

LINUX POCKET GUIDE

第2版

# LINUX

口袋书



DANIEL J. BARRETT 著

欧阳立博 严小商 任桥伟 译

O'REILLY®



电子工业出版社

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>

---

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

**备用QQ:2404062482**

## Linux口袋书 (第2版)

如果你每天都使用Linux, 这本流行的指南将会是你完美的工作伙伴。根据读者的要求, 第2版涵盖了更多的实用命令, 内容也不再局限于Fedora, 而是适用于任何发行版。

对于需要快速上手的Linux新人, 以及仅仅需要一个简明的功能性索引的老手来说, 这本《Linux口袋书》(第2版) 会快速提供你所需要的答案。对于Linux的使用, 本书也提供了一条有效的学习路径, 包括按照功能进行分类组织的大量常用命令及其选项。

本书涵盖的主题如下:

- 文件系统和Shell
- 用户账号管理
- 文件创建和编辑
- 成为超级用户
- 文本操作和管道
- 网络链接
- 备份和远程存储
- 音频和视频
- 软件安装
- 查看和控制进程
- Shell脚本编程

Daniel J. Barrett博士有25年从事软件工程师、系统管理员、幽默作家的工作经验。在此期间, 他在O'Reilly出版了大量书籍, 其中包括SSH, *The Secure Shell: The Definitive Guide*, *MediaWiki* 和 *Linux Security Cookbook*。

图书分类: Linux

oreilly.com.cn

责任编辑: 孙学瑛

封面设计: Karen Montgomery 张健



www.phei.com.cn

ISBN 978-7-121-20312-3



9 787121 20307

定价: 29.00元

O'Reilly Media, Inc. 授权电子工业出版社出版

此简体中文版仅限于中国大陆 (不包含中国香港、澳门特别行政区和中国台湾地区) 销售发行

This Authorized Edition for sale in the mainland of China (excluding Hong Kong, Macao SAR and Taiwan)

O'REILLY®

# Linux 口袋书 (第2版)

---

Linux Pocket Guide, Second Edition

Daniel J. Barrett 著

欧阳立博 严小商 任桥伟 译

电子工业出版社

Publishing House of Electronics Industry

北京•BEIJING



## 内 容 简 介

本书是一本精简的 Linux 入门指南,简要阐述了 Linux 的基本观念,涵盖 Linux 的许多命令和功能,并以精辟的范例示范如何利用它们进行日常工作,能够让你在短时间内有效提升工作效率。

©2012 by O'Reilly Media,Inc. Simplified Chinese Edition, jointly published by O'Reilly Media,Inc.and Publishing House of Electronics Industry, 2013.Authorized translation of the English edition,2012 O'Reilly Media,Inc.,the owner of all rights to publish and sell the same. All rights reserved including the rights of reproduction in whole or in part in any form.

本书简体中文版专有出版权由O'Reilly Media,Inc.授予电子工业出版社。未经许可,不得以任何方式复制或抄袭本书的任何部分。专有出版权受法律保护。

版权贸易合同登记号 图字: 01-2013-1793

### 图书在版编目(CIP)数据

Linux 口袋书 / (美) 巴瑞特 (Barrett,D.J.) 著; 欧阳立博, 严小商, 任桥伟译. —2 版. —北京: 电子工业出版社, 2013.6  
书名原文: Linux pocket guide,2nd edition  
ISBN 978-7-121-20307-7

I. ①L… II. ①巴… ②欧… ③严… ④任… III. ①Linux 操作系统 IV. ①TP316.89

中国版本图书馆 CIP 数据核字(2013)第 092350 号

责任编辑: 孙学瑛

封面设计: Karen Montgomery 张健

印 刷: 涿州市京南印刷厂

装 订: 涿州市京南印刷厂

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 850×1168 1/40 印张: 5.5 字数: 176 千字

印 次: 2013 年 6 月第 1 次印刷

定 价: 29.00 元

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

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线: (010) 88258888。

---

# 译者序

昨天参加一个 career talk 的活动，有邀请几位嘉宾作一些有关职业规划的讲座，其中一位前辈着重强调了 Mindset（心态）的问题：当我们是一个普通 engineer 的时候，我们这么写自己的 report “I worked on …”，而当我们 lead 一个 team 的时候，变成这么写 “We worked on …”，我们一直在简单地罗列自己参与的工作，而实际上不管是 I 或者是 We，我们都应该这么写 “I identified …, I contributed …”，去强调我们的贡献及我们的工作对他人的影响。

回顾一下自身，貌似在面对类似的场景时，我也一直在进行着各种重复：博客访问量……，写了……，写了……，进行了多少公益或非公益的讲座，……。从未或者说很少真正地沉下心去想想自己这些工作为他人贡献了些什么，又产生了多少有益的影响。

Anyway，回到本书，我相信它完全有能力成为每个人进入 Linux 的第一步，也相信它会成为很多 Linux 爱好者的随身伙伴，更希望在我以后的生活里，不会仅仅多

---

# 目录

Linux 口袋书（第 2 版）	1
本书内容	1
获取帮助	6
初识 Linux	8
文件系统	13
Shell	21
基本文件操作	36
目录操作	41
显示文件内容	43
文件的创建与编辑	53
文件属性	58
文件位置	68
文件文本操作	77
文件压缩和打包	90
文件比较	95
打印	100
拼写检查	101
磁盘和文件系统	103

备份和远程存储	108
查看进程	113
进程控制	117
任务调度	120
登录、注销与关机	125
用户和其操作环境	126
用户账号管理	130
成为超级用户	134
组管理	135
主机信息	137
主机位置	141
网络连接	145
电子邮件	149
网络浏览	154
Usenet News	158
即时消息	160
屏幕输出	162
数学计算	167
日期和时间	170
图像和屏幕保护程序	174
音频	177
视频	181
软件安装	182
Shell 脚本编程	187
后记	200
索引	203

## (第 2 版)

欢迎进入 Linux 的世界！对于 Linux 新人，本书可以作为一本入门指南，同时也可以作为一本常见和实用命令的快速索引。如果你已经拥有一定的 Linux 使用经验，则可以跳过前面这些介绍性的内容。

## 本书内容

本书只是一本精简的 Linux 入门指南，而不是一本完整全面的参考书。本书只介绍有助于你提高工作效率的重要内容，而不是逐条逐句地去详细解释每条命令及其参数（如果你没有找到自己想要的内容，请见谅），也不会去深入细致地探究操作系统的内部细节。精简、贴切和必要，是我们奉行的宗旨。

对于命令（*command*）的介绍是本书的重点。使用这些命令，你只需要输入很少的单词或字母就可以去指示一个 Linux 系统完成你所期望的工作。下面是一个命令的例子，用于统计文件 *myfiles* 的行数：

```
wc -l myfile
```

本书内容会涵盖对于一般用户而言最为重要的那些 Linux 命令，比如 *ls*（列出文件），*grep*（在文件中查找指定文本），*amarok*（播放音频文件），以及 *df*（计算空闲的磁盘空间）。限于篇幅，对 *GNOME* 和 *KDE* 之类的图形窗口环境，我们将只作简单介绍。

2> 为了提供一条流畅的学习路线，本书以功能为依据来编排内容。例如，我们将所有涉及文件内容显示的命令集中在一起进行介绍，`cat` 命令用于显示短文本文件，`less` 命令用于显示长文本文件，`od` 命令用于显示二进制文件，`acro read` 命令用于显示 PDF 文件等。然后，我们依次对每个命令进行解释，并提供其典型用法和常用选项的说明。

