

LINUX POCKET GUIDE

涵蓋  
Fedora Linux

# LINUX

## 隨身指南



O'REILLY

東南大學出版社

DANIEL J. BARRETT 著

O'REILLY TAIWAN 公司 编译

# LINUX 随身指南



如果你正在寻找Linux速成秘笈，这本就是。《Linux随身指南》简明扼要地阐述了Linux系统中的基本概念，并以精辟的例子示范如何利用Linux进行日常工作，让你在短时间内有效提高工作效率。

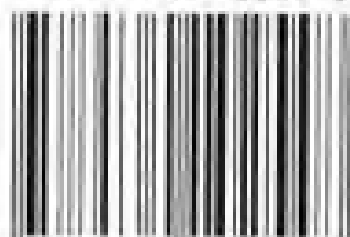
《Linux随身指南》提供了文件、目录、shell、X Window等Linux基本概念的说明，也汇总了常用命令的参考资料。所以，本手册不仅适合作为刚入门者的学习教材，更适合每位Linux用户随身携带查阅。

有别于一般参考手册以字母顺序排序的惯例，本手册按照功能类别来汇总各种命令的参考资料，因为这样更易于查找，在查阅的同时，也更容易了解还有哪些相关选择。对于每一个命令，本手册提供了语法、出处（所属的RPM包）、功能解释、用途、在磁盘上的位置、常用选项的意义以及典型用法的示范。

本手册的参考资料以Fedora Linux为准，但是大部分信息同样适用于其他Linux系统。如果你需要立刻学会Linux的实际应用，或是你身边需要一本简短、实用的参考手册，这本精简的随身指南正是你要找的。

本书作者 **Daniel Barrett** 也是《SSH: The Secure Shell》和《Linux Security Cookbook》这两本经典之作的作者之一。

ISBN 7-5641-0254-3



9 787564 102548 &gt;

**O'REILLY\***[www.oreilly.com.cn](http://www.oreilly.com.cn)

O'Reilly Media, Inc. 授权东南大学出版社出版

ISBN 7-5641-0254-3

定价：28.00 元

---

# Linux 随身指南

*Daniel J. Barrett* 著

*O'Reilly Taiwan* 公司 编译

**O'REILLY®**

*Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo*

O'Reilly Media, Inc. 授权东南大学出版社出版

东南大学出版社

## 图书在版编目 (CIP) 数据

Linux 随身指南 / (美) 巴雷特 (Barrett J., D.) 著,  
O'Reilly Taiwan 公司编译. — 南京: 东南大学出版社,  
2006.5

书名原文: Linux Pocket Guide

ISBN 7-5641-0254-3

I. L... II. ①巴... ②O... III. Linux 操作系统 IV. TP316.89

中国版本图书馆 CIP 数据核字 (2006) 第 013988 号

江苏省版权局著作权合同登记

图字: 10-2005-290 号

©2004 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and  
Southeast University Press, 2005. Authorized translation of the English edition,  
2004 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.

英文原版由 O'Reilly Media, Inc. 出版 2004。

简体中文版由东南大学出版社出版 2005。英文原版的翻译得到 O'Reilly Media,  
Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者 ——  
O'Reilly Media, Inc. 的许可。

版权所有。未得书面许可, 本书的任何部分和全部不得以任何形式重制。

书 名 / Linux 随身指南

书 号 / ISBN 7-5641-0254-3

责任编辑 / 张烨

封面设计 / Emma Colby, 张健

出版发行 / 东南大学出版社 (press.seu.edu.cn)

地 址 / 南京四牌楼 2 号 (邮政编码 210096)

印 刷 / 扬中市印刷有限公司

开 本 / 787 毫米 × 980 毫米 16 开本 14 印张 235 千字

版 次 / 2006 年 5 月第 1 版 2006 年 5 月第 1 次印刷

印 数 / 0001-4000 册

定 价 / 28.00 元 (册)

## O'Reilly Media, Inc. 介绍

为了满足读者对网络 and 软件技术知识的迫切需求,世界著名计算机图书出版机构 O'Reilly Media, Inc. 授权东南大学出版社, 翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc. 是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司, 同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》(被纽约公共图书馆评为二十世纪最重要的 50 本书之一) 到 GNN (最早的 Internet 门户和商业网站), 再到 WebSite (第一个桌面 PC 的 Web 服务器软件), O'Reilly Media, Inc. 一直处于 Internet 发展的最前沿。

许多书店的反馈表明, O'Reilly Media, Inc. 是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比, O'Reilly Media, Inc. 具有深厚的计算机专业背景, 这使得 O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员, 或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家, 而现在编写著作, O'Reilly Media, Inc. 依靠他们及时地推出图书。因为 O'Reilly Media, Inc. 紧密地与计算机业界联系着, 所以 O'Reilly Media, Inc. 知道市场上真正需要什么图书。

---

# 目录

<b>1. 本书内容 .....</b>	<b>1</b>
1.1 何谓 Linux? .....	2
1.2 何谓 Fedora Linux? .....	2
1.3 何谓“命令”? .....	3
1.4 普通用户与超级用户 .....	4
1.5 本书读法 .....	4
<b>2. 求助 .....</b>	<b>7</b>
<b>3. Fedora：乍看之下 .....</b>	<b>8</b>
3.1 shell 的角色 .....	10
3.2 如何启动 shell .....	10
<b>4. 登录、注销、关机 .....</b>	<b>11</b>
<b>5. 文件系统 .....</b>	<b>12</b>
5.1 个人目录 .....	14
5.2 系统目录 .....	14
5.3 操作系统目录 .....	18
5.4. 文件保护 .....	19

6. shell .....	20
6.1 shell 与程序的比较 .....	21
6.2 bash shell 概述 .....	21
6.3 任务控制 .....	31
6.4 结束执行中的命令 .....	34
6.5 结束 shell .....	35
6.6 Bash shell 的配置文件 .....	35
7. 安装软件 .....	35
7.1 tarball .....	40
8. 文件基本操作 .....	41
9. 目录操作 .....	45
10. 显示文件内容 .....	47
11. 文件的创建与编辑 .....	56
11.1 默认编辑器 .....	57
12. 文件属性 .....	62
13. 文件位置 .....	72
14. 文件文本操作 .....	78
14.1. 高级文件操作 .....	89
15. 文件压缩 .....	91
16. 文件比较 .....	96
17. 磁盘与文件系统 .....	101

18. 备份 .....	106
19. 文件打印 .....	113
20. 拼写检查 .....	115
21. 查看进程 .....	117
22. 控制进程 .....	121
23. 用户与其操作环境 .....	124
24. 管理用户账号 .....	129
25. 改变身份 .....	133
26. 组管理 .....	134
27. 基本的主机信息 .....	136
28. 检查远程主机 .....	139
29. 网络连接 .....	143
30. 电子邮件 .....	147
31. 网络浏览 .....	152
32. 新闻组 .....	157
33. 实时消息 .....	159
34. 屏幕输出 .....	161
35. 算术运算 .....	167



36. 日期与时间 .....	170
37. 调度工作 .....	174
38. 图像、屏幕保护程序 .....	179
39. 音频、视频 .....	182
40. shell script 程序设计 .....	185
40.1 空格与换行 .....	186
40.2 变量 .....	186
40.3 输入与输出 .....	187
40.4 逻辑值与返回值 .....	187
40.5 条件判断 .....	190
40.6 循环 .....	193
40.7 break 与 continue .....	195
40.8 shell script 的制作与运行 .....	196
40.9 命令行参数 .....	197
40.10 返回结束状态 .....	198
40.11 除了 shell Scripting 之外 .....	199
后 记 .....	199
致 谢 .....	199
索引 .....	201

---

# Linux 随身指南

欢迎来到 Linux 的世界！本书给新手提供 Linux（特别是 Fedora Linux）系统的一般性介绍，以及常见和实用命令的入门指南。对于已经有经验的 Linux 老手，则可略过介绍性的内容，将本书作为速查手册。

## 1. 本书内容

本书是精简的入门指南，而非完整的参考书。本书重点在于介绍有助于提高工作效率的 Linux 重点功能，所以，这里不会说明每个命令和每个选项（如果我们省略了你想要知道的部分，请见谅），也不会探究操作系统内部的细节。精简、贴切和必要是我们的宗旨。

书中重点放在命令（command）的介绍上，也就是在文本操作环境中用来指示 Linux 系统应该要做什么事的“魔幻咒语”，如 `ls`（列出文件）、`grep`（搜索文件中的特定文本）、`xmms`（播放音频文件）以及 `df`（计算磁盘空间用量）。至于 GNOME 和 KDE 之类的窗口环境，只能简单介绍，毕竟这只是一本篇幅有限的随身指南。

为了提供流畅的学习途径，本书题材的编排主要是以功能为依据。举例来说，提到显示文件内容的题材时，我们会把能够显示文件内容的命令集中在一起，像 `cat`（用于显示短篇文本文件）、`less`（用于显示长篇文本文件）、`od`（用于显示二进制文件）、`ghostview`（用于显示 Postscript）等，然后再依次说明每一个命令，并提供典型用法和常用选项的说明。

我们假设你已经有一个Linux系统的账号，而且知道如何以用户名称和密码登录该系统。如果不是，请咨询系统管理员；或者，如果你有自己的Linux系统，请使用在安装Linux时所建立的账号。

## 1.1 何谓Linux?

Linux是一种广受欢迎的开放源码（open source）计算机软件环境，其功能性与重要性足以与Microsoft Windows和Apple Macintosh相提并论。Linux大致由以下四大部分组成：

### 内核 (kernel)

操作系统中最底层、也最重要的部分，负责处理文件、磁盘、网络操作以及其他必要的基础功能。

### 应用程序

Linux备有数以千计的程序，范围涵盖文件操作、文本编辑、数学、排版、音频、视频、程序设计、网站设计、加密、光盘制作……应有尽有。

### shell

一种交互操作界面，可接受用户下达的命令，并显示运行结果。Linux系统中有各种各样的shell，如Bourne shell、Korn shell、C shell等。本书只讨论使用得最多的bash（Bourne Again shell），因为所有shell的基本功能都很相似。

### X

计算机视频硬件的驱动，并提供基本的GUI组件（窗口、菜单、图标、鼠标支持）绘制能力的图形系统。X本身只提供图形绘制能力，它还必须搭配KDE或GNOME之类的“桌面环境”才会有较好的窗口系统。

## 1.2 何谓Fedora Linux?

Fedora Linux的前身为知名的Red Hat Linux（注1），它是Red Hat公司和Fedora Project（<http://fedora.redhat.com>）所创建的Linux发行包

---

注1： Red Hat公司目前已经将业务重心转移到企业级的Linux产品（RHEL），而Fedora Project算是新版RHEL的试金石。

(distribution)。本书内容以2004年底开始发行的Fedora Core 3版为准，但是大部分的内容也适用于其他版本的发行包。

### 1.3 何谓“命令”？

你在shell环境下所输入的Linux命令(command)，通常包括“程序名称”、“选项”、“参数”。程序名称指出程序文件在磁盘上的存放位置；选项（通常以“-”符号开头）用来调整程序的行为；参数提供程序运行所需的数据（通常是I/O位置）。例如：

```
$ wc -l myfile
```

