自动补全。这时候你会听到嘟的一声。要是再按一下制表键，shell就会列出所有以`really`开头的文件名。这个特性可以让你观察究竟应该输入哪些内容才能完成自动补全。

3.6.4 链接文件

链接文件是Linux文件系统的优势。如需要在系统上维护同一文件的两份或多份副本，除了保存多份单独的物理文件副本之外，还可以采用保存一份物理文件副本和多个虚拟副本的方法。这种虚拟的副本就称为链接。链接是目录中指向文件真实位置的占位符。在Linux中有两种不同类型的文件链接：

- ❑ 符号链接
- ❑ 硬链接

符号链接就是一个实实在在的文件，它指向存放在虚拟目录结构中某个地方的另一个文件。这两个通过符号链接在一起的文件，彼此的内容并不相同。

要为一个文件创建符号链接，原始文件必须事先存在。然后可以使用`ln`命令以及`-s`选项来创建符号链接。

```
$ ls -l data_file
-rw-rw-r-- 1 christine christine 1092 May 21 17:27 data_file
$
$ ln -s data_file sl_data_file
$
$ ls -l *data_file
-rw-rw-r-- 1 christine christine 1092 May 21 17:27 data_file
lrwxrwxrwx 1 christine christine    9 May 21 17:29 sl_data_file -> data_file
$
```

在上面的例子中，注意符号链接的名字`sl_data_file`位于`ln`命令中的第二个参数位置上。显示在长列表中符号文件名后的`->`符号表明该文件是链接到文件`data_file`上的一个符号链接。

另外还要注意的，符号链接的文件大小与数据文件的文件大小。符号链接`sl_data_file`只有9个字节，而`data_file`有1092个字节。这是因为`sl_data_file`仅仅只是指向`data_file`而已。它们的内容并不相同，是两个完全不同的文件。

另一种证明链接文件是独立文件的方法是查看inode编号。文件或目录的inode编号是一个用于标识的唯一数字，这个数字由内核分配给文件系统中的每一个对象。要查看文件或目录的inode编号，可以给`ls`命令加入`-i`参数。

```
$ ls -i *data_file
296890 data_file 296891 sl_data_file
$
```

从这个例子中可以看出数据文件的inode编号是296890，而`sl_data_file`的inode编号则是296891。所以说它们是不同的文件。

硬链接会创建独立的虚拟文件，其中包含了原始文件的信息及位置。但是它们从根本上而言是同一个文件。引用硬链接文件等同于引用了源文件。要创建硬链接，原始文件也必须事先存在，只不过这次使用`ln`命令时不再需要加入额外的参数了。

```
$ ls -l code_file
-rw-rw-r-- 1 christine christine 189 May 21 17:56 code_file
$
$ ln code_file hl_code_file
$
$ ls -li *code_file
296892 -rw-rw-r-- 2 christine christine 189 May 21 17:56
code_file
296892 -rw-rw-r-- 2 christine christine 189 May 21 17:56
hl_code_file
$
```

在上面的例子中，我们使用`ls -li`命令显示了`*code_files`的inode编号以及长列表。注意，带有硬链接的文件共享inode编号。这是因为它们终归是同一个文件。还要注意的，链接计数（列表中第三项）显示这两个文件都有两个链接。另外，它们的文件大小也一模一样。

说明 只能对处于同一存储媒体的文件创建硬链接。要想在不同存储媒体的文件之间创建链接，只能使用符号链接。

复制链接文件的时候一定要小心。如果使用cp命令复制一个文件，而该文件又已经被链接到了另一个源文件上，那么你得到的其实是源文件的一个副本。这很容易让人犯晕。用不着复制链接文件，可以创建原始文件的另一个链接。同一个文件拥有多个链接，这完全没有问题。但是，千万别创建软链接文件的软链接。这会形成混乱的链接链，不仅容易断裂，还会造成各种麻烦。

你可能觉得符号链接和硬链接的概念不好理解。幸好下一节中的文件重命名容易明白得多。

3.6.5 重命名文件

在Linux中，重命名文件称为移动（moving）。mv命令可以将文件和目录移动到另一个位置或重新命名。

```
$ ls -li f?ll
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fall
296717 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fell
294561 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fill
296742 -rw-rw-r-- 1 christine christine 0 May 21 13:44 full
$
$ mv fall fzll
$
$ ls -li f?ll
296717 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fell
294561 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fill
296742 -rw-rw-r-- 1 christine christine 0 May 21 13:44 full
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fzll
$
```

注意，移动文件会将文件名从fall更改为fzll，但inode编号和时间戳保持不变。这是因为mv只影响文件名。

也可以使用mv来移动文件的位置。

```
$ ls -li /home/christine/fzll
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44
/home/christine/fzll
$
$ ls -li /home/christine/Pictures/
total 0
$ mv fzll Pictures/
$
$ ls -li /home/christine/Pictures/
total 0
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44 fzll
$
$ ls -li /home/christine/fzll
```

```
ls: cannot access /home/christine/fzll: No such file or directory
$
```

在上例中，我们使用mv命令把文件fzll从/home/christine移动到了/home/christine/Pictures。和刚才一样，这个操作并没有改变文件的inode编号或时间戳。

窍门 和cp命令类似，也可以在mv命令中使用-i参数。这样在命令试图覆盖已有的文件时，你就会得到提示。

3

唯一变化的就是文件的位置。/home/christine目录下不再有文件fzll，因为它已经离开了原先的位置，这就是mv命令所做的事情。

也可以使用mv命令移动文件位置并修改文件名称，这些操作只需一步就能完成。

```
$ ls -li Pictures/fzll
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44
Pictures/fzll
$
$ mv /home/christine/Pictures/fzll /home/christine/fall
$
$ ls -li /home/christine/fall
296730 -rw-rw-r-- 1 christine christine 0 May 21 13:44
/home/christine/fall
$
$ ls -li /home/christine/Pictures/fzll
ls: cannot access /home/christine/Pictures/fzll:
No such file or directory
```

在这个例子中，我们将文件fzll从子目录Pictures中移动到了主目录/home/christine，并将名字改为fall。文件的时间戳和inode编号都没有改变。改变的只有位置和名称。

也可以使用mv命令移动整个目录及其内容。

```
$ ls -li Mod_Scripts
total 26
296886 -rwxrw-r-- 1 christine christine 929 May 21 16:16
file_mod.sh
296887 -rwxrw-r-- 1 christine christine 54 May 21 16:27
my_script
296885 -rwxrw-r-- 1 christine christine 254 May 21 16:16
SGID_search.sh
296884 -rwxrw-r-- 1 christine christine 243 May 21 16:16
SUID_search.sh
$
$ mv Mod_Scripts Old_Scripts
$
$ ls -li Mod_Scripts
ls: cannot access Mod_Scripts: No such file or directory
$
$ ls -li Old_Scripts
total 26
296886 -rwxrw-r-- 1 christine christine 929 May 21 16:16
```

```
file_mod.sh
296887 -rwxrw-r-- 1 christine christine 54 May 21 16:27
my_script
296885 -rwxrw-r-- 1 christine christine 254 May 21 16:16
SGID_search.sh
296884 -rwxrw-r-- 1 christine christine 243 May 21 16:16
SUID_search.sh
$
```

目录内容没有变化。只有目录名发生了改变。

在知道了如何使用`mv`命令进行重命名……不对……移动文件之后,你应该发现这其实非常容易的。另一个简单但可能有危险的任务是删除文件。

3.6.6 删除文件

迟早有一天,你得删除已有的文件。不管是清理文件系统还是删除某个软件包,总有要删除文件的时候。

在Linux中,删除(deleting)叫作移除(removing)^①。bash shell中删除文件的命令是`rm`。rm命令的基本格式非常简单。

```
$ rm -i fall
rm: remove regular empty file 'fall'? y
$
$ ls -l fall
ls: cannot access fall: No such file or directory
$
```

注意, `-i`命令参数提示你是不是要真的删除该文件。bash shell中没有回收站或垃圾箱,文件一旦删除,就无法再找回。因此,在使用`rm`命令时,要养成总是加入`-i`参数的好习惯。

也可以使用通配符删除成组的文件。别忘了使用`-i`选项保护自己的文件。

```
$ rm -i f?ll
rm: remove regular empty file 'fell'? y
rm: remove regular empty file 'fill'? y
rm: remove regular empty file 'full'? y
$
$ ls -l f?ll
ls: cannot access f?ll: No such file or directory
$
```

`rm`命令的另外一个特性是,如果要删除很多文件且不受提示符的打扰,可以用`-f`参数强制删除。小心为妙!

^① 这里原文可理解为删除的功能实际上是移除(remove)命令`rm`完成的,在本书中,我们依然用“删除”这个大家已经习惯的叫法。

3.7 处理目录

在Linux中，有些命令（比如cp命令）对文件和目录都有效，而有些只对目录有效。创建新目录需要使用本节讲到的一个特殊命令。删除目录也很有意思，本节也会讲到。

3.7.1 创建目录

在Linux中创建目录很简单，用mkdir命令即可：

```
$ mkdir New_Dir
$ ls -ld New_Dir
drwxrwxr-x 2 christine christine 4096 May 22 09:48 New_Dir
$
```

系统创建了一个名为New_Dir的新目录。注意，新目录长列表是以d开头的。这表示New_Dir并不是文件，而是一个目录。

可以根据需要批量地创建目录和子目录。但是，如果你想单单靠mkdir命令来实现，就会得到下面的错误消息：

```
$ mkdir New_Dir/Sub_Dir/Under_Dir
mkdir: cannot create directory 'New_Dir/Sub_Dir/Under_Dir':
No such file or directory
$
```

要想同时创建多个目录和子目录，需要加入-p参数：

```
$ mkdir -p New_Dir/Sub_Dir/Under_Dir
$
$ ls -R New_Dir
New_Dir:
Sub_Dir

New_Dir/Sub_Dir:
Under_Dir

New_Dir/Sub_Dir/Under_Dir:
$
```

mkdir命令的-p参数可以根据需要创建缺失的父目录。父目录是包含目录树中下一级目录的目录。

当然，完事之后，你得知道怎么样删除目录，尤其是在把目录建错地方的时候。

3.7.2 删除目录

删除目录之所以很棘手，是有原因的。删除目录时，很有可能会发生一些不好的事情。shell会尽可能防止我们捅娄子。删除目录的基本命令是rmdir。

```
$ touch New_Dir/my_file
$ ls -li New_Dir/
```

```
total 0
294561 -rw-rw-r-- 1 christine christine 0 May 22 09:52 my_file
$
$ rm -i New_Dir
rm: failed to remove 'New_Dir': Directory not empty
$
```

默认情况下，`rm`命令只删除空目录。因为我们在`New_Dir`目录下创建了一个文件`my_file`，所以`rm`命令拒绝删除目录。

要解决这一问题，得先把目录中的文件删掉，然后才能在空目录上使用`rm`命令。

```
$ rm -i New_Dir/my_file
rm: remove regular empty file 'New_Dir/my_file'? y
$
$ rm -i New_Dir
$
$ ls -ld New_Dir
ls: cannot access New_Dir: No such file or directory
```

`rm`并没有`-i`选项来询问是否要删除目录。这也是为什么说`rm`只能删除空目录还是有好处的原因。

也可以在整个非空目录上使用`rm`命令。使用`-r`选项使得命令可以向下进入目录，删除其中的文件，然后再删除目录本身。

```
$ ls -l My_Dir
total 0
-rw-rw-r-- 1 christine christine 0 May 22 10:02 another_file
$
$ rm -ri My_Dir
rm: descend into directory 'My_Dir'? y
rm: remove regular empty file 'My_Dir/another_file'? y
rm: remove directory 'My_Dir'? y
$
$ ls -l My_Dir
ls: cannot access My_Dir: No such file or directory
$
```

这种方法同样可以向下进入多个子目录，当需要删除大量目录和文件时，这一点尤为有效。

```
$ ls -FR Small_Dir
Small_Dir:
a_file b_file c_file Teeny_Dir/ Tiny_Dir/

Small_Dir/Teeny_Dir:
e_file

Small_Dir/Tiny_Dir:
d_file
$
$ rm -ir Small_Dir
rm: descend into directory 'Small_Dir'? y
rm: remove regular empty file 'Small_Dir/a_file'? y
rm: descend into directory 'Small_Dir/Tiny_Dir'? y
```



```
rm: remove regular empty file 'Small_Dir/Tiny_Dir/d_file'? y
rm: remove directory 'Small_Dir/Tiny_Dir'? y
rm: descend into directory 'Small_Dir/Teeny_Dir'? y
rm: remove regular empty file 'Small_Dir/Teeny_Dir/e_file'? y
rm: remove directory 'Small_Dir/Teeny_Dir'? y
rm: remove regular empty file 'Small_Dir/c_file'? y
rm: remove regular empty file 'Small_Dir/b_file'? y
rm: remove directory 'Small_Dir'? y
$
$ ls -FR Small_Dir
ls: cannot access Small_Dir: No such file or directory
$
```

这种方法虽然可行，但很难用。注意，你依然要确认每个文件是否要被删除。如果该目录有很多个文件和子目录，这将非常琐碎。

说明 对rm命令而言，-r参数和-R参数的效果是一样的。-R参数同样可以递归地删除目录中的文件。shell命令很少会就相同的功能采用不同大小写的参数。

一口气删除目录及其所有内容的终极大法就是使用带有-r参数和-f参数的rm命令。

```
$ tree Small_Dir
Small_Dir
├─ a_file
├─ b_file
├─ c_file
├─ Teeny_Dir
│   └─ e_file
└─ Tiny_Dir
    └─ d_file

2 directories, 5 files
$
$ rm -rf Small_Dir
$
$ tree Small_Dir
Small_Dir [error opening dir]

0 directories, 0 files
$
```

rm -rf命令既没有警告信息，也没有声音提示。这肯定是一个危险的工具，尤其是在拥有超级用户权限的时候。务必谨慎使用，请再三检查你所要进行的操作是否符合预期。

说明 在上面的例子中，我们使用了tree工具。它能够以一种美观的方式展示目录、子目录及其中的文件。如果需要了解目录结构，尤其是在删除目录之前，这款工具正好能派上用场。不过它可能并没有默认安装在你所使用的Linux发行版中。请参阅第9章，学习如何安装软件。

在前面几节中，你看到了如何管理文件和目录。到此为止，除了如何查看文件内容，我们已经讲述了你所需要的有关文件的全部知识。

3.8 查看文件内容

Linux中有几个命令可以查看文件的内容，而不需要调用其他文本编辑器（参见第10章）。本节将演示一些可以帮助查看文件内容的命令。

3.8.1 查看文件类型

在显示文件内容之前，应该先了解一下文件的类型。如果打开了一个二进制文件，你会在屏幕上看到各种乱码，甚至会把你的终端仿真器挂起。

`file`命令是一个随手可得的便捷工具。它能够探测文件的内部，并决定文件是什么类型的：

```
$ file my_file
my_file: ASCII text
$
```

上面例子中的文件是一个`text`（文本）文件。`file`命令不仅能确定文件中包含的文本信息，还能确定该文本文件的字符编码，`ASCII`。

下面例子中的文件就是一个目录。因此，以后可以使用`file`命令作为另一种区分目录的方法：

```
$ file New_Dir
New_Dir: directory
$
```

第三个`file`命令的例子中展示了一个类型为符号链接的文件。注意，`file`命令甚至能够告诉你它链接到了哪个文件上：

```
$ file sl_data_file
sl_data_file: symbolic link to 'data_file'
$
```

下面的例子展示了`file`命令对脚本文件的返回结果。尽管这个文件是`ASCII text`，但因为它是一个脚本文件，所以可以在系统上执行（运行）：

```
$ file my_script
my_script: Bourne-Again shell script, ASCII text executable
$
```

最后一个例子是二进制可执行程序。`file`命令能够确定该程序编译时所面向的平台以及需要何种类型的库。如果你有从未知源处获得的二进制文件，这会是是个非常有用的特性：

```
$ file /bin/ls
/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.24,
[...]
$
```

现在你已经学会了如何快速查看文件类型，接着就可以开始学习文件的显示与浏览了。

3.8.2 查看整个文件

如果手头有一个很大的文本文件，你可能会想看看里面是什么内容。在Linux上有3个不同的命令可以完成这个任务。

1. cat命令

cat命令是显示文本文件中所有数据的得力工具。

```
$ cat test1
hello
```

```
This is a test file.
```

```
That we'll use to      test the cat command.
$
```

没什么特别的，就是文本文件的内容而已。这里还有一些可以和cat命令一起用的参数，可能对你有所帮助。

-n参数会给所有的行加上行号。

```
$ cat -n test1
 1 hello
 2
 3 This is a test file.
 4
 5
 6 That we'll use to      test the cat command.
$
```

这个功能在检查脚本时很有用。如果只想给有文本的行加上行号，可以用-b参数。

```
$ cat -b test1
 1 hello

 2 This is a test file.

 3 That we'll use to      test the cat command.
$
```

最后，如果不想让制表符出现，可以用-T参数。

```
$ cat -T test1
hello

This is a test file.

That we'll use to^Itest the cat command.
$
```

-T参数会用^I字符组合去替换文中的所有制表符。

对大型文件来说，cat命令有点繁琐。文件的文本会在显示器上一晃而过。好在有一个简单办法可以解决这个问题。

2. more命令

cat命令的主要缺陷是：一旦运行，你就无法控制后面的操作。为了解决这个问题，开发人员编写了more命令。more命令会显示文本文件的内容，但会在显示每页数据之后停下来。我们输入命令more /etc/bash.bashrc生成如图3-3中所显示的内容。

```
shopt -s checkwinsize

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, overwrite the one in /etc/profile)
PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '

# Commented out, don't overwrite xterm -T "title" -n "icontitle" by default.
# If this is an xterm set the title to user@host:dir
#case "$TERM" in
#xterm*|rxvt*)
#    PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME}: ${PWD}\007"'
#    ;;
#*)
#    ;;
#esac

# enable bash completion in interactive shells
#if ! shopt -oq posix; then
#    if [ -f /usr/share/bash-completion/bash_completion ]; then
#        . /usr/share/bash-completion/bash_completion
#    elif [ -f /etc/bash_completion ]; then
#        . /etc/bash_completion
#    fi
#fi

--More--(56%)
```

图3-3 使用more命令显示文本文件

注意图3-3中屏幕的底部，more命令显示了一个标签，其表明你仍然在more程序中以及你现在的位置在这个文本文件中的位置。这是more命令的提示符。

more命令是分页工具。在本章前面的内容里，当使用man命令时，分页工具会显示所选的bash手册页面。和在手册页中前后移动一样，你可以通过按空格键或回车键以逐行向前的方式浏览文本文件。浏览完之后，按q键退出。

more命令只支持文本文件中的基本移动。如果要更多高级功能，可以试试less命令。

3. less命令

从名字上看，它并不像more命令那样高级。但是，less命令的命名实际上是个文字游戏（从俗语“less is more”得来），它实为more命令的升级版。它提供了一些极为实用的特性，能够实现在文本文件中前后翻动，而且还有一些高级搜索功能。

less命令的操作和more命令基本一样，一次显示一屏的文件文本。除了支持和more命令相同的命令集，它还包括更多的选项。

窍门 要想查看`less`命令所有的可用选项，可以输入`man less`浏览对应的手册页。也可以这样查看`more`命令选项的参考资料。

其中一组特性就是`less`命令能够识别上下键以及上下翻页键（假设你的终端配置正确）。在查看文件内容时，这给了你全面的控制权。

3.8.3 查看部分文件

3

通常你要查看的数据要么在文本文件的开头，要么在文本文件的末尾。如果这些数据是在大型文件的起始部分，那你就得等`cat`或`more`加载完整个文件之后才能看到。如果数据是在文件的末尾（比如日志文件），那可能需要翻过成千上万行的文本才能到最后的内容。好在Linux有解决这两个问题的专用命令。

1. `tail`命令

`tail`命令会显示文件最后几行的内容（文件的“尾部”）。默认情况下，它会显示文件的末尾10行。

出于演示的目的，我们创建了一个包含20行文本的文本文件。使用`cat`命令显示该文件的全部内容如下：

```
$ cat log_file
line1
line2
line3
line4
line5
Hello World - line 6
line7
line8
line9
line10
line11
Hello again - line 12
line13
line14
line15
Sweet - line16
line17
line18
line19
Last line - line20
$
```

现在你已经看到了整个文件，可以再看看使用`tail`命令浏览文件最后10行的效果：

```
$ tail log_file
line11
Hello again - line 12
line13
```

```
line14
line15
Sweet - line16
line17
line18
line19
Last line - line20
$
```

可以向tail命令中加入-n参数来修改所显示的行数。在下面的例子中，通过加入-n 2使tail命令只显示文件的最后两行：

```
$ tail -n 2 log_file
line19
Last line - line20
$
```

-f参数是tail命令的一个突出特性。它允许你在其他进程使用该文件时查看文件的内容。tail命令会保持活动状态，并不断显示添加到文件中的内容。这是实时监测系统日志的绝妙方式。

2. head命令

head命令，顾名思义，会显示文件开头那些行的内容。默认情况下，它会显示文件前10行的文本：

```
$ head log_file
line1
line2
line3
line4
line5
Hello World - line 6
line7
line8
line9
line10
$
```

类似于tail命令，它也支持-n参数，这样就可以指定想要显示的内容了。这两个命令都允许你在破折号后面输入想要显示的行数：

```
$ head -5 log_file
line1
line2
line3
line4
line5
$
```

文件的开头通常不会改变，因此head命令并像tail命令那样支持-f参数特性。head命令是一种查看文件起始部分内容的便捷方法。

3.9 小结

本章涵盖了在shell提示符下操作Linux文件系统的基础知识。一开始我们讨论了bash shell，之后介绍了怎样和shell交互。命令行界面（CLI）采用提示符来表明你可以输入命令。bash shell提供了很多可用以创建和操作文件的工具。在开始操作文件之前，很有必要先了解一下Linux怎么存储文件。本章讨论了Linux虚拟目录的基础知识，然后展示了Linux如何引用存储设备。在描述了Linux文件系统之后，还带你逐步了解了如何使用cd命令在虚拟目录里切换目录。

在介绍如何进入指定目录后，我们又演示了怎样用ls命令列出目录中的文件和子目录。ls命令有很多参数可用来定制输出内容。可以通过ls命令获得有关文件和目录的信息。

touch命令非常有用，可以创建空文件和变更已有文件的访问时间或修改时间。本章还介绍了如何使用cp命令将已有文件复制到其他位置。另外还逐步介绍了如何链接文件，给出了一种简单的方法可以实现两个位置上拥有同一个文件且不用生成单独的副本。ln命令提供了这种链接功能。

接着我们讲了怎样用mv命令重命名文件（在Linux中称为移动文件），以及如何用rm命令删除文件（在Linux中称为移除文件），还介绍了怎样用mkdir和rmdir命令对目录执行相同的任务。

最后，本章以如何查看文件的内容作结。cat、more和less命令可以非常方便地查看文件全部内容，而且tail和head命令还可查看文件中的一小部分内容。

下章将继续讨论bash shell的命令，并了解更多管理Linux系统时经常用到的高级系统管理命令。

第 4 章

更多的bash shell命令

本章内容

- ❑ 管理进程
- ❑ 获取磁盘统计信息
- ❑ 挂载新磁盘
- ❑ 排序数据
- ❑ 归档数据

第3章介绍了Linux文件系统上切换目录以及处理文件和目录的基本知识。文件管理和目录管理是Linux shell的主要功能之一。不过，在开始脚本编程之前，我们还需要了解一下其他方面的知识。本章将详细介绍Linux系统管理命令，演示如何通过命令行命令来探查Linux系统的内部信息，最后介绍一些可以用来操作系统上数据文件的命令。

4.1 监测程序

Linux系统管理员面临的最复杂的任务之一就是跟踪运行在系统中的程序——尤其是现在，图形化桌面集成了大量的应用来生成一个完整的桌面环境。系统中总是运行着大量的程序。

好在有一些命令行工具可以使你的生活轻松一些。本节将会介绍一些能帮你在Linux系统上管理程序的基本工具及其用法。

4.1.1 探查进程

当程序运行在系统上时，我们称之为进程（process）。想监测这些进程，需要熟悉ps命令的用法。ps命令好比工具中的瑞士军刀，它能输出运行在系统上的所有程序的许多信息。

遗憾的是，随着它的稳健而来的还有复杂性——有数不清的参数，这或许让ps命令成了最难掌握的命令。大多数系统管理员在掌握了能提供他们需要信息的一组参数之后，就一直坚持只使用这组参数。

默认情况下，ps命令并不会提供那么多的信息：

```
$ ps
```



```
PID TTY          TIME CMD
3081 pts/0      00:00:00 bash
3209 pts/0      00:00:00 ps
$
```

没什么特别的吧？默认情况下，`ps`命令只会显示运行在当前控制台下的属于当前用户的进程。在此例中，我们只运行了**bash shell**（注意，**shell**也只是运行在系统上的另一个程序而已）以及**ps**命令本身。

上例中的基本输出显示了程序的进程ID（**Process ID**，**PID**）、它们运行在哪个终端（**TTY**）以及进程已用的CPU时间。

说明 `ps`命令叫人头疼的地方（也正是它如此复杂的原因）在于它曾经有两个版本。每个版本都有自己的命令行参数集，这些参数控制着输出什么信息以及如何显示。最近，Linux开发人员已经将这两种**ps**命令格式合并到了单个**ps**命令中（当然，也加入了他们自己的风格）。

Linux系统中使用的GNU `ps`命令支持3种不同类型的命令行参数：

- ❑ Unix风格的参数，前面加单破折线；
- ❑ BSD风格的参数，前面不加破折线；
- ❑ GNU风格的长参数，前面加双破折线。

下面将进一步解析这3种不同的参数类型，并举例演示它们如何工作。

1. Unix风格的参数

Unix风格的参数是从贝尔实验室开发的AT&T Unix系统上原有的**ps**命令继承下来的。这些参数如表4-1所示。

表4-1 Unix风格的ps命令参数

参 数	描 述
-A	显示所有进程
-N	显示与指定参数不符的所有进程
-a	显示除控制进程（ <i>session leader</i> ^① ）和无终端进程外的所有进程
-d	显示除控制进程外的所有进程
-e	显示所有进程
-C <i>cmdlist</i>	显示包含在 <i>cmdlist</i> 列表中的进程
-G <i>grplist</i>	显示组ID在 <i>grplist</i> 列表中的进程
-U <i>userlist</i>	显示属主的用户ID在 <i>userlist</i> 列表中的进程
-g <i>grplist</i>	显示会话或组ID在 <i>grplist</i> 列表中的进程 ^②
-p <i>pidlist</i>	显示PID在 <i>pidlist</i> 列表中的进程

① 关于*session leader*的概念，可参考《Unix环境高级编程（第3版）》第9章的内容。
② 这个在不同的Linux发行版中可能不尽相同，有的发行版中*grplist*代表会话ID，有的发行版中*grplist*代表有效组ID。

(续)

参 数	描 述
-s <i>sesslist</i>	显示会话ID在 <i>sesslist</i> 列表中的进程
-t <i>ttylist</i>	显示终端ID在 <i>ttylist</i> 列表中的进程
-u <i>userlist</i>	显示有效用户ID在 <i>userlist</i> 列表中的进程
-F	显示更多额外输出（相对-f参数而言）
-O <i>format</i>	显示默认的输出列以及 <i>format</i> 列表指定的特定列
-M	显示进程的安全信息
-C	显示进程的额外调度器信息
-f	显示完整格式的输出
-j	显示任务信息
-l	显示长列表
-o <i>format</i>	仅显示由 <i>format</i> 指定的列
-Y	不要显示进程标记（process flag，表明进程状态的标记）
-Z	显示安全标签（security context） ^① 信息
-H	用层级格式来显示进程（树状，用来显示父进程）
-n <i>namelist</i>	定义了WCHAN列显示的值
-w	采用宽输出模式，不限宽度显示
-L	显示进程中的线程
-V	显示ps命令的版本号

上面给出的参数已经很多了，不过还有很多。使用ps命令的关键不在于记住所有可用的参数，而在于记住最有用的那些参数。大多数Linux系统管理员都有自己的一组参数，他们会牢牢记住这些用来提取有用的进程信息的参数。举个例子，如果你想查看系统上运行的所有进程，可用-ef参数组合（ps命令允许你像这样把参数组合在一起）。

```
$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0 11:29 ?           00:00:01 init [5]
root           2        0  0 11:29 ?           00:00:00 [kthreadd]
root           3        2  0 11:29 ?           00:00:00 [migration/0]
root           4        2  0 11:29 ?           00:00:00 [ksoftirqd/0]
root           5        2  0 11:29 ?           00:00:00 [watchdog/0]
root           6        2  0 11:29 ?           00:00:00 [events/0]
root           7        2  0 11:29 ?           00:00:00 [khelper]
root          47        2  0 11:29 ?           00:00:00 [kblockd/0]
root          48        2  0 11:29 ?           00:00:00 [kacpid]
68          2349        1  0 11:30 ?           00:00:00 hald
root        3078      1981  0 12:00 ?           00:00:00 sshd: rich [priv]
rich        3080      3078  0 12:00 ?           00:00:00 sshd: rich@pts/0
rich        3081      3080  0 12:00 pts/0       00:00:00 -bash
rich        4445      3081  3 13:48 pts/0       00:00:00 ps -ef
$
```

^① security context也叫security label，是SELinux采用的声明资源的一种机制。

上例中，我们略去了输出中的不少行，以节约空间。但如你所见，Linux系统上运行着很多进程。这个例子用了两个参数：`-e`参数指定显示所有运行在系统上的进程；`-f`参数则扩展了输出，这些扩展的列包含了有用的信息。

- ❑ **UID**：启动这些进程的用户。
- ❑ **PID**：进程的进程ID。
- ❑ **PPID**：父进程的进程号（如果该进程是由另一个进程启动的）。
- ❑ **C**：进程生命周期中的CPU利用率。
- ❑ **STIME**：进程启动时的系统时间。
- ❑ **TTY**：进程启动时的终端设备。
- ❑ **TIME**：运行进程需要的累计CPU时间。
- ❑ **CMD**：启动的程序名称。

上例中输出了合理数量的信息，这也正是大多数系统管理员希望看到的。如果想要获得更多的信息，可采用`-l`参数，它会产生一个长格式输出。

```
$ ps -l
F S  UID PID  PPID  C PRI  NI ADDR SZ  WCHAN  TTY          TIME CMD
0 S  500 3081  3080  0  80   0 -  1173 wait pts/0    00:00:00 bash
0 R  500 4463  3081  1  80   0 -  1116 -    pts/0    00:00:00 ps
$
```

注意使用了`-l`参数之后多出的那些列。

- ❑ **F**：内核分配给进程的系统标记。
- ❑ **S**：进程的状态（O代表正在运行；S代表在休眠；R代表可运行，正等待运行；Z代表僵化，进程已结束但父进程已不存在；T代表停止）。
- ❑ **PRI**：进程的优先级（越大的数字代表越低的优先级）。
- ❑ **NI**：谦让度值用来参与决定优先级。
- ❑ **ADDR**：进程的内存地址。
- ❑ **SZ**：假如进程被换出，所需交换空间的大致大小。
- ❑ **WCHAN**：进程休眠的内核函数的地址。