我们假设你有一个 Linux 系统的账号，并且知道如何使用你的账号及密码登录系统。否则，请咨询你的系统管理员；如果是你自己的系统，那么可以使用你在安装 Linux 时所创建的账号。

## 何谓 Linux

Linux 是一个流行的开源操作系统，可以与微软的 Windows 和苹果的 Mac OS 相提并论。有两种操作 Linux 系统的方式。

- 图形用户接口，由窗口、图标、鼠标控制组成；
- 命令行接口，又称作 *shell*，可以输入和运行各种命令，比如前面曾提到过的命令 `wc`。

同样，Windows 和 Mac OS 也可以使用命令行（Windows 使用其 `cmd` 和 PowerShell 命令工具，OS X 使用其终端应用程序），但是它们的用户大多并不使用命令行方式操作。然而，在 Linux 系统中，*shell* 是非常关键的，如果使用 Linux 而不用 *shell*，那么你正在错过一些很重要的体验。

## 何谓发行版

Linux 拥有非常高的可配置性，包含了数以千计的应用程序。因此，出现了各种不同版本的 Linux，来满足不同人的需求和偏好。虽然它们都具有相同的核心，但是却包含不同的程序和文件，外表看起来也各不相同。每种 Linux，也被称为一个 *distro*（发行版，“distribution”的简称）。流行的发行版有：Ubuntu Linux、Red Hat Enterprise Linux、Slackware、Mint 等。本书涵盖了应用

3> 于每个发行版的那些核心部分。

## 何谓命令

一条 Linux 命令通常由程序名 (*program name*) 及其后的选项 (*option*) 和参数 (*argument*) 组成。命令需要在 shell 环境里输入, 比如:

```
$ wc -l myfile
```

程序名 (`wc`, 统计字数) 指向位于磁盘上的一个程序, shell 可以根据该名称定位程序位置并执行。选项通常以 “-” 开头, 用来影响程序的行为。上述命令中, “-l” 选项告诉程序 `wc` 对行数而不是字数进行统计。参数 `myfile` 指定了程序 `wc` 应读取和处理的文件。最前面的美元符号 (\$) 是 shell 的提示符, 表明 shell 正在等待命令输入。

同一个命令可以有多个选项和参数, 它们可以被分开指定:

```
$ wc -l -w myfile    (有两个分开的选项)
```

也可以被合并在一起:

```
$ wc -lw myfile    (与 -l -w 功效相同)
```

然而, 并不是所有的程序都能够接受合并的选项。多参数也是可行的, 比如:

```
$ wc -l myfile1 myfile2    (统计两个文件的行数)
```

选项的含义并没有被标准化。相同的选项符号 (如 -l) 对于不同的程序可能具有不同的含义: 同样的选项 -l, 对于程序 `wc`, 其含义是 “统计文本行数”, 而对于程序 `ls`, 其含义则改变为 “更长的输出”。另一方面, 两个不同的程序可能使用不同的选项, 来表达相同的含义。比如 -q (run quietly) 与 -s (run silently)。

同样, 参数也没有被标准化。它们通常是输入或输出文件名, 但也可以是其他东西, 如目录名或正则表达式。

相比一个单纯带选项的程序, 命令要更为复杂和有趣。

4

- 同一个命令可以同时运行多个程序，这些程序可以被依次执行，也可以使用上一个命令的输出作为下一个命令输入的管道操作方式同时执行。Linux 高手往往善于使用管道方式。
- Linux 命令行接口，即 *shell* 有一套内置的编程语言。因此，你的命令不仅可以这样写：“运行这个程序”，也可以这样写：“如果今天是星期二，则运行这个程序；否则，对每个扩展名为 *.txt* 的文件，运行另外一个程序 6 次”。

## 如何阅读本书

在这本书中，我们将描述许多 Linux 命令。每个命令的描述都有一个标准格式的标题。

图 1 显示了 *ls*（列出文件）命令的标题，它用一种简单的格式描述了 *ls* 命令的一般用法：

`ls [options] [files]`

这意味着你必须首先输入“*ls*”，是否提供选项及文件名可以视情况而定。方括号“*[*”和“*]*”并不需要实际进行输入，它们只是用来强调其所包含的内容是可选的。斜体字部分表示应该被替换成实际的特定值，比如实际的文件名。如果你看到两个选项或参数之间用 *|* 符号隔开，同时在它们外边可能还有一对圆括号：

`( file | directory )`

这意味着你将不得不面临选择：使用文件名作为参数，或者使用目录名称作为参数。

图 1 显示的命令标题包含了一个命令的 6 个属性，深色表示支持，浅色表示不支持。

*stdin*

命令从标准输入读取数据，例如，默认为键盘。参见第 11 页的“输入和输出”。



```
ls                               stdin  stdout  -file  --opt  --help  --version

ls [options] [files]
```

图1：标准的命令标题

### *stdout*

命令将数据送到标准输出，例如，默认为屏幕。参见第 11 页的“输入和输出”。

### *-file*

当用“-”符号代替输入文件名时，命令将从标准输入读取数据；同样，如果用“-”代替输出文件名，则意味着命令将向标准输出写入数据。例如，下面的例子中，`wc` 命令依次从文件 *file1*，文件 *file2*，标准输入，文件 *file3* 中读取数据。

```
$ wc file1 file2 - file3
```

### *--opt*

如果选项包括“--”符号，则意味着选项到此结束；其后出现的任何命令行部分均不能作为选项。在处理文件名以“-”符号开始的文件时，这种方式十分必要，否则这样的文件名将很容易被误认为是选项。例如，如果一个文件名为 *-foo*，那么直接运行命令 `wc -foo` 将会失败，因为 `-foo` 将会被误认为是选项，而 `wc -- -foo` 则可以正确运行。如果命令不支持“--”方式，你还可以在文件名前面加上当前路径“./”，如此一来，“-”符号将不会再被误认为是选项首字符，比如：

```
$ wc ./-foo
```

### *--help*

选项 `--help` 告诉命令打印帮助信息，显示该命令的正确用法而后退出。

### *--version*

选项 `--version` 告诉命令打印版本信息，而后退出。

## 6 Shell 提示符

本书中某些命令只有使用超级用户 (*superuser*) 才能成功运行。超级用户是可以在系统里进行任何操作的特定用户。这种情况下, 我们使用 “#” 作为 shell 的提示符:

```
# superuser command goes here
```

否则, 对于普通用户, 我们将用 “\$” 作为 shell 提示符:

```
$ ordinary command goes here
```

### 按键

本书使用特定的符号表示按键。像许多其他的 Linux 文档一样, 我们使用 “^” 符号表示 “按住 Ctrl 键不放”。例如, “^D” (发音 “control D”) 表示 “按住 Ctrl 键不放, 然后再按下 D 键”。我们也以 Esc 表示 “按下 Esc 键”。回车键和空格键之类的按键就不再解释了。

### 你的朋友, echo 命令

在本书的许多例子中, 我们使用 echo 命令 (详见第 162 页的 “屏幕输出”) 打印屏幕信息。echo 是最简单的命令之一: 它仅仅将 shell 处理过的参数显示到标准输出。

```
$ echo My dog has fleas
My dog has fleas
$ echo My name is $USER      // 环境变量 USER
My name is smith
```