上述命令可计算*myfile*文件中含有多少行文本。在此例中，*wc*是程序名称(word count, 计算字数)，*-l*是选项(line, 表示只计算行数)，*myfile*是参数(数据来源)。\$是shell的提示符，代表shell正等着你输入命令。

同一个命令中，可以同时存在多个选项，例如：

```
$ myprogram -a -b -c myfile    三个分开的选项
```

很多程序容许你将多个选项并在一起，变成下面这样：

```
$ myprogram -abc myfile    效果和 -a -b -c 一样
```

不过，并非所有程序都能接受合并形式的选项。

相比于运行单一程序，命令的形式可以更复杂：

- 同一个命令可以涉及多个程序，这些程序可能会被依次运行，也可能同时运行。它们彼此之间以|（管道）符号将I/O管道衔接在一起（将一个命令的输出导引到另一个命令的输入）。
- 选项并未标准化。同样形式的选项，在不同程序中分别有不同的含义。举例来说，同样的*-l*选项，对于*wc*程序，其意义是“计算文本行数”(line count)，但是对于*ls*程序，则代表“长格式输出”(long format)。即使是相同的功能，不同的程序也可能以不同的选项表示。以“安静模式”为例，某些程序所使用的选项可能是*-q*（意指run quietly），但是某些程序可能是用*-s*选项（意指run silently）来表示。

- 同理，参数也有没标准化。虽然它们多半是代表要输入或输出的文件名，但也可能是其他意义的数据，像目录名称或正则表达式 (regular expressions)。
- Linux 的命令行操作界面，shell，内置了一套程序语言。所以，你的命令不仅可以这样写：“运行此程序”，也可以这样写：“若今天是周二，运行此程序，否则，对每个扩展名为 .txt 的文件，各运行六次另一个程序”。

## 1.4 普通用户与超级用户

Linux 是多用户操作系统。Linux 系统里的每位用户，各有彼此不同的用户名 (username)，像 “smith” 或 “funkyguy”，而且每人各拥有系统的一部分 (合理的) 私有专区，以便进行个人工作。此外，还有一个特别指定的用户，其用户名称为 *root*，俗称“超级用户” (superuser)，他有可以在系统上执行任何工作的特权。普通用户受到限制，虽然他们可以运行大多数的程序，但是他们通常只能修改自己的文件。另一方面，超级用户却有权任意创建、修改或删除任何文件，并运行任何程序。

本书有不少命令只有 *root* 才能够成功运行。对于这种情况，我们以 # 为 shell 提示符 (跟 bash shell 的惯例一样)：

```
# supercommand goes here
```

如果是普通用户可以运行的命令，则以 \$ 作为 shell 提示符，

```
$ normalcommand goes here
```

要成为超级用户，你不必先注销然后再登录，只需使用 *su* 命令 (参阅 133 页，“25. 改变身份”) 就可以直接改变身份：

```
$ su -l
Password: *****    输入 root 的密码
#
```

## 1.5 本书读法

当我们描述某命令时，会先介绍该命令的一般语法。举例来说，*wc* (word count) 程序的一般语法如下：

```
wc [options] [files]
```

上述语法的意义是：你得照实输入 `wc`，但是可以视情况选择是否提供选项与文件名。方括号（`[]`）本身只是强调它们所包含的内容是选择性的，当你决定要提供选项或文件名时，并不需要加注方括号。以斜体字表示的部分，表示应该被替换成实际的特定值（比如说，实际文件的名称）。如果你见到选项或参数之间以 `|` 符号隔开，而这些选项或参数的外围有一对圆括号，则表示圆括号内的项目可择一而定。例如：

```
ls (file | directory)
```

此例表示 `ls` 可接受的参数可以是文件也可以是目录，根据实际需要而定。

## 输入与输出

大部分 Linux 程序的数据来源都是标准输入（standard input），而运行结果则是送到标准输出（standard output），如果程序发生错误，则将错误信息送到标准错误输出（standard error）上。这三个标准 I/O 是 Linux 设置的流（stream），分别以 `stdin`、`stdout` 与 `stderr` 来表示。在默认情况下，`stdin` 的来源是键盘，而 `stdout` 与 `stderr` 都是指向屏幕（注 2）。使用重定向符号（redirect operator）可改变个别程序的 I/O 流向，使其指向文件或管道。为了方便说明，当我们声称某命令“读取”时，其实是指该程序从 `stdin` 接受数据；若是说某命令“输出”，其真正含义是将信息送到 `stdout`，而不是真正用打印机打印出来。

## 标题格式

每个命令的正式说明之前，会有一段标题列出该命令的特性，以 `ls`（list files）命令为例：

```
ls [options] [files]
```

**coreutils**

```
/bin
```

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

横隔线上方左侧是命令名称（`ls`）与其语法，右侧是该命令所属的 RPM 包

---

注 2： 将错误信息与标准信息分成两个流，这样我们可分开处理它们。比如说，将程序的输出导入到一个文件，但是让错误信息仍输出到屏幕上。

---

(coreutils)；横隔线下方左侧是该命令的存放位置 (/bin)，右侧的六个项目，浅色表示不支持，深色表示支持。这六个项目的意义如下：

#### *stdin*

命令的默认行为是从标准输入（键盘）读取数据。

#### *stdout*

命令的默认行为是将信息送到标准输出（屏幕）。

#### *-file*

以 - 符号来代替输入文件名时，改从 *stdin* 读取数据；若是以 - 符号代替输出文件名时，则将信息送到 *stdout*。举例来说，下列 *wc* 命令会依次读取 *file1*、*file2*、*stdin* 与 *file3*：

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

#### *--opt*

以 -- 符号作为命令行选项时，表示“选项到此结束”，也就是说，命令行在 -- 之后的部分都不会被当成选项来解释。当你要操作的文件，其名称的第一个字符刚好是 -，就需要使用这种方法来避免误解。举例来说，如果想用 *wc* 计算 *-foo* 文件的行数，直接运行 *wc -foo* 命令将会失败，因为 *wc* 会告诉你它不认识 *-foo* 选项，你必须将命令改成 *wc -- -foo* 才有效。对于不支持 -- 语法的程序，你可以在文件名之前加上 ./（表示“当前工作目录”），迫使 shell 将该字符串当成文件名解释。例如：

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

#### *--help*

--help 选项促使命令输出更详细的语法解释信息。

#### *--version*

--version 选项促使命令输出其版本编号。

---

## 按键符号

本书以特定符号来表示按键顺序。按照 Linux 文件的惯例，^ 符号表示“按住 **Ctrl** 键不放”，所以，**^D** 的意思是按下 **Ctrl** 键不放，然后再按下 **D** 键，

`[ESC]`表示“按下Escape键”；而`[Enter]`和`[Spacebar]`之类的按键，就不再解释了。

## 你的好朋友，echo 命令

本书有很多例子会利用echo命令（参阅161页，“34 屏幕输出”）来输出信息。echo是最简单的命令之一，它的作用是将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 shell is $USER          环境变量USER
My name is smith
```

## 2. 求助

如果本书所提供的信息不敷所需，你还有几件事可做：

### 使用 man 命令

man可显示特定命令的在线说明（所谓的manpage）。比如说，若想查看ls命令的用法：

```
$ man ls
```

如果不清楚明确的命令，只有模糊的概念，则可利用-k选项来列出含有特定关键字（keyword）的说明：

```
$ man -k database
```

### 使用 info 命令

许多Linux程序提供了比manpage更详尽、更容易使用的info文件，这类文件可用info命令来查看。info的用法和man几乎一样：

```
$ info ls
```

当info找不到特定程序的info文件时，它会显示该程序的manpage。单独使用info而不加任何选项时，它会列出所有可找到的文件。关于如何使用info系统，请看info info。

### 试着使用 --help 选项

许多Linux程序内置简单的说明信息，使用--help选项通常能调出这类信息：



```
$ ls --help
```

#### 检查 `/usr/share/doc` 目录

许多程序将说明文件存放在此目录下,而且是以程序名称与版本编号组织在一起。举例来说,关于 Emacs 文本编辑器 21.3 版的说明文件,可在 `/usr/share/doc/emacs-21.3` 目录下找到。

#### GNOME 和 KDE 的辅助系统

在 GNOME 或 KDE 环境下,单击主菜单中的 Help 项,通常可获得很详尽的说明。

#### Fedora 专属网站

Fedora Project 的正式网站位于 <http://fedora.redhat.com>。在 <http://fedora.arioo.net> 可找到许多非正式的 FAQ。当然,本书也有专属网页:

<http://www.oreilly.com/catalog/linuxpg/>

#### Usenet 新闻组

关于 Linux 组的话题太多了,诸如 `comp.os.linux.misc` 和 `comp.os.linux.questions` 都颇为热门。关于 Red Hat 的信息,我们推荐 `alt.os.linux.redhat`, `comp.os.linux.redhat`, `linux.redhat.misc` 以及 `linux.redhat`。利用 Google 的新闻组搜索服务 (<http://groups.google.com>),可找到不少有助于解决问题的宝贵线索。

#### Google

如果需要更多更详尽的说明与教材,用 Google 找一找吧 (<http://www.google.com>)。

## 3. Fedora: 乍看之下

当你登录 Fedora (或其他) Linux 系统时,显示画面可能是一个图形化的桌面 (注 3),就像图 1 那样。在这个虚拟桌面上,可以找到下列东西:

- 顶端或底端有一个类似 Windows 任务栏的东西,以 GNOME 或 KDE 的术语来说,这种东西称为“面板”(panel)。面板上有许多很像按钮的东西:

---

注 3: 如果是通过网络从远程登录,则见到的是命令行环境。

- “红帽”以及“应用程序”图标。单击它们时，会跳出工具程序或应用程序的菜单。
  - 各种常见应用程序的图标，像Firefox浏览器、Evolution电子邮件程序、OpenOffice.org办公软件等。
  - 一组用于切换桌面的工具。此工具通常由四个方框组成，每个方框分别代表不同的虚拟桌面，让你可以迅速在不同桌面之间进行切换。
  - 一个代表系统软件是否更新到最新版的警示图标。蓝色打钩符号代表你的系统已经更新；若出现红色的惊叹号，表示又出现了更新版本的软件包。
  - 时钟
- 除了面板之外，桌面上还有其他图标，像垃圾桶、软盘、个人文件夹等。



图 1: Fedora 的图形桌面

Fedora 随附了多个类似风格的 GUI 环境，你所见到的不是 GNOME 就是 KDE。这两种环境的操作方式大同小异。事实上，GNOME 和 KDE 原本是两个风格迥异的桌面环境，但是 RedHat 随附的 KDE 包已经被刻意改成很接近 GNOME 的风格。尽管如此，由于 GNOME 和 KDE 的高度可塑性，只要你愿意，你随时可以动手改回来。顺带一提，Fedora Core 3 另外提供了 XFCE

桌面环境，专供资源有限的机器使用。如果你对 GNOME 和 KDE 的运行速度感到不满，或许 XFCE 是个好选择。

## 3.1 shell 的角色

广义而言，GNOME 和 KDE 也算是操作系统的 shell，因为它们提供了容易操作的 GUI 人机界面。对于某些用户而言，GUI shell 是他们使用 Linux 的主要方法。包括 Fedora 在内的许多发行包，竞相提供更优质、更容易操作、更花哨的 GUI 环境，让用户可以更轻易地编辑文件、阅读电子邮件和浏览网页。

在 shell 光彩照人的另一端，是单调而且不讨喜的文本命令行操作界面。奇怪的是，真正能有效发挥 Linux 潜力的 shell，反而是这些不起眼的 bash、csh、tcsh 等。相比较于 GUI 的花哨图形和菜单，文本模式的 shell 确实不容易上手，然而，一旦领略各种要领，那种得心应手的感觉，反而使得它们成为大多数 Linux 用户的最爱。

在本书后文，shell 一词将采取狭义定义，专指文本模式的 shell（即 bash），而这也是本书的主要重点。

## 3.2 如何启动 shell

GNOME、KDE 或 Linux 的任何其他 GUI 环境，可通过 *xterm*、*gnome-terminal*、*konsole* 之类的“终端机仿真程序”来开启 shell 窗口。这些程序的功能略有不同，有些提供花哨的字型变化，有些支持多种终端机的形式，但不管怎样，它们的基本作用都是一样的：提供一个让你下达文本命令的窗口。开启终端机窗口的方法随实际使用的桌面环境而异，但不外乎以下两种：“主菜单”→“系统工具”→“终端机”，或是在桌面空白处单击鼠标右键，然后点击菜单中的“开启终端机”。

请勿将“终端机仿真程序”（例如 *konsole*）与其中的 shell（bash）混为一谈。前者只是提供一个亮丽的容器，后者才是真正输出提示符供你使用的主角。

如果你不是在 GUI 环境下，比如说，通过网络从远程登录或是你就坐在没启动 X 的计算机之前，那么当你登录时，系统会立刻自动打开一个 shell 供你使唤。

## 4. 登录、注销、关机

我们假设你已经知道如何以你的Linux账号登录系统。若要从GNOME或KDE注销，单击面板上的红帽图标，然后选择菜单中的“注销”。若要结束终端机窗口中的shell，你可以直接关闭终端机窗口本身，也可以使用`exit`或`logout`命令来结束shell，当shell结束时，终端机窗口通常会跟着关闭。同样的命令也可以用来注销文本模式的shell（不管远程或本机）。

绝对不可贸然关闭Linux系统的电源。对于任何的多用户多任务操作系统而言，贸然断电会引发丢失数据的风险。你必须依据标准操作程序来关机。在GUI环境下，当你注销桌面环境后，所出现的登录画面会有一个可让你关机的选项；或者，在你注销的过程中，会出现一个对话框让你选择要注销、重新启动计算机还是关机。在文本模式下，则可以用`shutdown`命令来关闭系统。

**shutdown [options] time [message]**

**SysVinit**

/sbin

stdin stdout -file --opt --help --version

`shutdown`可关闭或重新启动Linux系统。只有superuser才可以使用此命令。下列命令会发布“scheduled maintenance”信息给所有已登录系统的用户，然后于十分钟之后关机：