2. BSD风格的参数

了解了Unix风格的参数之后，我们来一起看一下BSD风格的参数。伯克利软件发行版（Berkeley software distribution，BSD）是加州大学伯克利分校开发的一个Unix版本。它和AT&T Unix系统有许多细小的不同，这也导致了多年的Unix争论。BSD版的`ps`命令参数如表4-2所示。

表4-2 BSD风格的ps命令参数

参 数	描 述
T	显示跟当前终端关联的所有进程
a	显示跟任意终端关联的所有进程
g	显示所有的进程，包括控制进程
r	仅显示运行中的进程

(续)

参 数	描 述
x	显示所有的进程，甚至包括未分配任何终端的进程
U <i>userlist</i>	显示归 <i>userlist</i> 列表中某用户ID所有的进程
p <i>pidlist</i>	显示PID在 <i>pidlist</i> 列表中的进程
t <i>ttylist</i>	显示所关联的终端在 <i>ttylist</i> 列表中的进程
O <i>format</i>	除了默认输出的列之外，还输出由 <i>format</i> 指定的列
X	按过去的Linux i386寄存器格式显示
Z	将安全信息添加到输出中
j	显示任务信息
l	采用长模式
o <i>format</i>	仅显示由 <i>format</i> 指定的列
s	采用信号格式显示
u	采用基于用户的格式显示
v	采用虚拟内存格式显示
N <i>namelist</i>	定义在WCHAN列中使用的值
O <i>order</i>	定义显示信息列的顺序
S	将数值信息从子进程加到父进程上，比如CPU和内存的使用情况
c	显示真实的命令名称（用以启动进程的程序名称）
e	显示命令使用的环境变量
f	用分层格式来显示进程，表明哪些进程启动了哪些进程
h	不显示头信息
k <i>sort</i>	指定用以将输出排序的列
n	和WCHAN信息一起显示出来，用数值来表示用户ID和组ID
w	为较宽屏幕显示宽输出
H	将线程按进程来显示
m	在进程后显示线程
L	列出所有格式指定符
V	显示ps命令的版本号

如你所见，Unix和BSD类型的参数有很多重叠的地方。使用其中某种类型参数得到的信息也同样可以使用另一种获得。大多数情况下，你只要选择自己所喜欢格式的参数类型就行了（比如你在使用Linux之前就已经习惯BSD环境了）。

在使用BSD参数时，ps命令会自动改变输出以模仿BSD格式。下例是使用l参数的输出：

```
$ ps l
F  UID  PID PPID PRI  NI  VSZ  RSS WCHAN  STAT TTY          TIME COMMAND
0  500  3081 3080  20   0 4692 1432 wait    Ss   pts/0        0:00 -bash
0  500  5104 3081  20   0 4468  844 -        R+   pts/0        0:00 ps l
$
```

注意，其中大部分的输出列跟使用Unix风格参数时的输出是一样的，只有一小部分不同。

- ❑ VSZ: 进程在内存中的大小, 以千字节 (KB) 为单位。
 - ❑ RSS: 进程在未换出时占用的物理内存。
 - ❑ STAT: 代表当前进程状态的双字符状态码。
- 许多系统管理员都喜欢BSD风格的`l`参数。它能输出更详细的进程状态码 (STAT列)。双字符状态码能比Unix风格输出的单字符状态码更清楚地表示进程的当前状态。
- 第一个字符采用了和Unix风格`s`列相同的值, 表明进程是在休眠、运行还是等待。第二个参数进一步说明进程的状态。
- ❑ `<`: 该进程运行在高优先级上。
 - ❑ `N`: 该进程运行在低优先级上。
 - ❑ `L`: 该进程有页面锁定在内存中。
 - ❑ `s`: 该进程是控制进程。
 - ❑ `l`: 该进程是多线程的。
 - ❑ `+`: 该进程运行在前台。

从前面的例子可以看出, `bash`命令处于休眠状态, 但同时它也是一个控制进程 (在我的会话中, 它是主要进程), 而`ps`命令则运行在系统的前台。

3. GNU长参数

最后, GNU开发人员在这个新改进过的`ps`命令中加入了另外一些参数。其中一些GNU长参数复制了现有的Unix或BSD类型的参数, 而另一些则提供了新功能。表4-3列出了现有的GNU长参数。

表4-3 GNU风格的ps命令参数

参 数	描 述
<code>--deselect</code>	显示所有进程, 命令行中列出的进程
<code>--Group grplist</code>	显示组ID在 <code>grplist</code> 列表中的进程
<code>--User userlist</code>	显示用户ID在 <code>userlist</code> 列表中的进程
<code>--group grplist</code>	显示有效组ID在 <code>grplist</code> 列表中的进程
<code>--pid pidlist</code>	显示PID在 <code>pidlist</code> 列表中的进程
<code>--ppid pidlist</code>	显示父PID在 <code>pidlist</code> 列表中的进程
<code>--sid sidlist</code>	显示会话ID在 <code>sidlist</code> 列表中的进程
<code>--tty ttylist</code>	显示终端设备号在 <code>ttylist</code> 列表中的进程
<code>--user userlist</code>	显示有效用户ID在 <code>userlist</code> 列表中的进程
<code>--format format</code>	仅显示由 <code>format</code> 指定的列
<code>--context</code>	显示额外的安全信息
<code>--cols n</code>	将屏幕宽度设置为 <code>n</code> 列
<code>--columns n</code>	将屏幕宽度设置为 <code>n</code> 列
<code>--cumulative</code>	包含已停止的子进程的信息
<code>--forest</code>	用层级结构显示出进程和父进程之间的关系
<code>--headers</code>	在每页输出中都显示列的头
<code>--no-headers</code>	不显示列的头

(续)

参 数	描 述
--lines <i>n</i>	将屏幕高度设为 <i>n</i> 行
--rows <i>n</i>	将屏幕高度设为 <i>n</i> 排
--sort <i>order</i>	指定将输出按哪列排序
--width <i>n</i>	将屏幕宽度设为 <i>n</i> 列
--help	显示帮助信息
--info	显示调试信息
--version	显示ps命令的版本号

可以将GNU长参数和Unix或BSD风格的参数混用来定制输出。GNU长参数中一个着实让人喜爱的功能就是--forest参数。它会显示进程的层级信息，并用ASCII字符绘出可爱的图表。

```
1981 ?          00:00:00 sshd
3078 ?          00:00:00  \_ sshd
3080 ?          00:00:00    \_ sshd
3081 pts/0      00:00:00      \_ bash
16676 pts/0    00:00:00        \_ ps
```

这种格式让跟踪子进程和父进程变得十分容易。

4.1.2 实时监测进程

ps命令虽然在收集运行在系统上的进程信息时非常有用，但也有不足之处：它只能显示某个特定时间点的信息。如果想观察那些频繁换进换出的内存的进程趋势，用ps命令就不方便了。

而top命令刚好适用这种情况。top命令跟ps命令相似，能够显示进程信息，但它是实时显示的。图4-1是top命令运行时输出的截图。

输出的第一部分显示的是系统的概况：第一行显示了当前时间、系统的运行时间、登录的用户数以及系统的平均负载。

平均负载有3个值：最近1分钟的、最近5分钟的和最近15分钟的平均负载。值越大说明系统的负载越高。由于进程短期的突发性活动，出现最近1分钟的高负载值也很常见，但如果近15分钟内的平均负载都很高，就说明系统可能有问题。

说明 Linux系统管理的要点在于定义究竟到什么程度才算是高负载。这个值取决于系统的硬件配置以及系统上通常运行的程序。对某个系统来说是高负载的值可能对另一系统来说就是正常值。通常，如果系统的负载值超过了2，就说明系统比较繁忙了。

第二行显示了进程概要信息——top命令的输出中将进程叫作任务(task)：有多少进程处在运行、休眠、停止或是僵化状态（僵化状态是指进程完成了，但父进程没有响应）。

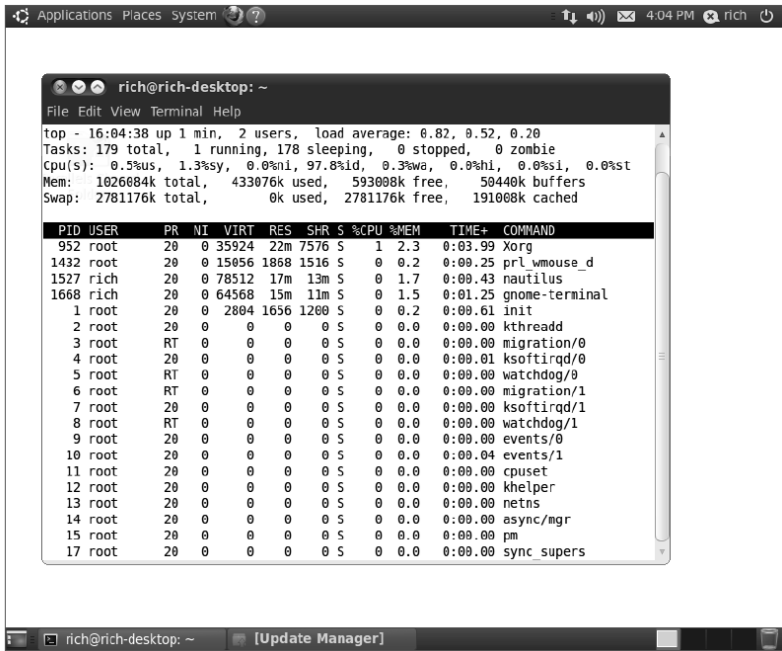


图4-1 top命令运行时的输出

下一行显示了CPU的概要信息。top根据进程的属主（用户还是系统）和进程的状态（运行、空闲还是等待）将CPU利用率分成几类输出。

紧跟其后的两行说明了系统内存的状态。第一行说的是系统的物理内存：总共有多少内存，当前用了多少，还有多少空闲。后一行说的是同样的信息，不过是针对系统交换空间（如果分配了的话）的状态而言的。

最后一部分显示了当前运行中的进程的详细列表，有些列跟ps命令的输出类似。

- ❑ PID：进程的ID。
- ❑ USER：进程属主的名字。
- ❑ PR：进程的优先级。
- ❑ NI：进程的谦让度值。
- ❑ VIRT：进程占用的虚拟内存总量。
- ❑ RES：进程占用的物理内存总量。
- ❑ SHR：进程和其他进程共享的内存总量。
- ❑ S：进程的状态（D代表可中断的休眠状态，R代表在运行状态，S代表休眠状态，T代表跟踪状态或停止状态，Z代表僵化状态）。
- ❑ %CPU：进程使用的CPU时间比例。
- ❑ %MEM：进程使用的内存占可用内存的比例。

- ❑ **TIME+**: 自进程启动到目前为止的CPU时间总量。
- ❑ **COMMAND**: 进程所对应的命令行名称, 也就是启动的程序名。

默认情况下, `top`命令在启动时会按照%CPU值对进程排序。可以在`top`运行时使用多种交互命令重新排序。每个交互式命令都是单字符, 在`top`命令运行时键入可改变`top`的行为。键入`f`允许你选择对输出进行排序的字段, 键入`d`允许你修改轮询间隔。键入`q`可以退出`top`。用户在`top`命令的输出上有很大的控制权。用这个工具就能经常找出占用系统大部分资源的罪魁祸首。当然了, 一旦找到, 下一步就是结束这些进程。这也正是接下来的话题。

4.1.3 结束进程

作为系统管理员, 很重要的一个技能就是知道何时以及如何结束一个进程。有时进程挂起了, 只需要动手让进程重新运行或结束就行了。但有时, 有的进程会耗尽CPU且不释放资源。在这两种情景下, 你就需要能控制进程的命令。Linux沿用了Unix进行进程间通信的方法。

在Linux中, 进程之间通过信号来通信。进程的信号就是预定义好的一个消息, 进程能识别它并决定忽略还是作出反应。进程如何处理信号是由开发人员通过编程来决定的。大多数编写完善的程序都能接收和处理标准Unix进程信号。这些信号都列在了表4-4中。

表4-4 Linux进程信号

信 号	名 称	描 述
1	HUP	挂起
2	INT	中断
3	QUIT	结束运行
9	KILL	无条件终止
11	SEGV	段错误
15	TERM	尽可能终止
17	STOP	无条件停止运行, 但不终止
18	TSTP	停止或暂停, 但继续在后台运行
19	CONT	在STOP或TSTP之后恢复执行

在Linux上有两个命令可以向运行中的进程发出进程信号。

1. **kill**命令

`kill`命令可通过进程ID (PID) 给进程发信号。默认情况下, `kill`命令会向命令行中列出的全部PID发送一个TERM信号。遗憾的是, 你只能用进程的PID而不能用命令名, 所以`kill`命令有时并不好用。

要发送进程信号, 你必须是进程的属主或登录为root用户。

```
$ kill 3940
-bash: kill: (3940) - Operation not permitted
$
```

TERM信号告诉进程可能的话就停止运行。不过, 如果有不服管教的进程, 那它通常会忽略

这个请求。如果要强制终止，`-s`参数支持指定其他信号（用信号名或信号值）。

你能从下例中看出，`kill`命令不会有任何输出。

```
# kill -s HUP 3940
#
```

要检查`kill`命令是否有效，可再运行`ps`或`top`命令，看看问题进程是否已停止。

2. `killall`命令

`killall`命令非常强大，它支持通过进程名而不是PID来结束进程。`killall`命令也支持通配符，这在系统因负载过大而变得很慢时很有用。

```
# killall http*
#
```

上例中的命令结束了所有以`http`开头的进程，比如Apache Web服务器的`httpd`服务。

警告 以`root`用户身份登录系统时，使用`killall`命令要特别小心，因为很容易就会误用通配符而结束了重要的系统进程。这可能会破坏文件系统。

4.2 监测磁盘空间

系统管理员的另一个重要任务就是监测系统磁盘的使用情况。不管运行的是简单的Linux台式机还是大型的Linux服务器，你都要知道还有多少空间可留给你的应用程序。

在Linux系统上有几个命令行命令可以用来帮助管理存储媒体。本节将介绍在日常系统管理中经常用到的核心命令。

4.2.1 挂载存储媒体

如第3章中讨论的，Linux文件系统将所有的磁盘都并入一个虚拟目录下。在使用新的存储媒体之前，需要把它放到虚拟目录下。这项工作称为挂载（`mounting`）。

在今天的图形化桌面环境里，大多数Linux发行版都能自动挂载特定类型的可移动存储媒体。可移动存储媒体指的是可从PC上轻易移除的媒体，比如CD-ROM、软盘和U盘。

如果用的发行版不支持自动挂载和卸载可移动存储媒体，就必须手动完成。本节将介绍一些可以帮你管理可移动存储设备的Linux命令行命令。

1. `mount`命令

Linux上用来挂载媒体的命令叫作`mount`。默认情况下，`mount`命令会输出当前系统上挂载的设备列表。

```
$ mount
/dev/mapper/VolGroup00-LogVol100 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
```

```

/dev/sda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/sdb1 on /media/disk type vfat
(rw,nosuid,nodev,uhelper=hal,shortname=lower,uid=503)
$

```

mount命令提供如下四部分信息：

- ❑ 媒体的设备文件名
- ❑ 媒体挂载到虚拟目录的挂载点
- ❑ 文件系统类型
- ❑ 已挂载媒体的访问状态

上面例子的最后一行输出中，U盘被GNOME桌面自动挂载到了挂载点/media/disk。vfat文件系统类型说明它是在Windows机器上被格式化的。

要手动在虚拟目录中挂载设备，需要以root用户身份登录，或是以root用户身份运行sudo命令。下面是手动挂载媒体设备的基本命令：

```
mount -t type device directory
```

type参数指定了磁盘被格式化的文件系统类型。Linux可以识别非常多的文件系统类型。如果是和Windows PC共用这些存储设备，通常得使用下列文件系统类型。

- ❑ vfat：Windows长文件系统。
- ❑ ntfs：Windows NT、XP、Vista以及Windows 7中广泛使用的高级文件系统。
- ❑ iso9660：标准CD-ROM文件系统。

大多数U盘和软盘会被格式化成vfat文件系统。而数据CD则必须使用iso9660文件系统类型。后面两个参数定义了该存储设备的设备文件的位置以及挂载点在虚拟目录中的位置。比如说，手动将U盘/dev/sdb1挂载到/media/disk，可用下面的命令：

```
mount -t vfat /dev/sdb1 /media/disk
```

媒体设备挂载到了虚拟目录后，root用户就有了对该设备的所有访问权限，而其他用户的访问则会被限制。你可以通过目录权限（将在第7章中介绍）指定用户对设备的访问权限。

如果要用到mount命令的一些高级功能，表4-5中列出了可用的参数。

表4-5 mount命令的参数

参 数	描 述
-a	挂载/etc/fstab文件中指定的所有文件系统
-f	使mount命令模拟挂载设备，但并不真的挂载
-F	和-a参数一起使用时，会同时挂载所有文件系统
-v	详细模式，将会说明挂载设备的每一步
-I	不启用任何/sbin/mount.filesystem下的文件系统帮助文件
-l	给ext2、ext3或XFS文件系统自动添加文件系统标签

(续)

参 数	描 述
-n	挂载设备，但不注册到/etc/mntab已挂载设备文件中
-p <i>num</i>	进行加密挂载时，从文件描述符 <i>num</i> 中获得密码短语
-s	忽略该文件系统不支持的挂载选项
-r	将设备挂载为只读的
-w	将设备挂载为可读写的（默认参数）
-L <i>label</i>	将设备按指定的 <i>label</i> 挂载
-U <i>uuid</i>	将设备按指定的 <i>uuid</i> 挂载
-O	和-a参数一起使用，限制命令只作用到特定的一组文件系统上
-o	给文件系统添加特定的选项

4

-o参数允许在挂载文件系统时添加一些以逗号分隔的额外选项。以下为常用的选项。

- ❑ ro：以只读形式挂载。
- ❑ rw：以读写形式挂载。
- ❑ user：允许普通用户挂载文件系统。
- ❑ check=none：挂载文件系统时不进行完整性校验。
- ❑ loop：挂载一个文件。

2. umount命令

从Linux系统上移除一个可移动设备时，不能直接从系统上移除，而应该先卸载。

窍门 Linux上不能直接弹出已挂载的CD。如果你在从光驱中移除CD时遇到麻烦，通常是因为该CD还挂载在虚拟目录里。先卸载它，然后再去尝试弹出。

卸载设备的命令是umount（是的，你没看错，命令名中并没有字母n，这一点有时候很让人困惑）。umount命令的格式非常简单：

```
umount [directory | device ]
```

umount命令支持通过设备文件或者是挂载点来指定要卸载的设备。如果有任何程序正在使用设备上的文件，系统就不会允许你卸载它：

```
[root@testbox mnt]# umount /home/rich/mnt
umount: /home/rich/mnt: device is busy
umount: /home/rich/mnt: device is busy
[root@testbox mnt]# cd /home/rich
[root@testbox rich]# umount /home/rich/mnt
[root@testbox rich]# ls -l mnt
total 0
[root@testbox rich]#
```

上例中，命令行提示符仍然在挂载设备的文件系统目录中，所以umount命令无法卸载该

镜像文件。一旦命令提示符移出该镜像文件的文件系统，umount命令就能卸载该镜像文件。^①

4.2.2 使用 df 命令

有时你需要知道在某个设备上还有多少磁盘空间。df命令可以让你很方便地查看所有已挂载磁盘的使用情况。

```
$ df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/sda2              18251068    7703964   9605024   45% /
/dev/sda1              101086      18680     77187    20% /boot
tmpfs                 119536        0    119536     0% /dev/shm
/dev/sdb1             127462      113892     13570    90% /media/disk
$
```

df命令会显示每个有数据的已挂载文件系统。如你在前例中看到的，有些已挂载设备仅限系统内部使用。命令输出如下：

- ❑ 设备的设备文件位置；
- ❑ 能容纳多少个1024字节大小的块；
- ❑ 已用了多少个1024字节大小的块；
- ❑ 还有多少个1024字节大小的块可用；
- ❑ 已用空间所占的比例；
- ❑ 设备挂载到了哪个挂载点上。

df命令有一些命令行参数可用，但基本上不会用到。一个常用的参数是-h。它会把输出中的磁盘空间按照用户易读的形式显示，通常用M来替代兆字节，用G替代吉字节。

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb2       18G   7.4G   9.2G   45% /
/dev/sda1       99M   19M    76M   20% /boot
tmpfs           117M    0   117M    0% /dev/shm
/dev/sdb1      125M  112M   14M   90% /media/disk
$
```

说明 Linux系统后台一直有进程来处理文件或使用文件。df命令的输出值显示的是Linux系统认为的当前值。有可能系统上有运行的进程已经创建或删除了某个文件，但尚未释放文件。这个值是不会算进闲置空间的。

① 如果在卸载设备时，系统提示设备繁忙，无法卸载设备，通常是有进程还在访问该设备或使用该设备上的文件。这时可用lsof命令获得使用它的进程信息，然后在应用中停止使用该设备或停止该进程。lsof命令的用法很简单：lsof /path/to/device/node，或者lsof /path/to/mount/point。

4.2.3 使用 du 命令

通过df命令很容易发现哪个磁盘的存储空间快没了。系统管理员面临的下一个问题是，发生这种情况时要怎么办。

另一个有用的命令是du命令。du命令可以显示某个特定目录（默认情况下是当前目录）的磁盘使用情况。这一方法可用来快速判断系统上某个目录下是不是有超大文件。

默认情况下，du命令会显示当前目录下所有的文件、目录和子目录的磁盘使用情况，它会以磁盘块为单位来表明每个文件或目录占用了多大存储空间。对标准大小的目录来说，这个输出会是一个比较长的列表。下面是du命令的部分输出：

```
$ du
484    ./gstreamer-0.10
8      ./Templates
8      ./Download
8      ./ccache/7/0
24     ./ccache/7
368    ./ccache/a/d
384    ./ccache/a
424    ./ccache
8      ./Public
8      ./gphpedit/plugins
32     ./gphpedit
72     ./gconfd
128    ./nautilus/metafiles
384    ./nautilus
72     ./bittorrent/data/metainfo
20     ./bittorrent/data/resume
144    ./bittorrent/data
152    ./bittorrent
8      ./Videos
8      ./Music
16     ./config/gtk-2.0
40     ./config
8      ./Documents
```

每行输出左边的数值是每个文件或目录占用的磁盘块数。注意，这个列表是从目录层级的最底部开始，然后按文件、子目录、目录逐级向上。

这么用du命令（不加参数，用默认参数）作用并不大。我们更想知道每个文件和目录占用了多大的磁盘空间，但如果还得逐页查找的话就没什么意义了。

下面是能让du命令用起来更方便的几个命令行参数。

- ❑ -c：显示所有已列出文件总的大小。
- ❑ -h：按用户易读的格式输出大小，即用K替代千字节，用M替代兆字节，用G替代吉字节。
- ❑ -s：显示每个输出参数的总计。

系统管理员接下来就是要使用一些文件处理命令操作大批量的数据。这正是下一节的主题。

4.3 处理数据文件

当你有大量数据时，通常很难处理这些信息及提取有用信息。正如在上节中学习的`du`命令，系统命令很容易输出过量的信息。

Linux系统提供了一些命令行工具来处理大量数据。本节将会介绍一些每个系统管理员以及日常Linux用户都应该知道的基本命令，这些命令能够让生活变得更加轻松。

4.3.1 排序数据

处理大量数据时的一个常用命令是`sort`命令。顾名思义，`sort`命令是对数据进行排序的。默认情况下，`sort`命令按照会话指定的默认语言的排序规则对文本文件中的数据行排序。

```
$ cat file1
one
two
three
four
five
$ sort file1
five
four
one
three
two
$
```

这相当简单。但事情并非总像看起来那样容易。看下面的例子。

```
$ cat file2
1
2
100
45
3
10
145
75
$ sort file2
1
10
100
145
2
3
45
75
$
```

如果你本期望这些数字能按值排序，就要失望了。默认情况下，`sort`命令会把数字当做字符来执行标准的字符排序，产生的输出可能根本就不是你要的。解决这个问题可用`-n`参数，它会

告诉sort命令把数字识别成数字而不是字符，并且按值排序。

```
$ sort -n file2
1
2
3
10
45
75
100
145
$
```

现在好多了！另一个常用的参数是-M，按月排序。Linux的日志文件经常会在每行的起始位置有一个时间戳，用来表明事件是什么时候发生的。

```
Sep 13 07:10:09 testbox smartd[2718]: Device: /dev/sda, opened
```

如果将含有时间戳日期的文件按默认的排序方法来排序，会得到类似于下面的结果。

```
$ sort file3
Apr
Aug
Dec
Feb
Jan
Jul
Jun
Mar
May
Nov
Oct
Sep
$
```

这并不是想要的结果。如果用-M参数，sort命令就能识别三字符的月份名，并相应地排序。

```
$ sort -M file3
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
$
```

还有其他一些方便的sort参数可用，如表4-6所示。

表4-6 sort命令参数

单破折线	双破折线	描 述
-b	--ignore-leading-blanks	排序时忽略起始的空白
-C	--check=quiet	不排序，如果数据无序也不要报告
-c	--check	不排序，但检查输入数据是不是已排序；未排序的话，报告
-d	--dictionary-order	仅考虑空白和字母，不考虑特殊字符
-f	--ignore-case	默认情况下，会将大写字母排在前面；这个参数会忽略大小写
-g	--general-number-sort	按通用数值来排序（跟-n不同，把值当浮点数来排序，支持科学计数法表示的值）
-i	--ignore-nonprinting	在排序时忽略不可打印字符
-k	--key=POS1[,POS2]	排序从POS1位置开始；如果指定了POS2的话，到POS2位置结束
-M	--month-sort	用三字符月份名按月份排序
-m	--merge	将两个已排序数据文件合并
-n	--numeric-sort	按字符串数值来排序（并不转换为浮点数）
-o	--output=file	将排序结果写出到指定的文件中
-R	--random-sort	按随机生成的散列表的键值排序
	--random-source=FILE	指定-R参数用到的随机字节的源文件
-r	--reverse	反序排序（升序变成降序）
-S	--buffer-size=SIZE	指定使用的内存大小
-s	--stable	禁用最后重排序比较
-T	--temporary-directory=DIR	指定一个位置来存储临时工作文件
-t	--field-separator=SEP	指定一个用来区分键位置的字符
-u	--unique	和-c参数一起使用时，检查严格排序；不和-c参数一起用时，仅输出第一例相似的两行
-z	--zero-terminated	用NULL字符作为行尾，而不是用换行符

-k和-t参数在对按字段分隔的数据进行排序时非常有用，例如/etc/passwd文件。可以用-t参数来指定字段分隔符，然后用-k参数来指定排序的字段。举个例子，要对前面提到的密码文件/etc/passwd根据用户ID进行数值排序，可以这么做：

```
$ sort -t ':' -k 3 -n /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```



```
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
```

现在数据已经按第三个字段——用户ID的数值排序。

`-n`参数在排序数值时非常有用，比如`du`命令的输出。

```
$ du -sh * | sort -nr
1008k  mrtg-2.9.29.tar.gz
972k   bldg1
888k   fbs2.pdf
760k   Printtest
680k   rsync-2.6.6.tar.gz
660k   code
516k   fig1001.tiff
496k   test
496k   php-common-4.0.4pl1-6mdk.i586.rpm
448k   MesaGLUT-6.5.1.tar.gz
400k   plp
```

注意，`-r`参数将结果按降序输出，这样就更容易看到目录下的哪些文件占用空间最多。

说明 本例中用到的管道命令（`|`）将`du`命令的输出重定向到`sort`命令。我们将在第11章中进一步讨论。

4.3.2 搜索数据

你会经常需要在大文件中找一行数据，而这行数据又埋藏在文件的中间。这时并不需要手动翻看整个文件，用`grep`命令来帮助查找就行了。`grep`命令的命令行格式如下。

```
grep [options] pattern [file]
```

`grep`命令会在输入或指定的文件中查找包含匹配指定模式的字符的行。`grep`的输出就是包含了匹配模式的行。

下面两个简单的例子演示了使用`grep`命令来对4.3.1节中用到的文件`file1`进行搜索。

```
$ grep three file1
three
$ grep t file1
two
three
$
```

第一个例子在文件`file1`中搜索能匹配模式`three`的文本。`grep`命令输出了匹配了该模式的行。第二个例子在文件`file1`中搜索能匹配模式`t`的文本。这个例子里，`file1`中有两行匹配了指定的模式，两行都输出了。

由于`grep`命令非常流行，它经历了大量的更新。有很多功能被加进了`grep`命令。如果查看一下它的手册页面，你会发现它是多么的无所不能。

如果要进行反向搜索（输出不匹配该模式的行），可加`-v`参数。

```
$ grep -v t file1
one
four
five
$
```

如果要显示匹配模式的行所在的行号，可加`-n`参数。

```
$ grep -n t file1
2:two
3:three
$
```

如果只要知道有多少行含有匹配的模式，可用`-c`参数。

```
$ grep -c t file1
2
$
```

如果要指定多个匹配模式，可用`-e`参数来指定每个模式。

```
$ grep -e t -e f file1
two
three
four
five
$
```

这个例子输出了含有字符`t`或字符`f`的所有行。

默认情况下，`grep`命令用基本的Unix风格正则表达式来匹配模式。Unix风格正则表达式采用特殊字符来定义怎样查找匹配的模式。

要想进一步了解正则表达式的细节，可以参考第20章的内容。

以下是在`grep`搜索中使用正则表达式的简单例子。

```
$ grep [tf] file1
two
three
four
five
$
```

正则表达式中的方括号表明`grep`应该搜索包含`t`或者`f`字符的匹配。如果不用正则表达式，`grep`就会搜索匹配字符串`tf`的文本。

`egrep`命令是`grep`的一个衍生，支持POSIX扩展正则表达式。POSIX扩展正则表达式含有更多的可以用来指定匹配模式的字符（参见第20章）。`fgrep`则是另外一个版本，支持将匹配模式指定为用换行符分隔的一列固定长度的字符串。这样就可以把这列字符串放到一个文件中，然后在`fgrep`命令中用其在一个大型文件中搜索字符串了。

4.3.3 压缩数据

如果你接触过Microsoft Windows，就必然用过zip文件。它如此流行，以至于微软从Windows XP开始，就已经将其集成进了自家的操作系统中。zip工具可以将大型文件（文本文件和可执行文件）压缩成占用更少空间的小文件。