## 获取帮助

如果本书提供的内容不能满足你的需求, 你可以做以下几样事情来获得帮助。

### 7 使用 man 命令

man 命令用于查看指定程序的在线说明, 即所谓的 *manpage*。例如, 若想查看 ls 命令的用法, 可以运行:

```
$ man ls
```

对于某些不明确的命令，可以通过 `-k` 选项指定关键词来查找某个特定主题的说明：

```
$ man -k database
```

使用 `info` 命令

`info` 命令提供了比 `man` 命令更为详尽的说明内容，很多 Linux 程序都对其进行支持。

```
$ info ls
```

当 `info` 命令正在运行时，我们可以使用一些按键进行辅助。

- 按 `h` 键获取帮助
- 按 `q` 键退出
- 使用空格键和回退键进行前、后翻页
- 按 `Tab` 键在超级链接之间转跳
- 按 `Enter` 键进入超级链接

如果某个给定程序没有提供相应的 `info` 文件，`info` 命令将显示该程序的 `manpage`。单独运行 `info` 命令可以列出所有可用的文档。要了解如何操作 `info` 命令，可以运行 `info info`。

使用 `--help` 选项（任何）

许多 Linux 命令都会响应 `--help` 选项，并输出一些简短的帮助信息。

```
$ ls --help
```

如果输出超出一屏，则将帮助信息通过管道方式传递给 `less` 程序进行多页显示（可以按 `q` 键退出）：

```
$ ls --help | less
```

检查 `usr/share/doc` 目录

这个目录包含了许多程序的说明文件。这些文件通常按程序名字及版本进行组织。例如，文本编辑器 `emacs` 23 版的

说明文件可能会在目录 `/usr/share/doc/emacs23` 下找到（依赖于具体的发行版）。

### GNOME 和 KDE 帮助

访问 <http://www.gnome.org> 或 <http://www.kde.org> 可获取 GNOME 或 KDE 的帮助。

### 版本说明网址

大多数 Linux 发行版都有官方网站，上面有文档、用来提问与解答的论坛，以及其他资源。只要在任何流行的搜索引擎中输入发行版的名字（如，“Ubuntu”），都可以找到它的网站。也可以访问本书的网站：<http://shop.oreilly.com/product/0636920023029.do>。

### Linux 帮助网站

有许多专门的 Linux 网站可以供我们寻找问题的答案，比如：<http://www.linuxquestions.org>，<http://unix.stackexchange.com>，<http://www.linuxhelp.net>，以及 <http://www.linuxforums.org>。

### 网页搜索

针对一些特定的 Linux 错误信息，通过在搜索引擎里进行查找，你很可能找到一些比较有帮助的结果。

## 初识 Linux

Linux 有四个主要的部分。

### 内核

涵盖底层操作系统、文件管理、磁盘、网络，以及其他我们认为必需的部分。大多数用户都几乎不会注意到内核。

### 支撑程序

包括数以千计的程序用于文件操作、文档编辑、数学计算、网页浏览、音频、视频、计算机编程、排版、加密、DVD 刻录……应有尽有。

用于输入命令、执行命令及显示命令结果的用户接口。Linux 有多种 shell : Bourne shell、Korn shell 和 C shell 等。本书内容基于 bash, 即 Bourne-Again Shell, 它通常也是用户的默认 shell。不管怎样, 所有 shell 都具有相似的基本功能。

## X

提供窗口、菜单、图标、鼠标支持, 以及其他常见 GUI 组件的图形系统。其他更加复杂的图形环境都是基于 X 的基础上建立的, 其中最流行的是 KDE 和 GNOME。我们将会讨论几个打开 X window 并运行的程序。

本书内容集中在上面的第二、第三部分: 即支撑程序和 shell。

## 图形化桌面

登录 Linux 系统时, 欢迎你的可能是如图 2 所示的图形化桌面<sup>注1</sup>。它包含:

- 主菜单或任务栏。它有可能位于屏幕的顶部、底部或者边上, 这要取决于你的发行版及系统设置。
- 表示计算机的电脑图标, 表示个人根目录的文件夹、垃圾桶, 以及其他更多的东西。
- 应用程序图标, 如 Firefox 网页浏览器及 Thunderbird 电子邮件程序。
- 开、关窗口及同时运行多个桌面的控件。
- 一个时钟, 以及其他的零星小图标。

---

注 1: 如果是通过网络远程登录, 则看到的是命令行环境。

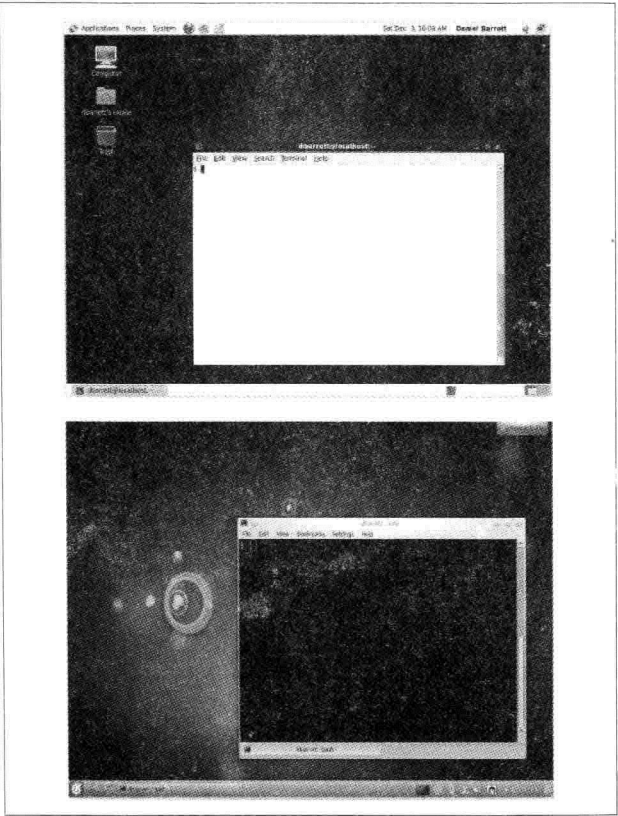


图2：图形桌面（CentOS Linux的GNOME，Ubuntu的KDE）。桌面的样式取决于你的发行版本及系统设置

11 Linux 系统有好几个图形界面，GNOME 和 KDE 是其中最为常用的。可以通过如下方式确定你的系统正在使用哪种图形界面：点击系统中相当于主菜单或开始菜单的地方，查找 GNOME、KDE、Kubuntu（基于 Ubuntu Linux 的 KDE）或类似的词。

## 运行 Shell

对某些用户而言，GNOME 和 KDE 中的图标和菜单是操作 Linux 的主要途径。使用这种方式来运行一些简单的任务，比如阅读电子邮件、浏览网页等都是非常方便的。然而，Linux 真正的力量却隐藏在图形界面的下面，在 shell 中。

为了更好地掌握 Linux 系统，你需要花费时间去熟悉 shell。（本书所有的内容都与它有关。）开始的时候 shell 操作也许比图标和菜单操作要困难，但是一旦你习惯了 shell，它将变得易于使用，并且非常强大。

如果是在 GNOME、KDE 或其他 Linux 图形用户界面中运行 shell，则必须先打开一个 shell 窗口（*shell window*）：一个容纳 shell 运行的窗口。图 2 显示了两个 shell 窗口，里面的 shell 提示符“\$”正在等待命令输入。你可以浏览你的系统菜单来发现这样的应用，比如 Terminal、xterm、gnome-terminal、konsole，以及 uxterm。