```
# shutdown -h +10 "scheduled maintenance"
```

`time` 参数的格式相当多变：+ 符号后接着一个十进制数，表示几分钟之后（例如，+10 表示十分钟之后）；`hh:mm` 是绝对时间（像 16:25）；也可以是 `now`（立即关机！）。

不指定选项时，`shutdown` 命令会让系统进入单用户模式（single-user mode）。这是一种专用于维护系统的操作模式，所有非必要服务都会被关闭，而且只容许一个人（从控制台）登录系统。有两种方法可离开单用户模式：再使用一次 `shutdown` 命令来关闭系统或让系统重新开机，或者，按下 **^D** 让系统进入正常的多用户模式。

### 常用选项

`-r` 重新开机。

- h 关闭系统。
- k 开玩笑 (Kidding): 对所有用户发布警告信息, 假装系统正要关闭, 但其实不是真正运行关机程序。
- c 撤销正在进行的关机程序 (省略 *time* 参数)。
- f 重新开机时, 跳过 *fsck* 程序 (参阅 102 页) 所进行的文件系统检查步骤。
- F 重新开机时, 强制检查文件系统。

关机、单用户模式以及各种系统状态的技术信息, 可参考 *init* 和 *inittab* 的在线说明。

## 5. 文件系统

在能够顺利运用任何 Linux 系统之前, 你必须先熟悉 Linux 的文件以及它们的组织方式。Linux 系统中的每个文件必定被收纳于某个集合中, 称为目录 (directory)。目录的地位, 就像 Windows 或 Macintosh 系统上的“文件夹”。所有目录共同组成一个树状结构, 这整个结构就称为目录树 (directory tree)。每个目录都可以包含其他目录, 称为子目录 (subdirectories), 而每个子目录本身也可以包含其他文件和子目录, 依此类推。最顶层的目录称为根目录 (root directory) (注 4), 代表根目录的符号是 */*。

描述文件与目录位置的字符串称为路径 (path)。举例来说:

```
/one/two/three/four
```

上述路径表示根目录 (*/*) 下的 *one* 子目录下的 *two* 子目录下的 *three* 子目录下的 *four* 文件 (或子目录)。以根目录 (*/*) 为起点的路径称为绝对路径 (absolute path), 否则称为相对路径 (relative path)。稍后有这方面的详细解释。

---

注 4: Linux 系统没有 Windows 或 DOS 的“磁盘驱动器符号”的概念, 而是以根目录来容纳所有文件与目录, 包括实际上存放于不同磁盘驱动器上的目录与文件。

当你进入某个 shell 环境时，该 shell 必定“位于”某个目录。更专业地说，shell 有一个当前工作目录（current working directory），你在 shell 环境所下达的命令，如果没有刻意指定作用对象，该命令就是作用在工作目录；如果命令行中含有相对路径，则该路径的起点是相对于（又是这个字眼！）当前的工作目录。

举例来说，假设 shell 当前的工作目录是 */one/two/three*，而你提供给命令的文件路径是 *myfile*，则该命令真正会操作的文件其实是 */one/two/three/myfile*；如果你给的路径是 *a/b/c*（相对路径），则其真正路径是 */one/two/three/a/b/c*。

为了描述目录之间的继承关系，shell 提供了两个特殊符号，分别是代表当前工作目录的“.”（一个点），以及代表上级目录的“..”（两个点）。举例来说，假设当前工作目录是 */one/two/three*，则 *.* 代表此目录，而 *..* 代表 */one/two* 目录。

使用 *cd* 命令可将 shell “搬移”到另一个目录，换个说法是改以另一个目录为工作目录：

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

以术语来说，上述命令将 shell 的工作目录切换到 */one/two/three*。*cd* 可接受绝对路径，也可以接受相对路径：

```
$ cd d           进入 d 目录
$ cd ../mydir     跳到上一层目录，然后再进入其 mydir 子目录
```

文件和目录的名称可以含有你所希望的大多数字符：大小写字母、数字、点、连字符、下划线以及除了“/”之外的大多数符号。“/”不能成为文件名，原因是它被当成路径分隔符使用。假如“/”是有效的文件名称或符号，shell 就无法判断 */one/two* 到底是 */one* 目录下的 *two* 文件，还是根目录下的 *one/two* 文件。

虽然可作为有效文件名的符号很多，但是在实际使用时，应该尽量避免使用空格、\*（星号）、（）（圆括号号）、[]（方括号）、?（问号）以及任何对 shell 有特殊意义的符号（例如 |）；否则，每次要描述含有特殊字符的文件名时，都得用引号包住整个名称，或是在每个特殊字符之前加上 \，才能够避免 shell 误解你的意思（参阅第 27、28 页的“界定”与“转义”小节）。

## 5.1 个人目录

Linux 系统上的每位用户都有一个专用于存放个人文件的目录，称为个人目录或主目录 (home directory)。一般用户的个人目录通常集中位于 */home* 目录下，而且目录名称与用户名称相同，例如 */home/smith*、*/home/jones* 等。系统管理员也有自己专用的个人目录，而且通常是固定位于 */root* 下。

有几种方法可找出或描述你的个人目录。

*cd*

如果没给任何参数，*cd* 会把你带回个人目录，成为 shell 的新工作目录。

### HOME 环境变量

环境变量 HOME（参阅第 23 页的“环境变量”小节）含有你的个人目录的绝对路径名称。

```
$ echo $HOME      echo 会输出其参数
/home/smith
```

~

*echo* 命令单独出现时，它相当于 HOME 环境变量，也就是当前用户的个人目录的绝对路径名称：

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

如果在后面接一个用户名称（像 *~smith*），则代表该用户的个人目录：