Linux包含了多种文件压缩工具。虽然听上去不错，但这实际上经常会在用户下载文件时造成混淆。表4-7列出了Linux上的文件压缩工具。

表4-7 Linux文件压缩工具

工 具	文件扩展名	描 述
bzip2	.bz2	采用Burrows-Wheeler块排序文本压缩算法和霍夫曼编码
compress	.Z	最初的Unix文件压缩工具，已经快没人用了
gzip	.gz	GNU压缩工具，用Lempel-Ziv编码
zip	.zip	Windows上PKZIP工具的Unix实现

compress文件压缩工具已经很少在Linux系统上看到了。如果下载了带.Z扩展名的文件，通常可以用第9章中介绍的软件包安装方法来安装compress包（在很多Linux发行版上叫作ncompress），然后再用uncompress命令来解压文件。gzip是Linux上最流行的压缩工具。

gzip软件包是GNU项目的产物，意在编写一个能够替代原先Unix中compress工具的免费版本。这个软件包包含有下面的工具。

- ❑ gzip：用来压缩文件。
- ❑ gzcat：用来查看压缩过的文本文件的内容。
- ❑ gunzip：用来解压文件。

这些工具基本上跟bzip2工具的用法一样。

```
$ gzip myprog
$ ls -l my*
-rwxrwxr-x 1 rich rich 2197 2007-09-13 11:29 myprog.gz
$
```

gzip命令会压缩你在命令行指定的文件。也可以在命令行指定多个文件名甚至用通配符来一次性批量压缩文件。

```
$ gzip my*
$ ls -l my*
-rwxr--r-- 1 rich rich 103 Sep 6 13:43 myprog.c.gz
-rwxr-xr-x 1 rich rich 5178 Sep 6 13:43 myprog.gz
-rwxr--r-- 1 rich rich 59 Sep 6 13:46 myscript.gz
-rwxr--r-- 1 rich rich 60 Sep 6 13:44 myscript2.gz
$
```

gzip命令会压缩该目录中匹配通配符的每个文件。

4.3.4 归档数据

虽然zip命令能够很好地将数据压缩和归档进单个文件,但它不是Unix和Linux中的标准归档工具。目前, Unix和Linux上最广泛使用的归档工具是tar命令。

tar命令最开始是用来将文件写到磁带设备上归档的, 然而它也能把输出写到文件里, 这种用法在Linux上已经普遍用来归档数据了。

下面是tar命令的格式:

```
tar function [options] object1 object2 ...
```

function参数定义了tar命令应该做什么, 如表4-8所示。

表4-8 tar命令的功能

功 能	长 名 称	描 述
-A	--concatenate	将一个已有tar归档文件追加到另一个已有tar归档文件
-c	--create	创建一个新的tar归档文件
-d	--diff	检查归档文件和文件系统的不同之处
	--delete	从已有tar归档文件中删除
-r	--append	追加文件到已有tar归档文件末尾
-t	--list	列出已有tar归档文件的内容
-u	--update	将比tar归档文件中已有的同名文件新的文件追加到该tar归档文件中
-x	--extract	从已有tar归档文件中提取文件

每个功能可用选项来针对tar归档文件定义一个特定行为。表4-9列出了这些选项中能 and tar命令一起使用的常见选项。

表4-9 tar命令选项

选 项	描 述
-C <i>dir</i>	切换到指定目录
-f <i>file</i>	输出结果到文件或设备 <i>file</i>
-j	将输出重定向给bzip2命令来压缩内容
-p	保留所有文件权限
-v	在处理文件时显示文件
-z	将输出重定向给gzip命令来压缩内容

这些选项经常合并到一起使用。首先, 你可以用下列命令来创建一个归档文件:

```
tar -cvf test.tar test/ test2/
```

上面的命令创建了名为test.tar的归档文件, 含有test和test2目录内容。接着, 用下列命令:

```
tar -tf test.tar
```

列出tar文件test.tar的内容(但并不提取文件)。最后, 用命令:

```
tar -xvf test.tar
```

通过这一命令从tar文件test.tar中提取内容。如果tar文件是从一个目录结构创建的，那整个目录结构都会在当前目录下重新创建。

如你所见，tar命令是给整个目录结构创建归档文件的简便方法。这是Linux中分发开源程序源码文件所采用的普遍方法。

窍门 下载了开源软件之后，你会经常看到文件名以.tgz结尾。这些是gzip压缩过的tar文件可以用命令tar -zxvf filename.tgz来解压。

4.4 小结

本章讨论了Linux系统管理员和程序员用到的一些高级bash命令。ps和top命令在判断系统的状态时特别重要，能看到哪些应用在运行以及它们消耗了多少资源。

在可移动存储普及的今天，系统管理员常谈到的另一个话题就是挂载存储设备。mount命令可以将一个物理存储设备挂载到Linux虚拟目录结构上。umount命令用来移除设备。

最后，本章讨论了各种处理数据的工具。sort工具能轻松地对大数据文件进行排序，便于组织数据；grep实用工具能快速检索大数据文件来查找特定信息。Linux上有一些不同的文件压缩工具，包括bzip2、gzip和zip。每种工具都能够压缩大型文件来节省文件系统空间。tar工具能将整个目录都归档到单个文件中，方便把数据迁移到另外一个系统上。

下一章将讨论各种Linux shell及其使用。Linux允许你在多个shell之间进行通信，这一点在脚本中创建子shell时非常有用。

本章内容

- ❑ 探究shell的类型
- ❑ 理解shell的父/子关系
- ❑ 别出心裁的子shell用法
- ❑ 探究内建的shell命令

现在你已经学到了一些shell的基础知识，例如如何进入shell以及初级的shell命令，是时候去一探shell进程的究竟了。要想理解shell，得先理解一些CLI。

shell不单单是一种CLI。它是一个时刻都在运行的复杂交互式程序。输入命令并利用shell来运行脚本会出现一些既有趣又令人困惑的问题。搞清楚shell进程以及它与系统之间的关系能够帮助你解决这些难题，或是完全避开它们。

本章将会带你全面学习shell进程。你会了解到如何创建子shell以及父shell与子shell之间的关系。探究各种用于创建子进程的命令和内建命令。另外还有一些shell的窍门和技巧等你一试。

5.1 shell 的类型

系统启动什么样的shell程序取决于你个人的用户ID配置。在/etc/passwd文件中，在用户ID记录的第7个字段中列出了默认的shell程序。只要用户登录到某个虚拟控制台终端或是在GUI中启动终端仿真器，默认的shell程序就会开始运行。

在下面的例子中，用户christine使用GNU bash shell作为自己的默认shell程序：

```
$ cat /etc/passwd
[...]  
Christine:x:501:501:Christine B:/home/Christine:/bin/bash  
$
```

bash shell程序位于/bin目录内。从长列表中可以看出/bin/bash (bash shell)是一个可执行程序：

```
$ ls -lF /bin/bash  
-rwxr-xr-x. 1 root root 938832 Jul 18 2013 /bin/bash*  
$
```

本书所使用的CentOS发行版中还有其他一些shell程序。其中包括tcsh，它源自最初的C shell：

```
$ ls -lF /bin/tcsh
-rwxr-xr-x. 1 root root 387328 Feb 21 2013 /bin/tcsh*
$
```

另外还包括ash shell的Debian版：

```
$ ls -lF /bin/dash
-rwxr-xr-x. 1 root root 109672 Oct 17 2012 /bin/dash*
$
```

最后，C shell的软链接（参见第3章）指向的是tcsh shell：

```
$ ls -lF /bin/csh
lrwxrwxrwx. 1 root root 4 Mar 18 15:16 /bin/csh -> tcsh*
$
```

这些shell程序各自都可以被设置成用户的默认shell。不过由于bash shell的广为流行，很少有人使用其他的shell作为默认shell。

5

说明 第1章对各种shell有一个简单的描述。如果你想进一步学习GNU bash shell之外的shell，第23章提供了更多的相关信息。

默认的交互shell会在用户登录某个虚拟控制台终端或在GUI中运行终端仿真器时启动。不过还有另外一个默认shell是/bin/sh，它作为默认的系统shell，用于那些需要在启动时使用的系统shell脚本。

你经常会看到某些发行版使用软链接将默认的系统shell设置成bash shell，如本书所使用的CentOS发行版：

```
$ ls -l /bin/sh
lrwxrwxrwx. 1 root root 4 Mar 18 15:05 /bin/sh -> bash
$
```

但要注意的是在有些发行版上，默认的系统shell和默认的交互shell并不相同，例如在Ubuntu发行版中：

```
$ cat /etc/passwd
[...]
christine:x:1000:1000:Christine,,,:/home/christine:/bin/bash
$
$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Apr 22 12:33 /bin/sh -> dash
$
```

注意，用户christine默认的交互shell是/bin/bash，也就是bash shell。但是作为默认系统shell的/bin/sh被设置为dash shell。

窍门 对bash shell脚本来说,这两种不同的shell(默认的交互shell和默认的系统shell)会造成问题。一定要阅读第11章中有关bash shell脚本首行的语法要求,以避免这些麻烦。

并不是必须一直使用默认的交互shell。可以使用发行版中所有可用的shell,只需要输入对应的文件名就行了。例如,你可以直接输入命令/bin/dash来启动dash shell。

```
$ /bin/dash
$
```

除启动了dash shell程序之外,看起来似乎什么都没有发生。提示符\$是dash shell的CLI提示符。可以输入exit来退出dash shell。

```
$ exit
exit
$
```

这一次好像还是什么都没有发生,但是dash shell程序已经退出了。为了理解这个过程,我们将在下一节中探究登录shell程序与新启动的shell程序之间的关系。

5.2 shell 的父子关系

用于登录某个虚拟控制器终端或在GUI中运行终端仿真器时所启动的默认的交互shell,是一个父shell。本书到目前为止都是父shell提供CLI提示符,然后等待命令输入。

在CLI提示符后输入/bin/bash命令或其他等效的bash命令时,会创建一个新的shell程序。这个shell程序被称为子shell(child shell)。子shell也拥有CLI提示符,同样会等待命令输入。

当输入bash、生成子shell的时候,你是看不到任何相关的信息的,因此需要另一条命令帮助我们理清这一切。第4章中讲过的ps命令能够派上用场,在生成子shell的前后配合选项-f来使用。

```
$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
501          1841   1840  0 11:50 pts/0        00:00:00 -bash
501          2429   1841  4 13:44 pts/0        00:00:00 ps -f
$
$ bash
$
$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
501          1841   1840  0 11:50 pts/0        00:00:00 -bash
501          2430   1841  0 13:44 pts/0        00:00:00 bash
501          2444   2430  1 13:44 pts/0        00:00:00 ps -f
$
```

第一次使用ps -f的时候,显示出了两个进程。其中一个进程的进程ID是1841(第二列),运行的是bash shell程序(最后一列)。另一个进程(进程ID为2429)对应的是命令ps -f。

说明 进程就是正在运行的程序。bash shell是一个程序，当它运行的时候，就成为了一个进程。一个运行着的shell就是某种进程而已。因此，在说到运行一个bash shell的时候，你经常会看到“shell”和“进程”这两个词交换使用。

输入命令bash之后，一个子shell就出现了。第二个ps -f是在子shell中执行的。可以从显示结果中看到有两个bash shell程序在运行。第一个bash shell程序，也就是父shell进程，其原始进程ID是1814。第二个bash shell程序，即子shell进程，其PID是2430。注意，子shell的父进程ID(PPID)是1841，指明了这个父shell进程就是该子shell的父进程。图5-1展示了这种关系。

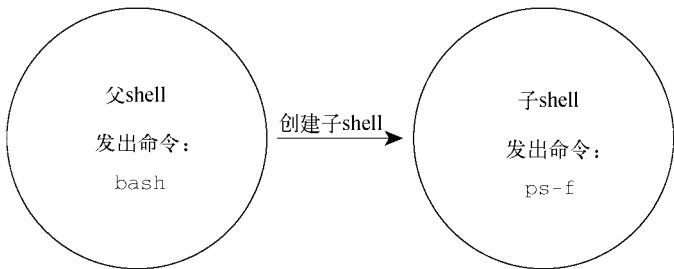


图5-1 bash shell进程的父亲关系

在生成子shell进程时，只有部分父进程的环境被复制到子shell环境中。这会对包括变量在内的一些东西造成影响，我们会在第6章中谈及相关的内容。

子shell（child shell，也叫subshell）可以从父shell中创建，也可以从另一个子shell中创建。

```
$ ps -f
UID          PID  PPID  C  STIME TTY          TIME CMD
501          1841  1840  0  11:50 pts/0        00:00:00 -bash
501          2532  1841  1  14:22 pts/0        00:00:00 ps -f
$
$ bash
$
$ bash
$
$ bash
$
$ ps --forest
  PID TTY          TIME CMD
 1841 pts/0        00:00:00 bash
 2533 pts/0        00:00:00 \_ bash
 2546 pts/0        00:00:00 \_ bash
 2562 pts/0        00:00:00 \_ bash
 2576 pts/0        00:00:00 \_ ps
$
```

在上面的例子中，bash命令被输入了三次。这实际上创建了三个子shell。ps -forest命令展示了这些子shell间的嵌套结构。图5-2中也展示了这种关系。

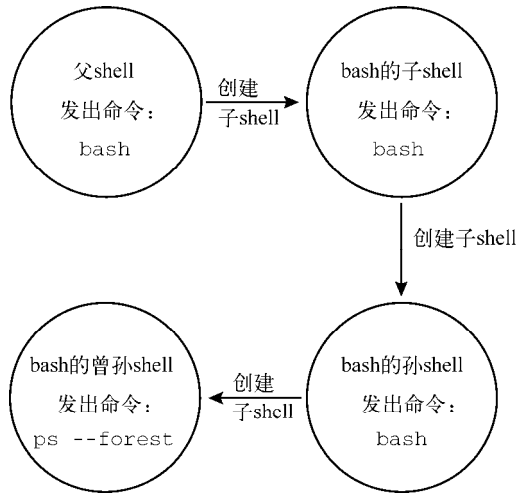


图5-2 子shell的嵌套关系

`ps -f`命令也能够表现子shell的嵌套关系，因为它能够通过PPID列显示出谁是谁的父进程。

```
$ ps -f
UID      PID  PPID  C  STIME TTY          TIME CMD
501      1841  1840  0  11:50 pts/0        00:00:00 -bash
501      2533  1841  0  14:22 pts/0        00:00:00 bash
501      2546  2533  0  14:22 pts/0        00:00:00 bash
501      2562  2546  0  14:24 pts/0        00:00:00 bash
501      2585  2562  1  14:29 pts/0        00:00:00 ps -f
$
```

`bash` shell程序可使用命令行参数修改shell启动方式。表5-1列举了`bash`中可用的命令行参数。

表5-1 bash命令行参数

参 数	描 述
-c string	从string中读取命令并进行处理
-i	启动一个能够接收用户输入的交互shell
-l	以登录shell的形式启动
-r	启动一个受限shell，用户会被限制在默认目录中
-s	从标准输入中读取命令

可以输入`man bash`获得关于`bash`命令的更多帮助信息，了解更多的命令行参数。`bash --help`命令也会提供一些额外的协助。

可以利用`exit`命令有条不紊地退出子shell。

```
$ exit
exit
$
$ ps --forest
```

```

    PID TTY          TIME CMD
    1841 pts/0        00:00:00 bash
    2533 pts/0        00:00:00  \_  bash
    2546 pts/0        00:00:00      \_  bash
    2602 pts/0        00:00:00        \_  ps
$
$ exit
exit
$
$ exit
exit
$
$ ps --forest
    PID TTY          TIME CMD
    1841 pts/0        00:00:00 bash
    2604 pts/0        00:00:00  \_  ps
$

```

exit命令不仅能退出子shell，还能用来登出当前的虚拟控制台终端或终端仿真器软件。只需要在父shell中输入exit，就能够从容退出CLI了。

运行shell脚本也能够创建出子shell。在第11章，你将会学习到相关话题的更多知识。

就算是不使用bash shell命令或是运行shell脚本，你也可以生成子shell。一种方法就是使用进程列表。

5.2.1 进程列表

你可以在一行中指定要依次运行的一系列命令。这可以通过命令列表来实现，只需要在命令之间加入分号(;)即可。

```

$ pwd ; ls ; cd /etc ; pwd ; cd ; pwd ; ls
/home/Christine
Desktop  Downloads  Music      Public     Videos
Documents junk.dat   Pictures   Templates
/etc
/home/Christine
Desktop  Downloads  Music      Public     Videos
Documents junk.dat   Pictures   Templates
$

```

在上面的例子中，所有的命令依次执行，不存在任何问题。不过这并不是进程列表。命令列表要想成为进程列表，这些命令必须包含在括号里。

```

$ (pwd ; ls ; cd /etc ; pwd ; cd ; pwd ; ls)
/home/Christine
Desktop  Downloads  Music      Public     Videos
Documents junk.dat   Pictures   Templates
/etc
/home/Christine
Desktop  Downloads  Music      Public     Videos
Documents junk.dat   Pictures   Templates
$

```

尽管多出来的括号看起来没有什么太大的不同，但起到的效果确是非同寻常。括号的加入使命令列表变成了进程列表，生成了一个子shell来执行对应的命令。

说明 进程列表是一种命令分组 (command grouping)。另一种命令分组是将命令放入花括号中，并在命令列表尾部加上分号 (;)。语法为 { command; }。使用花括号进行命令分组并不会像进程列表那样创建出子shell。

要想知道是否生成了子shell，得借助一个使用了环境变量的命令。(环境变量会在第6章中详述。) 这个命令就是 `echo $BASH_SUBSHELL`。如果该命令返回0，就表明没有子shell。如果返回1或者其他更大的数字，就表明存在子shell。

下面的例子中使用了一个命令列表，列表尾部是 `echo $BASH_SUBSHELL`。

```
$ pwd ; ls ; cd /etc ; pwd ; cd ; pwd ; ls ; echo $BASH_SUBSHELL
/home/Christine
Desktop    Downloads Music      Public     Videos
Documents  junk.dat  Pictures  Templates
/etc
/home/Christine
Desktop    Downloads Music      Public     Videos
Documents  junk.dat  Pictures  Templates
0
```

在命令输出的最后，显示的是数字0。这就表明这些命令不是在子shell中运行的。

要是使用进程列表的话，结果就不一样了。在列表最后加入 `echo $BASH_SUBSHELL`。

```
$ (pwd ; ls ; cd /etc ; pwd ; cd ; pwd ; ls ; echo $BASH_SUBSHELL)
/home/Christine
Desktop    Downloads Music      Public     Videos
Documents  junk.dat  Pictures  Templates
/etc
/home/Christine
Desktop    Downloads Music      Public     Videos
Documents  junk.dat  Pictures  Templates
1
```

这次在命令输入的最后显示出了数字1。这表明的确创建了子shell，并用于执行这些命令。

所以说，命令列表就是使用括号包围起来的一组命令，它能够创建出子shell来执行这些命令。你甚至可以在命令列表中嵌套括号来创建子shell的子shell。

```
$ ( pwd ; echo $BASH_SUBSHELL)
/home/Christine
1
$ ( pwd ; (echo $BASH_SUBSHELL))
/home/Christine
2
```

注意，在第一个进程列表中，数字1表明了一个子shell，这个结果和预期的一样。但是在第二个进程列表中，在命令 `echo $BASH_SUBSHELL` 外面又多出了一对括号。这对括号在子shell中

产生了另一个子shell来执行命令。因此数字2表明的就是这个子shell。

在shell脚本中，经常使用子shell进行多进程处理。但是采用子shell的成本不菲，会明显拖慢处理速度。在交互式的CLI shell会话中，子shell同样存在问题。它并非真正的多进程处理，因为终端控制着子shell的I/O。

5.2.2 别出心裁的子 shell 用法

在交互式的shell CLI中，还有很多更富有成效的子shell用法。进程列表、协程和管道（第11章会讲到）都利用了子shell。它们都可以有效地在交互式shell中使用。

在交互式shell中，一个高效的子shell用法就是使用后台模式。在讨论如果将后台模式与子shell搭配使用之前，你得先搞明白什么是后台模式。

1. 探索后台模式

在后台模式中运行命令可以在处理命令的同时让出CLI，以供他用。演示后台模式的一个经典命令就是sleep。

sleep命令接受一个参数，该参数是你希望进程等待（睡眠）的秒数。这个命令在脚本中常用于引入一段时间的暂停。命令sleep 10会将会话暂停10秒钟，然后返回shell CLI提示符。

```
$ sleep 10
$
```

要想将命令置入后台模式，可以在命令末尾加上字符&。把sleep命令置入后台模式可以让我们利用ps命令来小窥一番。

```
$ sleep 3000&
[1] 2396
$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
christi+    2338    2337  0 10:13 pts/9        00:00:00 -bash
christi+    2396    2338  0 10:17 pts/9        00:00:00 sleep 3000
christi+    2397    2338  0 10:17 pts/9        00:00:00 ps -f
$
```

sleep命令会在后台（&）睡眠3000秒（50分钟）。当它被置入后台，在shell CLI提示符返回之前，会出现两条信息。第一条信息是显示在方括号中的后台作业（background job）号（1）。第二条是后台作业的进程ID（2396）。

ps命令用来显示各种进程。我们可以注意到命令sleep 3000已经被列出来了。在第二列显示的进程ID（PID）和命令进入后台时所显示的PID是一样的，都是2396。

除了ps命令，你也可以使用jobs命令来显示后台作业信息。jobs命令可以显示出当前运行在后台模式中的所有用户的进程（作业）。

```
$ jobs
[1]+  Running                  sleep 3000 &
$
```

jobs命令在方括号中显示出作业号（1）。它还显示了作业的当前状态（running）以及对

应的命令 (`sleep 3000 &`)。

利用 `jobs` 命令的 `-l` (字母 `L` 的小写形式) 选项, 你还能够看到更多的相关信息。除了默认信息之外, `-l` 选项还能够显示出命令的 `PID`。

```
$ jobs -l
[1]+  2396 Running                  sleep 3000 &
$
```

一旦后台作业完成, 就会显示出结束状态。

```
[1]+  Done                          sleep 3000 &
$
```

窍门 需要提醒的是: 后台作业的结束状态可未必会一直等待到合适的时候才现身。当作业结束状态突然出现在屏幕上的时候, 你可别吃惊啊。

后台模式非常方便, 它可以让我们在 `CLI` 中创建出有实用价值的子 `shell`。

2. 将进程列表置入后台

之前说过, 进程列表是运行在子 `shell` 中的一条或多条命令。使用包含了 `sleep` 命令的进程列表, 并显示出变量 `BASH_SUBSHELL`, 结果和期望的一样。

```
$ (sleep 2 ; echo $BASH_SUBSHELL ; sleep 2)
1
$
```

在上面的例子中, 有一个2秒钟的暂停, 显示出的数字1表明只有一个子 `shell`, 在返回提示符之前又经历了另一个2秒钟的暂停。没什么大事。

将相同的进程列表置入后台模式会在命令输出上表现出些许不同。

```
$ (sleep 2 ; echo $BASH_SUBSHELL ; sleep 2)&
[2] 2401
$ 1

[2]+  Done                          ( sleep 2; echo $BASH_SUBSHELL; sleep 2 )
$
```

把进程列表置入后台会产生一个作业号和进程 `ID`, 然后返回到提示符。不过奇怪的是表明单一级子 `shell` 的数字1显示在了提示符的旁边! 不要不知所措, 只需要按一下回车键, 就会得到另一个提示符。

在 `CLI` 中运用子 `shell` 的创造性方法之一就是进程列表置入后台模式。你既可以在子 `shell` 中进行繁重的处理工作, 同时也不会让子 `shell` 的 `I/O` 受制于终端。

当然了, `sleep` 和 `echo` 命令的进程列表只是作为一个示例而已。使用 `tar` (参见第4章) 创建备份文件是有效利用后台进程列表的一个更实用的例子。

```
$ (tar -cf Rich.tar /home/rich ; tar -cf My.tar /home/christine)&
[3] 2423
$
```

将进程列表置入后台模式并不是子shell在CLI中仅有的创造性用法。协程就是另一种方法。

3. 协程

协程可以同时做两件事。它在后台生成一个子shell，并在这个子shell中执行命令。

要进行协程处理，得使用coproc命令，还有要在子shell中执行的命令。

```
$ coproc sleep 10
[1] 2544
$
```

除了会创建子shell之外，协程基本上就是将命令置入后台模式。当输入coproc命令及其参数之后，你会发现启用了后台作业。屏幕上会显示出后台作业号（1）以及进程ID（2544）。jobs命令能够显示出协程的处理状态。

```
$ jobs
[1]+  Running                  coproc COPROC sleep 10 &
$
```

在上面的例子中可以看到在子shell中执行的后台命令是coproc COPROC sleep 10。COPROC是coproc命令给进程起的名字。你可以使用命令的扩展语法自己设置这个名字。

```
$ coproc My_Job { sleep 10; }
[1] 2570
$
$ jobs
[1]+  Running                  coproc My_Job { sleep 10; } &
$
```

通过使用扩展语法，协程的名字被设置成My_Job。这里要注意的是，扩展语法写起来有点麻烦。必须确保在第一个花括号（{）和命令名之间有一个空格。还必须保证命令以分号（;）结尾。另外，分号和闭花括号（}）之间也得有一个空格。

说明 协程能够让你尽情发挥想象力，发送或接收来自子shell中进程的信息。只有在拥有多个协程的时候才需要对协程进行命名，因为你得和它们进行通信。否则的话，让coproc命令将其设置成默认的名字COPROC就行了。

你可以发挥才智，将协程与进程列表结合起来产生嵌套的子shell。只需要输入进程列表，然后把命令coproc放在前面就行了。

```
$ coproc ( sleep 10; sleep 2 )
[1] 2574
$
$ jobs
[1]+  Running                  coproc COPROC ( sleep 10; sleep 2 ) &
$
$ ps --forest
  PID TTY          TIME CMD
 2483 pts/12    00:00:00 bash
 2574 pts/12    00:00:00 \_ bash
 2575 pts/12    00:00:00 |  \_ sleep
```

```
2576 pts/12    00:00:00  \_ ps
$
```

记住，生成子shell的成本不低，而且速度还慢。创建嵌套子shell更是火上浇油！

在命令行中使用子shell能够获得灵活性和便利。要想获得这些优势，重要的是理解子shell的行为方式。对于命令也是如此。在下一节中，我们将研究内建命令与外部命令之间的行为差异。

5.3 理解 shell 的内建命令

在学习GNU bash shell期间，你可能听到过“内建命令”这个术语。搞明白shell的内建命令和非内建（外部）命令非常重要。内建命令和非内建命令的操作方式大不相同。

5.3.1 外部命令

外部命令，有时候也被称为文件系统命令，是存在于bash shell之外的程序。它们并不是shell程序的一部分。外部命令程序通常位于/bin、/usr/bin、/sbin或/usr/sbin中。

ps就是一个外部命令。你可以使用which和type命令找到它。

```
$ which ps
/bin/ps
$
$ type -a ps
ps is /bin/ps
$
$ ls -l /bin/ps
-rwxr-xr-x 1 root root 93232 Jan  6 18:32 /bin/ps
$
```

当外部命令执行时，会创建出一个子进程。这种操作被称为衍生（forking）。外部命令ps很方便显示出它的父进程以及自己所对应的衍生子进程。

```
$ ps -f
UID          PID  PPID  C  STIME TTY          TIME CMD
christi+  2743   2742  0  17:09 pts/9        00:00:00 -bash
christi+  2801   2743  0  17:16 pts/9        00:00:00 ps -f
$
```

作为外部命令，ps命令执行时会创建出一个子进程。在这里，ps命令的PID是2801，父PID是2743。作为父进程的bash shell的PID是2743。图5-3展示了外部命令执行时的衍生过程。

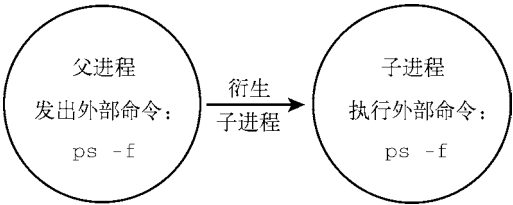


图5-3 外部命令的衍生

当进程必须执行衍生操作时，它需要花费时间和精力来设置新子进程的环境。所以说，外部命令多少还是有代价的。

说明 就算衍生出子进程或是创建了子shell，你仍然可以通过发送信号与其沟通，这一点无论是在命令行还是在脚本编写中都是极其有用的。发送信号（signaling）使得进程间可以通过信号进行通信。信号及其发送会在第16章中讲到。

5.3.2 内建命令

内建命令和外部命令的区别在于前者不需要使用子进程来执行。它们已经和shell编译成了一体，作为shell工具的组成部分存在。不需要借助外部程序文件来运行。

`cd`和`exit`命令都内建于bash shell。可以利用`type`命令来了解某个命令是否是内建的。

```
$ type cd
cd is a shell builtin
$
$ type exit
exit is a shell builtin
$
```

因为既不需要通过衍生出子进程来执行，也不需要打开程序文件，内建命令的执行速度要更快，效率也更高。附录A给出了GNU bash shell的内建命令列表。

要注意，有些命令有多种实现。例如`echo`和`pwd`既有内建命令也有外部命令。两种实现略有不同。要查看命令的不同实现，使用`type`命令的`-a`选项。

```
$ type -a echo
echo is a shell builtin
echo is /bin/echo
$
$ which echo
/bin/echo
$
$ type -a pwd
pwd is a shell builtin
pwd is /bin/pwd
$
$ which pwd
/bin/pwd
$
```

命令`type -a`显示出了每个命令的两种实现。注意，`which`命令只显示出了外部命令文件。

窍门 对于有多种实现的命令，如果想要使用其外部命令实现，直接指明对应的文件就可以了。例如，要使用外部命令`pwd`，可以输入`/bin/pwd`。

1. 使用history命令

一个有用的内建命令是history命令。bash shell会跟踪你用过的命令。你可以唤回这些命令并重新使用。

要查看最近用过的命令列表，可以输入不带选项的history命令。

```
$ history
 1 ps -f
 2 pwd
 3 ls
 4 coproc ( sleep 10; sleep 2 )
 5 jobs
 6 ps --forest
 7 ls
 8 ps -f
 9 pwd
10 ls -l /bin/ps
11 history
12 cd /etc
13 pwd
14 ls
15 cd
16 type pwd
17 which pwd
18 type echo
19 which echo
20 type -a pwd
21 type -a echo
22 pwd
23 history
```

在这个例子中，只显示了最近的23条命令。通常历史记录中会保存最近的1000条命令。这个数量可是不少的！

窍门 你可以设置保存在bash历史记录中的命令数。要想实现这一点，你需要修改名为HISTSIZE的环境变量（参见第6章）。

你可以唤回并重用历史列表中最近的命令。这样能够节省时间和击键量。输入!!，然后按回车键就能够唤出刚刚用过的那条命令来使用。

```
$ ps --forest
  PID TTY          TIME CMD
 2089 pts/0        00:00:00 bash
 2744 pts/0        00:00:00 \_ ps
$
$ !!
ps --forest
  PID TTY          TIME CMD
 2089 pts/0        00:00:00 bash
 2745 pts/0        00:00:00 \_ ps
$
```

当输入`!!`时, `bash`首先会显示出从shell的历史记录中唤回的命令。然后执行该命令。

命令历史记录被保存在隐藏文件`.bash_history`中, 它位于用户的主目录中。这里要注意的是, `bash`命令的历史记录是先存放在内存中, 当shell退出时才被写入到历史文件中。

```
$ history
[...]
```

```
25 ps --forest
26 history
27 ps --forest
28 history
```

```
$
$ cat .bash_history
pwd
ls
history
exit
$
```

注意, 当`history`命令运行时, 列出了28条命令。出于简洁性的考虑, 上面的例子中只摘取了一部分列表内容。但是文件`.bash_history`的内容被显示出来时, 其中只有4条命令, 与`history`命令的输出并不匹配。