不要混淆窗口程序（如 konsole）与其中运行的 shell。窗口仅仅是一个花哨的容器，但 shell 却是让你输入并执行命令的工具。

如果你没有运行图形用户界面，也就是说你正在通过网络远程登录，或者直接通过终端登录，在你登录时，shell 就开始运行了，而不再需要 shell 窗口。

这里只是进行初步的介绍，更多的细节请参见第 21 页的“shell”一章，而在第 187 页的“shell 脚本编程”中将包含更为高级的内容。

## 输入和输出

◀ 12

大部分 Linux 命令都接收输入并产生输出。输入可以来自于文件或标准输入（*standard input*），通常是键盘。同样，输出被写入到文件或标准输出（*standard output*），通常是 shell 窗口或者屏幕。错误信息被专门处理，并显示到标准错误输出（*standard*

*error*) 上, 通常指的是你的屏幕, 但与标准输出区分开。<sup>注 2</sup>

后面我们将看到如何将标准输入、输出, 以及标准错误输出重定向 (*redirect*) 为文件或管道。当我们说 “read” 时, 意思是从标准输入读; 当我们说 “write” 或 “print” 时, 则意味着向标准输出设备写 (除非我们正在讨论打印机)。

## 普通用户与超级用户

Linux 是一个多用户操作系统: 多个用户可以在同一时间操作同一台 Linux 计算机。在一台计算机上, 每个用户使用一个唯一的用户名 (*username*) 进行区分, 比如 “smith” 或 “funkyguy”, 并且每个人都拥有系统的一部分作为自己的私人空间, 完成一些私人的工作。同时, 也有一个叫做 *root* 的超级用户 (*Superuser*), 它可以在系统上进行任何操作而不受限制。普通用户都是受限的: 虽然他们可以运行绝大多数程序, 但是通常他们只能修改自己的文件。相反, 超级用户却可以创建、修改或者删除任何文件, 以及运行任何程序。

要成为超级用户, 你并不需要注销并重新登录; 只要运行 `su` 命令 (参见第 134 页 “成为超级用户”) 并提供超级用户密码:

```
$ su -l
Password: ****
#
```

13 超级用户提示符 # 意味着你可以开始运行超级用户命令。

另外一种获取超级用户权限的方式是, 运行 `sudo` 命令 (需要你的系统配置允许你使用它), 它只以超级用户的身份执行一条命令, 然后便会返回到原来的用户:

```
$ sudo ls /private/secrets // 查看一个私有的目录
Password:****
secretfile1 secretfile2 // 命令成功执行
$
```

---

注 2: 比如, 你能够在将标准输出导入到一个文件的同时, 将错误信息显示在屏幕上。



# 文件系统

为了很好地使用 Linux 系统，你必须适应 Linux 的文件及目录 (*directory*，又称为文件夹)。在窗口系统中，文件和目录直接被显示在屏幕上。而在命令行界面，比如 Linux shell 里，同样的文件和目录并没有被直观的显示，因此，你必须记住你目前在哪个目录里，以及这个目录与其他目录之间的层次关系。当然，你可以使用 shell 命令，比如 `cd` 与 `pwd`，在各个目录之间移动，以及定位你的位置。

让我们来解释一些术语。正如我们说过的，Linux 文件按照目录的方式来组织。如图 3 所示，目录呈树状 (*tree*) 层级分布：一个目录可能包括其他的目录，即子目录 (*Subdirectory*)。子目录本身也可能包含其他文件及子目录，等等，直至无穷。位于最高层的目录被称为根目录 (*root directory*)，并以 `/` 表示。<sup>注 3</sup>

14

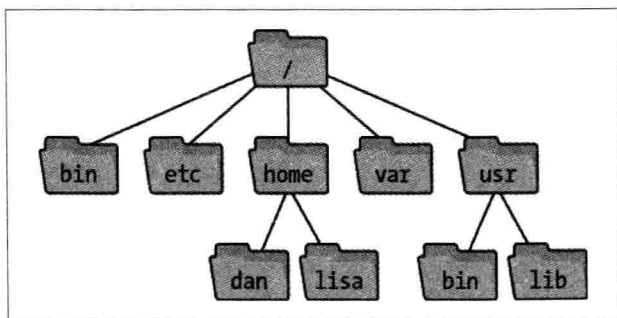


图3：Linux文件系统（部分）。根目录在顶部，“dan”文件夹的完整路径是/home/dan

Linux 使用“名字和斜线”的方式描述文件和目录的位置，也就是路径 (*path*)。例如：

`/one/two/three/four`

注 3：Linux 并没有 Windows 和 DOS 里“磁盘驱动器符号”的概念，而是所有的文件与目录都从一个根目录延伸下来。

上述路径表示根目录 / 下面 *one* 子目录下的 *two* 子目录下的 *three* 子目录下的 *four* 文件。如果一个路径以根目录为起点，则称为绝对路径；否则，就是一个相对路径。

无论何时进入 shell 环境，该 shell 都必定“位于”某个目录（抽象地说）。更专业地说，shell 有一个当前工作目录（*current working directory*），在 shell 中运行命令时，它就作用在这个目录。更具体地说，如果在 shell 中使用相对文件路径，它也是相对于当前工作目录而言的。例如，如果 shell 的当前工作目录是 */one/two/three*，提供给命令的文件路径是 *myfile*，那么这个命令真正操作的文件是 */one/two/three/myfile*。同理，相对路径 */a/b/c* 意味着真实的路径是 */one/two/three/a/b/c*。

有两个特殊的目录：（一个点）和 *..*（两个点）。前者表示当前目录；后者表示上一级目录。因此，如果你的当前目录是 */one/two/three* 时，则 *.* 代表此目录，*..* 代表目录 */one/two*。

可以使用 *cd* 命令从一个目录移动到另一个目录：

```
$ cd /one/two/three
```

更专业地说，这个命令将 shell 的当前目录更改为 */one/two/three*。对于这个例子来说，*cd* 使用的是绝对路径（因为目录以“/”开始）；当然，*cd* 也可以使用相对路径，比如：

```
$ cd d           // 进入子目录 d
$ cd ../mydir    // 回到上级目录，然后进入其子目录 mydir
```

文件和目录名可以包含大多数字符：大写、小写字母<sup>注4</sup>、数字、句点、破折号、下画线，以及大部分符号（除了“/”，因为它留作分割目录用）。但是，在实际使用中，要避免使用空格、星号、括号，以及其他在 shell 中有特殊意义的字符。否则，要给这些符号加上引号。（参见第 28 页“界定”。）

---

注 4：Linux 文件名对大小写敏感，因此相同字符大、小写意义不同。

## 主目录

Linux 里每个用户都有一个存放个人文件的目录，称为主目录 (home directory)。普通用户的个人目录通常位于 `/home` 目录下，典型的目录名称是 `/home/your-username`；`/home/smith`，`/home/jones`，等等。对于超级用户来说，主目录是 `/root`。

有几种方法可以找到或者引用你的主目录。

`cd`

如果不带参数，`cd` 命令会带你移动到你的主目录。

HOME 环境变量

环境变量 `HOME`（参见第 24 页“shell 变量”）包含你的主目录名称。

```
$echo $HOME           //echo 命令会显示它的参数
/home/smith
```

~

当使用字符“~”代替目录名时，shell 会自动将其扩展成主目录名，比如：

```
$ echo ~
/home/smith
```

当“~”后跟着用户名时（如 `~fred`），shell 将其扩展成该用户的主目录名，比如：

```
$ cd ~fred
$ pwd           // 打印工作目录命令
/home/fred
```

## 系统目录

典型的 Linux 系统有上万个系统目录 (system directory)。系统目录包含操作系统文件、应用程序、文档等，囊括除用户个人文件（通常位于 `/home` 目录下）外的所有文件。