```
$ cd ~alex
$ pwd      “Print Working Directory” 命令（译注 1）
/home/alex
```

## 5.2 系统目录

典型的 Linux 系统含有上千个系统目录 (system directory)。这类目录含有操作系统的文件、应用程序、说明文件以及除了用户个人私有文件（这类文件通常放在 */home* 目录之下）之外的任何文件。简单来说，除了 */root* 与 */home* 之外的其他目录都可称为系统目录。

---

译注 1： 正常的系统不容许你进入他人的个人目录，除非对方修改自己个人目录的访问模式。

---

系统管理员经常需要访问系统目录，而一般用户则不需要。虽然系统目录的数量很多，但其实组织方式颇有条理，只要具备一些基础知识，就很容易了解或猜出每个目录的内容或用途。

系统目录的完整路径名称通常可分成三部分，分别是范畴（scope）、分类（category）以及应用（application）（注5）。以 `/usr/local/share/emacs` 目录为例（图2），它所属的范畴为 `/usr/local`（个别系统所安装的文件），所属的分类为 `share`（特定程序的数据与说明文件），所属的应用是 `emacs`。合起来解释便是“供给安装于个别系统的Emacs应用程序（一种文本编辑器）的专用数据与说明文件”。以下依“分类”、“范畴”、“应用”顺序来解释这三部分的意义。



图2：目录内容的范畴、分类、应用

### 目录路径的“分类”（category）

分类（category）代表目录所存放文件的类型。举例来说，若分类为 `bin`，则可以合理猜测该目录下存放的是程序文件。以下是常见的分类：

#### 程序类

- `bin`            程序文件（通常是已编译成可直接执行的 binary 格式）。
- `sbin`          专供管理员使用的程序文件（通常也是 binary 格式）。
- `lib`            供程序使用的函数库。
- `libexec`       供其他程序（而非用户）调用的程序。可想象成“library of executable programs”。

#### 文件类

- `doc`            说明文件。
- `info`          可使用 Emacs 的内置辅助系统观看的说明文件。

---

注5： 这三个名词不是标准术语，但是有助于你了解系统目录的组织原则。



<i>man</i>	可用 <i>man</i> 查看的在线说明文件。这类文件通常存储成压缩格式，或是含有 <i>man</i> 才能解读的排版指令。
<i>share</i>	针对个别程序的额外文件，像安装指示与范例。
<b>配置类</b>	
<i>etc</i>	系统性的配置文件。
<i>init.d</i>	与开机程序有关的配置文件。 <i>rc1.d</i> 、 <i>rc2.d</i> 等也是相同性质。
<i>rc.d</i>	与 <i>init.d</i> 相同。
<b>编程类</b>	
<i>include</i>	C 程序语言的头文件 ( <i>.h</i> )。
<i>src</i>	程序源代码。
<b>网站类</b>	
<i>cgi-bin</i>	供网站服务器运行的程序。
<i>html</i>	网页。
<i>public_html</i>	网页，通常位于个人目录下。
<i>www</i>	网页。
<b>显示类</b>	
<i>fonts</i>	字体。
<i>X11</i>	X 窗口系统的文件。
<b>硬件类</b>	
<i>dev</i>	设备文件。内核提供给应用程序访问硬件的接口。
<i>mnt</i> 、 <i>misc</i>	挂载点：用于访问插入式设备（光盘、软盘）的目录。
<b>运行状态类</b>	
<i>var</i>	存放随个别计算机运行状态而异的文件。
<i>lock</i>	锁定文件。程序创建锁定文件，让其他程序（通常是另一个相同程序）知道自己的存在。这通常是为了避免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>	记录重要系统事件的日志文件。包括错误、警告以及信息性的消息。
<i>mail</i>	新进邮件的邮箱。
<i>run</i>	PID 文件，含有运行中进程的标识符。这类文件常被用来传送信号给特定进程。
<i>spool</i>	过渡性质的文件，像等待寄出的邮件、打印工作、调度工作等。
<i>tmp</i>	供程序或用户临时使用的存储空间。
<i>proc</i>	操作系统的状态。参阅第 18 页“5.3. 操作系统目录”。

### 目录路径的“范畴” (scope)

目录路径中的范畴 (scope)，从更高的角度描述该目录在整个目录结构中的地位。以下是几个比较常见的范畴：

<i>/</i>	Linux 提供的系统文件（称为“根目录”）。
<i>/usr</i>	更多提供给 Linux 的系统文件（发音为“user”）。
<i>/usr/games</i>	游戏（意外吧！）。
<i>/usr/kerberos</i>	构成 Kerberos 验证系统的文件。
<i>/usr/local</i>	个别计算机专用的系统文件。比如说，个人另外安装的软件包。
<i>/usr/X11R6</i>	构成 X 窗口系统的文件。

所以，Linux 系统中的 */lib*、*/usr/lib*、*/usr/local/lib*、*/usr/games/lib* 和 */usr/X11R6/lib* 等目录，其实都属于 *lib* 类别。如果你的系统中有类似 */my-company/lib*、*/my-division/lib* 之类的目录，那可能是系统管理员创建给特定部门来安装程序库所使用的目录。

在实际使用中，*/* 和 */usr* 之间并没有很明确的差别，但是在概念上，*/* 属于较低的层次，而且比较接近于操作系统。所以，*/bin* 含有最基本的程序（例如 *ls*、*cat*），*/usr/bin* 含有 Linux 发行版本所提供的各种应用程序，而 */usr/local/bin* 含有系统管理员另外安装的软件。这些虽然不是硬性规定，但却是约定俗成的惯例。

## 目录路径的“应用” (application)

目录路径中的“应用”部分通常是程序的名称。在范畴与分类之后 (比如说 */usr/local/doc*)，个别程序可以有自己的子目录 (例如 */usr/local/doc/myprogram*) 来容纳所需文件。

## 5.3 操作系统目录

### */boot*

用于启动系统的文件。这是内核映像文件 (文件名通常是 */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*

Linux 内核用于提供进程运行状态的虚构目录。

*/proc* 目录下的文件并不实际存在于磁盘中，而是内核虚拟出来的。当你读取它们时，内核才实时提供状态信息。因此，*/proc* 下的所有文件的长度都是 0，而且大部分都是只读的 (也有少部分可写的文件，它们可用于改变系统行为)，日期为当天：

```
$ ls -l /proc/version
-r--r--r-- 1 root root 0 Dec 20 13:24 /proc/version
```

虽然长度为 0，但是当你读取时，它们却会很神奇地告诉你关于 Linux 内核的版本信息：

```
$ cat /proc/version
Linux version 2.4.22-1.2115.nptl ...
```

*/proc* 下的文件主要是提供给应用程序使用，一般用户很少需要用到它们。不过，建议你花点时间探索它们的内容，那将有助于了解系统的运行状态。这里有些值得探讨的文件：

*/proc/ioproports*    I/O 硬件的列表。

*/proc/version*    操作系统的版本。uname 命令可输出相同的信息。

*/proc/uptime*    系统开机时间。也就是从系统开始运行至目前所经过的秒数。uptime 命令可提供较容易解读的相同信息。

- /proc/nnn* 系统中的每一个进程都有一个自己专属的 */proc/nnn* 目录，其中的 *nnn* 是进程的标识符 (PID)。此目录下含有个别进程的环境信息 (context)。
- /proc/self* 正在运行中的当前进程的环境信息。这其实是某个 */proc/nnn* 文件的符号链接，由内核动态更新。使用 `watch ls -l /proc/self` 命令可看到 */proc/self* 经常改变其链接对象。

## 5.4. 文件保护

同一个Linux系统中可能同时具有多个可供用户登录的账号。为了维护私密与安全性，每位用户都只能访问系统中的某些文件，而不是全部文件。访问控制的原则取决于下列两个问题的答案：

用户是谁？

对于任何文件与目录，用户可分成三种身份：拥有者 (owner)、用户组 (group)、其他人 (other)。

拥有者具有完整的操作权限，可对文件或目录做任何事。一般而言，文件的制作者就是该文件的拥有者，但是也有许多例外。

用户组是系统管理员预先定义的一组用户，他们对文件或目录拥有相同的权限，而且同是某个组的成员。参阅第 134 页的“26. 组管理”。

既不是拥有者也非用户组成员的其他任何用户，都视为“其他人”。

何种权限？

文件的操作权限分为三种：可读 (read)、可写 (write)、可执行 (execute)。可写表示可修改文件内容，通常能写就应该能读。目录也有同样的三种操作权限，但是意义不一样：“可读”表示可看到目录所含的文件；“可写”表示可新增或删除该目录下的文件；“可执行”表示可以 `cd` 命令进入该目录。

用户身份与各种身份的操作权限合称为访问模式 (access mode)。使用 `ls` 的 `-l` 选项，可查看文件或目录的访问模式。例如，若要查看特定文件的访问模式，可使用下列形式的命令：

```
$ ls -l filename
```

若是要查看特定目录的访问模式，多加一个 `-d` 选项：

```
$ ls -ld dirname
```

ls 输出信息中，最左侧的 10 个字符就是该文件或目录的访问模式。第一个字符代表文件的类型，其余的 9 个字符，每三个一组，分别代表“拥有者”、“用户组”、“其他人”的权限，每组权限由三个权限位构成：“可读”(r)、“可写”(w)、“可执行”(x) 权限。举例来说：

```
$ ls -dl /usr/src
drwxr-xr  4 root wheel 4096 Nov 11 14:20 /usr/src
```

在此例中，/usr/src 目录的拥有者用户 root，所属用户组为 wheel。拥有者 root 具有完整的权限，wheel 用户组的成员能读取此目录的内容，也能进入该目录，但是不能在该目录下新建或删除文件。不是 root 也不是 wheel 用户组成员的其他人，只能查看该目录下有哪些文件。

各种权限符号的位置与意义汇总如下：

位置（最左侧为 1）	意义
1	文件类型：- = 文件，d = 目录， l = 符号链接，p = 具名管道， c = 字符设备，b = 块设备
2-4	文件拥有者的读、写、执行权限
5-7	用户组成员的读、写、执行权限
8-10	其他人的读、写、执行权限

第 41 页的“8. 文件基本操作”会详细解释 ls 的细节。用于改变文件或目录拥有者或所属组的命令是 chown 与 chgrp，改变访问模式的命令是 chmod。这三个命令分述于第 62 页的“12. 文件属性”。

## 6. shell

要让 Linux 系统能执行你的命令，需要有一个能够承接命令的地方，这个“地方”就是所谓的 shell，也就是 Linux 的命令行操作界面：你输入一行命令，然后按下回车键，shell 便会运行你所要求的程序（或一组程序）。第 8 页的“3. Fedora：乍看之下”解释了如何启动 shell。

举例来说，要看当前有哪些用户登录系统，可在 shell 下达 who 命令：

```
$ who
lin      vc/2      Dec 16 10:12
me       pts/2     Dec 20 16:12 (192.168.0.185)
```

此例中的\$是shell的提示符，当它出现时，表示shell已经准备好接收你的命令。在一个命令里，可同时调用多个程序，甚至将不同程序连接在一起，使它们接力完成工作。下列命令将who的输出导入到wc程序：

```
$ who | wc -l
2
```

在此例中，who逐个输出每一位已登录的用户（每人各一行）（注6），而wc -l计算输入数据的行数。两者中间的|符号，称为管道（pipe），其作用是将左边程序的输出作为右边程序的输入。所以，上述命令便可算出当前系统中有多少用户。

shell本身其实也是一个程序，事实上，Linux系统提供了好几种shell程序供你自由挑选。本书只讨论默认的Bash（the “Bourne-Again shell”），它位于/bin/bash。

## 6.1 shell与程序的比较

当命令运行时，它可能启动某个Linux程序（例如who），但也可能是运行内置命令（built-in command），也就是shell本身的功能。type可告诉你特定命令属于哪一种类型：

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

搞清楚哪些是shell内置的功能，而哪些又是Linux系统所提供的外部程序，将有助于你编写shell script。以下几个小节简要说明shell的一些重要功能。

## 6.2 bash shell概述

除了运行命令之外，shell还能够做很多的事。它提供许多有助于你轻松完成

---

注6： 精确来说，应该是共有多少个shell被启动。如果有人同时启动了两个shell，who所汇报的行数就不等于实际使用系统的人数。

工作的功能。比如说，“通配符”（wildcard）可让你轻易地描述一组有相似特征的文件；“I/O 重定向”（I/O redirection）可让命令改以特定文件为数据来源或输出目的地；“管道”（pipe）可将一个程序的输出作为另一个程序的输入，使两者可接力完成工作；“别名”（alias）可让你用更简短的字符串来表示常用命令；“变量”（variable）可作为程序之间的数据传递管道，也可用来影响 shell 的行为。然而，这里也只能粗略介绍这些功能的皮毛而已，如需完整的说明，请参阅 `info bash`。

## 通配符

当你要描述一组具有相似名称的文件时，通配符（wildcard）可让你很方便地完成。举例来说，`a*` 代表文件名第一个字符为小写字母 `a` 的所有文件。当 shell 在命令行遇到“`a*`”时，会试着找出所有符合条件的文件，然后替换成实际的文件名（这过程称为“展开”）。所以，假设你输入的命令是：