可以在退出shell会话之前强制将命令历史记录写入`.bash_history`文件。要实现强制写入, 需要使用`history`命令的`-a`选项。

```
$ history -a
$
$ history
[...]
```

```
25 ps --forest
26 history
27 ps --forest
28 history
29 ls -a
30 cat .bash_history
31 history -a
32 history
```

```
$
$ cat .bash_history
[...]
```

```
ps --forest
history
ps --forest
history
ls -a
cat .bash_history
history -a
```

由于两处输出内容都太长, 因此都做了删减。注意, `history`命令和`.bash_history`文件的输入是一样的, 除了最近的那条`history`命令, 因为它是在`history -a`命令之后出现的。

说明 如果你打开了多个终端会话，仍然可以使用`history -a`命令在打开的会话中向`.bash_history`文件中添加记录。但是对于其他打开的终端会话，历史记录并不会自动更新。这是因为`.bash_history`文件只有在打开首个终端会话时才会被读取。要想强制重新读取`.bash_history`文件，更新终端会话的历史记录，可以使用`history -n`命令。

你可以唤回历史列表中任意一条命令。只需输入惊叹号和命令在历史列表中的编号即可。

```
$ history
[...]
```

13	pwd
14	ls
15	cd
16	type pwd
17	which pwd
18	type echo
19	which echo
20	type -a pwd
21	type -a echo

```
[...]
```

32	history -a
33	history
34	cat .bash_history
35	history

```
$
$ !20
type -a pwd
pwd is a shell builtin
pwd is /bin/pwd
$
```

编号为20的命令从命令历史记录中被取出。和执行最近的命令一样，`bash shell`首先显示出从`shell`历史记录中唤回的命令，然后执行该命令。

使用`bash shell`命令历史记录能够大大地节省时间。利用内建的`history`命令能够做到的事情远不止这里所描述的。可以通过输入`man history`来查看`history`命令的`bash`手册页面。

2. 命令别名

`alias`命令是另一个`shell`的内建命令。命令别名允许你为常用的命令（及其参数）创建另一个名称，从而将输入量减少到最低。

你所使用的Linux发行版很有可能已经为你设置好了一些常用命令的别名。要查看当前可用的别名，使用`alias`命令以及选项`-p`。

```
$ alias -p
[...]
```

alias	egrep='egrep --color=auto'
alias	fgrep='fgrep --color=auto'
alias	grep='grep --color=auto'
alias	l='ls -CF'
alias	la='ls -A'

```
alias ll='ls -alF'
alias ls='ls --color=auto'
$
```

注意，在该Ubuntu Linux发行版中，有一个别名取代了标准命令ls。它自动加入了--color选项，表明终端支持彩色模式的列表。

可以使用alias命令创建属于自己的别名。

```
$ alias li='ls -li'
$
$ li
total 36
529581 drwxr-xr-x. 2 Christine Christine 4096 May 19 18:17 Desktop
529585 drwxr-xr-x. 2 Christine Christine 4096 Apr 25 16:59 Documents
529582 drwxr-xr-x. 2 Christine Christine 4096 Apr 25 16:59 Downloads
529586 drwxr-xr-x. 2 Christine Christine 4096 Apr 25 16:59 Music
529587 drwxr-xr-x. 2 Christine Christine 4096 Apr 25 16:59 Pictures
529584 drwxr-xr-x. 2 Christine Christine 4096 Apr 25 16:59 Public
529583 drwxr-xr-x. 2 Christine Christine 4096 Apr 25 16:59 Templates
532891 -rwxr--r--. 1 Christine Christine 36 May 30 07:21 test.sh
529588 drwxr-xr-x. 2 Christine Christine 4096 Apr 25 16:59 Videos
$
```

在定义好别名之后，你随时都可以在shell中使用它，就算在shell脚本中也没问题。要注意，因为命令别名属于内部命令，一个别名仅在它所被定义的shell进程中才有效。

```
$ alias li='ls -li'
$
$ bash
$
$ li
bash: li: command not found
$
$ exit
exit
$
```

不过好在有办法能够让别名在不同的子shell中都奏效。下一章中就会讲到具体的做法，另外还会介绍环境变量。

5.4 小结

本章讨论了复杂的交互式程序：GNU bash shell。其中包括理解shell进程及其关系，如何生成子shell，以及子shell与父shell的关系。还探究了那些能够创建子进程的命令和不能创建子进程的命令。

当用户登录终端的时候，通常会启动一个默认的交互式shell。系统究竟启动哪个shell，这取决于用户ID配置。一般这个shell都是/bin/bash。默认的系统shell (/bin/sh) 用于系统shell脚本，如那些需要在系统启动时运行的脚本。

子shell可以利用bash命令来生成。当使用进程列表或coproc命令时也会产生子shell。将子shell运用在命令行中使得我们能够创造性地高效使用CLI。子shell还可以嵌套，生成子shell的子shell，子shell的子shell的子shell。创建子shell的代价可不低，因为还必须为子shell创建出一个全新的环境。

在最后，我们学习了两种不同类型的命令：内建命令和外部命令。外部命令会创建出一个包含全新环境的子进程，而内建命令则不会。相比之下，外部命令的使用成本更高。内建命令因为不需要创建新环境，所以更高效，不会受到环境变化的影响。

shell、子shell、进程和衍生进程都会受到环境变量的影响。下一章，我们会探究环境变量的影响方式以及如何在不同的上下文中使用环境变量。

本章内容

- ❑ 什么是环境变量
- ❑ 创建自己的局部变量
- ❑ 删除环境变量
- ❑ 默认shell环境变量
- ❑ 设置PATH环境变量
- ❑ 定位环境文件
- ❑ 数组变量

Linux环境变量能帮你提升Linux shell体验。很多程序和脚本都通过环境变量来获取系统信息、存储临时数据和配置信息。在Linux系统上有很多地方可以设置环境变量，了解去哪里设置相应的环境变量很重要。

本章将带你逐步了解Linux环境变量：它们存储在哪里，怎样使用，以及怎样创建自己的环境变量。最后以数组变量的用法作结。

6.1 什么是环境变量

bash shell用一个叫作环境变量（environment variable）的特性来存储有关shell会话和工作环境的信息（这也是它们被称作环境变量的原因）。这项特性允许你在内存中存储数据，以便程序或shell中运行的脚本能够轻松访问到它们。这也是存储持久数据的一种简便方法。

在bash shell中，环境变量分为两类：

- ❑ 全局变量
- ❑ 局部变量

本节将描述以上环境变量，并演示怎么查看和使用它们。

说明 尽管bash shell使用一致的专有环境变量，但不同的Linux发行版经常会添加其自有的环境变量。你在本章中看到的环境变量的例子可能会跟你安装的发行版中看到的结果略微不同。如果遇到本书未讲到的环境变量，可以查看你的Linux发行版上的文档。

6.1.1 全局环境变量

全局环境变量对于shell会话和所有生成的子shell都是可见的。局部变量则只对创建它们的shell可见。这让全局环境变量对那些所创建的子shell需要获取父shell信息的程序来说非常有用。

Linux系统在你开始bash会话时就设置了一些全局环境变量（如想了解此时设置了哪些变量，请参见6.6节）。系统环境变量基本上都是使用全大写字母，以区别于普通用户的环境变量。

要查看全局变量，可以使用env或printenv命令。

```
$ printenv
HOSTNAME=server01.class.edu
SELINUX_ROLE_REQUESTED=
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
[...]
HOME=/home/Christine
LOGNAME=Christine
[...]
G_BROKEN_FILENAMES=1
_=/usr/bin/printenv
```

系统为bash shell设置的全局环境变量数目众多，我们不得不在展示的时候进行删减。其中有很多是在登录过程中设置的，另外，你的登录方式也会影响到所设置的环境变量。

要显示个别环境变量的值，可以使用printenv命令，但是不要用env命令。

```
$ printenv HOME
/home/Christine
$
$ env HOME
env: HOME: No such file or directory
$
```

也可以使用echo显示变量的值。在这种情况下引用某个环境变量的时候，必须在变量前面加上一个美元符（\$）。

```
$ echo $HOME
/home/Christine
$
```

在echo命令中，在变量名前加上\$可不仅仅是要显示变量当前的值。它能够让变量作为命令行参数。

```
$ ls $HOME
```



```
Desktop    Downloads Music      Public    test.sh
Documents  junk.dat  Pictures  Templates Videos
$
$ ls /home/Christine
Desktop    Downloads Music      Public    test.sh
Documents  junk.dat  Pictures  Templates Videos
$
```

正如前面提到的，全局环境变量可用于进程的所有子shell。

```
$ bash
$
$ ps -f
UID      PID  PPID  C  STIME TTY          TIME CMD
501      2017  2016  0  16:00 pts/0        00:00:00 -bash
501      2082  2017  0  16:08 pts/0        00:00:00 bash
501      2095  2082  0  16:08 pts/0        00:00:00 ps -f
$
$ echo $HOME
/home/Christine
$
$ exit
exit
$
```

在这个例子中，用bash命令生成一个子shell后，显示了HOME环境变量的当前值，这个值和父shell中的一模一样，都是/home/Christine。

6.1.2 局部环境变量

顾名思义，局部环境变量只能在定义它们的进程中可见。尽管它们是局部的，但是和全局环境变量一样重要。事实上，Linux系统也默认定义了标准的局部环境变量。不过你也可以定义自己的局部变量，如你所想，这些变量被称为用户定义局部变量。

查看局部环境变量的列表有点复杂。遗憾的是，在Linux系统并没有一个只显示局部环境变量的命令。set命令会显示为某个特定进程设置的所有环境变量，包括局部变量、全局变量以及用户定义变量。

```
$ set
BASH=/bin/bash
[...]
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
[...]
colors=/etc/DIR_COLORS
my_variable='Hello World'
[...]
$
```

可以看到,所有通过`printenv`命令能看到的全局环境变量都出现在了`set`命令的输出中。但在`set`命令的输出中还有其他一些环境变量,即局部环境变量和用户定义变量。

说明 命令`env`、`printenv`和`set`之间的差异很细微。`set`命令会显示出全局变量、局部变量以及用户定义变量。它还会按照字母顺序对结果进行排序。`env`和`printenv`命令同`set`命令的区别在于前两个命令不会对变量排序,也不会输出局部变量和用户定义变量。在这种情况下,`env`和`printenv`的输出是重复的。不过`env`命令有一个`printenv`没有的功能,这使得它要更有用一些。

6.2 设置用户定义变量

可以在`bash shell`中直接设置自己的变量。本节将介绍怎样在交互式`shell`或`shell`脚本程序中创建自己的变量并引用它们。

6.2.1 设置局部用户定义变量

一旦启动了`bash shell`(或者执行一个`shell`脚本),就能创建在这个`shell`进程内可见的局部变量了。可以通过等号给环境变量赋值,值可以是数值或字符串。

```
$ echo $my_variable

$ my_variable=Hello
$
$ echo $my_variable
Hello
```

非常简单!现在每次引用`my_variable`环境变量的值,只要通过`$my_variable`引用即可。如果要给变量赋一个含有空格的字符串值,必须用单引号来界定字符串的首和尾。

```
$ my_variable=Hello World
-bash: World: command not found
$
$ my_variable="Hello World"
$
$ echo $my_variable
Hello World
$
```

没有单引号的话,`bash shell`会以为下一个词是另一个要执行的命令。注意,你定义的局部环境变量用的是小写字母,而到目前为止你所看到的系统环境变量都是大写字母。

窍门 所有的环境变量名均使用大写字母,这是`bash shell`的标准惯例。如果是你自己创建的局部变量或是`shell`脚本,请使用小写字母。变量名区分大小写。在涉及用户定义的局部变量时坚持使用小写字母,这能够避免重新定义系统环境变量可能带来的灾难。

记住，变量名、等号和值之间没有空格，这一点非常重要。如果在赋值表达式中加上了空格，bash shell就会把值当成一个单独的命令：

```
$ my_variable = "Hello World"
-bash: my_variable: command not found
$
```

设置了局部环境变量后，就能在shell进程的任何地方使用它了。但是，如果生成了另外一个shell，它在子shell中就不可用。

```
$ my_variable="Hello World"
$
$ bash
$
$ echo $my_variable

$ exit
exit
$
$ echo $my_variable
Hello World
$
```

在这个例子中生成了一个子shell。在子shell中无法使用用户定义变量my_variable。通过命令echo \$my_variable所返回的空行就能够证明这一点。当你退出子shell并回到原来的shell时，这个局部环境变量依然可用。

类似地，如果你在子进程中设置了一个局部变量，那么一旦你退出了子进程，那个局部环境变量就不可用。

```
$ echo $my_child_variable

$ bash
$
$ my_child_variable="Hello Little World"
$
$ echo $my_child_variable
Hello Little World
$
$ exit
exit
$
$ echo $my_child_variable

$
```

当我们回到父shell时，子shell中设置的局部变量就不存在了。可以通过将局部的用户定义变量变成全局变量来改变这种情况。

6.2.2 设置全局环境变量

在设定全局环境变量的进程所创建的子进程中，该变量都是可见的。创建全局环境变量的方

法是先创建一个局部环境变量，然后再把它导出到全局环境中。

这个过程通过`export`命令来完成，变量名前面不需要加`$`。

```
$ my_variable="I am Global now"
$
$ export my_variable
$
$ echo $my_variable
I am Global now
$
$ bash
$
$ echo $my_variable
I am Global now
$
$ exit
exit
$
$ echo $my_variable
I am Global now
$
```

在定义并导出局部环境变量`my_variable`后，`bash`命令启动了一个子shell。在这个子shell中能够正确的显示出变量`my_variable`的值。该变量能够保留住它的值是因为`export`命令使其变成了全局环境变量。

修改子shell中全局环境变量并不会影响到父shell中该变量的值。

```
$ my_variable="I am Global now"
$ export my_variable
$
$ echo $my_variable
I am Global now
$
$ bash
$
$ echo $my_variable
I am Global now
$
$ my_variable="Null"
$
$ echo $my_variable
Null
$
$ exit
exit
$
$ echo $my_variable
I am Global now
$
```

在定义并导出变量`my_variable`后，`bash`命令启动了一个子shell。在这个子shell中能够正确显示出全局环境变量`my_variable`的值。子shell随后改变了这个变量的值。但是这种改变仅在

子shell中有效，并不会被反映到父shell中。

子shell甚至无法使用`export`命令改变父shell中全局环境变量的值。

```
$ my_variable="I am Global now"
$ export my_variable
$
$ echo $my_variable
I am Global now
$
$ bash
$
$ echo $my_variable
I am Global now
$
$ my_variable="Null"
$
$ export my_variable
$
$ echo $my_variable
Null
$
$ exit
exit
$
$ echo $my_variable
I am Global now
$
```

尽管子shell重新定义并导出了变量`my_variable`，但父shell中的`my_variable`变量依然保留着原先的值。

6.3 删除环境变量

当然，既然可以创建新的环境变量，自然也能删除已经存在的环境变量。可以用`unset`命令完成这个操作。在`unset`命令中引用环境变量时，记住不要使用`$`。

```
$ echo $my_variable
I am Global now
$
$ unset my_variable
$
$ echo $my_variable
$
```

窍门 在涉及环境变量名时，什么时候该使用`$`，什么时候不该使用`$`，实在让人摸不着头脑。记住一点就行了：如果要用到变量，使用`$`；如果要操作变量，不使用`$`。这条规则的一个例外就是使用`printenv`显示某个变量的值。

在处理全局环境变量时,事情就有点棘手了。如果你是在子进程中删除了一个全局环境变量,这只对子进程有效。该全局环境变量在父进程中依然可用。

```
$ my_variable="I am Global now"
$
$ export my_variable
$
$ echo $my_variable
I am Global now
$
$ bash
$
$ echo $my_variable
I am Global now
$
$ unset my_variable
$
$ echo $my_variable

$ exit
exit
$
$ echo $my_variable
I am Global now
$
```

和修改变量一样,在子shell中删除全局变量后,你无法将效果反映到父shell中。

6.4 默认的 shell 环境变量

默认情况下, bash shell 会用一些特定的环境变量来定义系统环境。这些变量在你的Linux系统上都已经设置好了,只管放心使用。bash shell源自当初的Unix Bourne shell,因此也保留了Unix Bourne shell里定义的那些环境变量。

表6-1列出了bash shell提供的与Unix Bourne shell兼容的环境变量。

表6-1 bash shell支持的Bourne变量

变 量	描 述
CDPATH	冒号分隔的目录列表,作为cd命令的搜索路径
HOME	当前用户的主目录
IFS	shell用来将文本字符串分割成字段的一系列字符
MAIL	当前用户收件箱的文件名 (bash shell会检查这个文件,看看有没有新邮件)
MAILPATH	冒号分隔的当前用户收件箱的文件名列表 (bash shell会检查列表中的每个文件,看看有没有新邮件)
OPTARG	getopts命令处理的最后一个选项参数值
OPTIND	getopts命令处理的最后一个选项参数的索引号
PATH	shell查找命令的目录列表,由冒号分隔
PS1	shell命令行界面的主提示符
PS2	shell命令行界面的次提示符

除了默认的Bourne的环境变量，bash shell还提供一些自有的变量，如表6-2所示。

表6-2 bash shell环境变量

变 量	描 述
BASH	当前shell实例的全路径名
BASH_ALIASES	含有当前已设置别名的关联数组
BASH_ARGC	含有传入子函数或shell脚本的参数总数的数组变量
BASH_ARCV	含有传入子函数或shell脚本的参数的数组变量
BASH_CMDS	关联数组，包含shell执行过的命令的所在位置
BASH_COMMAND	shell正在执行的命令或马上就执行的命令
BASH_ENV	设置了的话，每个bash脚本会在运行前先尝试运行该变量定义的启动文件
BASH_EXECUTION_STRING	使用bash -c选项传递过来的命令
BASH_LINENO	含有当前执行的shell函数的源代码行号的数组变量
BASH_REMATCH	只读数组，在使用正则表达式的比较运算符=~进行肯定匹配（positive match）时，包含了匹配到的模式和子模式
BASH_SOURCE	含有当前正在执行的shell函数所在源文件名的数组变量
BASH_SUBSHELL	当前子shell环境的嵌套级别（初始值是0）
BASH_VERSINFO	含有当前运行的bash shell的主版本号和次版本号的数组变量
BASH_VERSION	当前运行的bash shell的版本号
BASH_XTRACEFD	若设置成了有效的文件描述符（0、1、2），则'set -x'调试选项生成的跟踪输出可被重定向。通常用来将跟踪输出到一个文件中
BASHOPTS	当前启用的bash shell选项的列表
BASHPID	当前bash进程的PID
COLUMNS	当前bash shell实例所用终端的宽度
COMP_CWORD	COMP_WORDS变量的索引值，后者含有当前光标的位置
COMP_LINE	当前命令行
COMP_POINT	当前光标位置相对于当前命令起始的索引
COMP_KEY	用来调用shell函数补全功能的最后一个键
COMP_TYPE	一个整数值，表示所尝试的补全类型，用以完成shell函数补全
COMP_WORDBREAKS	Readline库中用于单词补全的词分隔字符
COMP_WORDS	含有当前命令行所有单词的数组变量
COMPREPLY	含有由shell函数生成的可能填充代码的数组变量
COPROC	占用未命名的协进程的I/O文件描述符的数组变量
DIRSTACK	含有目录栈当前内容的数组变量
EMACS	设置为't'时，表明emacs shell缓冲区正在工作，而行编辑功能被禁止
ENV	如果设置了该环境变量，在bash shell脚本运行之前会先执行已定义的启动文件（仅用于当bash shell以POSIX模式被调用时）
EUID	当前用户的有效用户ID（数字形式）
FCEDIT	供fc命令使用的默认编辑器
FIGIGNORE	在进行文件名补全时可以忽略后缀名列表，由冒号分隔
FUNCNAME	当前执行的shell函数的名称

(续)

变 量	描 述
FUNCNEST	当设置成非零值时,表示所允许的最大函数嵌套级数(一旦超出,当前命令即被终止)
GLOBIGNORE	冒号分隔的模式列表,定义了在进行文件名扩展时可以忽略的一组文件名
GROUPS	含有当前用户属组列表的数组变量
histchars	控制历史记录扩展,最多可有3个字符
HISTCMD	当前命令在历史记录中的编号
HISTCONTROL	控制哪些命令留在历史记录列表中
HISTFILE	保存shell历史记录列表的文件名(默认是.bash_history)
HISTFILESIZE	最多在历史文件中存多少行
HISTTIMEFORMAT	如果设置了且非空,就用作格式化字符串,以显示bash历史中每条命令的时间戳
HISTIGNORE	由冒号分隔的模式列表,用来决定历史文件中哪些命令会被忽略
HISTSIZE	最多在历史文件中存多少条命令
HOSTFILE	shell在补全主机名时读取的文件名称
HOSTNAME	当前主机的名称
HOSTTYPE	当前运行bash shell的机器
IGNOREEOF	shell在退出前必须收到连续的EOF字符的数量(如果这个值不存在,默认是1)
INPUTRC	Readline初始化文件名(默认是.inputrc)
LANG	shell的语言环境类别
LC_ALL	定义了一个语言环境类别,能够覆盖LANG变量
LC_COLLATE	设置对字符串排序时用的排序规则
LC_CTYPE	决定如何解释出现在文件名扩展和模式匹配中的字符
LC_MESSAGES	在解释前面带有\$的双引号字符串时,该环境变量决定了所采用的语言环境设置
LC_NUMERIC	决定着格式化数字时采用的语言环境设置
LINENO	当前执行的脚本的行号
LINES	定义了终端上可见的行数
MACHTYPE	用“CPU-公司-系统”(CPU-company-system)格式定义的系统类型
MAPFILE	一个数组变量,当mapfile命令未指定数组变量作为参数时,它存储了mapfile所读入的文本
MAILCHECK	shell查看新邮件的频率(以秒为单位,默认值是60)
OLDPWD	shell之前的工作目录
OPTERR	设置为1时,bash shell会显示getopts命令产生的错误
OSTYPE	定义了shell所在的操作系统
PIPESTATUS	含有前台进程的退出状态列表的数组变量
POSIXLY_CORRECT	设置的话,bash会以POSIX模式启动
PPID	bash shell父进程的PID
PROMPT_COMMAND	设置的话,在命令行主提示符显示之前会执行这条命令
PROMPT_DIRTRIM	用来定义当启用了\w或\W提示符字符串转义时显示的尾部目录名的数量。被删除的目录名会用一组英文句点替换
PS3	select命令的提示符

(续)

变 量	描 述
PS4	如果使用了bash的-x选项，在命令行之前显示的提示信息
PWD	当前工作目录
RANDOM	返回一个0~32767的随机数（对其的赋值可作为随机数生成器的种子）
READLINE_LINE	当使用bind -x命令时，存储Readline缓冲区的内容
READLINE_POINT	当使用bind -x命令时，表示Readline缓冲区内容插入点的当前位置
REPLY	read命令的默认变量
SECONDS	自从shell启动到现在的秒数（对其赋值将会重置计数器）
SHELL	bash shell的全路径名
SHELLOPTS	已启用bash shell选项列表，列表项之间以冒号分隔
SHLVL	shell的层级；每次启动一个新bash shell，该值增加1
TIMEFORMAT	指定了shell的时间显示格式
TMOUT	select和read命令在没输入的情况下等待多久（以秒为单位）。默认值为0，表示无限长
TMPDIR	目录名，保存bash shell创建的临时文件
UID	当前用户的真实用户ID（数字形式）

6

你可能已经注意到，不是所有的默认环境变量都会在运行set命令时列出。尽管这些都是默认环境变量，但并不是每一个都必须有一个值。

6.5 设置 PATH 环境变量

当你在shell命令行界面中输入一个外部命令时（参见第5章），shell必须搜索系统来找到对应的程序。PATH环境变量定义了用于进行命令和程序查找的目录。在本书所用的Ubuntu系统中，PATH环境变量的内容是这样的：

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games:/usr/local/games
$
```

输出中显示了有8个可供shell用来查找命令和程序。PATH中的目录使用冒号分隔。

如果命令或者程序的位置没有包括在PATH变量中，那么如果不使用绝对路径的话，shell是没法找到的。如果shell找不到指定的命令或程序，它会产生一个错误信息：

```
$ myprog
-bash: myprog: command not found
$
```

问题是，应用程序放置可执行文件的目录常常不在PATH环境变量所包含的目录中。解决的办法是保证PATH环境变量包含了所有存放应用程序的目录。

可以把新的搜索目录添加到现有的PATH环境变量中，无需从头定义。PATH中各个目录之间

是用冒号分隔的。你只需引用原来的PATH值，然后再给这个字符串添加新目录就行了。可以参考下面的例子。

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
/sbin:/bin:/usr/games:/usr/local/games
$
$ PATH=$PATH:/home/christine/Scripts
$
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games:/usr/local/games:/home/christine/Scripts
$
$ myprog
The factorial of 5 is 120.
$
```

将目录加到PATH环境变量之后，你现在就可以在虚拟目录结构中的任何位置执行程序。

```
$ cd /etc
$
$ myprog
The factorial of 5 is 120
$
```

窍门 如果希望子shell也能找到你的程序的位置，一定要记得把修改后的PATH环境变量导出。

程序员通常的办法是将单点符也加入PATH环境变量。该单点符代表当前目录（参见第3章）。

```
$ PATH=$PATH:.
$
$ cd /home/christine/Old_Scripts
$
$ myprog2
The factorial of 6 is 720
$
```

对PATH变量的修改只能持续到退出或重启系统。这种效果并不能一直持续。在下一节中，你会学到如何永久保持环境变量的修改效果。

6.6 定位系统环境变量

环境变量在Linux系统中的用途很多。你现在已经知道如何修改系统环境变量，也知道了如何创建自己的环境变量。接下来的问题是怎样让环境变量的作用持久化。

在你登入Linux系统启动一个bash shell时，默认情况下bash会在几个文件中查找命令。这些文件叫作启动文件或环境文件。bash检查的启动文件取决于你启动bash shell的方式。启动bash shell有3种方式：

- ❑ 登录时作为默认登录shell

- ❑ 作为非登录shell的交互式shell
- ❑ 作为运行脚本的非交互shell

下面几节介绍了bash shell在不同的方式下启动文件。

6.6.1 登录 shell

当你登录Linux系统时，bash shell会作为登录shell启动。登录shell会从5个不同的启动文件里读取命令：

- ❑ /etc/profile
- ❑ \$HOME/.bash_profile
- ❑ \$HOME/.bashrc
- ❑ \$HOME/.bash_login
- ❑ \$HOME/.profile

/etc/profile文件是系统上默认的bash shell的主启动文件。系统上的每个用户登录时都会执行这个启动文件。

6

说明 要留意的是有些Linux发行版使用了可拆卸式认证模块（Pluggable Authentication Modules，PAM）。在这种情况下，PAM文件会在bash shell启动之前处理，这些文件中可能会包含环境变量。PAM文件包括/etc/environment文件和\$HOME/.pam_environment文件。PAM更多的相关信息可以在<http://linux-pam.org>中找到。

另外4个启动文件是针对用户的，可根据个人需求定制。我们来仔细看一下各个文件。

1. /etc/profile文件

/etc/profile文件是bash shell默认的主启动文件。只要你登录了Linux系统，bash就会执行/etc/profile启动文件中的命令。不同的Linux发行版在这个文件里放了不同的命令。在本书所用的Ubuntu Linux系统上，它看起来是这样的：

```
$ cat /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
```

```
        PS1='$ '
    fi
fi
fi

# The default umask is now handled by pam_umask.
# See pam_umask(8) and /etc/login.defs.

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

这个文件中的大部分命令和语法都会在第12章以及后续章节中具体讲到。每个发行版的/etc/profile文件都有不同的设置和命令。例如，在上面所显示的Ubuntu发行版的/etc/profile文件中，涉及了一个叫作/etc/bash.bashrc的文件。这个文件包含了系统环境变量。

但是，在下面显示的CentOS发行版的/etc/profile文件中，并没有出现这个文件。另外要注意的是，该发行版的/etc/profile文件还在内部导出了一些系统环境变量。

```
$ cat /etc/profile
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

# It's NOT a good idea to change this file unless you know what you
# are doing. It's much better to create a custom.sh shell script in
# /etc/profile.d/ to make custom changes to your environment, to
# prevent the need for merging in future updates.

pathmunge () {
    case "${PATH}:" in
        *:"$1":*)
            ;;
        *)
            if [ "$2" = "after" ] ; then
                PATH=$PATH:$1
            else
                PATH=$1:$PATH
            fi
    esac
}

if [ -x /usr/bin/id ]; then
    if [ -z "$EUID" ]; then
        # ksh workaround
```

```

        EUID=`id -u`
        UID=`id -ru`
    fi
    USER="`id -un`"
    LOGNAME=$USER
    MAIL="/var/spool/mail/$USER"
fi

# Path manipulation
if [ "$EUID" = "0" ]; then
    pathmunge /sbin
    pathmunge /usr/sbin
    pathmunge /usr/local/sbin
else
    pathmunge /usr/local/sbin after
    pathmunge /usr/sbin after
    pathmunge /sbin after
fi

HOSTNAME=`/bin/hostname 2>/dev/null`
HISTSIZE=1000
if [ "$HISTCONTROL" = "ignorespace" ] ; then
    export HISTCONTROL=ignoreboth
else
    export HISTCONTROL=ignoredups
fi

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL

# By default, we want umask to get set. This sets it for login shell
# Current threshold for system reserved uid/gids is 200
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 002
else
    umask 022
fi

for i in /etc/profile.d/*.sh ; do
    if [ -r "$i" ]; then
        if [ "${-#*i}" != "$-" ]; then
            . "$i"
        else
            . "$i" >/dev/null 2>&1
        fi
    fi
done

unset i
unset -f pathmunge
$

```

这两个发行版的/etc/profile文件都用到了同一个特性：for语句。它用来迭代/etc/profile.d目

录下的所有文件。（该语句会在第13章中详述。）这为Linux系统提供了一个放置特定应用程序启动文件的地方，当用户登录时，shell会执行这些文件。在本书所用的Ubuntu Linux系统中，/etc/profile.d目录下包含以下文件：

```
$ ls -l /etc/profile.d
total 12
-rw-r--r-- 1 root root  40 Apr 15 06:26 appmenu-qt5.sh
-rw-r--r-- 1 root root 663 Apr  7 10:10 bash_completion.sh
-rw-r--r-- 1 root root 1947 Nov 22  2013 vte.sh
$
```

在CentOS系统中，/etc/profile.d目录下的文件更多：