除非你是一名系统管理员，否则大多数系统目录你都几乎不用访问，但是只要有点基础知识，你就可以理解或猜测出这些目录的作用。这些目录的名字通常包含三部分，我们称之为范畴（scope）、分类（category），以及应用（application）。（这三个名词并不是标准的术语，但是可以帮助你了解系统目录的组织原则。）例如，目录 `/usr/local/share/emacs` 包含了文本编辑器 emacs 的数据。它的 scope 是 `/usr/local`（安装系统文件的地方），category 是 `share`（特定应用的数据与文档），application 是 `emacs`（文本编辑器），如图 4 所示。后面按照 category，scope，application 的顺序分别对它们进行解释。



图4：目录的scope，category，application

## 目录路径的 category

*category* 告诉你目录下文件的类型。例如，如果 category 是 *bin*，你可以大致确定该目录下包含了程序。常见的 category 如下。

### 17 程序类

<i>bin</i>	程序（通常是二进制文件）
<i>sbin</i>	专为超级用户使用的程序（通常是二进制文件）
<i>lib</i>	程序使用的库文件
<i>libexec</i>	供其他程序调用的程序，可当作“可执行程序库（library of executable programs）”

### 文档类

<i>doc</i>	文档
<i>info</i>	emacs 内置帮助系统的文档
<i>man</i>	可用 man 程序查看在线手册；这些文件通常是压缩过的，或者含有 man 命令才能解释的排版指令
<i>share</i>	特定程序的文件，如示例、安装说明等

## 配置类

<i>etc</i>	系统配置文件
<i>init.d</i>	与 Linux 启动有关的配置文件
<i>rc.d</i>	与 Linux 启动有关的配置文件；也可能是 <i>rc1.d</i> , <i>rc2.d</i> , ……

## 编程类

<i>include</i>	程序的头文件
<i>src</i>	程序的源码

## 网站类

<i>cgi-bin</i>	网站运行的脚本和程序
<i>html</i>	网页
<i>public_html</i>	网页，通常位于用户主目录
<i>WWW</i>	网页

## 显示类

<i>fonts</i>	字体
<i>X11</i>	X window 系统文件

## 硬件类

<i>dev</i>	设备文件，内核提供给应用访问磁盘及其他硬件的接口
<i>media</i>	挂载点，用于访问磁盘的目录
<i>mnt</i>	挂载点，用于访问磁盘的目录
<i>misc</i>	挂载点，用于访问磁盘的目录

## 运行状态类

<i>var</i>	存放随当前计算机的运行而改变的文件
<i>lock</i>	锁定文件，程序创建它用来表明自己正在执行。它的存在是用来阻止其他程序进行某些操作，或是避免同一个系统上同时运行多个一样的程序
<i>log</i>	日志文件，记录系统的重要事件，包含错误、警告，以及一些仅仅是信息性的消息等
<i>mail</i>	接收邮件的邮箱
<i>run</i>	PID 文件，包含正在执行的进程的 ID；这些文件常用来对特定的进程进行追踪或者终止其运行

<i>spool</i>	过渡文件，比如等待寄出的 email、打印任务、计划任务等
<i>tmp</i>	供程序或者用户临时使用的空间
<i>proc</i>	操作系统状态，参见第 18 页“操作系统目录”

## 目录路径的 scope

目录路径的 *scope* 是从更高的层次上描述整个目录的体系结构。比较常见的 *scope* 是：

<i>/</i>	Linux 提供的系统文件（称为“root”）
<i>/usr</i>	更多 Linux 提供的系统文件（称为“user”）
<i>/usr/games</i>	游戏
<i>/usr/local</i>	个别组织或计算机专用的系统文件，比如，个人另外安装 的软件包
<i>/usr/X11R6</i>	X window 系统的文件

因此，对于 *lib* (library) 类，你的 Linux 系统可能存在如 */lib*，*/usr/lib*，*/usr/local/lib*，*/usr/games/lib* 和 */usr/X11R6/lib* 等目录。

19 实际上，“/”与“/usr”两个目录间并没有很明确的区别。但是感觉上，“/”属于更底的层次，离操作系统更近。比如，*/bin* 目录包含了 *ls*、*cat* 之类最基本的程序文件，*/usr/bin* 目录包含了 Linux 发行版提供的各种应用程序，*/usr/local/bin* 目录包含了系统管理员另外安装的程序。这些只是约定俗成的管理，不是硬性的规定。

## 目录路径的 application

目录路径的 *application* 部分通常是程序的名字。在 *scope* 和 *category*（比如说 */usr/local/doc*）之后，程序可以有自己的子目录（比如 */usr/local/doc/myprogram*）来包含自己所需的文件。

## 操作系统目录

这些目录包含了支持 Linux 内核（即 Linux 操作系统的最底层部分）的文件。

*/boot*

与系统启动相关的文件。它是内核映像文件（通常是 */boot/vmlinuz*）所在的位置。

*/lost+found*

供磁盘修复工具存放恢复好的受损文件。

*/proc*

描述进程的运行状态。

*/proc* 目录下的文件描述了内核的运行状况。与其他文件相比，它们有一些特殊的地方，比如它们总是 0 字节大小、只读的、日期为当天的文件。

```
$ ls -l /proc/version
-r--r--r-- 1 root root 0 Oct 3 22:55 /proc/version
```

但是，神奇的是它们包含了 Linux 内核的相关信息，如：

```
$ cat /proc/version
Linux version 2.6.32 -71.el6.i686...
```

*/proc* 目录下的文件主要是供程序使用，但是花点时间浏览它们 20 的内容有助于我们了解系统的运行状态。下面是一些示例：

*/proc/ioprots* 输入和输出设备列表。

*/proc/version* 操作系统版本，命令 `uname` 同样可以打印该信息。

*/proc/uptime* 系统运行时间，比如上次启动后运行的秒数。`uptime` 命令可以提供易读的相同信息。

*/proc/nnn* 系统中每一个进程都有一个专属的 */proc/nnn* 目录，*nnn* 为进程的 ID。此目录包含了该进程的相关信息。

*/proc/self* 当前进程的信息；它是某个 */proc/nnn* 目录的符号链接，会自动更新。多次执行 `ls -l /proc/self`，你将看到 */proc/self* 不断改变它链接的对象。

## 文件保护

一个 Linux 系统中，可能有许多用户拥有登录账号。为保护隐私和安全，多数用户只能访问一些文件，而不是所有文件。访

问控制主要体现在下面两个方面。

### 谁有权限

每份文件和目录都有一个拥有者 (*owner*)，他有权限对这些文件、目录进行任何操作。通常，用户创建一个文件后，即成为该文件的拥有者，但是也有例外。

此外，预定义的用户组 (*group*) 也有访问文件的权限。用户组由系统管理员定义，详见第 135 页“组管理”。

最后，文件或目录也可以向系统中所有用户开放。这些既不是拥有者也非用户组成员的其他任何用户，都被称为“其他人” (*other*)。

### 授予何种权限

文件拥有者、用户组，以及所有其他人都可能有读 (*read*)、写 (*write*)、执行 (*execute*) 某些文件的权限。目录也同样有这三种权限：可读（访问目录中的文件），可写（创建和删除目录中的文件），可执行（可使用 `cd` 命令进入该目录）。

查看特定文件的访问模式，可以执行：

```
$ ls -l myfile
-rw-r--r-- 1 smith smith 7384 Jan 04 22:40 myfile
```