```
$ ls a*
```

而当前工作目录下只有 `apple` 和 `admin.txt` 两个文件符合条件，则上述命令会被改写成：

```
ls apple admin.txt
```

`ls` 程序本身并不知道你使用了带有通配符的文件名，它只见到 shell 展开后的结果。

通配字符	意义
<code>*</code>	除了首位“ <code>.</code> ”之外的任何字符
<code>?</code>	任何单个字符
<code>[set]</code>	出现于 <code>set</code> 中的任何字符。例如， <code>[aeiouAEIOU]</code> 代表所有元音字符， <code>[A-Z]</code> 代表所有大写字符
<code>[^set]</code> 或 <code>[!set]</code>	不出现于 <code>set</code> 中的任何字符

使用 `[set]` 语法时，如果要描述的字符序列中含有特殊字符，可通过改变它们出现于字符序列中的位置来避免 shell 误解。例如，若要描述 `-` 字符，可将该字符放在第一个或最后一个，就可以避免 shell 将它当成“范畴”来解释。如果要描述 `]` 字符，那就将它放在首位；若要描述 `^` 或 `!` 字符，只要避开首位即可。

## 花括号

类似于通配符，含有花括号的表达式也可以展开成多个参数。以逗号分隔的花括号表达式：

```
{a,b,cc,ddd}
```

会被 shell 展开成下列参数：

```
a b cc ddd
```

花括号可包括任何字符串，不像通配符仅限于文件名。举例来说，`sand{X,Y,ZZZ}wich`可展开成：

```
$ echo sand{X,Y,ZZZ}wich
sandXwich sandYwich sandZZZwich
```

不管当前工作目录下有没有那些文件。

## 波浪符号

当 shell 遇到单独的 `~` 符号或是在前缀遇到它时，会将它展开为当前用户的个人目录或是指定用户的个人目录：

```
~          你的个人目录
~smith     smith 的个人目录
```

## 环境变量

使用类似赋值的语法，就可以定义环境变量的值：

```
$ MYVAR=3
```

要取用变量值时，只要在变量名称之前冠一个 `$` 符号即可：

```
$ echo $MYVAR
3
```

Linux 预先定义了一组标准变量，当你登录你的 shell 时，它们会自动生效。

变量名称	意义
DISPLAY	X 窗口画面的名称
HOME	个人目录的路径名称



变量名称	意义
LOGNAME	登录名称
MAIL	收件邮箱的路径
OLDPWD	前次工作目录
PATH	搜索路径（目录之间以:符号分隔）
PWD	当前的工作目录
SHELL	shell 程序所在的路径（例如 <i>/bin/bash</i> ）
TERM	终端机的类型（例如 <i>xterm</i> 或 <i>vt100</i> ）
USER	当前身份的名称（通常与 LOGNAME 一样，除非以 <i>su</i> 命令变换身份）

显示环境变量的命令是 `printenv`：

```
$ printenv
```

变量的默认有效范围仅及于定义该变量的 shell 本身，若要让范围扩及 shell 所启动的其他程序（即 subshell），则必须以 `export` 命令来定义变量：

```
$ export MYVAR
```

或者，“定义”与“设值”两种动作一气呵成：

```
$ export MYVAR=3
```

严格来说，唯有被 `export` 定义的变量才有资格被称为“环境变量”（environment variable），效力仅限于个别 shell 的变量，称为 shell 变量。

有时候，我们需要定义环境变量，但是希望其时效性仅限于某特定命令，当该命令结束后，变量立刻回到原值。对于这种情况，只要将 `variable=value` 放在命令行开头处即可，例如：

```
$ echo $HOME
/home/smith
$ HOME=/home/sally echo "My home is $HOME"
My home is /home/sally
$ echo $HOME
/home/smith
```

原本的值不受影响

## 搜索路径

在诸多环境变量中，最重要的莫过于PATH。此变量的值是一系列以冒号(:)分隔的目录名称，当shell需要查找外部程序时，就是根据这些目录进行查找，所以PATH被称为“搜索路径”(search path)。

举例来说，PATH的典型值如下：

```
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:
/home/smith/bin
```

上述搜索路径的意义是：当shell要寻找某外部命令时，会优先到*/usr/local/bin*目录下去查找，如果没找到，则依次寻找*/bin*、*/usr/bin*等目录。所以，若who程序的位置是在*/usr/bin*目录下，而你在*/usr/local/bin*目录下存放一个同名程序，则shell会优先运行*/usr/local/bin*目录下的版本。

我们怎样确定shell找到的是哪个目录下的程序呢？将要运行的程序名称告诉which，它就会告诉你答案：

```
$ which who
/usr/bin/who
```

上述命令告诉你，当shell执行who命令时，它会运行*/usr/bin*目录下的版本。

若要临时让某目录成为搜索路径之一，只要直接修改PATH变量即可。比如说，若想将*/usr/sbin*加入搜索路径：

```
$ PATH $PATH:/usr/sbin
$ echo $PATH
/usr/local/bin: bin:/usr/bin:/usr/X11R6/bin:
/home/smith/bin:/usr/sbin
```

要让这项改变永久有效，请修改你的*~/.bash\_profile*配置文件，让PATH变量含有你要的目录，然后追溯(source)此配置文件：

```
$ source ~/.bash_profile
```

从此以后，每当你登录时，新的设置会自动生效。相关细节，请参阅第35页的“6.6 Bash shell的配置文件”。

## 命令别名

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

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

上述命令定义了一个新命令，`ll`，其效果相当于 `ls -l`。所以：

```
$ 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
...
```

照例，在命令行定义的别名在 `shell` 结束后就消失了。要让你定义的别名延续到以后每次登录系统时，你必须将别名定义写在 `~/.bashrc` 配置文件（参阅第 35 页“6.6 Bash shell 的配置文件”）。

直接输入 `alias` 而不给任何参数，它会列出所有已定义的别名。在 Fedora Linux 系统中，或许你会惊讶地发现，前一小节介绍的 `which` 命令，它本身其实也是一个别名。如果你觉得别名似乎不是很有用的功能，那是因为你还没见识过别名于 `shell script`（参阅 185 页“40. shell script 程序设计”）中的应用。

## I/O 重定向

之前说过，每个程序都有三个标准流：`stdin`、`stdout`、`stderr`，它们默认指向控制台（键盘与屏幕），但是你可以要求 `shell` 将它们指向文件。也就是说，任何能够读取 `stdin` 的程序都可改以文件为输入数据来源；而任何会将信息输出到 `stdout` 与 `stderr` 的程序，你都可以将其输出信息导入你所指定的文件里。改变 `stdin` 输入来源的重定向符号是 `<`：

```
$ myCommand < infile
```

用于改变 `stdout` 输出目的地的重定向符号是 `>` 与 `>>`：

```
$ myCommand > outfile      创建或覆盖 outfile
$ myCommand >> outfile      附加到 outfile 原有内容的末端
```

改变 stderr 输出目的地的重定向符号是 2> 与 2>> (译注 2):

```
$ mycommand 2> errorfile
```

如果要将 stdout 和 stderr 两者都导入文件, 方法如下:

```
$ mycommand > outfile 2> errorfile    分别导入不同文件
$ mycommand > outfile 2> &1           导入同一个文件
```

## 管道

除了改变 I/O 来源与目的地之外, shell 还提供管道 (pipe, |) 来让你将一个程序的输出连接到另一个程序的输入。举例来说,

```
$ who | sort
```

上述命令将 who 的输出接到 sort 的输入, 这样, 我们可得到一份有序的已登录用户列表。

## 组合命令

在同一命令行可依次启动多个命令:

```
$ cmd1 ; cmd2 ; cmd3
```

使用 ; 符号来分隔命令时, shell 会依次启动它们。注意, 即使 cmd1 运行失败, 或是 cmd1 还没运行结束, cmd2 都会被启动。如果命令之间有关联性, 那么任一命令失败就必须立刻停止运行。对于这种情况请将 ; 改成 &&:

```
$ cmd1 && cmd2 && cmd3
```

如果命令之间有顺序性, 也就是下一个命令必须等待前一个命令运行完毕之后方可启动, 请改用 || 分隔符:

```
$ cmd1 || cmd2 || cmd3
```

## 界定

正常情况下, shell 将空格符视为单词分隔符, 但如果想要描述任何含有空格

---

译注 2: 2 是 stderr 的文件句柄 (file handle)。如你所料, stdin 和 stdout 的代码分别是 0 和 1。

的参数时（比如说，含有空格的文件名），则必须以一对单引号或双引号来界定字符串范围，这样 shell 才知道要将该字符串视为一个单位。

以单引号界定的字符串，shell 会以字面意义来看待它们，而不加任何特殊解释；如果是双引号界定的字符串，则 shell 会解读其中的特殊字符，并展开成相对应的模式。举例来说：

```
$ 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
```

如你所见，在单引号内，\$HOME 被视为字符串解释，但是在双引号内，\$HOME 会被展开成实际的个人目录路径。

偶尔，我们希望将某命令的运行结果作为另一个命令的参数，或是换个角度想，我们希望某个参数被 shell 当成命令解释，而不是字符串。在这种情况下，可用反向单引号（```）来界定那个要被作为命令解释的字符串。例如：

```
$ /usr/bin/whoami
smith
$ echo My name is ` /usr/bin/whoami `
My name is smith
```

在此例中，shell 会先运行 `/usr/bin/whoami` 命令，并将它的运行结果（smith）作为 echo 命令的参数，然后才运行 echo 命令。

## 转义

如果你需要使用 shell 的特殊字符（\*、?、\$ 等），但是又不想 shell 将它们当成特殊字符解释，这时候可以在这些特殊字符之前加上 \ 转义符（escaping operator），迫使 shell “跳脱” 原来的解释。例如：

\$ echo a*	作为通配符解释
aardvark agnostic apple	
\$ echo a\*	作为普通星号解释
a*	
\$ echo "I live in \$HOME"	展开环境变量
I live in /home/smith	
\$ echo "I live in \\$HOME"	作为普通美元符号解释
I live in \$HOME	

shell 提供了一个用来抑制其他控制符的控制符 `^V`，让你可将控制符（tab、

换行、**^D**等)作为一般字符使用。举例来说, shell 将 ASCII 字符 9 (**Tab** 或 **^I**) 视为文件名补齐控制符(参阅第 30 页“文件名补齐”)。如果你的字符串中要含有 ASCII 字符 9, 你可以先按下 **^V**, 然后接着按 **^I** (或是按住 **Ctrl** 不放, 然后依次按下 **V**、**I**):

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

如果没先按 **^V**, 则当你按下 **^I** 时, shell 会试着找出名称开始为 here 的文件, 而不是插入一个 ASCII 9 字符在字符串里。

## 命令行编辑

bash 也支持 emacs 和 vi 文本编辑器(参阅第 56 页“11. 文件的创建与编辑”)的编辑快捷键, 让惯用 emacs 或 vi 的用户可以用习惯的快捷键来编辑命令行。让 emacs 快捷键生效的方法是:

```
$ set -o emacs
```

如果你比较习惯 vi 编辑模式:

```
$ set -o vi
```

将上述命令编进你的 `~/.bash_profile`, 可使编辑模式的设定永久有效。

emacs 快捷键	vi 快捷键 (先按 <b>ESC</b> )	意义
<b>^P</b> 或 <b>↑</b>	<b>k</b>	调出命令历史记录中的前一个命令
<b>^N</b> 或 <b>↓</b>	<b>j</b>	调出命令历史记录中的下一个命令
<b>^F</b> 或 <b>→</b>	<b>l</b>	光标向右移动一个字符
<b>^B</b> 或 <b>←</b>	<b>h</b>	光标向左移动一个字符
<b>^A</b>	<b>o</b>	光标移到行首
<b>^E</b>	<b>\$</b>	光标移到行尾
<b>^D</b>	<b>x</b>	删除下一个字符
<b>^U</b>	<b>^U</b>	消掉一整行

## 命令历史记录

bash 能记录你最近输入的一系列命令，这些已成“历史”（history）的命令，称为命令历史记录（command history）。你可利用 `[↑]` 和 `[↓]` 快捷键来调出某个历史命令，并稍加修改使其变成新命令。除了快捷键之外，bash 还提供了 history 与 ! 来操作命令历史记录。

命令	意义
history	列出完整的命令历史记录
history N	列出最近 N 次命令
history -c	清空命令历史记录
!!	直接运行前次命令
!N	直接运行历史记录中编号为 N 的命令
!-N	直接运行倒数第 N 个命令
!*	代表前次命令的所有参数
!\$	代表前次命令的最后一个参数。例如：

```
$ ls a*
$ rm !$
```

这可让你在删除文件之前，先确定有哪些文件会被删除

---

## 文件名补齐

有些文件名又臭又长，要在命令行一字不漏地输入正确文件名，确实苦不堪言。幸好，bash 可以帮上忙，你只要输入文件的前几个字，然后按下 `[Tab]` 键，bash 就会帮你补齐不足的部分。

如果你给的提示不够多，无法让 bash 找出唯一的结果，这时候连按两次 `[Tab]` 键，bash 就会列出所有可能的文件名，然后重现未完成的命令，等待你继续编辑命令行。请试着练习以下操作：

```
$ cd /usr/bin
$ ls un<TAB><TAB>
uname          unexpand      unicode_stop  unix-lpr.sh   unzip
uncompress     unicode_start uniq           unlink        unzipsfx
```

```
$ ls un [?]TAB
$ ls unzip [TAB] [TAB]
unzip      unzipsfx
$ ls unzip [s] [TAB]
$ ls unzipsfx
```

小心[TAB]会让你上瘾。

## 6.3 任务控制

jobs	列出所有后台任务。
&	将任务放到后台执行。
[^Z]	暂停当前（前台）任务。
suspend	暂停 shell。
fg	恢复任务（将任务带到前台）。
bg	将暂停的任务放到后台继续执行。

任务控制 (job control) 是任何 Linux shell 都必备的能力, 也就是在后台 (多任务) 与前台 (实时现场) 运行程序的能力。任务 (job) 是 shell 的工作单位, 当你以交互方式下达一个命令, shell 就将该命令视为一项任务。当命令结束时, 相关的任务也就跟着完成消失了。别将 Linux 内核的工作单位——进程 (process), 与 shell 的任务 (job) 混为一谈, “任务” 是比 “进程” 更高级的概念: 一个任务可能涉及好几个进程, 但是一个进程几乎不可能完成一个任务。事实上, Linux 内核对 “任务” 的概念一无所知, 真正控制任务的是 shell。关于任务控制的几个重要术语, 分述如下:

### 前台任务 (foreground job)

直接由 shell 在现场执行的任务, 在任务完毕之前, 提示符不会出现。一个 shell 每次只能有一个前台任务。

### 后台任务 (background job)

由 shell 于暗地执行的任务, shell 不等待后台任务结束, 便会立刻再输出提示符, 让用户可以继续使用同一个 shell。

### 挂起 (suspend)

暂时停止执行前台任务。



恢复 (resume)

让挂起的任务继续执行。

---

## jobs

shell 的内置命令之一，用于列出当前 shell 所有未完成的任务。

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

左侧方括号内的数字是任务编号，方括号右侧的 + 符号代表该任务是 fg (foreground, 前台) 与 bg (background, 后台) 命令默认会影响的任务。

---

## &

在命令行末端加注一个 & 符号，表示要求 shell 将该命令放到后台去运行。

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

shell 会输出一个任务编号（此例中为 2）以及该命令的进程编号 (PID, 此例中为 28090)，然后就立刻输出提示符，不管后台任务是否执行完毕。

---

## ^Z

当前台任务仍在执行中，提示符还没输出之前，在 shell 中按下 **^Z**，可立即挂起任务。被挂起的任务会暂停运作，其状态会被记录下来，但是不会结束。

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

对于已挂起的任务，你可以用 bg 将它放到后台继续执行或是以 fg 使其恢复前台执行。

---

## **suspend**

shell 的内置命令之一，其作用是挂起当前的 shell 本身。只有 subshell 才可以被挂起，你不能挂起 login shell。

举例来说，当你用 su 暂时切换到 root 身份时（或另一位用户的身份），这时就是进入了一个 subshell，你可以用 suspend 暂时回到原本的 shell：

```
$ whoami
smith
$ su -l
Password: *****
# whoami
root          进入 subshell
# suspend
[1]+ Stopped   su
$ whoami      回到 login shell
smith
```

---

## **bg [% jobnumber]**

shell 的内置命令之一，其作用是将挂起的任务放到后台继续执行。没有指定参数时，bg 作用在最近一次挂起的任务上若要影响前几次所挂起的特定任务，可以先用 jobs 命令查出该任务的编号，然后以下列语法来将该任务放到后台：

```
$ bg %2
```

某些类型的交互任务不能留在后台。例如，正在等待你输入数据的任务就不适合放在后台，如果你尝试这样做，shell 将挂起该任务，然后告诉你：

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

这时候可用 fg 将该任务拉到前台，使其继续运行。

---

## **fg [% jobnumber]**

shell 的内置命令之一，其作用是将挂起的任务或后台任务拉到前台，使其继续执行。没有指定参数时，fg 会选择最近被挂起或放到后台的任务；若要

指定特定任务，可以先用 `jobs` 命令查出该任务的编号，然后以下列语法来将该任务拉到前台：

```
$ fg %2
```

---

## 6.4 结束执行中的命令

当你在 shell 前台触动了一个命令，但是随后立刻后悔了，这时候可直接按下 `^C` 来终止该命令。shell 将 `^C` 解读为“立刻终止当前的前台命令”。所以，如果你用 `cat` 观看一个非常长的文件，但是忘了使用 `more` 或 `less` 之类的分页工具来辅助浏览，这时候就可用 `^C` 来终止噩梦：

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

若要终止后台程序，你可以先用 `fg` 命令将它拉到前台，然后用 `^C` 停止；或者，你也可以用 `kill` 命令（参阅 121 页“22. 进程控制”）来直接终止该程序。

一般而言，使用 `^C` 是相当粗鲁的手段，如果程序本身有自己的结束方式，那就应该尽可能使用。因为使用 `^C` 的效果是立即终止程序，程序没有任何清理自己的机会，特别是当你终止前台程序时，有可能使你的 shell 处于一种奇怪的状态。比如说，屏幕上可能出现一堆乱码，你打的字不是没显示，就是输出怪字等。

发生终端机脱序现象时，可试着使用下列步骤来重设（reset）终端机：

1. 按下 `^J` 来夺回 shell 提示符。`^J` 的作用其实与 `Enter` 一样，但是可以在 `Enter` 失效时使用。
2. 径直输入 `reset` 这五个字母，然后再按一次 `^J` 来运行该命令。输入的时候，别管屏幕上会出现什么（或不会出现什么），照输就对了。如果顺利，这应该能救回你的 shell。

请注意，`^C` 必须被输进 shell 才有效，如果要被终止的前台程序会拦截按键（例如 `emacs`），或是 shell 本身在终端机窗口里运行，`^C` 不仅可能没有作用，甚至可能造成前台程序误解。

## 6.5 结束 shell

离开 shell 的命令是 `exit` 或 `logout`，这两个命令的作用是一样的：

```
$ exit          或 logout 也可以
```

此外，使用 `^D`（注 7）也有结束 shell 的效果。

## 6.6 Bash shell 的配置文件

几乎每位 shell 用户或多或少都会改变 shell 的默认行为，使 shell 的表现符合个人的使用习惯。比如说：修改提示符、设置某些环境变量、定义新命令别名、增添新的搜索路径等。不过，若是直接在 shell 环境进行手动调整，一切改变在注销 shell 之后就消失了。

很显然，没人愿意每次开始使用 shell 时，都要重新设置一大堆东西，事实上也没有必要。你可将所有设置语句都写在个人目录下的 `.bash_profile` 或 `.bashrc` 文件中。这两个文件都是 *shell script*（脚本程序），当你登录 shell 时（login shell），`~/.bash_profile` 会自动运行一次；当 shell 被开启时（subshell），`~/.bashrc` 会自动运行一次。

除了个人的 `~/.bash_profile` 和 `~/.bashrc` 之外，在 `/etc` 目录下还有一个系统性的 `/etc/profile`。此文件的效力及于整个系统，因此只包含全体用户都适用的环境设置语句，也正因为会影响其他用户，所以只有管理员才能修改 `/etc/profile` 文件。

由于 bash 的配置文件都是以脚本程序（shell script）的形式存在，所以，这些文件内容的语法与命令行一样，你在命令行是怎么做的，在 `~/.bash_profile` 或 `~/.bashrc` 里就怎么写。关于 shell script 的细节，请参阅 185 页“40. shell script 程序设计”。

## 7. 安装软件

系统用久了，难免需要安装额外的新软件或是升级旧版软件。在 Linux 系统

---

注 7：许多会读取 `stdin` 的程序，将 `^D` 解读为“文件结尾”（End of File），在这种情况下，结束的可能是 shell 所运行的程序，而非 shell 本身。

中，软件是以封装包（package）的形式发布的，但是包本身的封装格式与管理软件随 Linux 发行版本而异。各种包系统分述如下：

### Red Hat Package Manager (RPM)

这是 RedHat 公司所开发的包系统，它是目前最广为使用的同时也是 Fedora 所采用的包系统。RPM 包名称的扩展名为 *.rpm*，这类文件含有已经编译好的软件以及该软件所需的相关文件。用于管理 *.rpm* 文件的工具是 *rpm*（手动）、*up2date*（自动）和 *yum*（半自动）（译注 3）。另一种常见的 RPM 包文件是 *.src.rpm*，它们封装未被编译的源代码。你可使用 *rpmbuild* 来将 *.src.rpm* 编译成适合自己平台的 *.rpm* 包文件。

### source tarball（简称 tarball）

这类包通常被封装成 *.tar.gz*、*.tar.Z* 或 *.tar.bz2* 之类的压缩格式，它们含有未编译的源代码。你可用 *tar* 搭配 *gunzip*、*uncompress*、*bunzip2* 等工具来解开这类文件，然后使用 GNU 开发工具来设定编译选项，并进行编译与安装等步骤。严格来说，source tarball 不应该算是包，因为它们通常来自软件的原创作者，不依循特定文件布局，也没有统一的包管理系统。有些软件相当容易就被编译成功，有些软件则需要费一番工夫。尽管如此，tarball 是许多 Linux 高手的最爱，因为最新版的软件通常是以这种形式来发布的。

除了 RPM 与 tarball 之外，还有许多其他包系统存在。例如，Debian 的 Advanced Package Tools (APT) 和 Gentoo Linux 的 Portage，这两种包系统也都各自有很多的拥护者。不过，本手册只介绍 Fedora Linux 的 RPM 系统。

使用 RPM 安装新软件时，由于需要写入文件到系统目录下，所以必须事先使用 *su*（或其他等效命令）获得 root 权限：