```
$ ls -l /etc/profile.d
total 80
-rw-r--r-- 1 root root 1127 Mar  5 07:17 colorls.csh
-rw-r--r-- 1 root root 1143 Mar  5 07:17 colorls.sh
-rw-r--r-- 1 root root  92 Nov 22  2013 cvs.csh
-rw-r--r-- 1 root root  78 Nov 22  2013 cvs.sh
-rw-r--r-- 1 root root 192 Feb 24 09:24 glib2.csh
-rw-r--r-- 1 root root 192 Feb 24 09:24 glib2.sh
-rw-r--r-- 1 root root  58 Nov 22  2013 gnome-ssh-askpass.csh
-rw-r--r-- 1 root root  70 Nov 22  2013 gnome-ssh-askpass.sh
-rwxr-xr-x 1 root root 373 Sep 23  2009 kde.csh
-rwxr-xr-x 1 root root 288 Sep 23  2009 kde.sh
-rw-r--r-- 1 root root 1741 Feb 20 05:44 lang.csh
-rw-r--r-- 1 root root 2706 Feb 20 05:44 lang.sh
-rw-r--r-- 1 root root 122 Feb  7  2007 less.csh
-rw-r--r-- 1 root root 108 Feb  7  2007 less.sh
-rw-r--r-- 1 root root 976 Sep 23  2011 qt.csh
-rw-r--r-- 1 root root 912 Sep 23  2011 qt.sh
-rw-r--r-- 1 root root 2142 Mar 13 15:37 udisks-bash-completion.sh
-rw-r--r-- 1 root root  97 Apr  5  2012 vim.csh
-rw-r--r-- 1 root root 269 Apr  5  2012 vim.sh
-rw-r--r-- 1 root root 169 May 20  2009 which2.sh
$
```

不难发现，有些文件与系统中的特定应用有关。大部分应用都会创建两个启动文件：一个供bash shell使用（使用.sh扩展名），一个供c shell使用（使用.csh扩展名）。

lang.csh和lang.sh文件会尝试去判定系统上所采用的默认语言字符集，然后设置对应的LANG环境变量。

2. \$HOME目录下的启动文件

剩下的启动文件都起着同一个作用：提供一个用户专属的启动文件来定义该用户所用到的环境变量。大多数Linux发行版只用这四个启动文件中的一到两个：

- ❑ \$HOME/.bash_profile
- ❑ \$HOME/.bashrc
- ❑ \$HOME/.bash_login
- ❑ \$HOME/.profile

注意，这四个文件都以点号开头，这说明它们是隐藏文件（不会在通常的`ls`命令输出列表中出现）。它们位于用户的HOME目录下，所以每个用户都可以编辑这些文件并添加自己的环境变量，这些环境变量会在每次启动bash shell会话时生效。

说明 Linux发行版在环境文件方面存在的差异非常大。本节中所列出的\$HOME下的那些文件并非每个用户都有。例如有些用户可能只有一个\$HOME/.bash_profile文件。这很正常。

shell会按照按照下列顺序，运行第一个被找到的文件，余下的则被忽略：

```
$HOME/.bash_profile
$HOME/.bash_login
$HOME/.profile
```

注意，这个列表中并没有\$HOME/.bashrc文件。这是因为该文件通常通过其他文件运行的。

窍门 记住，\$HOME表示的是某个用户的主目录。它和波浪号（~）的作用一样。

CentOS Linux系统中的.bash_profile文件的内容如下：

```
$ cat $HOME/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
$
```

.bash_profile启动文件会先去检查HOME目录中是不是还有一个叫.bashrc的启动文件。如果有的话，会先执行启动文件里面的命令。

6.6.2 交互式 shell 进程

如果你的bash shell不是登录系统时启动的（比如是在命令行提示符下敲入bash时启动），那么你启动的shell叫作交互式shell。交互式shell不会像登录shell一样运行，但它依然提供了命令行提示符来输入命令。

如果bash是作为交互式shell启动的，它就不会访问/etc/profile文件，只会检查用户HOME目录中的.bashrc文件。

在本书所用的CentOS Linux系统上，这个文件看起来如下：

```
$ cat .bashrc

# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific aliases and functions
$
```

.bashrc文件有两个作用：一是查看/etc目录下通用的bashrc文件，二是为用户提供一个定制自己的命令别名（参见第5章）和私有脚本函数（将在第17章中讲到）的地方。

6.6.3 非交互式 shell

最后一种shell是非交互式shell。系统执行shell脚本时用的就是这种shell。不同的地方在于它没有命令行提示符。但是当你在系统上运行脚本时，也许希望能够运行一些特定启动的命令。

窍门 脚本能以不同的方式执行。只有其中的某一些方式能够启动子shell。你会在第11章中学习到shell不同的执行方式。

为了处理这种情况，bash shell提供了BASH_ENV环境变量。当shell启动一个非交互式shell进程时，它会检查这个环境变量来查看要执行的启动文件。如果有指定的文件，shell会执行该文件里的命令，这通常包括shell脚本变量设置。

在本书所用的CentOS Linux发行版中，这个环境变量在默认情况下并未设置。如果变量未设置，printenv命令只会返回CLI提示符：

```
$ printenv BASH_ENV
$
```

在本书所用的Ubuntu发行版中，变量BASH_ENV也没有被设置。记住，如果变量未设置，echo命令会显示一个空行，然后返回CLI提示符：

```
$ echo $BASH_ENV

$
```

那如果BASH_ENV变量没有设置，shell脚本到哪里去获得它们的环境变量呢？别忘了有些shell脚本是通过启动一个子shell来执行的（参见第5章）。子shell可以继承父shell导出过的变量。

举例来说，如果父shell是登录shell，在/etc/profile、/etc/profile.d/*.sh和\$HOME/.bashrc文件中设置并导出了变量，用于执行脚本的子shell就能够继承这些变量。

要记住，由父shell设置但并未导出的变量都是局部变量。子shell无法继承局部变量。

对于那些不启动子shell的脚本，变量已经存在于当前shell中了。所以就算没有设置BASH_ENV，也可以使用当前shell的局部变量和全局变量。

6.6.4 环境变量持久化

现在你已经了解了各种shell进程以及对应的环境文件，找出永久性环境变量就容易多了。也可以利用这些文件创建自己的永久性全局变量或局部变量。

对全局环境变量来说（Linux系统中所有用户都需要使用的变量），可能更倾向于将新的或修改过的变量设置放在/etc/profile文件中，但这可不是什么好主意。如果你升级了所用的发行版，这个文件也会跟着更新，那你所有定制过的变量设置可就都没有了。

最好是在/etc/profile.d目录中创建一个以.sh结尾的文件。把所有新的或修改过的全局环境变量设置放在这个文件中。

在大多数发行版中，存储个人用户永久性bash shell变量的地方是\$HOME/.bashrc文件。这一点适用于所有类型的shell进程。但如果设置了BASH_ENV变量，那么记住，除非它指向的是\$HOME/.bashrc，否则你应该将非交互式shell的用户变量放在别的地方。

说明 图形化界面组成部分（如GUI客户端）的环境变量可能需要在另外一些配置文件中设置，这和设置bash shell环境变量的地方不一样。

想想第5章中讲过的alias命令设置就是不能持久的。你可以把自己的alias设置放在\$HOME/.bashrc启动文件中，使其效果永久化。

6.7 数组变量

环境变量有一个很酷的特性就是，它们可作为数组使用。数组是能够存储多个值的变量。这些值可以单独引用，也可以作为整个数组来引用。

要给某个环境变量设置多个值，可以把值放在括号里，值与值之间用空格分隔。

```
$ mytest=(one two three four five)
$
```

没什么特别的地方。如果你想把数组像普通的环境变量那样显示，你会失望的。

```
$ echo $mytest
one
$
```

只有数组的第一个值显示出来了。要引用一个单独的数组元素，就必须用代表它在数组中位置的数值索引值。索引值要用方括号括起来。

```
$ echo ${mytest[2]}
three
$
```

窍门 环境变量数组的索引值都是从零开始。这通常会带来一些困惑。

要显示整个数组变量，可用星号作为通配符放在索引值的位置。

```
$ echo ${mytest[*]}
one two three four five
$
```

也可以改变某个索引值位置的值。

```
$ mytest[2]=seven
$
$ echo ${mytest[*]}
one two seven four five
$
```

甚至能用unset命令删除数组中的某个值，但是要小心，这可能会有点复杂。看下面的例子。

```
$ unset mytest[2]
$
$ echo ${mytest[*]}
one two four five
$
$ echo ${mytest[2]}

$ echo ${mytest[3]}
four
$
```

这个例子用unset命令删除在索引值为2的位置上的值。显示整个数组时，看起来像是索引里面已经没这个索引了。但当专门显示索引值为2的位置上的值时，就能看到这个位置是空的。

最后，可以在unset命令后跟上数组名来删除整个数组。

```
$ unset mytest
$
$ echo ${mytest[*]}

$
```

有时数组变量会让事情很麻烦，所以在shell脚本编程时并不常用。对其他shell而言，数组变量的可移植性并不好，如果需要在不同的shell环境下从事大量的脚本编写工作，这会带来很多不便。有些bash系统环境变量使用了数组（比如BASH_VERSINFO），但总体上不会太频繁用到。

6.8 小结

本章介绍了Linux的环境变量。全局环境变量可以在对其作出定义的父进程所创建的子进程中使用。局部环境变量只能在定义它们的进程中使用。

Linux系统使用全局环境变量和局部环境变量存储系统环境信息。可以通过shell的命令行界面或者在shell脚本中访问这些信息。bash shell沿用了最初Unix Bourne shell定义的那些系统环境变量，也支持很多新的环境变量。PATH环境变量定义了bash shell在查找可执行命令时的搜索目录。可以修改PATH环境变量来添加自己的搜索目录（甚至是当前目录符号），以方便程序的运行。

也可以创建自用的全局和局部环境变量。一旦创建了环境变量，它在整个shell会话过程中都是可用的。

bash shell会在启动时执行几个启动文件。这些启动文件包含了环境变量的定义，可用于为每个bash会话设置标准环境变量。每次登录Linux系统，bash shell都会访问/etc/profile启动文件以及3个针对每个用户的本地启动文件：`$HOME/.bash_profile`、`$HOME/.bash_login`和`$HOME/.profile`。用户可以在这些文件中定制自己想要的环境变量和启动脚本。

最后，我们还讨论了环境变量数组。这些环境变量可在单个变量中包含多个值。你可以通过指定索引值来访问其中的单个值，或是通过环境变量数组名来引用所有的值。

下章将会深入介绍Linux文件的权限。对Linux新手来说，这可能是最难懂的。然而要写出优秀的shell脚本，就必须明白文件权限的工作原理以及如何在Linux系统中使用它们。

第 7 章

理解Linux文件权限



本章内容

- ❑ 理解Linux的安全性
- ❑ 解读文件权限
- ❑ 使用Linux组

缺乏安全性的系统不是完整的系统。系统中必须有一套能够保护文件免遭非授权用户浏览或修改的机制。Linux沿用了Unix文件权限的办法，即允许用户和组根据每个文件和目录的安全性设置来访问文件。本章将介绍如何在必要时利用Linux文件安全系统保护和共享数据。

7.1 Linux 的安全性

Linux安全系统的核心是用户账户。每个能进入Linux系统的用户都会被分配唯一的用户账户。用户对系统中各种对象的访问权限取决于他们登录系统时用的账户。

用户权限是通过创建用户时分配的用户ID（User ID，通常缩写为UID）来跟踪的。UID是数值，每个用户都有唯一的UID，但在登录系统时用的不是UID，而是登录名。登录名是用户用来登录系统的最长八字符的字符串（字符可以是数字或字母），同时会关联一个对应的密码。

Linux系统使用特定的文件和工具来跟踪和管理系统上的用户账户。在我们讨论文件权限之前，先来看一下Linux是怎样处理用户账户的。本节会介绍管理用户账户需要的文件和工具，这样在处理文件权限问题时，你就知道如何使用它们了。

7.1.1 /etc/passwd 文件

Linux系统使用一个专门的文件来将用户的登录名匹配到对应的UID值。这个文件就是/etc/passwd文件，它包含了一些与用户有关的信息。下面是Linux系统上典型的/etc/passwd文件的一个例子。

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

```

bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
rpm:x:37:37:/:/var/lib/rpm:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
mailnull:x:47:47:/:/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51:/:/var/spool/mqueue:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
ntp:x:38:38:/:etc/ntp:/sbin/nologin
nscd:x:28:28:NSCD Daemon:/:/sbin/nologin
tcpdump:x:72:72:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
avahi:x:70:70:Avahi daemon:/:/sbin/nologin
hsqldb:x:96:96:/:/var/lib/hsqldb:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
haldaemon:x:68:68:HAL daemon:/:/sbin/nologin
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
gdm:x:42:42:/:/var/gdm:/sbin/nologin
rich:x:500:500:Rich Blum:/home/rich:/bin/bash
mama:x:501:501:Mama:/home/mama:/bin/bash
katie:x:502:502:katie:/home/katie:/bin/bash
jessica:x:503:503:Jessica:/home/jessica:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
$

```

root用户账户是Linux系统的管理员，固定分配给它的UID是0。就像上例中显示的，Linux系统会为各种各样的功能创建不同的用户账户，而这些账户并不是真的用户。这些账户叫作系统账户，是系统上运行的各种服务进程访问资源用的特殊账户。所有运行在后台的服务都需要用一个系统用户账户登录到Linux系统上。

在安全成为一个大问题之前，这些服务经常会用root账户登录。遗憾的是，如果有非授权的用户攻陷了这些服务中的一个，他立刻就能作为root用户进入系统。为了防止发生这种情况，现在运行在Linux服务器后台的几乎所有的服务都是用自己的账户登录。这样的话，即使有人攻入了某个服务，也无法访问整个系统。

Linux为系统账户预留了500以下的UID值。有些服务甚至要用特定的UID才能正常工作。为

普通用户创建账户时，大多数Linux系统会从500开始，将第一个可用UID分配给这个账户（并非所有的Linux发行版都是这样）。

你可能已经注意到/etc/passwd文件中还有很多用户登录名和UID之外的信息。/etc/passwd文件的字段包含了如下信息：

- ❑ 登录用户名
- ❑ 用户密码
- ❑ 用户账户的UID（数字形式）
- ❑ 用户账户的组ID（GID）（数字形式）
- ❑ 用户账户的文本描述（称为备注字段）
- ❑ 用户HOME目录的位置
- ❑ 用户的默认shell

/etc/passwd文件中的密码字段都被设置成了x，这并不是说所有的用户账户都用相同的密码。在早期的Linux上，/etc/passwd文件里有加密后的用户密码。但鉴于很多程序都需要访问/etc/passwd文件获取用户信息，这就成了一个安全隐患。随着用来破解加密密码的工具的不断演进，用心不良的人开始忙于破解存储在/etc/passwd文件中的密码。Linux开发人员需要重新考虑这个策略。

现在，绝大多数Linux系统都将用户密码保存在另一个单独的文件中（叫作shadow文件，位置在/etc/shadow）。只有特定的程序（比如登录程序）才能访问这个文件。

/etc/passwd是一个标准的文本文件。你可以用任何文本编辑器在/etc/passwd文件里直接手动进行用户管理（比如添加、修改或删除用户账户）。但这样做极其危险。如果/etc/passwd文件出现损坏，系统就无法读取它的内容了，这样会导致用户无法正常登录（即便是root用户）。用标准的Linux用户管理工具去执行这些用户管理功能就会安全许多。

7.1.2 /etc/shadow 文件

/etc/shadow文件对Linux系统密码管理提供了更多的控制。只有root用户才能访问/etc/shadow文件，这让它比起/etc/passwd安全许多。

/etc/shadow文件为系统上的每个用户账户都保存了一条记录。记录就像下面这样：

```
rich:$1$.FfcK0ns$f1UgiyHQ25wrB/hykCn020:11627:0:99999:7:::
```

在/etc/shadow文件的每条记录中都有9个字段：

- ❑ 与/etc/passwd文件中的登录名字段对应的登录名
- ❑ 加密后的密码
- ❑ 自上次修改密码后过去的天数密码（自1970年1月1日开始计算）
- ❑ 多少天后才能更改密码
- ❑ 多少天后必须更改密码
- ❑ 密码过期前提前多少天提醒用户更改密码

- ❑ 密码过期后多少天禁用用户账户
- ❑ 用户账户被禁用的日期（用自1970年1月1日到当天的天数表示）
- ❑ 预留字段给将来使用

使用shadow密码系统后，Linux系统可以更好地控制用户密码。它可以控制用户多久更改一次密码，以及什么时候禁用该用户账户，如果密码未更新的话。

7.1.3 添加新用户

用来向Linux系统添加新用户的主要工具是useradd。这个命令简单快捷，可以一次性创建新用户账户及设置用户HOME目录结构。useradd命令使用系统的默认值以及命令行参数来设置用户账户。系统默认值被设置在/etc/default/useradd文件中。可以使用加入了-D选项的useradd命令查看所用Linux系统中的这些默认值。

```
# /usr/sbin/useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
#
```

说明 一些Linux发行版会把Linux用户和组工具放在/usr/sbin目录下，这个目录可能不在PATH环境变量里。如果你的Linux系统是这样的话，可以将这个目录添加进PATH环境变量，或者用绝对文件路径名来使用这些工具。

在创建新用户时，如果你不在命令行中指定具体的值，useradd命令就会使用-D选项所显示的那些默认值。这个例子列出的默认值如下：

- ❑ 新用户会被添加到GID为100的公共组；
- ❑ 新用户的HOME目录将会位于/home/loginname；
- ❑ 新用户账户密码在过期后不会被禁用；
- ❑ 新用户账户未被设置过期日期；
- ❑ 新用户账户将bash shell作为默认shell；
- ❑ 系统会将/etc/skel目录下的内容复制到用户的HOME目录下；
- ❑ 系统为该用户账户在mail目录下创建一个用于接收邮件的文件。

倒数第二个值很有意思。useradd命令允许管理员创建一份默认的HOME目录配置，然后把它作为创建新用户HOME目录的模板。这样就能自动在每个新用户的HOME目录里放置默认的系统文件。在Ubuntu Linux系统上，/etc/skel目录有下列文件：

```
$ ls -al /etc/skel
```

```
total 32
drwxr-xr-x  2 root root  4096 2010-04-29 08:26 .
drwxr-xr-x 135 root root 12288 2010-09-23 18:49 ..
-rw-r--r--  1 root root   220 2010-04-18 21:51 .bash_logout
-rw-r--r--  1 root root  3103 2010-04-18 21:51 .bashrc
-rw-r--r--  1 root root   179 2010-03-26 08:31 examples.desktop
-rw-r--r--  1 root root   675 2010-04-18 21:51 .profile
$
```

根据第6章的内容，你应该能知道这些文件是做什么的。它们是bash shell环境的标准启动文件。系统会自动将这些默认文件复制到你创建的每个用户的HOME目录。

可以用默认系统参数创建一个新用户账户，然后检查一下新用户的HOME目录。

```
# useradd -m test
# ls -al /home/test
total 24
drwxr-xr-x 2 test test 4096 2010-09-23 19:01 .
drwxr-xr-x 4 root root 4096 2010-09-23 19:01 ..
-rw-r--r-- 1 test test  220 2010-04-18 21:51 .bash_logout
-rw-r--r-- 1 test test 3103 2010-04-18 21:51 .bashrc
-rw-r--r-- 1 test test  179 2010-03-26 08:31 examples.desktop
-rw-r--r-- 1 test test  675 2010-04-18 21:51 .profile
#
```

默认情况下，useradd命令不会创建HOME目录，但是-m命令行选项会使其创建HOME目录。你能在此例中看到，useradd命令创建了新HOME目录，并将/etc/skel目录中的文件复制了过来。

说明 运行本章中提到的用户账户管理命令，需要以root用户账户登录或者通过sudo命令以root用户账户身份运行这些命令。

要想在创建用户时改变默认值或默认行为，可以使用命令行参数。表7-1列出了这些参数。

表7-1 useradd命令行参数

参 数	描 述
-c <i>comment</i>	给新用户添加备注
-d <i>home_dir</i>	为主目录指定一个名字（如果不想用登录名作为主目录名的话）
-e <i>expire_date</i>	用YYYY-MM-DD格式指定一个账户过期的日期
-f <i>inactive_days</i>	指定这个账户密码过期后多少天这个账户被禁用；0表示密码一过期就立即禁用，1表示禁用这个功能
-g <i>initial_group</i>	指定用户登录组的GID或组名
-G <i>group ...</i>	指定用户除登录组之外所属的一个或多个附加组
-k	必须和-m一起使用，将/etc/skel目录的内容复制到用户的HOME目录
-m	创建用户的HOME目录
-M	不创建用户的HOME目录（当默认设置里要求创建时才使用这个选项）
-n	创建一个与用户登录名同名的新组

(续)

参 数	描 述
-r	创建系统账户
-p <i>passwd</i>	为用户账户指定默认密码
-s <i>shell</i>	指定默认的登录shell
-u <i>uid</i>	为账户指定唯一的UID

你会发现，在创建新用户账户时使用命令行参数可以更改系统指定的默认值。但如果总需要修改某个值的话，最好还是修改一下系统的默认值。

可以在-D选项后跟上一个指定的值来修改系统默认的新用户设置。这些参数如表7-2所示。

表7-2 useradd更改默认值的参数

参 数	描 述
-b <i>default_home</i>	更改默认的创建用户HOME目录的位置
-e <i>expiration_date</i>	更改默认的新账户的过期日期
-f <i>inactive</i>	更改默认的新用户从密码过期到账户被禁用的天数
-g <i>group</i>	更改默认的组名称或GID
-s <i>shell</i>	更改默认的登录shell

更改默认值非常简单：

```
# useradd -D -s /bin/tsh
# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/tsh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
#
```

现在，useradd命令会将tsh shell作为所有新建用户的默认登录shell。

7.1.4 删除用户

如果你想从系统中删除用户，userdel可以满足这个需求。默认情况下，userdel命令会只删除/etc/passwd文件中的用户信息，而不会删除系统中属于该账户的任何文件。

如果加上-r参数，userdel会删除用户的HOME目录以及邮件目录。然而，系统上仍可能存有已删除用户的其他文件。这在有些环境中会造成问题。

下面是用userdel命令删除已有用户账户的一个例子。

```
# /usr/sbin/userdel -r test
# ls -al /home/test
ls: cannot access /home/test: No such file or directory
#
```

加了-r参数后，用户先前的那个/home/test目录已经不存在了。

警告 在有大量用户的环境中使用-r参数时要特别小心。你永远不知道用户是否在其HOME目录下存放了其他用户或其他程序要使用的重要文件。记住，在删除用户的HOME目录之前一定要检查清楚！

7.1.5 修改用户

Linux提供了一些不同的工具来修改已有用户账户的信息。表7-3列出了这些工具。

表7-3 用户账户修改工具

命 令	描 述
usermod	修改用户账户的字段，还可以指定主要组以及附加组的所属关系
passwd	修改已有用户的密码
chpasswd	从文件中读取登录名密码对，并更新密码
chage	修改密码的过期日期
chfn	修改用户账户的备注信息
chsh	修改用户账户的默认登录shell

每种工具都提供了特定的功能来修改用户账户信息。下面的几节将具体介绍这些工具。

1. usermod

usermod命令是用户账户修改工具中最强大的一个。它能用来修改/etc/passwd文件中的大部分字段，只需用与想修改的字段对应的命令行参数就可以了。参数大部分跟useradd命令的参数一样（比如，-c修改备注字段，-e修改过期日期，-g修改默认的登录组）。除此之外，还有另外一些可能派上用场的选项。

- ❑ -l修改用户账户的登录名。
- ❑ -L锁定账户，使用户无法登录。
- ❑ -p修改账户的密码。
- ❑ -U解除锁定，使用户能够登录。

-L选项尤其实用。它可以将账户锁定，使用户无法登录，同时无需删除账户和用户的数据。要让账户恢复正常，只要用-U选项就行了。

2. passwd和chpasswd

改变用户密码的一个简便方法就是用passwd命令。

```
# passwd test
Changing password for user test.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
#
```

如果只用passwd命令，它会改你自己的密码。系统上的任何用户都能改自己的密码，但只有root用户才有权限改别人的密码。

-e选项能强制用户下次登录时修改密码。你可以先给用户设置一个简单的密码，之后再强制在下一次登录时改成他们能记住的更复杂的密码。

如果需要为系统中的大量用户修改密码，chpasswd命令可以事半功倍。chpasswd命令能从标准输入自动读取登录名和密码对（由冒号分割）列表，给密码加密，然后为用户账户设置。你也可以用重定向命令来将含有userid:passwd对的文件重定向给该命令。

```
# chpasswd < users.txt
#
```

3. chsh、chfn和chage

chsh、chfn和chage工具专门用来修改特定的账户信息。chsh命令用来快速修改默认的用户登录shell。使用时必须用shell的全路径名作为参数，不能只用shell名。

```
# chsh -s /bin/csh test
Changing shell for test.
Shell changed.
#
```

chfn命令提供了在/etc/passwd文件的备注字段中存储信息的标准方法。chfn命令会将用于Unix的finger命令的信息存进备注字段，而不是简单地存入一些随机文本（比如名字或昵称之类的），或是将备注字段留空。finger命令可以非常方便地查看Linux系统上的用户信息。

```
# finger rich
Login: rich                               Name: Rich Blum
Directory: /home/rich                     Shell: /bin/bash
On since Thu Sep 20 18:03 (EDT) on pts/0 from 192.168.1.2
No mail.
No Plan.
#
```

说明 出于安全性考虑，很多Linux系统管理员会在系统上禁用finger命令，不少Linux发行版甚至都没有默认安装该命令。

如果在使用chfn命令时没有参数，它会向你询问要将哪些适合的内容加进备注字段。

```
# chfn test
Changing finger information for test.
Name []: Ima Test
Office []: Director of Technology
Office Phone []: (123)555-1234
Home Phone []: (123)555-9876

Finger information changed.
# finger test
Login: test                               Name: Ima Test
Directory: /home/test                     Shell: /bin/csh
Office: Director of Technology             Office Phone: (123)555-1234
```

```
Home Phone: (123)555-9876
Never logged in.
No mail.
No Plan.
#
```

查看/etc/passwd文件中的记录，你会看到下面这样的结果。

```
# grep test /etc/passwd
test:x:504:504:Ima Test,Director of Technology,(123)555-
1234,(123)555-9876:/home/test:/bin/csh
#
```

所有的指纹信息现在都存在/etc/passwd文件中了。

最后，chage命令用来帮助管理用户账户的有效期。你需要对每个值设置多个参数，如表7-4所示。

表7-4 chage命令参数

参 数	描 述
-d	设置上次修改密码到现在的天数
-E	设置密码过期的日期
-I	设置密码过期到锁定账户的天数
-m	设置修改密码之间最少要多少天
-W	设置密码过期前多久开始出现提醒信息

chage命令的日期值可以用下面两种方式中的任意一种：

- ❑ YYYY-MM-DD格式的日期
- ❑ 代表从1970年1月1日起到该日期天数的数值

chage命令中有个好用的功能是设置账户的过期日期。有了它，你就能创建在特定日期自动过期的临时用户，再也不需要记住删除用户了！过期的账户跟锁定的账户很相似：账户仍然存在，但用户无法用它登录。

7.2 使用 Linux 组

用户账户在控制单个用户安全性方面很好用，但涉及在共享资源的一组用户时就捉襟见肘了。为了解决这个问题，Linux系统采用了另外一个安全概念——组（group）。

组权限允许多个用户对系统中的对象（比如文件、目录或设备等）共享一组共用的权限。（更多内容会在7.3节中细述。）

Linux发行版在处理默认组的成员关系时略有差异。有些Linux发行版会创建一个组，把所有用户都当作这个组的成员。遇到这种情况要特别小心，因为文件很有可能对其他用户也是可读的。有些发行版会为每个用户创建单独的一个组，这样可以更安全一些。^①

^① 例如，Ubuntu就会为每个用户创建一个单独的与用户账户同名的组。在添加用户前后可用grep命令或tail命令查看/etc/group文件的内容比较（grep USERNAME /etc/group或tail /etc/group）。

每个组都有唯一的GID——跟UID类似，在系统上这是个唯一的数值。除了GID，每个组还有唯一的组名。Linux系统上有一些组工具可以创建和管理你自己的组。本节将细述组信息是如何保存的，以及如何用组工具创建新组和修改已有的组。

7.2.1 /etc/group 文件

与用户账户类似，组信息也保存在系统的一个文件中。`/etc/group`文件包含系统上用到的每个组的信息。下面是一些来自Linux系统上`/etc/group`文件中的典型例子。

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
rich:x:500:
mama:x:501:
katie:x:502:
jessica:x:503:
mysql:x:27:
test:x:504:
```

和UID一样，GID在分配时也采用了特定的格式。系统账户用的组通常会分配低于500的GID值，而用户组的GID则会从500开始分配。`/etc/group`文件有4个字段：

- ❑ 组名
- ❑ 组密码
- ❑ GID
- ❑ 属于该组的用户列表

组密码允许非组内成员通过它临时成为该组成员。这个功能并不很普遍，但确实存在。

千万不能通过直接修改`/etc/group`文件来添加用户到一个组，要用`usermod`命令（在7.1节中介绍过）。在添加用户到不同的组之前，首先得创建组。

说明 用户账户列表某种意义上有些误导人。你会发现，在列表中，有些组并没有列出用户。这并不是说这些组没有成员。当一个用户在`/etc/passwd`文件中指定某个组作为默认组时，用户账户不会作为该组成员再出现在`/etc/group`文件中。多年以来，被这个问题难倒的系统管理员可不是一两个呢。

7.2.2 创建新组

`groupadd`命令可在系统上创建新组。

```
# /usr/sbin/groupadd shared
# tail /etc/group
haldaemon:x:68:
```

```
xfs:x:43:
gdm:x:42:
rich:x:500:
mama:x:501:
katie:x:502:
jessica:x:503:
mysql:x:27:
test:x:504:
shared:x:505:
#
```

在创建新组时，默认没有用户被分配到该组。groupadd命令没有提供将用户添加到组中的选项，但可以用usermod命令来弥补这一点。

```
# /usr/sbin/usermod -G shared rich
# /usr/sbin/usermod -G shared test
# tail /etc/group
haldaemon:x:68:
xfs:x:43:
gdm:x:42:
rich:x:500:
mama:x:501:
katie:x:502:
jessica:x:503:
mysql:x:27:
test:x:504:
shared:x:505:rich, test
#
```

shared组现在有两个成员：test和rich。usermod命令的-G选项会把这个新组添加到该用户账户的组列表里。

说明 如果更改了已登录系统账户所属的用户组，该用户必须登出系统后再登录，组关系的更改才能生效。

警告 为用户账户分配组时要格外小心。如果加了-g选项，指定的组名会替换掉该账户的默认组。-G选项则将该组添加到用户的属组的列表里，不会影响默认组。

7.2.3 修改组

在/etc/group文件中可以看到，需要修改的组信息并不多。groupmod命令可以修改已有组的GID（加-g选项）或组名（加-n选项）。

```
# /usr/sbin/groupmod -n sharing shared
# tail /etc/group
haldaemon:x:68:
```

```

xfs:x:43:
gdm:x:42:
rich:x:500:
mama:x:501:
katie:x:502:
jessica:x:503:
mysql:x:27:
test:x:504:
sharing:x:505:test,rich
#