查看特定目录的访问模式，可以执行：

```
$ ls -ld dirname
drwxr-x--- 3 smith smith 4096 Jan 08 15:02 dirname
```

输出的信息中，最左边的 10 个字符是文件的访问权限，r（读）、w（写）、x（执行），与其他字母及“-”符号的组合，例如：

```
-rwxr-x---
```

上述字母及符号的意义如下。



位置	意义
1	文件类型：- = 文件，d = 目录，l = 符号链接，p = 命名管道，c = 字符设备，d = 块设备
2~4	文件拥有者的读、写、执行权限
5~7	用户组的读、写、执行权限
8~10	其他人的读、写、执行权限

因此，上述示例中的 `-rwxr-x---` 就表示文件可被拥有者读、写、执行，被用户组读、执行，并禁止其他人访问。对 `ls` 命令更详细地描述在第 36 页“基本文件操作”。`chown`、`chgrp`、`chmod` 命令可分别改变文件的拥有者、用户组和访问权限，详见第 58 页“文件属性”。

## Shell

22

要让 Linux 系统执行你的命令，则得有一个能输入命令的地方。这个地方即是所谓的 *shell*，也就是 Linux 的命令行操作界面：输入命令，然后按下 Enter 键，shell 将运行任何你所要求的程序。（参见第 11 页“运行 shell”，学习如何打开 shell 窗口。）

例如，要查看当前有哪些用户登录系统，可在 shell 中执行如下命令：

```
$ who
silver      :0      sep 23 20:44
byrnes     pts/0    sep 15 13:51
barrett    pts/1    sep 22 21:15
silver     pts/2    sep 22 21:18
```

（符号 `$` 是 shell 提示符，表示 shell 已做好运行命令的准备。）在一条命令里，可以同时调用多个程序，甚至将几个不同的程序联接在一起，协作完成某项工作。下面的命令将 `who` 程序的输出作为 `wc` 程序的输入，`wc` 程序用于统计文件中的文本行数，所以整个命令的执行结果就是 `who` 输出的行数。

```
$ who | wc -l
4
```

该命令运行的结果是告诉你当前系统中有多少用户<sup>注5</sup>。命令中的“|”被称为管道 (*pipe*)，用于联接 `who` 和 `wc`，将左边程序的输出当作右边程序的输入。

实际上，shell 本身也是一个程序，Linux 系统提供了好几种 shell 以供选择。本书只讨论 `bash` (Bourne-Again Shell)，它位于 `/bin/bash`，是众多 Linux 发行版的默认 shell。

## 23 Shell 与程序

一条命令运行时，它可能启动一个 Linux 程序（如 `who`），或者运行 shell 的内置命令 (*built-in command*)，也就是 shell 本身的功能。`type` 命令可以告诉你特定的命令属于哪一种类型。

```
$ type who
who is /usr/bin/who
$ type cd
cd is a shell builtin
```

`type` 命令有助于你了解哪些是 shell 的内置命令，哪些是 Linux 提供的外部程序。以下几节简要描述了 shell 的一些特点。

## bash Shell 概述

除了运行命令，shell 还能做许多其他的事情。它提供了许多有用的功能，有助于你轻松完成工作。比如，通配符 (*wildcard*) 可以匹配文件的名字；“历史命令 (*command history*)”可以快速调用以前使用的命令；管道 (*pipe*) 可以将一个命令的输出作为另一个命令的输入；变量 (*variable*) 存储 shell 用到的数据，等等。多花点时间了解这些功能，可以让你更快速有效地使用 Linux。这里只是粗略地介绍一下这些功能的皮毛，如需完整的说明，请运行 `info bash`。

---

注 5：准确地说，是共有多少个 shell 被启动。如果有用户同时启动了两个 shell，`who` 所汇报的行数就不等于实际使用系统的人数。

## 通配符

通配符可以让你方便地描述一组有相似名称的文件。例如，`a*` 代表文件名以小写字母 “a” 开头的文件。shell 在遇到通配符后，会将其展开为符合要求的一系列文件的名称。因此，如果输入的命令是：

```
$ ls a*
```

shell 会首先将 `a*` 展开为当前目录下所有以 “a” 开头的文件的名称，从而上述命令会被改写为：

```
$ ls aardvark adamantium apple
```

`ls` 命令本身并不知道你使用了带有通配符的文件名，它只看到了 shell 将通配符展开以后的文件名的列表。重要的是，这意味着 ◀ 24 每一条 Linux 命令，无论它来自哪里，都可以与通配符及其他 shell 功能协同工作。

通配符不能匹配两个符号：文件名开头的 “.”，以及表示目录的 “/”。这种情况下，必须明确写出这些符号，比如使用 `.pro*` 来匹配 `.profile`，或者使用 `/etc/*conf` 匹配目录 `/etc` 下所有以 `conf` 结尾的文件名。

### 隐藏文件

名字以 “.” 开头的文件称作隐藏文件 (*dot file*)，是 Linux 系统中的一种特殊文件。当你给某个文件起的名字以 “.” 开始时，一些程序将不会显示它，比如：

- `ls` 将在文件目录列表中忽略它，除非你添加 `-a` 选项；
- shell 通配符不会与文件名开头的 “.” 进行匹配。

很明显，如果没有明确要求，隐藏文件将是隐藏的。

通配符	意义
*	零个或者多个连续的字符
?	任何单个字符
[set]	出现在 set 中的任何单个字符。set 通常都是一个字符序列，比如，[aeiouAEIOU] 代表所有元音字符，[A-Z] 代表所有大写字母
[^set]	任何在 set 中未出现的单个字符
[!set]	同 ^

使用 [set] 语法时，如果希望包含特殊字符，可通过改变它们在字符序列中的位置来避免 shell 误解。比如，如果希望在字符序列中包含 “-”，可将它放在第一个或最后一个；如果要包含 “]”，可将其放在第一个；如果要包含 “^” 或 “!”，要避免将其放在首位。

## 25 花括号

与通配符类似，含花括号的表达式也可以扩展成多个参数。表达式以逗号分隔，比如：

```
{X,YY,ZZ}
```

在命令行中，首先扩展成 X，然后是 YY，最后是 ZZZ，就像下面一样：

```
$ echo sand{X,YY,ZZ}wich
sandXwich sandYYwich sandZZwich
```

花括号可包含任何字符串，不像通配符那样仅限于文件名。无论当前目录下那些文件是否存在，上述例子都能正常运行。

## Shell 变量

可以像下面这样定义变量的值：

```
$ MYVAR=3
```

引用该变量时，只需要在变量名前添加 “\$” 符号：

```
$ echo $MYVAR
3
```

通常在你登录时，有些预定义的变量就会自动生效。

变量	意义
DISPLAY	X window 界面的名字
HOME	个人根目录，如 <i>/home/smith</i>
LOGNAME	登录名，如 <i>smith</i>
MAIL	收件箱，如 <i>/var/spool/mail/smith</i>
OLDPWD	shell 的前一个工作目录，最近一次输入 <i>cd</i> 命令前的工作目录
PATH	shell 的搜索路径，目录之间由冒号分隔
PWD	shell 的当前工作目录
SHELL	shell 程序的所在路径，如 <i>/bin/bash</i>
TERM	终端类型，如 <i>xterm</i> 或 <i>vt100</i>
USER	登录名

26

显示所有 shell 变量的命令是运行：

```
$ printenv
```

变量的默认作用范围只包括定义该变量的 shell 本身。要使变量作用范围涵盖被 shell 调用的程序（比如 subshells），必须使用 *export* 命令定义变量：

```
$ export MYVAR
```

或者同时完成变量的定义及赋值：

```
$ export MYVAR=3
```

这样定义的变量就是所谓的环境变量 (*environment variable*)，因为它的作用范围涵盖当前 shell 环境下所有程序。因此，上述例子中，变量 *MYVAR* 可以被其所在的 shell 环境下所有正在运行的程序使用（包括 shell 脚本，参见第 188 页“变量”）。

如果需要变量仅对于特定的命令有效，并且该命令结束后变量

立刻回到原值，则必须在命令行的开头加上 `variable=value`：

```
$ echo $HOME
/home/smith
$ HOME=/home/sally printenv HOME
/home/sally
$ echo $HOME
/home/smith
```

// 原先的值不受影响

## 搜索路径

Linux 文件系统中，程序文件随处可见，如目录 `/bin` 和 `/usr/bin` 中就有很多程序文件。通过 shell 命令运行程序时，shell 是如何找到该程序的呢？环境变量 `PATH` 用于告诉 shell 到哪里去寻找命令所要运行的程序。输入如下命令时：

```
$ who
```

27 shell 会查找 Linux 目录，直至找到 `who` 程序。这个过程中，环境变量 `PATH` 起到了非常重要的作用，它的值是一系列由冒号隔开的目录名称：

```
$ echo $ PATH
/usr/local/bin:/bin:/usr/bin:/home/smith/bin
```

shell 在这些目录中逐个搜索，查找 `who` 程序（如 `/usr/bin/who`），找到后即运行。如果所有目录中均未找到 `who` 程序，它将返回如下提示：

```
bash: who: command not found
```

若要临时增加目录到 shell 的搜索路径中，需要修改 `PATH` 变量的值。例如，将 `/usr/sbin` 加入 shell 的搜索路径，需要进行如下操作：

```
$ PATH=$PATH:/usr/sbin
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/home/smith/bin:/usr/sbin
```

这些操作只能改变当前 shell 的搜索路径。要永久改变，需要修改配置文件 `~/.bash_profile` 中的 `PATH` 变量，然后重新登录。细节描述参见第 36 页“配置 shell”一节。

## 命令别名

内置命令 `alias` 用于定义命令的别名，让你用较短的字符串代替一长串常用的复杂命令。比如：

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

定义了新的命令 `ll`，它代表命令“`ls -l`”，`ll` 命令运行如下：

```
$ ll
total 436
-rw-r--r-- 1 smith 3584 Oct 11 14:59 file1
-rwxr-xr-x 1 smith 72 Aug 6 23:04 file2
...
```

无论你何时登录系统，`~/.bashrc` 文件中定义的别名（参见第 36 页“配置 shell”一节）都是有效的。<sup>注 6</sup>`alias` 命令可以列出所有已定义的别名。如果你觉得别名并不是非常有用的功能（别名不能带参数和分支），可以看看第 187 页的“Shell 脚本编程”，运行 `info bash` 命令，并仔细研究一下 `shell` 函数。

## 输入 / 输出重定向

Shell 可以将标准输入（`stdin`）、标准输出（`stdout`）、标准错误输出（`stderr`）重定向为文件。也就是说，任何命令都可以用 shell 的“`<`”操作符将输入数据来源从 `stdin` 重定向为文件，比如：

```
$ mycommand < infile
```

同样，任何命令都可以将数据输出的目的地从 `stdout` 重定向为文件。

```
$ mycommand > outfile      // 创建或覆盖 outfile
$ mycommand >> outfile      // 附加到 outfile 原有内容的末尾
```

一个命令也能够将 `stderr` 重定向到文件，同时仍然将写入标准输出的内容打印在屏幕上：

```
$ mycommand 2> errorfile
```

---

注 6：有些系统启动时使用独立的别名定义文件，如 `~/.bash_aliases`。

下面的命令将 stdout 及 stderr 都重定向到文件：

```
$ mycommand > outfile 2> errorfile // 输出到不同文件
$ mycommand >& outfile // 输出到一个文件
```

## 管道

shell 管道操作 (|) 可以将一个程序的输出连接到另一个程序的输入，例如：

```
$ who | sort
```

上述命令将程序 who 的输出导入到程序 sort，作为其输入，结果得到一份按字母顺序列出的已登录用户列表。同理可知多级管道的使用方式，比如，将 who 的输出进行排列，并提取用户信息的第一列（使用 awk 命令），然后将结果按页显示（使用 less 命令）：

```
$ who | sort | awk '{print $1}' | less
```

## 29 组合命令

在同一命令行中可以启动多个连续命令，各个命令之间以 “;” 隔开，比如：

```
$ command1 ; command2 ; command3
```

如果各个命令之间存在关联性，要求任意命令运行失败其他命令也必须停止运行，则可以使用 “&&” 替代 “;”：

```
$ command1 && command2 && command3
```

如果各个命令之间存在顺序性，要求上一个命令运行完毕，下一个命令才能够开始运行，则需要改用 “||”：

```
$ command1 || command2 || command3
```

## 界定

通常情况下，shell 将空格符视为命令行中单词的分隔符。如果命令行中的某个参数包含空格（比如包含空格的文件名），则



需要用一对单引号或双引号来界定该参数的范围，这样 shell 才会将其视为一个整体。shell 以字面意义看待单引号界定的字符串，并不会附加其他的解释；而对于使用双引号界定的字符串，shell 将会解读其中的特殊字符，并展开。比如：

```
$ echo 'The variable HOME has value $HOME'
The variable HOME has value $HOME
$ echo "The variable HOME has value $HOME"
The variable HOME has value /home/smith注 7
```

反向单引号 “`” 界定的内容将被视为 shell 命令，其内容在执行过程中被该命令的输出自动替换，比如：

```
$ whoami          // 显示用户名的程序
smith
$ echo My name is `whoami`
My name is smith
```

## 转义

如果你想要使用 shell 的某个特殊字符，但是又希望 shell 将它们当做一般的字符处理（比如，“\*” 作为星形符号处理，而不是通配符），此时你可以在特殊字符前添加 “\”。这就是所谓的特殊字符的转义（*escaping*），“\” 也被称作转义符，比如：

```
$ echo a*          /* 作为通配符解释
aardvark agnostic apple
$ echo a\*         /* 作为一般字符解释
a*
$ echo "I live in $HOME"  //$ 表示展开环境变量
I live in /home/smith
$ echo "I live in \$HOME"  //$ 作为一般字符解释
I live in $HOME
```

30

命令行中，可以在控制字符（如制表符 Tab、换行符、^D 等符号）前加上 ^V，从而将它们当做一般字符使用。制表符 Tab (^I) 常被 shell 用作文件名补齐控制符（参见第 31 页“文件名补齐”），比如：

---

注 7：单引号中，\$HOME 被视为字符串；双引号中，\$HOME 被视为变量，并展开为它实际的值。——译者注

```
$ echo "There is a tab between here^V^I and here"
There is a tab between here      and here 注 8
```

## 编辑命令行

bash 支持文本编辑器 emacs 和 vi 的编辑快捷键(参见第 53 页“文件的创建与编辑”),让习惯 emacs 或 vi 的用户可以使用习惯的快捷键来编辑命令行。运行如下命令(将其放在 `~/.bash_profile` 文件中,可以使设定永久有效)可以让 emacs 的编辑快捷键生效:

```
$ set -o emacs
```