```

修改组名时，GID和组成员不会变，只有组名改变。由于所有的安全权限都是基于GID的，你可以随意改变组名而不会影响文件的安全性。

7.3 理解文件权限

现在你已经了解了用户和组，是时候解读ls命令输出时所出现的谜一般的文件权限了。本节将会介绍如何对权限进行分析以及它们的来历。

6

7.3.1 使用文件权限符

如果你还记得第3章，那应该知道ls命令可以用来查看Linux系统上的文件、目录和设备的权限。

```

$ ls -l
total 68
-rw-rw-r-- 1 rich rich  50 2010-09-13 07:49 file1.gz
-rw-rw-r-- 1 rich rich  23 2010-09-13 07:50 file2
-rw-rw-r-- 1 rich rich  48 2010-09-13 07:56 file3
-rw-rw-r-- 1 rich rich  34 2010-09-13 08:59 file4
-rwxrwxr-x 1 rich rich 4882 2010-09-18 13:58 myprog
-rw-rw-r-- 1 rich rich  237 2010-09-18 13:58 myprog.c
drwxrwxr-x 2 rich rich 4096 2010-09-03 15:12 test1
drwxrwxr-x 2 rich rich 4096 2010-09-03 15:12 test2
$

```

输出结果的第一个字段就是描述文件和目录权限的编码。这个字段的第一个字符代表了对象的类型：

- -代表文件
- d代表目录
- l代表链接
- c代表字符型设备
- b代表块设备
- n代表网络设备

之后有3组三字符的编码。每一组定义了3种访问权限：

- r代表对象是可读的
- w代表对象是可写的

□ x代表对象是可执行的

若没有某种权限，在该权限位会出现单破折线。这3组权限分别对应对象的3个安全级别：

□ 对象的属主

□ 对象的属组

□ 系统其他用户

这个概念在图7-1中进行了分解。

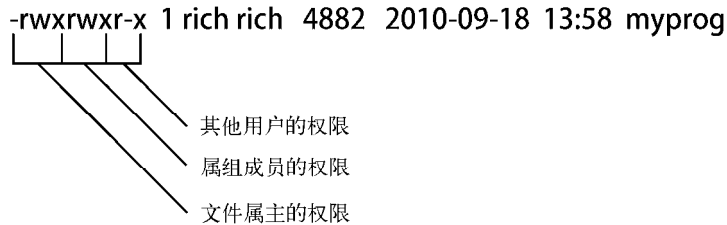


图7-1 Linux文件权限

讨论这个问题的最简单的办法就是找个例子，然后逐个分析文件权限。

```
-rwxrwxr-x 1 rich rich 4882 2010-09-18 13:58 myprog
```

文件myprog有下面3组权限。

□ rwx：文件的属主（设为登录名rich）。

□ rwx：文件的属组（设为组名rich）。

□ r-x：系统上其他人。

这些权限说明登录名为rich的用户可以读取、写入以及执行这个文件（可以看作有全部权限）。

类似地，rich组的成员也可以读取、写入和执行这个文件。然而不属于rich组的其他用户只能读取和执行这个文件：w被单破折线取代了，说明这个安全级别没有写入权限。

7.3.2 默认文件权限

你可能会问这些文件权限从何而来，答案是umask。umask命令用来设置所创建文件和目录的默认权限。

```
$ touch newfile
$ ls -al newfile
-rw-r--r-- 1 rich rich 0 Sep 20 19:16 newfile
$
```

touch命令用分配给我的用户账户的默认权限创建了这个文件。umask命令可以显示和设置这个默认权限。

```
$ umask
0022
$
```


遗憾的是，umask命令设置没那么简单明了，想弄明白其工作原理就更混乱了。第一位代表了一项特别的安全特性，叫作粘着位（sticky bit）。这部分内容会在7.5节详述。

后面的3位表示文件或目录对应的umask八进制值。要理解umask是怎么工作的，得先理解八进制模式的安全性设置。

八进制模式的安全性设置先获取这3个rwx权限的值，然后将其转换成3位二进制值，用一个八进制值来表示。在这个二进制表示中，每个位置代表一个二进制位。因此，如果读权限是唯一置位的权限，权限值就是r--，转换成二进制值就是100，代表的八进制值是4。表7-5列出了可能会遇到的组合。

表7-5 Linux文件权限码

权 限	二进制值	八进制值	描 述
---	000	0	没有任何权限
--x	001	1	只有执行权限
-w-	010	2	只有写入权限
-wx	011	3	有写入和执行权限
r--	100	4	只有读取权限
r-x	101	5	有读取和执行权限
rw-	110	6	有读取和写入权限
rwx	111	7	有全部权限

八进制模式先取得权限的八进制值，然后再把这三组安全级别（属主、属组和其他用户）的八进制值顺序列出。因此，八进制模式的值664代表属主和属组成员都有读取和写入的权限，而其他用户都只有读取权限。

了解八进制模式权限是怎么工作的之后，umask值反而更叫人困惑了。我的Linux系统上默认的八进制的umask值是0022，而我所创建的文件八进制权限却是644，这是如何得来的呢？

umask值只是个掩码。它会屏蔽掉不想授予该安全级别的权限。接下来我们还得再多进行一些八进制运算才能搞明白来龙去脉。

要把umask值从对象的全权限值中减掉。对文件来说，全权限的值是666（所有用户都有读和写的权限）；而对目录来说，则是777（所有用户都有读、写、执行权限）。

所以在上例中，文件一开始的权限是666，减去umask值022之后，剩下的文件权限就成了644。

在大多数Linux发行版中，umask值通常会设置在/etc/profile启动文件中（参见第6章），不过有一些是设置在/etc/login.defs文件中的（如Ubuntu）。可以用umask命令为默认umask设置指定一个新值。

```
$ umask 026
$ touch newfile2
$ ls -l newfile2
-rw-r----- 1 rich    rich      0 Sep 20 19:46 newfile2
$
```

在把umask值设成026后，默认的文件权限变成了640，因此新文件现在对组成员来说是只读的，而系统里的其他成员则没有任何权限。

umask值同样会作用在创建目录上。

```
$ mkdir newdir
$ ls -l
drwxr-x--x    2 rich    rich          4096 Sep 20 20:11 newdir/
$
```

由于目录的默认权限是777，umask作用后生成的目录权限不同于生成的文件权限。umask值026会从777中减去，留下来751作为目录权限设置。

7.4 改变安全性设置

如果你已经创建了一个目录或文件，需要改变它的安全性设置，在Linux系统上有一些工具能够完成这项任务。本节将告诉你如何更改文件和目录的已有权限、默认文件属主以及默认属组。

7.4.1 改变权限

chmod命令用来改变文件和目录的安全性设置。该命令的格式如下：

```
chmod options mode file
```

mode参数可以使用八进制模式或符号模式进行安全性设置。八进制模式设置非常直观，直接用期望赋予文件的标准3位八进制权限码即可。

```
$ chmod 760 newfile
$ ls -l newfile
-rwxrw----    1 rich    rich          0 Sep 20 19:16 newfile
$
```

八进制文件权限会自动应用到指定的文件上。符号模式的权限就没这么简单了。

与通常用到的3组三字符权限字符不同，chmod命令采用了另一种方法。下面是在符号模式下指定权限的格式。

```
[ugoa...][[+-=][rwxXstugo...]
```

非常有意义，不是吗？第一组字符定义了权限作用的对象：

- ☐ u代表用户
- ☐ g代表组
- ☐ o代表其他
- ☐ a代表上述所有

下一步，后面跟着的符号表示你是想在现有限权基础上增加权限(+)，还是在现有限权基础上移除权限(-)，或是将权限设置成后面的值(=)。

最后，第三个符号代表作用到设置上的权限。你会发现，这个值要比通常的rwx多。额外的设置有以下几项。

- ❑ x: 如果对象是目录或者它已有执行权限, 赋予执行权限。
- ❑ s: 运行时重新设置UID或GID。
- ❑ t: 保留文件或目录。
- ❑ u: 将权限设置为跟属主一样。
- ❑ g: 将权限设置为跟属组一样。
- ❑ o: 将权限设置为跟其他用户一样。

像这样使用这些权限。

```
$ chmod o+r newfile
$ ls -lF newfile
-rwxrw-r--  1 rich      rich          0 Sep 20 19:16 newfile*
$
```

不管其他用户在这一安全级别之前都有什么权限, o+r都给这一级别添加读取权限。

```
$ chmod u-x newfile
$ ls -lF newfile
-rw-rw-r--  1 rich      rich          0 Sep 20 19:16 newfile
$
```

u-x移除了属主已有的执行权限。注意ls命令的-F选项, 它能够在具有执行权限的文件名后加一个星号。

options为chmod命令提供了另外一些功能。-R选项可以让权限的改变递归地作用到文件和子目录。你可以使用通配符指定多个文件, 然后利用一条命令将权限更改应用到这些文件上。

7.4.2 改变所属关系

有时你需要改变文件的属主, 比如有人离职或开发人员创建了一个在产品环境中需要归属在系统账户下的应用。Linux提供了两个命令来实现这个功能: chown命令用来改变文件的属主, chgrp命令用来改变文件的默认属组。

chown命令的格式如下。

```
chown options owner[.group] file
```

可用登录名或UID来指定文件的新属主。

```
# chown dan newfile
# ls -l newfile
-rw-rw-r--  1 dan      rich          0 Sep 20 19:16 newfile
#
```

非常简单。chown命令也支持同时改变文件的属主和属组。

```
# chown dan.shared newfile
# ls -l newfile
-rw-rw-r--  1 dan      shared        0 Sep 20 19:16 newfile
#
```

如果你不嫌麻烦, 可以只改变一个目录的默认属组。

```
# chown .rich newfile
# ls -l newfile
-rw-rw-r-- 1 dan rich 0 Sep 20 19:16 newfile
#
```

最后，如果你的Linux系统采用和用户登录名匹配的组名，可以只用一个条目就改变二者。

```
# chown test. newfile
# ls -l newfile
-rw-rw-r-- 1 test test 0 Sep 20 19:16 newfile
#
```

chown命令采用一些不同的选项参数。-R选项配合通配符可以递归地改变子目录和文件的所属关系。-h选项可以改变该文件的所有符号链接文件的所属关系。

说明 只有root用户能够改变文件的属主。任何属主都可以改变文件的属组，但前提是属主必须是原属组和目标属组的成员。

chgrp命令可以更改文件或目录的默认属组。

```
$ chgrp shared newfile
$ ls -l newfile
-rw-rw-r-- 1 rich shared 0 Sep 20 19:16 newfile
$
```

用户账户必须是这个文件的属主，除了能够更换属组之外，还得是新组的成员。现在shared组的任意一个成员都可以写这个文件了。这是Linux系统共享文件的一个途径。然而，在系统中给一组用户共享文件也会变得很复杂。下一节会介绍如何实现。

7.5 共享文件

可能你已经猜到了，Linux系统上共享文件的方法是创建组。但在一个完整的共享文件的环境中，事情会复杂得多。

在7.3节中你已经看到，创建新文件时，Linux会用你默认的UID和GID给文件分配权限。想让别人也能访问文件，要么改变其他用户所在安全组的访问权限，要么就给文件分配一个包含其他用户的新默认属组。

如果你想在大量环境中创建文档并将文档与人共享，这会很烦琐。幸好有一种简单的方法可以解决这个问题。

Linux还为每个文件和目录存储了3个额外的信息位。

- ❑ **设置用户ID（SUID）**：当文件被用户使用时，程序会以文件属主的权限运行。
- ❑ **设置组ID（SGID）**：对文件来说，程序会以文件属组的权限运行；对目录来说，目录中创建的新文件会以目录的默认属组作为默认属组。
- ❑ **粘着位**：进程结束后文件还驻留（粘着）在内存中。

SGID位对文件共享非常重要。启用SGID位后，你可以强制在一个共享目录下创建的新文件

都属于该目录的属组，这个组也就成为了每个用户的属组。

SGID可通过chmod命令设置。它会加到标准3位八进制值之前（组成4位八进制值），或者在符号模式下用符号s。

如果你用的是八进制模式，你需要知道这些位的位置，如表7-6所示。

表7-6 chmod SUID、SGID和粘着位的八进制值

二进制值	八进制值	描 述
000	0	所有位都清零
001	1	粘着位置位
010	2	SGID位置位
011	3	SGID位和粘着位都置位
100	4	SUID位置位
101	5	SUID位和粘着位都置位
110	6	SUID位和SGID位都置位
111	7	所有位都置位

因此，要创建一个共享目录，使目录里的新文件都能沿用目录的属组，只需将该目录的SGID位置位。

```
$ mkdir testdir
$ ls -l
drwxrwxr-x    2 rich      rich      4096 Sep 20 23:12 testdir/
$ chgrp shared testdir
$ chmod g+s testdir
$ ls -l
drwxrwsr-x    2 rich      shared    4096 Sep 20 23:12 testdir/
$ umask 002
$ cd testdir
$ touch testfile
$ ls -l
total 0
-rw-rw-r--    1 rich      shared      0 Sep 20 23:13 testfile
$
```

首先，用mkdir命令来创建希望共享的目录。然后通过chgrp命令将目录的默认属组改为包含所有需要共享文件的用户的组（你必须是该组的成员）。最后，将目录的SGID位置位，以保证目录中新建文件都用shared作为默认属组。

为了让这个环境能正常工作，所有组成员都需把他们的umask值设置成文件对属组成员可写。在前面的例子中，umask改成了002，所以文件对属组是可写的。

做完了这些，组成员就能到共享目录下创建新文件了。跟期望的一样，新文件会沿用目录的属组，而不是用户的默认属组。现在shared组的所有用户都能访问这个文件了。

7.6 小结

本章讨论了管理Linux系统安全性需要知道的一些命令行命令。Linux通过用户ID和组ID来限制对文件、目录以及设备的访问。Linux将用户账户的信息存储在/etc/passwd文件中,将组信息存储在/etc/group文件中。每个用户都会被分配唯一的用户ID,以及在系统中识别用户的文本登录名。组也会被分配唯一的组ID以及组名。组可以包含一个或多个用户以支持对系统资源的共享访问。

有若干命令可以用来管理用户账户和组。useradd命令用来创建新的用户账户,groupadd命令用来创建新的组账户。修改已有用户账户,我们用usermod命令。类似的groupmod命令用来修改组账户信息。

Linux采用复杂的位系统来判定文件和目录的访问权限。每个文件都有三个安全等级:文件的属主、能够访问文件的默认属组以及系统上的其他用户。每个安全等级通过三个访问权限位来定义:读取、写入以及执行,对应于符号rwx。如果某种权限被拒绝,权限对应的符号会用单破折线代替(比如r--代表只读权限)。

这种符号权限通常以八进制值来描述。3位二进制组成一个八进制值,3个八进制值代表了3个安全等级。umask命令用来设置系统中所创建的文件和目录的默认安全设置。系统管理员通常会在/etc/profile文件中设置一个默认的umask值,但你可以随时通过umask命令来修改自己的umask值。

chmod命令用来修改文件和目录的安全设置。只有文件的属主才能改变文件或目录的权限。不过root用户可以改变系统上任意文件或目录的安全设置。chown和chgrp命令可用来改变文件默认的属主和属组。

本章最后讨论了如何使用设置组ID位来创建共享目录。SGID位会强制某个目录下创建的新文件或目录都沿用该父目录的属组,而不是创建这些文件的用户的属组。这可以为系统的用户之间共享文件提供一个简便的途径。

现在你已经了解了文件权限,下面就可以进一步了解如何使用实际的Linux文件系统了。下一章将会介绍如何使用命令行在Linux上创建新的分区,以及如何格式化新分区以使其可用于Linux虚拟目录。

本章内容

- ❑ 文件系统基础
- ❑ 日志文件系统与写时复制文件系统
- ❑ 文件系统管理
- ❑ 逻辑卷布局
- ❑ 使用Linux逻辑卷管理器

使用Linux系统时，需要作出的决策之一就是为存储设备选用什么文件系统。大多数Linux发行版在安装时会非常贴心地提供默认的文件系统，大多数入门级用户想都不想就用了默认的那个。

使用默认文件系统未必就不好，但了解一下可用的选择有时也会有所帮助。本章将探讨Linux世界里可选用的不同文件系统，并向你演示如何在命令行上进行创建和管理。

8.1 探索 Linux 文件系统

第3章讨论了Linux如何通过文件系统来在存储设备上存储文件和目录。Linux的文件系统为我们在硬盘中存储的0和1和应用中使用的文件与目录之间搭建起了一座桥梁。

Linux支持多种类型的文件系统管理文件和目录。每种文件系统都在存储设备上实现了虚拟目录结构，仅特性略有不同。本章将带你逐步了解Linux环境中较常用的文件系统的优点和缺陷。

8.1.1 基本的 Linux 文件系统

Linux最初采用的是一种简单的文件系统，它模仿了Unix文件系统的功能。本节讨论了这种文件系统的演进过程。

1. ext文件系统

Linux操作系统中引入的最早的文件系统叫作扩展文件系统(extended filesystem, 简记为ext)。它为Linux提供了一个基本的类Unix文件系统：使用虚拟目录来操作硬件设备，在物理设备上按定长的块来存储数据。

ext文件系统采用名为索引节点的系统来存放虚拟目录中所存储文件的信息。索引节点系统在每个物理设备中创建一个单独的表（称为索引节点表）来存储这些文件的信息。存储在虚拟目录中的每一个文件在索引节点表中都有一个条目。ext文件系统名称中的extended部分来自其跟踪的每个文件的额外数据，包括：

- ❑ 文件名
- ❑ 文件大小
- ❑ 文件的属主
- ❑ 文件的属组
- ❑ 文件的访问权限
- ❑ 指向存有文件数据的每个硬盘块的指针

Linux通过唯一的数值（称作索引节点号）来引用索引节点表中的每个索引节点，这个值是创建文件时由文件系统分配的。文件系统通过索引节点号而不是文件全名及路径来标识文件。

2. ext2文件系统

最早的ext文件系统有不少限制，比如文件大小不得超过2 GB。在Linux出现后不久，ext文件系统就升级到了第二代扩展文件系统，叫作ext2。

如你所猜测的，ext2文件系统是ext文件系统基本功能的一个扩展，但保持了同样的结构。ext2文件系统扩展了索引节点表的格式来保存系统上每个文件的更多信息。

ext2的索引节点表为文件添加了创建时间值、修改时间值和最后访问时间值来帮助系统管理员追踪文件的访问情况。ext2文件系统还将允许的最大文件大小增加到了2 TB（在ext2的后期版本中增加到了32 TB），以容纳数据库服务器中常见的大文件。

除了扩展索引节点表外，ext2文件系统还改变了文件在数据块中存储的方式。ext文件系统常见的问题是在文件写入到物理设备时，存储数据用的块很容易分散在整个设备中（称作碎片化，fragmentation）。数据块的碎片化会降低文件系统的性能，因为需要更长的时间在存储设备中查找特定文件的所有块。

保存文件时，ext2文件系统通过按组分配磁盘块来减轻碎片化。通过将数据块分组，文件系统在读取文件时不需要为了数据块查找整个物理设备。

多年来，ext文件系统一直都是Linux发行版采用的默认文件系统。但它也有一些限制。索引节点表虽然支持文件系统保存有关文件的更多信息，但会对系统造成致命的问题。文件系统每次存储或更新文件，它都要用新信息来更新索引节点表。问题在于这种操作并非总是一气呵成的。

如果计算机系统在存储文件和更新索引节点表之间发生了什么，这二者的内容就不同步了。ext2文件系统由于容易在系统崩溃或断电时损坏而臭名昭著。即使文件数据正常保存到了物理设备上，如果索引节点表记录没完成更新的话，ext2文件系统甚至都不知道那个文件存在！

很快开发人员就开始尝试开发不同的Linux文件系统了。

8.1.2 日志文件系统

日志文件系统为Linux系统增加了一层安全性。它不再使用之前先将数据直接写入存储设备再更新索引节点表的做法，而是先将文件的更改写入到临时文件（称作日志，journal）中。在数据成功写到存储设备和索引节点表之后，再删除对应的日志条目。

如果系统在数据被写入存储设备之前崩溃或断电了，日志文件系统下次会读取日志文件并处理上次留下的未写入的数据。

Linux中有3种广泛使用的日志方法，每种的保护等级都不相同，如表8-1所示。

表8-1 文件系统日志方法

方 法	描 述
数据模式	索引节点和文件都会被写入日志；丢失数据风险低，但性能差
有序模式	只有索引节点数据会被写入日志，但只有数据成功写入后才删除；在性能 and 安全性之间取得了良好的折中
回写模式	只有索引节点数据会被写入日志，但不控制文件数据何时写入；丢失数据风险高，但仍比不用日志好

数据模式日志方法是目前为止最安全的数据保护方法，但同时也是最慢的。所有写到存储设备上的数据都必须写两次：第一次写入日志，第二次写入真正的存储设备。这样会导致性能很差，尤其是对要做大量数据写入的系统而言。

这些年来，在Linux上还出现了一些其他日志文件系统。后面几节将会讲述常见的Linux日志文件系统。

1. ext3文件系统

2001年，ext3文件系统被引入Linux内核中，直到最近都是几乎所有Linux发行版默认的文件系统。它采用和ext2文件系统相同的索引节点表结构，但给每个存储设备增加了一个日志文件，以将准备写入存储设备的数据先记入日志。

默认情况下，ext3文件系统用有序模式的日志功能——只将索引节点信息写入日志文件，直到数据块都被成功写入存储设备才删除。你可以在创建文件系统时用简单的一个命令行选项将ext3文件系统的日志方法改成数据模式或回写模式。

虽然ext3文件系统为Linux文件系统添加了基本的日志功能，但它仍然缺少一些功能。例如ext3文件系统无法恢复误删的文件，它没有任何内建的数据压缩功能（虽然有个需单独安装的补丁支持这个功能），ext3文件系统也不支持加密文件。鉴于这些原因，Linux项目的开发人员选择再接再厉，继续改进ext3文件系统。

2. ext4文件系统

扩展ext3文件系统功能的结果是ext4文件系统（你可能也猜出来了）。ext4文件系统在2008年受到Linux内核官方支持，现在已是大多数流行的Linux发行版采用的默认文件系统，比如Ubuntu。

除了支持数据压缩和加密，ext4文件系统还支持一个称作区段（extent）的特性。区段在存储设备上按块分配空间，但在索引节点表中只保存起始块的位置。由于无需列出所有用来存储文件

中数据的数据块，它可以在索引节点表中节省一些空间。

ext4还引入了块预分配技术（block preallocation）。如果你想在存储设备上给一个你知道要变大的文件预留空间，ext4文件系统可以为文件分配所有需要用到的块，而不仅仅是那些现在已经用到的块。ext4文件系统用0填满预留的数据块，不会将它们分配给其他文件。

3. Reiser文件系统

2001年，Hans Reiser为Linux创建了第一个称为ReiserFS的日志文件系统。ReiserFS文件系统只支持回写日志模式——只把索引节点表数据写到日志文件。ReiserFS文件系统也因此成为Linux上最快的日志文件系统之一。

有两个有意思的特性被引入了ReiserFS文件系统：一个是你可以在线调整已有文件系统的大小；另一个是被称作尾部压缩（tailpacking）的技术，该技术能将一个文件的数据填进另一个文件的数据块中的空白空间。如果你必须为已有文件系统扩容来容纳更多的数据，在线调整文件系统大小功能非常好用。

4. JFS文件系统

作为可能依然在用的最老的日志文件系统之一，JFS（Journaled File System，日志化文件系统^①）是IBM在1990年为其Unix衍生版AIX开发的。然而直到第2版，它才被移植到Linux环境中。

说明 IBM官方称JFS文件系统的第2版为JFS2，但大多数Linux系统提到它时都只用JFS。

JFS文件系统采用的是有序日志方法，即只在日志中保存索引节点表数据，直到真正的文件数据被写进存储设备时才删除它。这个方法在ReiserFS的速度和数据模式日志方法的完整性之间的采取的一种折中。

JFS文件系统采用基于区段的文件分配，即为每个写入存储设备的文件分配一组块。这样可以减少存储设备上的碎片。

除了用在IBM Linux上外，JFS文件系统并没有流行起来，但你有可能在同Linux打交道的日子中碰到它。

5. XFS文件系统

XFS日志文件系统是另一种最初用于商业Unix系统而如今走进Linux世界的文件系统。美国硅图公司（SGI）最初在1994年为其商业化的IRIX Unix系统开发了XFS。2002年，它被发布到了适用于Linux环境的版本。

XFS文件系统采用回写模式的日志，在提供了高性能的同时也引入了一定的风险，因为实际数据并未存进日志文件。XFS文件系统还允许在线调整文件系统的大小，这点类似于ReiserFS文件系统，除了XFS文件系统只能扩大不能缩小。

^① 此处“日志化文件系统”是指Journaled File System这一Journal File System概念的具体实现。为防止读者混淆，后文中都将用JFS缩写代替。

8.1.3 写时复制文件系统

采用了日志式技术，你就必须在安全性和性能之间做出选择。尽管数据模式日志提供了最高的安全性，但是会对性能带来影响，因为索引节点和数据都需要被日志化。如果是回写模式日志，性能倒是可以接受，但安全性就会受到损害。

就文件系统而言，日志式的另一种选择是一种叫作写时复制（copy-on-write, COW）的技术。COW利用快照兼顾了安全性和性能。如果要修改数据，会使用克隆或可写快照。修改过的数据并不会直接覆盖当前数据，而是被放入文件系统中的一个位置上。即便是数据修改已经完成，之前的旧数据也不会被重写。

COW文件系统已日渐流行，接下来会简要概览其中最流行的两种（Btrfs和ZFS）。

1. ZFS文件系统

COW文件系统ZFS是由Sun公司于2005年研发的，用于OpenSolaris操作系统，从2008年起开始向Linux移植，最终在2012年投入Linux产品的使用。

ZFS是一个稳定的文件系统，与Reiser4、Btrfs和ext4势均力敌。它最大的弱项就是没有使用GPL许可。自2013年发起的OpenZFS项目有可能改变这种局面。但是，在获得GPL许可之前，ZFS有可能终无法成为Linux默认的文件系统。

2. Btrfs文件系统

Btrfs文件系统是COW的新人，也被称为B树文件系统。它是由Oracle公司于2007年开始研发的。Btrfs在Reiser4的诸多特性的基础上改进了可靠性。另一些开发人员最终也加入了开发过程，帮助Btrfs快速成为了最流行的文件系统。究其原因，则要归于它的稳定性、易用性以及能够动态调整已挂载文件系统的大小。OpenSUSE Linux发行版最近将Btrfs作为其默认文件系统。除此之外，该文件系统也出现在了其他Linux发行版中（如RHEL），不过并不是作为默认文件系统。

8.2 操作文件系统

Linux提供了一些不同的工具，我们可以利用它们轻松地在命令行中进行文件系统操作。可使用键盘随心所欲地创建新的文件系统或者修改已有的文件系统。本节将会带你逐步了解命令行下的文件系统交互的命令。

8.2.1 创建分区

一开始，你必须在存储设备上创建分区来容纳文件系统。分区可以是整个硬盘，也可以是部分硬盘，以容纳虚拟目录的一部分。

fdisk工具用来帮助管理安装在系统上的任何存储设备上的分区。它是个交互式程序，允许你输入命令来逐步完成硬盘分区操作。

要启动fdisk命令，你必须指定要分区的存储设备的设备名，另外还得有超级用户权限。如果在没有对应权限的情况下使用该命令，你会得到类似于下面这种错误提示。

```
$ fdisk /dev/sdb

Unable to open /dev/sdb
$
```

说明 有时候，创建新磁盘分区最麻烦的事情就是找出安装在Linux系统中的物理磁盘。Linux采用了一种标准格式来为硬盘分配设备名称，但是你得熟悉这种格式。对于老式的IDE驱动器，Linux使用的是/dev/hdx。其中x表示一个字母，具体是什么要根据驱动器的检测顺序（第一个驱动器是a，第二个驱动器是b，以此类推）。对于较新的SATA驱动器和SCSI驱动器，Linux使用/dev/sdx。其中的x具体是什么也要根据驱动器的检测顺序（和之前一样，第一个驱动器是a，第二个驱动器是b，以此类推）。在格式化分区之前，最好再检查一下是否正确指定了驱动器。

如果你拥有超级用户权限并指定了正确的驱动器，那就可以进入fdisk工具的操作界面了。下面展示了该命令在CentOS发行版中的使用情景。

```
$ sudo fdisk /dev/sdb
[sudo] password for Christine:
Device contains neither a valid DOS partition table,
nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0xd3f759b5.
Changes will remain in memory only
until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will
be corrected by w(rite)

[...]
Command (m for help):
```

窍门 如果这是你第一次给该存储设备分区，fdisk会警告你设备上没有分区表。

fdisk交互式命令提示符使用单字母命令来告诉fdisk做什么。表8-2显示了fdisk命令提示符下的可用命令。

表8-2 fdisk命令

命 令	描 述
a	设置活动分区标志
b	编辑BSD Unix系统用的磁盘标签
c	设置DOS兼容标志
d	删除分区

(续)

命 令	描 述
l	显示可用的分区类型
m	显示命令选项
n	添加一个新分区
o	创建DOS分区表
p	显示当前分区表
q	退出，不保存更改
s	为Sun Unix系统创建一个新磁盘标签
t	修改分区的系统ID
u	改变使用的存储单位
v	验证分区表
w	将分区表写入磁盘
x	高级功能

尽管看上去很恐怖，但实际上你在日常工作中用到的只有几个基本命令。
对于初学者，可以用p命令将一个存储设备的详细信息显示出来。

Command (m for help): **p**

```
Disk /dev/sdb: 5368 MB, 5368709120 bytes
255 heads, 63 sectors/track, 652 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x11747e88
```

Device	Boot	Start	End	Blocks	Id	System
--------	------	-------	-----	--------	----	--------

Command (m for help):

输出显示这个存储设备有5368 MB（5 GB）的空间。存储设备明细后的列表说明这个设备上是否已有分区。这个例子中的输出中没有显示任何分区，所以设备还未分区。

下一步，可以使用n命令在该存储设备上创建新的分区。

Command (m for help): **n**

Command action

 e extended

 p primary partition (1-4)

p

Partition number (1-4): **1**

First cylinder (1-652, default 1): **1**

Last cylinder, +cylinders or +size{K,M,G} (1-652, default 652): **+2G**

Command (m for help):

分区可以按主分区（primary partition）或扩展分区（extended partition）创建。主分区可以被

文件系统直接格式化，而扩展分区则只能容纳其他主分区^①。扩展分区出现的原因是每个存储设备上只能有4个分区。可以通过创建多个扩展分区，然后在扩展分区内创建主分区进行扩展。^②上例中创建了一个主分区，在存储设备上给它分配了分区号1，然后给它分配了2 GB的存储设备空间。你可以再次使用p命令查看结果。

```
Command (m for help): p
```

```
Disk /dev/sdb: 5368 MB, 5368709120 bytes
255 heads, 63 sectors/track, 652 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x029aa6af
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdb1		1	262	2104483+	83	Linux

```
Command (m for help):
```

从输出中现在可以看到，该存储设备上有了一个分区（叫作/dev/sdb1）。Id列定义了Linux怎么对待该分区。fdisk允许创建多种分区类型。使用l命令列出可用的不同类型。默认类型是83，该类型定义了一个Linux文件系统。如果你想为其他文件系统创建一个分区（比如Windows的NTFS分区），只要选择一个不同的分区类型即可。

可以重复上面的过程，将存储设备上剩下的空间分配给另一个Linux分区。创建了想要的分区之后，用w命令将更改保存到存储设备上。

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
$
```

存储设备的分区信息被写入分区表中，Linux系统通过ioctl()调用来获知新分区的出现。设置好分区之后，可以使用Linux文件系统对其进行格式化。

窍门 有些发行版和较旧的发行版在生成新分区之后并不会自动提醒Linux系统。如果是这样的话，你要么使用partprob或hdparm命令（参考相应的手册页），要么重启系统，让系统读取更新过的分区表。

① 此处说法有误。扩展分区内容纳的应该是“逻辑分区”（logical partition）。可参考https://en.wikipedia.org/wiki/Extended_boot_record及<https://technet.microsoft.com/en-us/library/cc976786.aspx>。

② 此处正确的说法应是：“可以通过创建一个扩展分区，然后在扩展分区内创建逻辑分区进行扩展。”

8.2.2 创建文件系统

在将数据存储到分区之前，你必须用某种文件系统对其进行格式化，这样Linux才能使用它。每种文件系统类型都用自己的命令行程序来格式化分区。表8-3列出了本章中讨论的不同文件系统所对应的工具。

表8-3 创建文件系统的命令行程序

工 具	用 途
mkefs	创建一个ext文件系统
mke2fs	创建一个ext2文件系统
mkfs.ext3	创建一个ext3文件系统
mkfs.ext4	创建一个ext4文件系统
mkreiserfs	创建一个ReiserFS文件系统
jfs_mkfs	创建一个JFS文件系统
mkfs.xfs	创建一个XFS文件系统
mkfs.zfs	创建一个ZFS文件系统
mkfs.btrfs	创建一个Btrfs文件系统

并非所有文件系统工具都已经默认安装了。要想知道某个文件系统工具是否可用，可以使用type命令。

```
$ type mkfs.ext4
mkfs.ext4 is /sbin/mkfs.ext4
$
$ type mkfs.btrfs
-bash: type: mkfs.btrfs: not found
$
```

据上面这个取自Ubuntu系统的例子显示，mkfs.ext4工具是可用的。而Btrfs工具则不可用。请参阅第9章中有关如何在Linux发行版中安装软件和工具的相关内容。

每个文件系统命令都有很多命令行选项，允许你定制如何在分区上创建文件系统。要查看所有可用的命令行选项，可用man命令来显示该文件系统命令的手册页面（参见第3章）。所有的文件系统命令都允许通过不带选项的简单命令来创建一个默认的文件系统。

```
$ sudo mkfs.ext4 /dev/sdb1
[sudo] password for Christine:
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
131648 inodes, 526120 blocks
26306 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=541065216
```

```
17 block groups
32768 blocks per group, 32768 fragments per group
7744 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 23 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
$
```

这个新的文件系统采用ext4文件系统类型，这是Linux上的日志文件系统。注意，创建过程中有一步是创建新的日志。

为分区创建了文件系统之后，下一步是将它挂载到虚拟目录下的某个挂载点，这样就可以将数据存储在新文件系统中了。你可以将新文件系统挂载到虚拟目录中需要额外空间的任何位置。

```
$ ls /mnt
$
$ sudo mkdir /mnt/my_partition
$
$ ls -al /mnt/my_partition/
$
$ ls -dF /mnt/my_partition
/mnt/my_partition/
$
$ sudo mount -t ext4 /dev/sdb1 /mnt/my_partition
$
$ ls -al /mnt/my_partition/
total 24
drwxr-xr-x. 3 root root 4096 Jun 11 09:53 .
drwxr-xr-x. 3 root root 4096 Jun 11 09:58 ..
drwx-----. 2 root root 16384 Jun 11 09:53 lost+found
$
```

mkdir命令（参见第3章）在虚拟目录中创建了挂载点，mount命令将新的硬盘分区添加到挂载点。mount命令的-t选项指明了要挂载的文件系统类型（ext4）。现在你可以在新分区中保存新文件和目录了！

说明 这种挂载文件系统的方法只能临时挂载文件系统。当重启Linux系统时，文件系统并不会自动挂载。要强制Linux在启动时自动挂载新的文件系统，可以将其添加到/etc/fstab文件。

现在文件系统已经被挂载到了虚拟目录中，可以投入日常使用了。遗憾的是，在日常使用过程中有可能会出现一些严重的问题，例如文件系统损坏。下一节将演示如何应对这种问题。

8.2.3 文件系统的检查与修复

就算是现代文件系统，碰上突然断电或者某个不规矩的程序在访问文件时锁定了系统，也会出现错误。幸而有一些命令行工具可以帮你将文件系统恢复正常。

每个文件系统都有各自可以和文件系统交互的恢复命令。这可能会让局面变得不太舒服，随着Linux环境中可用的文件系统变多，你也不得不去掌握大量对应的命令。好在有个通用的前端程序，可以决定存储设备上的文件系统并根据要恢复的文件系统调用适合的文件系统恢复命令。

`fsck`命令能够检查和修复大部分类型的Linux文件系统，包括本章早些时候讨论过的`ext`、`ext2`、`ext3`、`ext4`、`ReiserFS`、`JFS`和`XFS`。该命令的格式是：

```
fsck options filesystem
```

你可以在命令行上列出多个要检查的文件系统。文件系统可以通过设备名、在虚拟目录中的挂载点以及分配给文件系统的唯一UUID值来引用。

窍门 尽管日志式文件系统的用户需要用到`fsck`命令，但是COW文件系统的用户是否也得使用该命令还存在争议。实际上，ZFS文件系统甚至都没有提供`fsck`工具的接口。

`fsck`命令使用`/etc/fstab`文件来自动决定正常挂载到系统上的存储设备的文件系统。如果存储设备尚未挂载（比如你刚刚在新的存储设备上创建了个文件系统），你需要用`-t`命令行选项来指定文件系统类型。表8-4列出了其他可用的命令行选项。

表8-4 `fsck`的命令行选项

选 项	描 述
<code>-a</code>	如果检测到错误，自动修复文件系统
<code>-A</code>	检查 <code>/etc/fstab</code> 文件中列出的所有文件系统
<code>-C</code>	给支持进度条功能的文件系统显示一个进度条（只有 <code>ext2</code> 和 <code>ext3</code> ）
<code>-N</code>	不进行检查，只显示哪些检查会执行
<code>-r</code>	出现错误时提示
<code>-R</code>	使用 <code>-A</code> 选项时跳过根文件系统
<code>-s</code>	检查多个文件系统时，依次进行检查
<code>-t</code>	指定要检查的文件系统类型
<code>-T</code>	启动时不显示头部信息
<code>-V</code>	在检查时产生详细输出
<code>-y</code>	检测到错误时自动修复文件系统

你可能注意到了，有些命令行选项是重复的。这是为多个命令实现通用的前端带来的部分问题。有些文件系统修复命令有一些额外的可用选项。如果要做更高级的错误检查，就需要查看这个文件系统修复工具的手册页面来确定是不是有该文件系统专用的扩展选项。

窍门 只能在未挂载的文件系统上运行`fsck`命令。对大多数文件系统来说，你只需卸载文件系统来进行检查，检查完成之后重新挂载就好了。但因为根文件系统含有所有核心的Linux命令和日志文件，所以无法在处于运行状态的系统上卸载它。

这正是亲身体验Linux LiveCD的好时机！只需用LiveCD启动系统即可，然后在根文件系统上运行`fsck`命令。

到目前为止，本章讲解了如何处理物理存储设备中的文件系统。Linux还有另一些方法可以为文件系统创建逻辑存储设备。下一节将告诉你如何使用逻辑存储设备。

8.3 逻辑卷管理

如果用标准分区在硬盘上创建了文件系统，为已有文件系统添加额外的空间多少是一种痛苦的体验。你只能在同一个物理硬盘的可用空间范围内调整分区大小。如果硬盘上没有地方了，你就必须弄一个更大的硬盘，然后手动将已有的文件系统移动到新的硬盘上。

这时候可以通过将另外一个硬盘上的分区加入已有文件系统，动态地添加存储空间。Linux逻辑卷管理器（logical volume manager, LVM）软件包正好可以用来做这个。它可以让你在无需重建整个文件系统的情况下，轻松地管理磁盘空间。

8.3.1 逻辑卷管理布局

逻辑卷管理的核心在于如何处理安装在系统上的硬盘分区。在逻辑卷管理的世界里，硬盘称作物理卷（physical volume, PV）。每个物理卷都会映射到硬盘上特定的物理分区。

多个物理卷集中在一起可以形成一个卷组（volume group, VG）。逻辑卷管理系统将卷组视为一个物理硬盘，但事实上卷组可能是由分布在多个物理硬盘上的多个物理分区组成的。卷组提供了一个创建逻辑分区的平台，而这些逻辑分区则包含了文件系统。

整个结构中的最后一层是逻辑卷（logical volume, LV）。逻辑卷为Linux提供了创建文件系统的分区环境，作用类似于到目前为止我们一直在探讨的Linux中的物理硬盘分区。Linux系统将逻辑卷视为物理分区。

可以使用任意一种标准Linux文件系统来格式化逻辑卷，然后再将它加入Linux虚拟目录中的某个挂载点。

图8-1显示了典型Linux逻辑卷管理环境的基本布局。

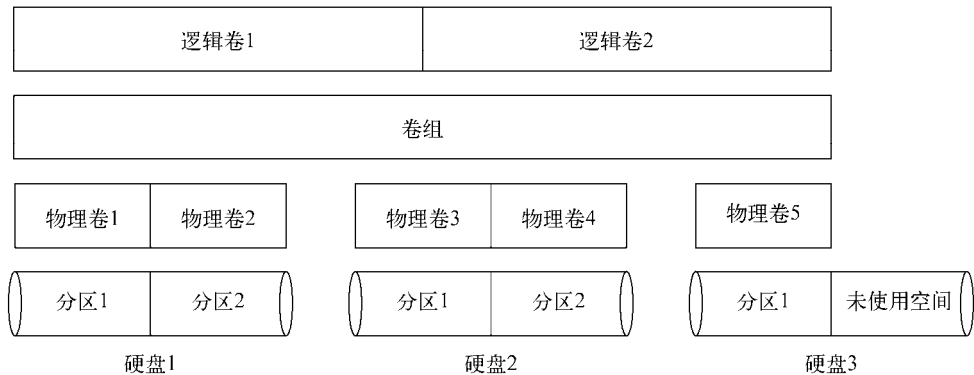


图8-1 逻辑卷管理环境

图8-1中的卷组横跨了三个不同的物理硬盘，覆盖了五个独立的物理分区。在卷组内部有两个独立的逻辑卷。Linux系统将每个逻辑卷视为一个物理分区。每个逻辑卷可以被格式化成ext4文件系统，然后挂载到虚拟目录中某个特定位置。

注意，图8-1中，第三个物理硬盘有一个未使用的分区。通过逻辑卷管理，你随后可以轻松地将这个未使用分区分配到已有卷组：要么用它创建一个新的逻辑卷，要么在需要更多空间时用它们来扩展已有的逻辑卷。

类似地，如果你给系统添加了一块硬盘，逻辑卷管理系统允许你将它添加到已有卷组，为某个已有的卷组创建更多空间，或是创建一个可用来挂载的新逻辑卷。这种扩展文件系统的方法要好得多！

8.3.2 Linux 中的 LVM

Linux LVM是由Heinz Mauelshagen开发的，于1998年发布到了Linux社区。它允许你在Linux上用简单的命令行命令管理一个完整的逻辑卷管理环境。

Linux LVM有两个可用的版本。

❑ **LVM1**：最初的LVM包于1998年发布，只能用于Linux内核2.4版本。它仅提供了基本的逻辑卷管理功能。

❑ **LVM2**：LVM的更新版本，可用于Linux内核2.6版本。它在标准的LVM1功能外提供了额外的功能。

大部分采用2.6或更高内核版本的现代Linux发行版都提供对LVM2的支持。除了标准的逻辑卷管理功能外，LVM2还提供了另外一些好用的功能。

1. 快照

最初的Linux LVM允许你在逻辑卷在线的状态下将其复制到另一个设备。这个功能叫作快照。在备份由于高可靠性需求而无法锁定的重要数据时，快照功能非常给力。传统的备份方法在将文件复制到备份媒体上时通常要将文件锁定。快照允许你在复制的同时，保证运行关键任务的

Web服务器或数据库服务器继续工作。遗憾的是，LVM1只允许你创建只读快照。一旦创建了快照，就不能再写入东西了。

LVM2允许你创建在线逻辑卷的可读写快照。有了可读写的快照，就可以删除原先的逻辑卷，然后将快照作为替代挂载上。这个功能对快速故障转移或涉及修改数据的程序试验（如果失败，需要恢复修改过的数据）非常有用。

2. 条带化

LVM2提供的另一个引人注目的功能是条带化（striping）。有了条带化，可跨多个物理硬盘创建逻辑卷。当Linux LVM将文件写入逻辑卷时，文件中的数据块会被分散到多个硬盘上。每个后继数据块会被写到下一个硬盘上。

条带化有助于提高硬盘的性能，因为Linux可以将一个文件的多个数据块同时写入多个硬盘，而无需等待单个硬盘移动读写磁头到多个不同位置。这个改进同样适用于读取顺序访问的文件，因为LVM可同时从多个硬盘读取数据。

说明 LVM条带化不同于RAID条带化。LVM条带化不提供用来创建容错环境的校验信息。事实上，LVM条带化会增加文件因硬盘故障而丢失的概率。单个硬盘故障可能会造成多个逻辑卷无法访问。

3. 镜像

通过LVM安装文件系统并不意味着文件系统就不会再出问题。和物理分区一样，LVM逻辑卷也容易受到断电和磁盘故障的影响。一旦文件系统损坏，就有可能再也无法恢复。

LVM快照功能提供了一些安慰，你可以随时创建逻辑卷的备份副本，但对有些环境来说可能还不够。对于涉及大量数据变动的系统，比如数据库服务器，自上次快照之后可能要存储成百上千条记录。

这个问题的一个解决办法就是LVM镜像。镜像是一个实时更新的逻辑卷的完整副本。当你创建镜像逻辑卷时，LVM会将原始逻辑卷同步到镜像副本中。根据原始逻辑卷的大小，这可能需要一些时间才能完成。

一旦原始同步完成，LVM会为文件系统的每次写操作执行两次写入——一次写入到主逻辑卷，一次写入到镜像副本。可以想到，这个过程会降低系统的写入性能。就算原始逻辑卷因为某些原因损坏了，你手头也已经有了一个完整的最新副本！

8.3.3 使用 Linux LVM

现在你已经知道Linux LVM可以做什么了，本节将讨论如何创建LVM来帮助组织系统上的硬盘空间。Linux LVM包只提供了命令程序来创建和管理逻辑卷管理系统中所有组件。有些Linux发行版则包含了命令行命令对应的图形化前端，但为了完全控制你的LVM环境，最好习惯直接使用这些命令。

1. 定义物理卷

创建过程的第一步就是将硬盘上的物理分区转换成Linux LVM使用的物理卷区段。我们的朋友fdisk命令可以帮忙。在创建了基本的Linux分区之后，你需要通过t命令改变分区类型。

```
[...]
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): 8e
Changed system type of partition 1 to 8e (Linux LVM)

Command (m for help): p

Disk /dev/sdb: 5368 MB, 5368709120 bytes
255 heads, 63 sectors/track, 652 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xa8661341

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1             1           262     2104483+   8e  Linux LVM

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
$
```

分区类型8e表示这个分区将会被用作Linux LVM系统的一部分，而不是一个直接的文件系统（就像你在前面看到的83类型的分区）。

说明 如果下一步中的pvcreate命令不能正常工作，很可能是因为LVM2软件包没有默认安装。可以使用软件包名lvm2，按照第9章中介绍的软件安装方法安装这个包。

下一步是用分区来创建实际的物理卷。这可以通过pvcreate命令来完成。pvcreate定义了用于物理卷的物理分区。它只是简单地将分区标记成Linux LVM系统中的分区而已。

```
$ sudo pvcreate /dev/sdb1
dev_is_mpath: failed to get device for 8:17
Physical volume "/dev/sdb1" successfully created
$
```

说明 别被吓人的消息dev_is_mpath: failed to get device for 8:17或类似的消息唬住了。只要看到了successfully created就没问题。pvcreate命令会检查分区是否为多路（multi-path，mpath）设备。如果不是的话，就会发出上面那段消息。

如果你想查看创建进度的话，可以使用`pvdiskdisplay`命令来显示已创建的物理卷列表。

```
$ sudo pvdiskdisplay /dev/sdb1
"/dev/sdb1" is a new physical volume of "2.01 GiB"
--- NEW Physical volume ---
PV Name                /dev/sdb1
VG Name
PV Size                2.01 GiB
Allocatable            NO
PE Size                0
Total PE               0
Free PE                0
Allocated PE           0
PV UUID                0FiUq2-LBod-IOWt-8VeN-tglm-Q2ik-rGU2w7

$
```

`pvdiskdisplay`命令显示出`/dev/sdb1`现在已经被标记为物理卷。注意，输出中的VG Name内容为空，因为物理卷还不属于某个卷组。

2. 创建卷组

下一步是从物理卷中创建一个或多个卷组。究竟要为系统创建多少卷组并没有既定的规则，你可以将所有的可用物理卷加到一个卷组，也可以结合不同的物理卷创建多个卷组。

要从命令行创建卷组，需要使用`vgcreate`命令。`vgcreate`命令需要一些命令行参数来定义卷组名以及你用来创建卷组的物理卷名。

```
$ sudo vgcreate Vol1 /dev/sdb1
Volume group "Vol1" successfully created
$
```

输出结果平淡无奇。如果你想看看新创建的卷组的细节，可用`vgdisplay`命令。

```
$ sudo vgdisplay Vol1
--- Volume group ---
VG Name                Vol1
System ID
Format                lvm2
Metadata Areas         1
Metadata Sequence No   1
VG Access               read/write
VG Status               resizable
MAX LV                 0
Cur LV                 0
Open LV                 0
Max PV                 0
Cur PV                 1
Act PV                 1
VG Size                2.00 GiB
PE Size                4.00 MiB
Total PE               513
Alloc PE / Size         0 / 0
Free PE / Size          513 / 2.00 GiB
VG UUID                oe4I7e-5RA9-G9ti-ANoI-QKLz-qkX4-58Wj6e
```

\$

这个例子使用/dev/sdb1分区上创建的物理卷，创建了一个名为Vol1的卷组。
创建一个或多个卷组后，就可以创建逻辑卷了。

3. 创建逻辑卷

Linux系统使用逻辑卷来模拟物理分区，并在其中保存文件系统。Linux系统会像处理物理分区一样处理逻辑卷，允许你定义逻辑卷中的文件系统，然后将文件系统挂载到虚拟目录上。

要创建逻辑卷，使用lvcreate命令。虽然你通常不需要在其他Linux LVM命令中使用命令行选项，但lvcreate命令要求至少输入一些选项。表8-5显示了可用的命令行选项。

表8-5 lvcreate的选项

选 项	长选项名	描 述
-c	--chunksize	指定快照逻辑卷的单位大小
-C	--contiguous	设置或重置连续分配策略
-i	--stripes	指定条带数
-I	--stripesize	指定每个条带的大小
-l	--extents	指定分配给新逻辑卷的逻辑区段数，或者要用的逻辑区段的百分比
-L	--size	指定分配给新逻辑卷的硬盘大小
	--minor	指定设备的次设备号
-m	--mirrors	创建逻辑卷镜像
-M	--persistent	让次设备号一直有效
-n	--name	指定新逻辑卷的名称
-p	--permission	为逻辑卷设置读/写权限
-r	--readahead	设置预读扇区数
-R	--regionsize	指定将镜像分成多大的区
-s	snapshot	创建快照逻辑卷
-Z	--zero	将新逻辑卷的前1 KB数据设置为零

虽然命令行选项看起来可能有点吓人，但大多数情况下你用到的只是少数几个选项。

```
$ sudo lvcreate -l 100%FREE -n lvtest Vol1
Logical volume "lvtest" created
$
```

如果想查看你创建的逻辑卷的详细情况，可用lvdisplay命令。

```
$ sudo lvdisplay Vol1
--- Logical volume ---
LV Path                /dev/Vol1/lvtest
LV Name                 lvtest
VG Name                 Vol1
LV UUID                 4W2369-pLXy-jWmb-lIFN-SMNx-xZnN-3KN208
LV Write Access         read/write
LV Creation host, time  ... -0400
LV Status                available
```

```

# open                0
LV Size               2.00 GiB
Current LE            513
Segments              1
Allocation             inherit
Read ahead sectors    auto
- currently set to    256
Block device          253:2

$

```

现在可以看到你刚刚创建的逻辑卷了！注意，卷组名（Vol1）用来标识创建新逻辑卷时要使用的卷组。

-l选项定义了要为逻辑卷指定多少可用的卷组空间。注意，你可以按照卷组空闲空间的百分比来指定这个值。本例中为新逻辑卷使用了所有的空闲空间。

你可以用-l选项来按可用空间的百分比来指定这个大小，或者用-L选项以字节、千字节（KB）、兆字节（MB）或吉字节（GB）为单位来指定实际的大小。-n选项允许你为逻辑卷指定一个名称（在本例中称作lvtest）。

4. 创建文件系统

运行完lvcreate命令之后，逻辑卷就已经产生了，但它还没有文件系统。你必须使用相应的命令行程序来创建所需要的文件系统。

```

$ sudo mkfs.ext4 /dev/Vol1/lvtest
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
131376 inodes, 525312 blocks
26265 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=541065216
17 block groups
32768 blocks per group, 32768 fragments per group
7728 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 28 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
$

```

在创建了新的文件系统之后，可以用标准Linux mount命令将这个卷挂载到虚拟目录中，就跟它是物理分区一样。唯一的不同是你需要用特殊的路径来标识逻辑卷。


```
$ sudo mount /dev/Vol1/lvtest /mnt/my_partition
$
$ mount
/dev/mapper/vg_server01-lv_root on / type ext4 (rw)
[...]
/dev/mapper/Vol1-lvtest on /mnt/my_partition type ext4 (rw)
$
$ cd /mnt/my_partition
$
$ ls -al
total 24
drwxr-xr-x. 3 root root 4096 Jun 12 10:22 .
drwxr-xr-x. 3 root root 4096 Jun 11 09:58 ..
drwx-----. 2 root root 16384 Jun 12 10:22 lost+found
$
```

注意,mkfs.ext4和mount命令中用到的路径都有点奇怪。路径中使用了卷组名和逻辑卷名,而不是物理分区路径。文件系统被挂载之后,就可以访问虚拟目录中的这块新区域了。

5. 修改LVM

Linux LVM的好处在于能够动态修改文件系统,因此最好有工具能够让你实现这些操作。在Linux有一些工具允许你修改现有的逻辑卷管理配置。

如果你无法通过一个很炫的图形化界面来管理你的Linux LVM环境,也不是什么都干不了。在本章中你已经看到了一些Linux LVM命令行程程序的实际用法。还有一些其他的命令可以用来管理LVM的设置。表8-6列出了在Linux LVM包中的常见命令。

表8-6 Linux LVM命令

命 令	功 能
vgchange	激活和禁用卷组
vgremove	删除卷组
vgextend	将物理卷加到卷组中
vgreduce	从卷组中删除物理卷
lvextend	增加逻辑卷的大小
lvreduce	减小逻辑卷的大小

通过使用这些命令行程程序,就能完全控制你的Linux LVM环境。

窍门 在手动增加或减小逻辑卷的大小时,要特别小心。逻辑卷中的文件系统需要手动修整来处理大小上的改变。大多数文件系统都包含了能够重新格式化文件系统的命令行程程序,比如用于ext2、ext3和ext4文件系统的resize2fs程序。

8.4 小结

在Linux上使用存储设备需要懂一点文件系统的知识。当工作在Linux系统下时，懂得如何在命令行下创建和处理文件系统能帮上你的忙。本章讨论了如何使用Linux命令行处理文件系统。

Linux系统和Windows的不同之处在于前者支持大量不同的存储文件和目录的方法。每个文件系统方法都有不同的特性，使其适用于不同的场景。另外，每种文件系统都使用不同的命令与存储设备打交道。

在将文件系统安装到存储设备之前，你得先备好设备。`fdisk`命令用来对存储设备进行分区，以便安装文件系统。在分区存储设备时，必须定义在上面使用什么类型的文件系统。

划分完存储设备分区后，你可以为该分区选用一种文件系统。流行的Linux文件系统包括ext3和ext4。两者都提供了日志文件系统功能，降低它们在Linux系统崩溃时遇到错误或问题的几率。

在存储设备分区上直接创建文件系统的限制因素是，如果硬盘空间用完了，你无法轻易地改变文件系统的大小。但Linux支持逻辑卷管理，这是一种跨多个存储设备创建虚拟分区的方法。这种方法允许你轻松地扩展一个已有文件系统，而不用完全重建。Linux LVM包提供了跨多个存储设备创建逻辑卷的命令行命令。

现在你已经了解了核心的Linux命令行命令，差不多是时候开始编写一些shell脚本程序了。但在开始编码前，我们还有另一件事情需要讨论：安装软件。如果你打算写shell脚本，就需要一个环境来完成你的杰作。下一章将讨论如何在不同的Linux环境中从命令行下安装和管理软件包。

本章内容

- ❑ 安装软件
- ❑ 使用Debian包
- ❑ 使用Red Hat包

在Linux的早期，安装软件是一件痛苦的事。幸好Linux开发人员已经通过把软件打包成更易于安装的预编译包，我们的生活因此舒坦了一些。但你多少还是得花点功夫安装软件包，尤其是准备从命令行下安装的时候。本章将介绍Linux上能见到的各种包管理系统（package management system, PMS），以及用来进行软件安装、管理和删除的命令行工具。

9.1 包管理基础

在深入了解Linux软件包管理之前，本章将先介绍一些基础知识。各种主流Linux发行版都采用了某种形式的包管理系统来控制软件和库的安装。PMS利用一个数据库来记录各种相关内容：

- ❑ Linux系统上已安装了什么软件包；
- ❑ 每个包安装了什么文件；
- ❑ 每个已安装软件包的版本。

软件包存储在服务器上，可以利用本地Linux系统上的PMS工具通过互联网访问。这些服务器称为仓库（repository）。可以用PMS工具来搜索新的软件包，或者是更新系统上已安装软件包。

软件包通常会依赖其他的包，为了前者能够正常运行，被依赖的包必须提前安装在系统中。PMS工具将会检测这些依赖关系，并在安装需要的包之前先安装好所有额外的软件包。

PMS的不足之处在于目前还没有统一的标准工具。不管你用的是哪个Linux发行版，本书到目前为止所讨论的bash shell命令都能工作，但对于软件包管理可就不一定了。

PMS工具及相关命令在不同的Linux发行版上有很大的不同。Linux中广泛使用的两种主要的PMS基础工具是dpkg和rpm。

基于Debian的发行版（如Ubuntu和Linux Mint）使用的是dpkg命令，这些发行版的PMS工具也是以该命令为基础的。dpkg会直接和Linux系统上的PMS交互，用来安装、管理和删除软件包。

基于Red Hat的发行版(如Fedora、openSUSE及Mandriva)使用的是rpm命令,该命令是其PMS的底层基础。类似于dpkg命令, rpm命令能够列出已安装包、安装新包和删除已有软件。

注意,这两个命令是它们各自PMS的核心,并非全部的PMS。许多使用dpkg或rpm命令的Linux发行版都有各自基于这些命令的特定PMS工具,这些工具能够助你事半功倍。随后几节将带你逐步了解主流Linux发行版上的各种PMS工具命令。

9.2 基于 Debian 的系统

dpkg命令是基于Debian系PMS工具的核心。包含在这个PMS中的其他工具有:

- ❑ apt-get
- ❑ apt-cache
- ❑ aptitude

到目前为止,最常用的命令行工具是aptitude,这是有原因的。aptitude工具本质上是apt工具和dpkg的前端。dpkg是软件包管理系统工具,而aptitude则是完整的软件包管理系统。

命令行下使用aptitude命令有助于避免常见的软件安装问题,如软件依赖关系缺失、系统环境不稳定及其他一些不必要的麻烦。本节将会介绍如何在命令行下使用aptitude命令工具。

9.2.1 用 aptitude 管理软件包

Linux系统管理员面对的一个常见任务是确定系统上已经安装了什么软件包。好在aptitude有个很方便的交互式界面可以轻松完成这项任务。

如果使用的Linux发行版中已经安装了aptitude,只需要在shell提示符键入aptitude并按下回车键就行了。紧接着就会进入aptitude的全屏模式,如图9-1所示。

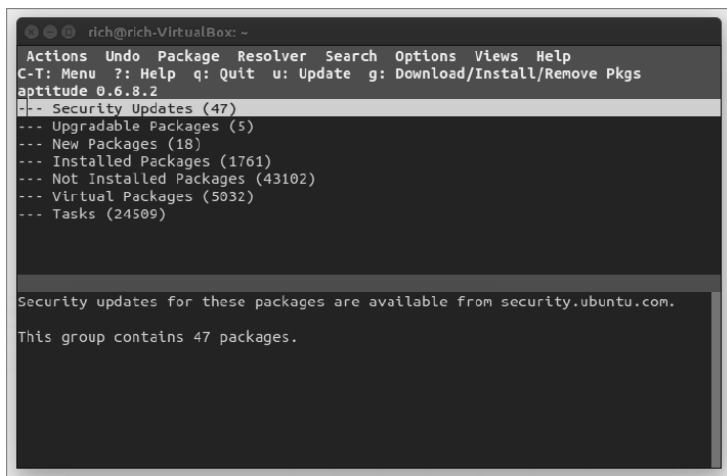


图9-1 aptitude主窗口

可以用方向键在菜单上移动。选择菜单选项 **Installed Packages** 来查看已安装了什么软件包。你可以看到几组软件包，比如编辑器等。每组后面的括号里都有个数字，表示这个组包含多少个软件包。

使用方向键高亮显示一个组，按回车键来查看每个软件包分组。你会看到每个单独的软件包名称以及它们的版本号。在软件包上按回车键可以获得更详细的信息，比如软件包的描述、主页、大小和维护人员等。

看完了已安装软件包后，按 **q** 键来退出显示。你可以继续用方向键和回车键打开或关闭软件包和它们所在的分组。如果想退出，多按几次 **q** 键，直到看到弹出的屏幕提示 “Really quit Aptitude?”。

如果你已经知道了系统上的那些软件包，只想快速显示某个特定包的详细信息，就没必要到 aptitude 的交互式界面。可以在命令行下以单个命令的方式使用 aptitude。

```
aptitude show package_name
```

下面的例子显示了包 `mysql-client` 的详情。

```
$ aptitude show mysql-client
Package: mysql-client
State: not installed
Version: 5.5.38-0ubuntu0.14.04.1
Priority: optional
Section: database
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Architecture: all
Uncompressed Size: 129 k
Depends: mysql-client-5.5
Provided by: mysql-client-5.5
Description: MySQL database client (metapackage depending on the latest version)
 This is an empty package that depends on the current "best" version of
 mysql-client (currently mysql-client-5.5), as determined by the MySQL
 maintainers. Install this package if in doubt about which MySQL version you
 want, as this is the one considered to be in the best shape by the Maintainers.
 Homepage: http://dev.mysql.com/

$
```

说明 `aptitude show` 命令显示上面例子中的软件包还没有安装到系统上。它输出的软件包相关的详细信息来自于软件仓库。

无法通过 aptitude 看到的一个细节是所有跟某个特定软件包相关的所有文件的列表。要得到这个列表，就必须用 `dpkg` 命令。

```
dpkg -L package_name
```

下面这个例子是用 `dpkg` 列出 `vim-common` 软件包所安装的全部文件。

```
$
$ dpkg -L vim-common
```

```

/.
/usr
/usr/bin
/usr/bin/xxd
/usr/bin/help2tags
/usr/lib
/usr/lib/mime
/usr/lib/mime/packages
/usr/lib/mime/packages/vim-common
/usr/share
/usr/share/man
/usr/share/man/ru
/usr/share/man/ru/man1
/usr/share/man/ru/man1/vim.1.gz
/usr/share/man/ru/man1/vimdiff.1.gz
/usr/share/man/ru/man1/xxd.1.gz
/usr/share/man/it
/usr/share/man/it/man1
[...]
$

```

同样可以进行反向操作，查找某个特定文件属于哪个软件包。

```
dpkg --search absolute_file_name
```

注意，在使用的时候必须用绝对文件路径。

```

$
$ dpkg --search /usr/bin/xxd
vim-common: /usr/bin/xxd
$

```

从输出中可以看出/usr/bin/xxd文件是作为vim-common包的一部分被安装的。

9.2.2 用 aptitude 安装软件包

了解了怎样在系统中列出软件包信息之后，本节将带你逐步学习怎样安装软件包。首先，要确定准备安装的软件包名称。怎么才能找到特定的软件包呢？用aptitude命令加search选项。

```
aptitude search package_name
```

search选项的妙处在于你无需在*package_name*周围加通配符。通配符会隐式添加。下面是用aptitude来查找wine软件包的例子。

```

$
$ aptitude search wine
p  gnome-wine-icon-theme          - red variation of the GNOME- ...
v  libkwineffects1-api            -
p  libkwineffects1a              - library used by effects...
p  q4wine                        - Qt4 GUI for wine (W.I.N.E)
p  shiki-wine-theme              - red variation of the Shiki- ...
p  wine                          - Microsoft Windows Compatibility ...
p  wine-dev                      - Microsoft Windows Compatibility ...
p  wine-gecko                    - Microsoft Windows Compatibility ...
p  wine1.0                       - Microsoft Windows Compatibility ...