对于 vi 来讲,该命令如下:

```
$ set -o vi
```

emacs 快捷键	vi 快捷键 (先按 Esc 键)	含义
^P 或 方向键上	k	调出上一个命令
^N 或 方向键下	j	调出下一个命令
^F 或 方向键右	l	光标右移一个字符
^B 或 方向键左	h	光标左移一个字符
^A	o	光标移到行首
^E	\$	光标移到行末
^D	x	删除下一个字符
^U	^U	整行消除

## 31 命令历史记录

你可以通过命令历史记录 (*history*) 重新调用并执行之前运行过的命令。与之相关的一些命令和快捷键如下:

命令	含义
history	打印出完整的历史命令记录
history <i>N</i>	打印出最近的 <i>N</i> 条历史命令记录
history -C	删除历史命令记录

---

注 8: 如果没有添加 ^V, shell 将会尝试找出名称以 here 开始的文件,而不是插入一个制表符在字符串里。——译者注

命令	含义
!!	重新运行上一条命令
!N	运行历史记录中编号为 <i>N</i> 的命令
!-N	运行历史记录中倒数第 <i>N</i> 条命令
!\$	代表前一命令的最后一个参数，让你可以在删除文件前对文件进行检查：
	<pre>\$ ls a* acorn.txt  affidavit \$ rm !\$</pre>
!*	代表前一命令的所有参数：
	<pre>\$ ls a b c a b c \$ wc !* 103    252    2904   a 12     25     384    b 25473  65510  988215  c 25588  65787  991503  total</pre>

## 文件名补齐

在输入文件名的过程中，按下 Tab 键，shell 将自动补齐完整的文件名。如果多个文件的名称开头几个字符与你输入的字符相同，shell 将发出“哔”的声音，提示你匹配并不唯一。这种情况下，再按一次 Tab 键，shell 将列出所有的可选文件。比如，输入以下命令：

```
$ cd /usr/bin
$ ls un<TAB><TAB>
```

则 shell 将显示目录 `/usr/bin` 下所有名字以字符 `un` 开头的文件，◀ 32如 `uniq`，`units` 和 `unzip`。再多输入几个字符，并再次按下 Tab 键时，shell 将自动补齐文件名。

# Shell 任务控制

jobs	列出所有运行的任务
&	将任务放到后台运行
^Z	暂停当前（前台）任务
suspend	暂停 shell
fg	恢复任务，前台运行该任务
bg	将已暂停的任务放到后台运行

所有 Linux 的 shell 都具备任务控制的能力,即有能力在后台（后台多任务运行）或前台（实时现场）运行程序。任务只是 shell 的工作单位。交互运行命令时,shell 将其视为任务,并进行记录。命令结束时,相关任务也就结束了。任务是比 Linux 进程更高一层次的概念。事实上, Linux OS 对任务一无所知,任务仅是由 shell 进行控制的。关于任务控制的几个重要术语介绍如下：

## 前台任务 (Foreground job)

在 shell 中运行,任务完成前 shell 提示符不会出现,因而不能同时运行其他任务。

## 后台任务 (Background job)

在 shell 中运行,但不独占 shell,任务完成前 shell 提示符就能出现,因而可以在同一个 shell 下同时运行其他任务。

## 暂停 (Suspend)

临时停止执行前台任务。

## 恢复 (Resume)

让暂停的任务继续执行。

---

### 33 jobs

内置命令 jobs 用于列出 shell 中正在运行的任务。

```
$ jobs
[1]- Running          emacs  myfile  &
[2]+ Stopped          su
```

左侧的数字是任务编号,+ 表示该任务默认是受 fg (foreground, 前台) 和 bg (background, 后台) 命令影响的。

---

## &

命令行末尾加上 & 符号,表示要求 shell 将该命令放到后台执行。

```
$ emacs myfile &
[2] 28090
```

shell 运行命令后, 输出任务编号 2 和命令的进程 ID : 28090。然后立刻输出提示符, 不管后台任务是否执行完毕。

---

## ^Z

前台任务正在运行时, 按下 ^Z, 该任务将被挂起。挂起的任务只是暂停运行, 但是其状态会被继续记录。

```
$ mybigprogram
^Z
[1]+  Stopped                  mybigprogram
$
```

对于已挂起的任务 mybigprogram, 可以通过 bg 命令将其放到后台继续执行, 也可以使用 fg 命令将其恢复到前台继续执行。

---

## suspend

内置命令 suspend 的作用是暂停 shell, 就像对 shell 本身按下了 ^Z 一样。例如, 运行 su 命令进入另一个 shell 后, 想回到原来的 shell 中, 则可以运行以下命令 :

```
$ whoami
smith
$ su -l
Password: *****
# whoami
root
# suspend
[1]+  Stopped                  su
```

```
$ whoami  
smith
```

---

## bg

**bg [%jobnumber]**

内置命令 **bg** 的作用是将挂起的任务放到后台运行。**bg** 命令没有指定参数时，只对最近被挂起的任务起作用。要影响其他某个特定的任务时（可以通过命令 **jobs** 列出任务的详细参数），可以在该任务编号前加上百分号，比如：

```
$ bg %2
```

某些交互式任务不能保持在后台运行，比如正在等待输入数据的任务。如果你尝试将其放到后台，shell 将暂停该任务并显示：

```
[2]+ Stopped command line here
```

这时候可用 **fg** 将任务放到前台继续运行。

---

## fg

**fg[%jobnumber]**

内置命令 **fg** 的作用是将挂起或后台运行的任务放到前台运行。没有指定参数时，**fg** 将自动选择一个任务，通常是最近挂起的任务或者最近放到后台运行的任务。要影响其他某个特定的任务时（可以通过命令 **jobs** 列出任务的详细参数），可在该任务编号前加上百分号，比如：

```
$ fg %2
```

## 35 结束运行中的命令

如果你在 shell 中启动了一个命令，但是随后希望能立即停止该命令，这时可以按 **^C**。shell 会将 **^C** 解读为“立即终止当前的前台命令”。因此，在浏览非常长的文件（比如使用 **cat** 命令）过程中，可以按 **^C** 停止显示：

```
$ cat bigfile
This is a very long file with many lines. Blah blah
blah
blah blah blah blah blah blah ^C
$
```

要终止后台运行的程序，可以先用 `fg` 命令将其放到前台，然后再按下 `^C`；或者，使用 `kill` 命令终止它运行（参见第 117 页“进程控制”）。

通过按 `^C` 来终止程序是一种不太友好的方式。如果程序自身有退出的方法，那就应该尽可能去使用。

## 程序终止之后的清理

使用 `^C` 终止程序可能会使 shell 处于一种奇怪的状态，比如屏幕显示的不是你输入的字符，而是其他的东西。这是由于程序被终止后，没有机会清理自身。发生这种现象时，可以按以下步骤解决问题：

1. 按 `^J` 使 shell 输出提示符。这和 Enter 键(换行)的作用一样，但在 Enter 键失效时还可以使用。
2. 输入 `reset` 命令（即使在你输入时它没有显示，也照样输入），然后再次按下 `^J`，运行该命令。这将让 shell 回到正常状态。

`^C` 仅在 shell 环境下才起作用，非 shell 窗口内按 `^C` 是不起作用的。而且，有些程序会拦截 `^C` 并忽略，比如文本编辑器 `emacs`。

## 结束 shell

终止 shell 运行的方法有两个：执行 `exit` 命令或按 `^D`<sup>注9</sup>：

```
$ exit
```

---

注9：`^D` 会向许多读取标准输入的程序发送“文件结束”的消息。因此，在这种情况下，结束的可能是 shell 所运行的程序，而不是 shell 本身。

---

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明：

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢！

**备用QQ:2404062482**