```

```

p   wine1.0-dev                - Microsoft Windows Compatibility ...
p   wine1.0-gecko              - Microsoft Windows Compatibility ...
p   wine1.2                    - Microsoft Windows Compatibility ...
p   wine1.2-dbg               - Microsoft Windows Compatibility ...
p   wine1.2-dev               - Microsoft Windows Compatibility ...
p   wine1.2-gecko             - Microsoft Windows Compatibility ...
p   winefish                  - LaTeX Editor based on Bluefish
$

```

注意，在每个包名字之前都有一个p或i。如果看到一个i，说明这个包现在已经安装到了你的系统上了。如果看到一个p或v，说明这个包可用，但还没安装。我们在上面的列表中可以看到系统中尚未安装wine，但是在软件仓库中可以找到这个包。

在系统上用aptitude从软件仓库中安装软件包非常简单。

```
aptitude install package_name
```

一旦通过search选项找到了软件包名称，只要将它通过install选项插入aptitude命令。

```

$
$ sudo aptitude install wine
The following NEW packages will be installed:
  cabextract{a} esound-clients{a} esound-common{a} gnome-exe-thumbnailer
{a}
  icoutils{a} imagemagick{a} libaudio2{a} libaudiofile0{a} libcdt4{a}
  libesd0{a} libgraph4{a} libgvc5{a} liblmbase6{a} libmagickcore3-extra
{a}
  libmpg123-0{a} libnetpbm10{a} libopenall{a} libopenexr6{a}
  libpathplan4{a} libxdot4{a} netpbm{a} ttf-mscorefonts-installer{a}
  ttf-symbol-replacement{a} winbind{a} wine wine1.2{a} wine1.2-gecko{a}
0 packages upgraded, 27 newly installed, 0 to remove and 0 not upgraded.
Need to get 0B/27.6MB of archives. After unpacking 121MB will be used.
Do you want to continue? [Y/n/?] Y
Preconfiguring packages ...
[...]
All done, no errors.
All fonts downloaded and installed.
Updating fontconfig cache for /usr/share/fonts/truetype/msttcorefonts
Setting up winbind (2:3.5.4~dfsg-1ubuntu7) ...
  * Starting the Winbind daemon winbind
  [ OK ]
Setting up wine (1.2-0ubuntu5) ...
Setting up gnome-exe-thumbnailer (0.6-0ubuntu1) ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place

$

```

说明 在上面的例子中，在aptitude命令之前出现了sudo命令。sudo命令允许你以root用户身份运行一个命令。可以用sudo命令进行管理任务，比如安装软件。

要检查安装过程是否正常，只要再次使用search选项就可以了。这次你应该可以看到在wine

软件包出现了i u，这说明它已经安装好了。

你可能还会注意到这里的另外一些包前面也有i u。这是因为aptitude自动解析了必要的包依赖关系，并安装了需要的额外的库和软件包。这是许多包管理系统都有的非常好的功能。

9.2.3 用 aptitude 更新软件

尽管aptitude可以帮忙解决安装软件时遇到的问题，但解决有依赖关系的多个包的更新会比较烦琐。要用软件仓库中的新版本妥善地更新系统上所有的软件包，可用safe-upgrade选项。

```
aptitude safe-upgrade
```

注意，这个命令不需要使用软件包名称作为参数。因为safe-upgrade选项会将所有已安装的包更新到软件仓库中的最新版本，更有利于系统稳定。

这里是aptitude safe-upgrade命令的输出示例。

```
$
$ sudo aptitude safe-upgrade
The following packages will be upgraded:
  evolution evolution-common evolution-plugins gsfonts libevolution
  xserver-xorg-video-geode
6 packages upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 9,312kB of archives. After unpacking 0B will be used.
Do you want to continue? [Y/n/?] Y
Get:1 http://us.archive.ubuntu.com/ubuntu/ maverick/main
  libevolution i386 2.30.3-1ubuntu4 [2,096kB]
[...]
Preparing to replace xserver-xorg-video-geode 2.11.9-2
(using ../xserver-xorg-video-geode_2.11.9-3_i386.deb) ...
Unpacking replacement xserver-xorg-video-geode ...
Processing triggers for man-db ...
Processing triggers for desktop-file-utils ...
Processing triggers for python-gmenu ...
[...]
Current status: 0 updates [-6].
$
```

还有一些不那么保守的软件升级选项：

☐ aptitude full-upgrade

☐ aptitude dist-upgrade

这些选项执行相同的任务，将所有软件包升级到最新版本。它们同safe-upgrade的区别在于，它们不会检查包与包之间的依赖关系。整个包依赖关系问题非常麻烦。如果不是很确定各种包的依赖关系，那还是坚持用safe-upgrade选项吧。

说明 显然，应该定期运行aptitude的safe-upgrade选项来保持系统处于最新状态。这点在安装了一个全新的发行版之后尤其重要。通常在发行版推出最新的完整发布之后，就会跟着出现很多新的安全补丁和更新。

9.2.4 用 aptitude 卸载软件

用aptitude卸载软件包与安装及更新它们一样容易。你要作出的唯一选择就是要不要保留软件数据和配置文件。

要想只删除软件包而不删除数据和配置文件，可以使用aptitude的remove选项。要删除软件包和相关的数据和配置文件，可用purge选项。

```
$ sudo aptitude purge wine
[sudo] password for user:
The following packages will be REMOVED:
  cabextract{u} esound-clients{u} esound-common{u} gnome-exe-thumbnailer
{u}
  icoutils{u} imagemagick{u} libaudio2{u} libaudiofile0{u} libcdt4{u}
  libesd0{u} libgraph4{u} libgvc5{u} libilmbase6{u} libmagickcore3-extra
{u}
  libmpeg123-0{u} libnetpbm10{u} libopenall{u} libopenexr6{u}
  libpathplan4{u} libxdot4{u} netpbm{u} ttf-mscorefonts-installer{u}
  ttf-symbol-replacement{u} winbind{u} wine{p} wine1.2{u} wine1.2-gecko
{u}
0 packages upgraded, 0 newly installed, 27 to remove and 6 not upgraded.
Need to get 0B of archives. After unpacking 121MB will be freed.
Do you want to continue? [Y/n/?] Y
(Reading database ... 120968 files and directories currently installed.)
Removing ttf-mscorefonts-installer ...
[...]
Processing triggers for fontconfig ...
Processing triggers for ureadahead ...
Processing triggers for python-support ...

$
```

要看软件包是否已删除，可以再用aptitude的search选项。如果在软件包名称的前面看到一个c，意味着软件已删除，但配置文件尚未从系统中清除；如果前面是个p的话，说明配置文件也已删除。

9.2.5 aptitude 仓库

aptitude默认的软件仓库位置是在安装Linux发行版时设置的。具体位置存储在文件/etc/apt/sources.list中。

很多情况下，根本不需要添加或删除软件仓库，所以也没必要接触这个文件。但aptitude只会从这些仓库中下载文件。另外，在搜索软件进行安装或更新时，aptitude同样只会检查这些库。如果需要为你的PMS添加一些额外的软件仓库，就在这个文件中设置吧。

窍门 Linux发行版的开发人员下了大工夫，以保证添加到软件仓库的包版本不会互相冲突。通常通过库来升级或安装软件包是最安全的。即使在其他地方有更新的版本，也应该等到该版本出现在你的Linux发行版仓库中的时候再安装。

下面是Ubuntu系统中sources.list文件的例子。

```
$ cat /etc/apt/sources.list
#deb cdrom:[Ubuntu 14.04 LTS _Trusty Tahr_ - Release i386 (20140417)]/
  trusty main restricted

# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.
deb http://us.archive.ubuntu.com/ubuntu/ trusty main restricted
deb-src http://us.archive.ubuntu.com/ubuntu/ trusty main restricted

## Major bug fix updates produced after the final release of the
## distribution.
deb http://us.archive.ubuntu.com/ubuntu/ trusty-updates main restricted
deb-src http://us.archive.ubuntu.com/ubuntu/ trusty-updates main restricted

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team. Also, please note that software in universe WILL NOT receive any
## review or updates from the Ubuntu security team.
deb http://us.archive.ubuntu.com/ubuntu/ trusty universe
deb-src http://us.archive.ubuntu.com/ubuntu/ trusty universe
deb http://us.archive.ubuntu.com/ubuntu/ trusty-updates universe
deb-src http://us.archive.ubuntu.com/ubuntu/ trusty-updates universe
[...]
## Uncomment the following two lines to add software from Canonical's
## 'partner' repository.
## This software is not part of Ubuntu, but is offered by Canonical and the
## respective vendors as a service to Ubuntu users.
# deb http://archive.canonical.com/ubuntu trusty partner
# deb-src http://archive.canonical.com/ubuntu trusty partner

## This software is not part of Ubuntu, but is offered by third-party
## developers who want to ship their latest software.
deb http://extras.ubuntu.com/ubuntu trusty main
deb-src http://extras.ubuntu.com/ubuntu trusty main
$
```

首先，我们注意到文件里满是帮助性的注释和警告。使用下面的结构来指定仓库源。

```
deb (or deb-src) address distribution_name package_type_list
```

deb或deb-src的值表明了软件包的类型。deb值说明这是一个已编译程序源，而deb-src值则说明这是一个源代码的源。

address条目是软件仓库的Web地址。distribution_name条目是这个特定软件仓库的发行版版本的名称。在这个例子中，发行版名称是trusty。这未必就是说你使用的发行版就是Ubuntu Trusty Tahr，它只是说明这个Linux发行版正在用Ubuntu Trusty Tahr软件仓库！举个例子，在Linux Mint的sources.list文件中，你能看到混用了Linux Mint和Ubuntu的软件仓库。

最后，package_type_list条目可能并不止一个词，它还表明仓库里面有什么类型的包。你可以看到诸如main、restricted、universe和partner这样的值。

当需要给你的source_list文件添加软件仓库时，你可以自己发挥，但一般会带来问题。通常

软件仓库网站或各种包开发人员网站上都会有一行文本,你可以直接复制,然后粘贴到sources.list文件中。最好选择较安全的途径并且只复制/粘贴。

aptitude前端界面提供了智能命令行选项来配合基于Debian的dpkg工具。现在是时候了解基于Red Hat的发行版的rpm工具和它的各种前端界面了。

9.3 基于 Red Hat 的系统

和基于Debian的发行版类似,基于Red Hat的系统也有几种不同的可用前端工具。常见的有以下3种。

- ❑ yum: 在Red Hat和Fedora中使用。
- ❑ urpm: 在Mandriva中使用。
- ❑ zypper: 在openSUSE中使用。

这些前端都是基于rpm命令行工具的。下一节会讨论如何用这些基于rpm的工具来管理软件包。重点是在yum上,但也会讲到zypper和urpm。

9.3.1 列出已安装包

要找出系统上已安装的包,可在shell提示符下输入如下命令:

```
yum list installed
```

输出的信息可能会在屏幕上一闪而过,所以最好是将已安装包的列表重定向到一个文件中。可以用more或less命令(或一个GUI编辑器)按照需要查看这个列表。

```
yum list installed > installed_software
```

要列出openSUSE或Mandriva发行版上的已安装包,可参考表9-1中的命令。遗憾的是,Mandriva中采用的urpm工具无法生成当前已安装软件列表。因此,你需要转向底层的rpm工具。

表9-1 如何用zypper和urpm列出已安装软件

版 本	前端工具	命 令
Mandriva	urpm	rpm -qa > installed_software
openSUSE	zypper	zypper search -I > installed_software

yum擅长找出某个特定软件包的详细信息。它能给出关于包的非常详尽的描述,另外你还可以通过一条简单的命令查看包是否已安装。

```
# yum list xterm
Loaded plugins: langpacks, presto, refresh-packagekit
Adding en_US to language list
Available Packages
xterm.i686 253-1.el6
#
# yum list installed xterm
Loaded plugins: refresh-packagekit
```

```
Error: No matching Packages to list
#
```

用urpm和zypper列出详细软件包信息的命令见表9-2。还可用zypper命令的info选项从库中获得一份更详细的包信息。

表9-2 如何用zypper和urpm查看各种包详细信息

信息类型	前端工具	命 令
包信息	urpm	urpmq -i <i>package_name</i>
是否安装	urpm	rpm -q <i>package_name</i>
包信息	zypper	zypper search -s <i>package_name</i>
是否安装	zypper	同样的命令，注意在Status列查找i

最后，如果需要找出系统上的某个特定文件属于哪个软件包，万能的yum可以做到！只要输入命令：

```
yum provides file_name
```

这里有个查找配置文件/etc/yum.conf归属的例子。

```
#
# yum provides /etc/yum.conf
Loaded plugins: fastestmirror, refresh-packagekit, security
Determining fastest mirrors
 * base: mirror.web-ster.com
 * extras: centos.chi.host-engine.com
 * updates: mirror.umd.edu
yum-3.2.29-40.el6.centos.noarch : RPM package installer/updater/manager
Repo      : base
Matched from:
Filename   : /etc/yum.conf

yum-3.2.29-43.el6.centos.noarch : RPM package installer/updater/manager
Repo      : updates
Matched from:
Filename   : /etc/yum.conf

yum-3.2.29-40.el6.centos.noarch : RPM package installer/updater/manager
Repo      : installed
Matched from:
Other      : Provides-match: /etc/yum.conf

#

#
```

yum会分别查找三个仓库：base、updates和installed。从其中两个仓库中得到的答案都是：该文件是yum软件包提供的！

9.3.2 用 yum 安装软件

用yum安装软件包极其简单。下面这个简单的命令会从仓库中安装软件包、所有它需要的库以及依赖的其他包：

```
yum install package_name
```

下面的例子是安装在第2章中讨论过的xterm包。

```
$ su -
Password:
# yum install xterm
Loaded plugins: fastestmirror, refresh-packagekit, security
Determining fastest mirrors
 * base: mirrors.bluehost.com
 * extras: mirror.5ninesolutions.com
 * updates: mirror.san.fastserv.com
Setting up Install Process
Resolving Dependencies
--> Running transaction check
--> Package xterm.i686 0:253-1.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
[...]
Installed:
  xterm.i686 0:253-1.el6

Complete!
#
```

9

说明 在上面的例子中，我们在运行yum命令之前使用了su-命令。这个命令允许你切换到root用户。在Linux系统上，#表明你是以root用户身份登录的。应该只有在运行管理性的任务时才临时切换到root用户（比如安装和更新软件）。也可以使用sudo命令。

也可以手动下载rpm安装文件并用yum安装，这叫作本地安装。基本的命令是：

```
yum localinstall package_name.rpm
```

你现在应该能发现yum的优点之一就是它的命令富有逻辑性，而且对用户也友好。

表9-3显示了如何用urpm和zypper安装包。注意，如果不是以root用户身份登录，你会在使用urpm时得到一个“command not found”的错误消息。

表9-3 如何用zypper和urpm安装软件

前端工具	命 令
urpm	urpmi package_name
zypper	zypper install package_name

9.3.3 用 yum 更新软件

在大多数Linux发行版上，如果你是在GUI上工作，就会看到一些好看的小通知图标，告诉你需要更新了。在命令行下的话，就得费点事了。

要列出所有已安装包的可用更新，输入如下命令：

```
yum list updates
```

如果这个命令没有输出就太好了，因为它说明你没有任何需要更新的！但如果发现某个特定软件包需要更新，输入如下命令：

```
yum update package_name
```

如果想对更新列表中的所有包进行更新，只要输入如下命令：

```
yum update
```

Mandriva和openSUSE上用来更新软件包的命令列在了表9-4中。在使用urpm时，软件仓库数据库会自动更新，软件包也会更新。

表9-4 如何用zypper和urpm更新软件

前端工具	命 令
urpm	urpmi --auto-update --update
zypper	zypper update

9.3.4 用 yum 卸载软件

yum工具还提供了一种简单的方法来卸载系统中不再想要的*应用*。和aptitude一样，你需要决定是否保留软件包的数据和配置文件。

只删除软件包而保留配置文件和数据文件，就用如下命令：

```
yum remove package_name
```

要删除软件和他所有的文件，就用erase选项：

```
yum erase package_name
```

在表9-5中不难发现，用urpm和zypper删除软件同样简单。这两个工具的作用类似于yum的erase选项。

表9-5 如何用zypper和urpm卸载软件

前端工具	命 令
urpm	urpme package_name
zypper	zypper remove package_name

有了PMS包的生活尽管安逸了不少，但也不是风平浪静。偶尔也会有一些波澜，好在总有解决的办法。

9.3.5 处理损坏的包依赖关系

有时在安装多个软件包时，某个包的软件依赖关系可能会被另一个包的安装覆盖掉。这叫作损坏的包依赖关系（broken dependency）。

如果系统出现了这个问题，先试试下面的命令：

```
yum clean all
```

然后试着用yum命令的update选项。有时，只要清理了放错位置的文件就可以了。

如果这还解决不了问题，试试下面的命令：

```
yum deplist package_name
```

这个命令显示了所有包的库依赖关系以及什么软件可以提供这些库依赖关系。一旦知道某个包需要的库，你就能安装它们了。下面是确定xterm包依赖关系的例子。

```
# yum deplist xterm

Loaded plugins: fastestmirror, refresh-packagekit, security
Loading mirror speeds from cached hostfile
* base: mirrors.bluehost.com
* extras: mirror.5ninesolutions.com
* updates: mirror.san.fastserv.com
Finding dependencies:
package: xterm.i686 253-1.el6
dependency: libncurses.so.5
  provider: ncurses-libs.i686 5.7-3.20090208.el6
dependency: libfontconfig.so.1
  provider: fontconfig.i686 2.8.0-3.el6
dependency: libXft.so.2
  provider: libXft.i686 2.3.1-2.el6
dependency: libXt.so.6
  provider: libXt.i686 1.1.3-1.el6
dependency: libX11.so.6
  provider: libX11.i686 1.5.0-4.el6
dependency: rtld(GNU_HASH)
  provider: glibc.i686 2.12-1.132.el6
  provider: glibc.i686 2.12-1.132.el6_5.1
  provider: glibc.i686 2.12-1.132.el6_5.2
dependency: libICE.so.6
  provider: libICE.i686 1.0.6-1.el6
dependency: libXaw.so.7
```

```
provider: libXaw.i686 1.0.11-2.el6
dependency: libtinfo.so.5
provider: ncurses-libs.i686 5.7-3.20090208.el6
dependency: libutempter.so.0
provider: libutempter.i686 1.1.5-4.1.el6
dependency: /bin/sh
provider: bash.i686 4.1.2-15.el6_4
dependency: libc.so.6 (GLIBC_2.4)
provider: glibc.i686 2.12-1.132.el6
provider: glibc.i686 2.12-1.132.el6_5.1
provider: glibc.i686 2.12-1.132.el6_5.2
dependency: libXmu.so.6
provider: libXmu.i686 1.1.1-2.el6
#
```

如果这样仍未解决问题，还有最后一招：

```
yum update --skip-broken
```

--skip-broken选项允许你忽略依赖关系损坏的那个包，继续去更新其他软件包。这可能救不了损坏的包，但至少可以更新系统上的其他包。

表9-6中列出了用urpm和zypper来尝试修复损坏的依赖关系的命令。用zypper时，只有一个命令能够用来验证和修复损坏的依赖关系。用urpm时，如果clean选项不工作，你可以跳过更新那些有问题的包。要这么做的话，就必须将有问题包的名字添加到文件/etc/urpmi/skip.list。

表9-6 用zypper和urpm修复损坏的依赖关系

前端工具	命 令
urpm	urpmi -clean
zypper	zypper verify

9.3.6 yum 软件仓库

类似于aptitude系统，yum也是在安装发行版的时候设置的软件仓库。这些预设的仓库就能很好地满足你的大部分需求。但如果需要从其他仓库安装软件，有些事情你得知道。

窍门 聪明的系统管理委员会坚持使用通过审核的仓库。通过审核的仓库是指该发行版官方网站上指定的库。如果你添加了未通过审核的库，就失去了稳定性方面的保证，可能陷入损坏的依赖关系惨剧中。

要想知道你现在正从哪些仓库中获取软件，输入如下命令：

```
yum repolist
```

如果仓库中没有需要的软件，你可以编辑一下配置文件。yum的仓库定义文件位于/etc/yum.repos.d。你需要添加正确的URL，并获得必要的加密密钥。

像rpmfusion.org这种优秀的仓库站点会列出必要的使用步骤。有时这些仓库网站会提供一个可下载的rpm文件，可以用yum localinstall命令进行安装。这个rpm文件在安装过程会为你完成所有的仓库设置工作。现在方便多了！

urpm称它的仓库为媒体。查看urpm媒体和zypper仓库的命令列在了表9-7中。注意，用这两个前端工具时不需要编辑配置文件。只需要输入命令就可以添加媒体或仓库。

表9-7 zypper和urpm的库

动 作	前端工具	命 令
显示仓库	urpm	urpmq --list-media
添加仓库	urpm	urpmi.addmedia path_name
显示仓库	zypper	zypper repos
添加仓库	zypper	zypper addrepo path_name

基于Debian的和基于Red Hat的系统都使用包管理系统来简化管理软件的过程。现在我们就离开包管理系统的世界，看看稍微麻烦一点的：直接从源码安装。

9.4 从源码安装

第4章中讨论了tarball包——如何通过tar命令行命令进行创建和解包。在好用的rpm和dpkg工具出现之前，管理员必须知道如何从tarball来解包和安装软件。

如果你经常在开源软件环境中工作，就很可能遇到打包成tarball形式的软件。本节就带你逐步了解这种软件的解包与安装过程。

在这个例子中用到了软件包sysstat。sysstat提供了各种系统监测工具，非常好用。

首先需要将sysstat的tarball下载到你的Linux系统上。通常能在各种Linux网站上找到sysstat包，但最好是直接到程序的官方站点下载（<http://sebastien.godard.pagesperso-orange.fr/>）。

单击Download（下载）链接，就会转入文件下载页面。本书编写时的最新版本是11.1.1，发行文件名是sysstat-11.1.1.tar.gz。

将文件下载到你的Linux系统上，然后解包。要解包一个软件的tarball，用标准的tar命令。

```
#
# tar -zxvf sysstat-11.1.1.tar.gz
sysstat-11.1.1/
sysstat-11.1.1/cifsiostat.c
sysstat-11.1.1/FAQ
sysstat-11.1.1/ioconf.h
sysstat-11.1.1/rd_stats.h
sysstat-11.1.1/COPYING
sysstat-11.1.1/common.h
sysstat-11.1.1/sysconfig.in
sysstat-11.1.1/mpstat.h
sysstat-11.1.1/rndr_stats.h
[...]
```

```

sysstat-11.1.1/activity.c
sysstat-11.1.1/sar.c
sysstat-11.1.1/iostat.c
sysstat-11.1.1/rd_sensors.c
sysstat-11.1.1/prealloc.in
sysstat-11.1.1/sa2.in
#
#

```

现在，tarball已经完成了解包，所有文件都已顺利放到了一个叫sysstat-11.1.1的目录中，你可以跳到那个目录下继续了。

首先，用cd命令进入这个新目录中，然后列出这个目录的内容。

```

$ cd sysstat-11.1.1
$ ls
activity.c      iconfig          prealloc.in     sa.h
build           INSTALL         pr_stats.c      sar.c
CHANGES       ioconf.c        pr_stats.h      sa_wrap.c
cifsiostat.c   ioconf.h        rd_sensors.c    sysconfig.in
cifsiostat.h   iostat.c        rd_sensors.h    sysstat-11.1.1.lsm
common.c       iostat.h        rd_stats.c      sysstat-11.1.1.spec
common.h       json_stats.c    rd_stats.h      sysstat.in
configure      json_stats.h    README          sysstat.ioconf
configure.in   Makefile.in     rndr_stats.c    sysstat.service.in
contrib        man             rndr_stats.h    sysstat.sysconfig.in
COPYING        mpstat.c        sa1.in          version.in
count.c        mpstat.h        sa2.in          xml
count.h        nfsiostat-sysstat.c sa_common.c     xml_stats.c
CREDITS        nfsiostat-sysstat.h sadc.c          xml_stats.h
cron           nls             sadf.c
FAQ            pidstat.c       sadf.h
format.c       pidstat.h       sadf_misc.c
$

```

在这个目录的列表中，应该能看到README或AAAREADME文件。读这个文件非常重要。该文件中包含了软件安装所需要的操作。

按照README文件中的建议，下一步是为系统配置sysstat。它会检查你的Linux系统，确保它拥有合适的编译器能够编译源代码，另外还要具备正确的库依赖关系。

```

# ./configure

Check programs:
.
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
[...]
checking for ANSI C header files... (cached) yes
checking for dirent.h that defines DIR... yes
checking for library containing opendir... none required
checking ctype.h usability... yes

```

```

checking ctype.h presence... yes
checking for ctype.h... yes
checking errno.h usability... yes
checking errno.h presence... yes
checking for errno.h... yes
[...]
Check library functions:
.
checking for strchr... yes
checking for strcspn... yes
checking for strspn... yes
checking for strstr... yes
checking for sensors support... yes
checking for sensors_get_detected_chips in -lsensors... no
checking for sensors lib... no
.
Check system services:
.
checking for special C compiler options needed for large files... no
checking for _FILE_OFFSET_BITS value needed for large files... 64
.
Check configuration:
[...]
Now create files:
[...]
config.status: creating Makefile

Sysstat version:          11.1.1
Installation prefix:      /usr/local
rc directory:             /etc/rc.d
Init directory:           /etc/rc.d/init.d
Systemd unit dir:
Configuration directory:  /etc/sysconfig
Man pages directory:      ${datarootdir}/man
Compiler:                 gcc
Compiler flags:           -g -O2

#

```

如果哪里有错了，在configure步骤中会显示一条错误消息说明缺失了什么东西。如果你所用的Linux发行版中没有安装GNU C编译器，那只会得到一条错误信息。对于其他问题，你会看到好几条消息，说明安装了什么，没有安装什么。

下一步就是用make命令来构建各种二进制文件。make命令会编译源码，然后链接器会为此个包创建最终的可执行文件。和configure命令一样，make命令会在编译和链接所有的源码文件的过程中产生大量的输出。

```

# make
-gcc -o sadc.o -c -g -O2 -Wall -Wstrict-prototypes -pipe -O2
-DSA_DIR=\"/var/log/sa\" -DSADC_PATH=\"/usr/local/lib/sa/sadc\"
-DUSE_NLS -DPACKAGE=\"sysstat\"
-DLOCALEDIR=\"/usr/local/share/locale\" sadc.c
gcc -o act_sadc.o -c -g -O2 -Wall -Wstrict-prototypes -pipe -O2

```

```
-DSOURCE_SADC -DSA_DIR=\"/var/log/sa\"
-DSADC_PATH=\"/usr/local/lib/sa/sadc\"
-DUSE_NLS -DPACKAGE=\"sysstat\"
-DLOCALEDIR=\"/usr/local/share/locale\" activity.c
[...]
#
```

make步骤结束时，可运行的sysstat软件程序就会出现在目录下！但是从这个目录下运行程序有些不便。你会想将它安装到Linux系统中常用的位置上。要这样的话，就必须以root用户身份登录（或者用sudo命令，如果你的Linux发行版偏好这个的话），然后用make命令的install选项。

```
# make install
mkdir -p /usr/local/share/man/man1
mkdir -p /usr/local/share/man/man5
mkdir -p /usr/local/share/man/man8
rm -f /usr/local/share/man/man8/sa1.8*
install -m 644 -g man man/sa1.8 /usr/local/share/man/man8
rm -f /usr/local/share/man/man8/sa2.8*
install -m 644 -g man man/sa2.8 /usr/local/share/man/man8
rm -f /usr/local/share/man/man8/sadc.8*
[...]
install -m 644 -g man man/sadc.8 /usr/local/share/man/man8
install -m 644 FAQ /usr/local/share/doc/sysstat-11.1.1
install -m 644 *.lsm /usr/local/share/doc/sysstat-11.1.1
#
```

现在，sysstat包已经安装在系统上了！虽然不像使用PMS安装那样简单，但是通过tarball安装软件也没那么难。

9.5 小结

本章讨论了如何用软件包管理系统（PMS）在命令行下安装、更新或删除软件。虽然大部分Linux发行版都使用漂亮的GUI工具进行软件包管理，但是你也可以在命令行下完成同样的工作。

基于Debian的Linux发行版使用dpkg工具作为命令行与PMS的接口。dpkg工具的一个前端是aptitude，它提供了处理dpkg格式软件包的简单命令行选项。

基于Red Hat的Linux发行版都以rpm工具为基础，但在命令行下采用了不同的前端工具。Red Hat和Fedora用yum安装和管理软件包。openSUSE发行版采用zypper来管理软件，而Mandriva发行版则采用urpm。

本章讨论了如何安装仅以源代码tarball形式发布的软件包。tar命令可以从tarball中解包出源代码文件，然后使用configure和make命令从源代码中构建出最终的可执行程序。

下章将讲述Linux发行版中可用的编辑器。如果你已经准备好开始编写shell脚本，那么了解哪些编辑器可用将会助你一臂之力。