```
$ su -l
Password: *****
# rpm -ivh mypackage.rpm
...
```

---

译注 3：yum 是 Fedora Core 2 以后新增的工具，它移植自 Yellow Dog Linux (RedHat Linux 在 PowerPC/Mac 计算机上的版本)。yum 不像 up2date 那么方便，但是它有自动解决软件依靠性的能力，这又使得它比 rpm 方便许多，所以将它归类为“半自动”。本手册没提到 yum，有兴趣的读者请参考 <http://linux.duke.edu/projects/yum/>。

RPM 包文件的来源很多，除了你的 Linux 光盘之外，还有下列网站：

```
http://freshmeat.net/  
http://freshrpms.net/  
http://rpmfind.net/  
http://sourceforge.net/
```

## **up2date [options] [packages]**

**up2date**

*/usr/bin*

**stdin stdout -file --opt --help --version**

up2date 是让你的 Fedora 系统保持在最新状态 (up to date) 的最容易的方法。在 GUI 环境下，以 root 身份运行下列命令：

```
# up2date
```

然后依照画面指示操作。或者在命令行环境下，以下列命令找出所有适用于你的系统的 RPM 更新包：

```
# up2date -l
```

然后下载想要更新的包：

```
# up2date -d packages
```

安装 up2date -d 所下载的特定 RPM 包：

```
# up2date -i packages
```

由于 up2date 是通过 Internet 从 Red Hat 或 Fedora 的相关服务器下载 RPM 包，所以，第一次使用 up2date 时，可能要先向 Red Hat 注册你的系统（译注 4）。

有兴趣学习 yum 或 apt 包系统的读者，可分别在 <http://linux.duke.edu/projects/yum/> 和 <http://ayo.freshrpms.net/> 中找到相关信息。

---

译注 4：Red Hat 公司在发表 FC2 之前就宣布停止 up2date 的注册政策。目前只有 Red Hat Enterprise Linux 的用户才需要向 Red Hat 公司注册他们的系统。

---

爱好自己动手安装 RPM 包的人，可以使用 rpm 来满足 DIY 的欲望。rpm 其实是隐藏在 up2date 背后的灵魂，它不仅会安装软件，还会检查你的系统是否符合先决条件。举例来说，假设 *superstuff* 包需要搭配 *otherstuff* 包，在你的系统没安装 *otherstuff* 包之前，rpm 会拒绝帮你安装 *superstuff* 包。唯有在你的系统成功通过条件检验后，rpm 才会安装你所指定的软件。

RPM 包文件的名称有一定的命名法则：

*pkgname-version.architecture.rpm*

例如，*emacs-20.7-17.i386.rpm* 文件所封装的软件是适用于 i386 机器（Intel 80386 以及更高级的处理器）的 Emacs 20.7-17 版。要留意的是，rpm 的参数有时候需要的是包文件的“文件名”（例如 *emacs-20.7-17.i386.rpm*），但有时候则是需要“包名称”（例如 *emacs*）。大原则是：涉及已安装包的操作以包名称表示；涉及包文件或未安装包的操作以文件名表示。

用于操作 RPM 包的常用命令如下：

```
rpm -q package_name
```

找出 *package\_name* 是否安装于系统中以及所安装的版本，例如：  
`rpm -q textutils`。如果你不知道包名称，可先列出所有已安装包的名称，并以 `grep` 找出可能的名称：

```
$ rpm -qa | grep i likely_name
```

```
rpm -ql package_name
```

列出特定（已安装）包所含的文件。范例：

```
$ rpm -ql emacs
```

```
rpm -qi package_name
```

显示特定（已安装）包的一般信息。

```
rpm -qlp package.rpm
```

列出 RPM 文件的内容。你需要先取得 RPM 文件（可不安装），才可进行此操作。使用 `-qip` 可显示关于 RPM 文件的一般信息。

```
rpm -qa
```

列出所有已安装的 **RPM** 包。经常搭配 `grep` 来找出特定包的名称（译注 5）：

```
$ rpm -qa | grep i emacs
```

```
rpm -qf filename
```

显示 `filename` 所属包的名称，例如：

```
$ rpm -qf /usr/bin/who  
sh-utils 2.0-11
```

上述命令表示 `/usr/bin/who` 是 `sh-utils` 包所提供的文件。

```
rpm -ivh package1.rpm package2.rpm ...
```

安装先前未曾安装过的新包。

```
rpm -Fvh package1.rpm package2.rpm ...
```

使用指定的 **RPM** 文件来更新已安装的旧包。如果指定了未安装的包，则予以忽略（不安装）。

```
rpm -e package_names
```

删除某个已安装包。请注意，此命令所需的参数是不含版本编号的包名称。举例来说，如果你安装了 `emacs-20.7-17.i386.rpm` 包，要删除此包：

```
$ rpm -e emacs           正确  
$ rpm -e emacs 20.7-17   错误  
$ rpm -e emacs-20.7-17.rpm 错误
```

---

译注 5：运行一次 `rpm -qa` 需要很长的时间（因为需要找遍整个包数据库），如果要查找多个包或是真的不确定包的可能名称（这表示你不知道要 `grep` 什么），建议你将 `rpm -qa` 的输出交给 `tee`，并存入一个临时文件，例如：

```
rpm -qa | tee /tmp/rpmlist | grep likely_pkgname
```

如果这样找不出结果，则可以用 `less` 查看临时文件，或是直接用 `grep` 搜索临时文件的内容：

```
grep likely_pkgname2 /tmp/rpmlist
```

这样就可以不必等待 `rpm -qa` 的漫长搜索时间了。



## 7.1 tarball

绝大多数“第一手”的软件几乎都是封装成所谓的“tarball”，这类包是以标准的tar将源程序与相关文件封装成一个.tar文件，然后再以gzip、compress、bzip2之类的工具压缩成适合通过网络传输的文件。所以，tarball的扩展名通常是.tar.gz、.tar.bz2或.tar.Z。

既然tarball封装的是未编译的源程序，所以你的系统必须事先安装GNU开发工具，而且安装步骤也没有一致的标准（这就是为何需要RPM包管理系统的原因）。尽管如此，安装过程大致上仍可分为下列几个步骤：

1. 从相关网站取得tarball，并存储于适当目录下（习惯是放在/usr/src目录下）。

2. 解开包：

```
$ mkdir pkgdir
$ cd pkgdir                      (确保解开后的文件不影响原有文件)
$ tar xvzf package.tar.gz        (对于gzip压缩文件)
```

或

```
$ tar xvjf package.tar.bz2      (对于bzip2压缩文件)
```

3. 在解开后的文件中找出INSTALL或README之类的说明文件，这类文件应该含有详细的安装指示。

```
$ less INSTALL
```

4. 依照指示设定编译选项，然后开始编译。大多数tarball包都会随附一个configure脚本程序，使用下列方式可查看并暂存所有编译选项的说明：

```
$ ./configure help | tee /tmp/cfgopt
```

实际需要设定的编译选项根据你要安装的软件与系统环境而定。由于编译选项通常很多，所以将其另外存储到一个临时文件中，以便日后参考。决定好编译选项之后，便可以开始进行编译：

```
$ ./configure options
$ make
```

5. 如果编译失败，你可能要回到前两个步骤，仔细阅读说明文件，看看还有什么地方需要调整；如果编译成功，就可以切换到root身份，并进行安装：

```
$ su -  
Password: *****  
# make install
```

6. 安装好软件之后,请依照包随附的说明文件来设定该软件。大多数软件会随附一些配置文件范例,而且范例里会详细注释如何设定。

## 8. 文件基本操作

ls 列出目录内容(文件)。

cp 复制文件。

mv 改变文件位置(相当于修改文件名)。

rm 删除文件。

ln 制作链接(替换文件名)。

学习Linux系统的第一课,就是学会如何操作文件:复制、更名、删除等。

### ls [options] [files]

coreutils

/bin

stdin stdout -file --opt --help --version

ls命令(发音与字母拼法一样,ell ess)可列出文件与目录的属性。不加任何参数时,它直接列出当前工作目录的内容:

```
$ ls
```

列出指定目录的内容:

```
$ ls dir1 dir2 dir3
```

或是列出个别文件:

```
$ ls file1 file2 file3
```

最重要的选项是-a与-l。一般,ls不显示隐藏文件与隐藏目录(名称第一个字符为“.”的文件或目录),而-a选项可显示所有文件。-l选项以长格式显示文件名属性明细:

```
-rw-r--r  1 smith users 149 Oct 28 2002 my.data
```

从左至右,依次是文件的访问模式(-rw-r--r--)、拥有者(smith)、所属组

(users)、大小 (149 bytes)、最后一次修改日期 (Oct 28 2002)、文件名。访问模式也就是三种用户对此文件的访问权限，详情请参阅第 19 页“5.4 文件保护”。

## 常用选项

- a 列出所有文件（包括隐藏文件）。
- l 以长格式显示文件属性。加上 -h (human-readable) 选项表示以千字节、兆字节等适合用户阅读的单位来取代字节。
- F 以象征类型的符号来修饰特定文件名。各种符号象征的类型如下：

/ 目录

\* 可执行文件

@ 符号链接 (symbolic links)

| 命名管道 (named pipes)

= 通信管道 (sockets)

上述符号附加在文件名之后，可帮助你一眼看出它们的类型。提醒你，这些符号并非文件名的一部分，而是 `ls` 在显示时自己加上去的。

- i 在文件名之前加注该文件的 inode 编号。
- s 在文件名之前加注该文件所占的块数（1 个块相当于 1024 bytes）。
- R 遇到目录时，递归列出其内容。
- d 列出目录本身，而不是其内容。

## **cp [options] files (file|dir)**

**coreutils**

/bin

stdin stdout - file --opt --help --version

`cp` 命令通常只复制一个文件：

```
$ cp file file2    将file复制到file2
```

或是将多个文件复制到某目录下：

```
$ cp file1 file2 file3 file4 dir
```

使用 `-a` 或 `-R` 选项，可以递归复制整个子目录。

## 常用选项

- p 连同原文件的访问模式、时间戳也一并复制给新文件,如果你的权限允许,新文件的拥有者与所属组也和旧文件一样。如果没刻意指定本选项,新文件的拥有者是你(因为该文件是你创建的),修改时间是当前时间,而访问模式则是原文件的访问模式与你的umask的交集。
- a 复制整个目录树,而且保留特殊文件、访问模式、符号链接与硬链接的关系。本选项相当于同时指定-R(包含特殊文件在内的递归复制)、-p(访问模式)与-d(链接)三个选项。
  - i 交互模式。在覆盖目标文件之前,先征求用户的意见。
- f 强行复制。即使目标文件已存在,也一律无条件改写。

**mv [options] source target**

**coreutils**

/bin

stdin stdout -file --opt --help --version

仔细想想,你会发现“移动”和“改名”其实是同一回事。这就是为何mv(move)命令可改变文件名:

```
$ mv file1 file2
```

或是将文件与目录移动到另一个目录下:

```
$ mv file1 file2 dir3 dir4 destination_directory
```

## 常用选项

- i 交互模式。覆盖目标文件之前,先征求用户的意见。
- f 强制移动。即使目标文件已存在,也一律无条件覆盖。

**rm [options] files | directories**

**coreutils**

/bin

stdin stdout -file --opt --help --version

rm(remove)命令可删除文件:

```
$ rm file1 file2 file3
```

也能删除整个子目录:

```
$ rm -r dir1 dir2
```

## 常用选项

- i 交互模式。删除每个文件之前，都要先征求用户的意见。
- f 强制删除。忽略任何错误或警告。
- r 递归删除目录与其内容。谨慎使用，特别是搭配 -f 选项时。

**ln [options] source target**

**coreutils**

/bin

stdin stdout -file --opt --help --version

链接 (link) 是对另一个文件的“引用” (reference)，由 ln 命令所建立。链接分成两种。符号链接 (symbolic link) 以“路径”来表示另一个文件，就像 Windows 的“快捷方式” (shortcut) 或 Macintosh 的“别名” (alias)。

```
$ ln -s myfile softlink
```

若删除原文件，符号链接将因为指向不存在路径而变得无效。另一方面，硬链接 (hard link) 是磁盘上某物理文件的第二个名称 (以术语来说，硬链接指向同一个 inode)。删除原文件并不会造成链接失败，你仍然可通过硬链接来访问原文件的内容。

```
$ ln myfile hardlink
```

符号链接可跨越磁盘分区，因为它们只是对原文件路径的引用而已。但是硬链接则不可以，因为一个分区上的 inode 对其他分区是没有意义的。符号链接的对象可以是目录，但是硬链接不可以，除非你是 superuser 而且使用 -d 选项。

## 常用选项

- s 制作符号链接 (默认行为是制作硬链接)。
- i 交互模式。覆盖目标文件前先征求用户的意见。
- f 强制模式，无条件予以覆盖。
- d 容许 superuser 制作目录的硬链接。

使用下列命令之一，可轻易找出符号链接的对象为何：

```
$ readlink linkname
$ ls -l linkname
```

---

## 9. 目录操作

**cd**            改变工作目录。

**pwd**          显示当前工作目录的名称。

**basename**    显示文件路径中的末段部分。

**dirname**     显示文件路径中的末段之外的部分。

**mkdir**       创建目录。

**rmdir**       删除空目录。

**rm -r**       删除非空目录及其内容。

我们曾在第 12 页“5. 文件系统”探讨过 Linux 的目录树结构，现在我们要说明用来操作目录树结构的命令，也就是如何创建、修改、删除目录。

---

<b>cd [directory]</b>	<b>bash</b>
<i>shell 内置</i>	<b>stdin stdout -file --opt --help --version</b>

---

**cd** (change directory) 命令可设定当前的新工作目录。没指定任何目录时，**cd** 默认切换到你的个人目录。

---

<b>pwd</b>	<b>bash</b>
<i>shell 内置</i>	<b>stdin stdout -file --opt --help --version</b>

---

**pwd** 命令显示当前工作目录的绝对路径：

```
$ pwd
/users/smith/mydir
```

---

<b>basename path</b>	<b>coreutils</b>
<i>/bin</i>	<b>stdin stdout -file --opt --help --version</b>

---

**basename** 命令显示文件路径中的末段部分，也就是真正的文件名称或目录名称。例如：

```
$ basename /users.smith/mydir
mydir
```

---

**dirname path****coreutils****/usr/bin****stdin stdout -file --opt --help --version**

---

dirname 命令显示末段之外的文件路径，也就是所属目录的路径名称：

```
$ dirname /users/smith/mydir
/users/smith
```

dirname 的作用纯粹是分析目录名称字符串，它不影响工作目录。

---

**mkdir [options] directories****coreutils****/bin****stdin stdout -file --opt --help --version**

---

mkdir 可创建一个或多个目录：

```
$ mkdir d1 d2 d3
```

## 常用选项

**-p** 自动创建上级目录。使用此选项时，必须提供完整的目录路径（而不是只有目录名称），mkdir 会自动建立不存在的目录结构。例如：

```
$ mkdir -p /one/two/three
```

假设 */one* 存在，但是 */one/two* 不存在，则上述命令会自动创建 */one/two* 目录，然后才创建 */one/two/three*；如果 */one* 也不存在，则会先创建 */one*，然后才依序创建其下的子目录树。

**-m mode** 建好新目录后，将该目录的访问模式设定为 *mode*。例如：

```
$ mkdir -m 0755 mydir
```

没有刻意指定访问模式时，新目录的访问模式由 shell 的 *umask* 来控制。参阅第 19 页“5.4 文件保护”与第 62 页“12. 文件属性”中的 *chmod* 命令。

**rmdir** (remove directory, 删除目录) 可删除一个或多个空目录。若要删除非空目录与其内容, 请改用 **rm -r directory** (谨慎使用)。使用 **rm -ri directory** 可在删除目录之前获得确认机会, **rm -rf directory** 可毫不犹豫地删除子目录树。

### 常用选项

**-p** 如果提供一个目录路径 (而非只有目录名称), 则连同路径上的目录也一并自动删除, 不过, 前提是每个目录都必须是空目录。所以, **rmdir -p /one/two/three** 不仅会删除 */one/two/three*, 连同 */one/two* 和 */one* 也会跟着消失 (如果 */one* 和 */one/two* 都是空目录的话)。

## 10. 显示文件内容

<b>cat</b>	原样显示文件内容。
<b>less</b>	分页显示。
<b>head</b>	只显示文件内容的开始处。
<b>tail</b>	只显示文件内容的结尾处。
<b>nl</b>	标示行号。
<b>od</b>	以八进制 (或其他格式) 来显示数据。
<b>xxd</b>	以十六进制来显示数据。
<b>gv</b>	显示 Postscript 或 PDF 文件。
<b>xdvi</b>	显示 TeX DVI 文件。

在 Linux 中, 你会遇到各种各样的数据文件, 包括普通文本、Postscript、二进制数据、各种图像文件等。这里介绍如何显示各种常见类型的文件。关于显示图像文件的命令, 请参阅 179 页 “38. 图像、屏幕保护程序”, 关于播放视频文件的工具, 请参阅 182 页 “39. 音频、视频”。



最简单的显示工具是 `cat`，它能将文件内容“转接”（concatenating）到 `stdout`，因此而得名。使用 `cat` 显示大文件时，文件内容会很快闪过画面，让你来不及详看。所以，通常 `cat` 会搭配 `less` 或 `more` 之类的分页程序命令来使用。然而，由于 `less` 本身就有分页显示文件的能力，所以 `cat` 已经很少被用来显示文件内容（毕竟 `less file` 比 `cat file | less` 简单），但是若要传送一组文件的内容进入 `shell` 管道，`cat` 还是最方便的。

`cat` 也能有限地影响它的输出，例如，以特殊符号代替显示不出来的字符、加注行编号（虽然 `nl` 也行，而且功能更强）、消除空格等。

### 常用选项

- T 以 “^I” 表示制表符（tab）。
- E 以 “\$” 代替换行符（newlines）。
- v 以一种适合阅读的格式来表示不可显示的字符。
- n 每行之前加注编号。
- b 非空白行之前加注编号。
- s 将连续多行空白压缩成一行空白。

`less` 可一次显示一页文本。它不仅适合用来查看长篇文本文件，也适合放在管道末端，好让我们能分页显示那些会输出长篇信息的命令。例如：

```
$ command1 | command2 | command3 | command4 | less
```

当运行 `less` 时，有几个快捷键可以用来控制 `less`。

---

注8： 技术应用中，`less` 也可以放在管道中间，或是将它的输出导入文件，但是这样做没有意义。

快捷键	意义
<code>[h][H]</code>	显示辅助说明
<code>[Spacebar][f][^V][^F]</code>	下一页
<code>[Enter]</code>	前进一行
<code>[b][^B][ESC-b]</code>	回到前一页
<code>[/]</code>	进入查找模式。接着输入一个正则表达式 (regular expression)，然后按 <code>[Enter]</code> ，less 会往下找到第一行符合条件的地方
<code>[?]</code>	同 <code>[/]</code> ，但是查找方向相反（往回找）
<code>[n]</code>	重复运行最近一次的顺向查找
<code>[N]</code>	重复运行最近一次的逆向查找
<code>[V]</code>	以默认编辑器 <sup>注</sup> 来编辑当前文件
<code>[&lt;]</code>	回到文件开始处
<code>[&gt;]</code>	直接跳到末段
<code>[:n]</code>	跳到下一个文件
<code>[:p]</code>	回到前一个文件

注： 优先考虑环境变量 VISUAL 所定义的编辑器，如果没定义，则考虑环境变量 EDITOR，再没有，就使用 vi。

less 的功能多到你眼花，我们只介绍最常用的几项。建议你仔细阅读它的在线说明文件。

## 常用选项

- c 显示下一页之前先清屏。
- m 显示较为详尽的提示信息以及已显示的文件内容百分比。
- N 加注行号。
- r 直接显示控制符。正常情况下，less 改以适合阅读的格式来显示控制符。

- s 将多行空白压缩成单行空白。
- S 截断超过画面宽度的长文本行（不换行）。

## **head [options] [files]**

coreutils

/usr/bin

stdin stdout -file --opt --help --version

head 显示文件内容的前 10 行。很适合用来预览文件内容。

```
$ head myfile
```

```
$ head * | less
```

预览当前目录下的所有文件

## 常用选项

- N 输出前  $N$  行内容。
- n  $N$
- c  $N$  输出前  $N$  个字节。
- q 安静模式：同时处理多个文件时，不在每个文件之前显示标题。正常情况下，head 每处理一个文件，会先显示文件的标题，然后才输出文件内容。

## **tail [options] [files]**

coreutils

/usr/bin

stdin stdout -file --opt --help --version

tail 命令可显示文件内容的最后 10 行，此外还有一些有用的功能。

```
$ tail myfile
```

## 常用选项

- N 显示最后  $N$  行。
- n  $N$
- + $N$  从开始的前  $N$  行之后开始显示。
- c  $N$  显示最后  $N$  个字节。
- f 保持文件开启，每当有新内容添加到文件末端时，就显示它们。此功能常被用来观察日志文件的变化。如果要观察的文件尚不存在，可加上 --retry 选项来等待文件出现。

-q            安静模式：同时处理多个文件时，不在每个文件之前显示标题。正常情况下，tail 在处理每个文件之前，会先显示文件的标题，然后才显示文件内容。

## **nl [options] [files]**

coreutils

/usr/bin

stdin stdout -file --opt --help --version

nl 将文件复制到 stdout，在每行之前加注编号。nl 比 cat 的 -n 和 -b 选项更灵活，因为它提供了各种控制编号方式的选项。nl 主要应用在两方面：处理一般的文本文件，处理含有特殊标记的文件。

### 常用选项

-b [a|t|n|pR]    加注编号到每一行 (a)，或非空白行，(t)，或不编号 (n)，或是符合 R 正则表达式的行才予以编号 (默认为 a)。

-v N              从整数 N 开始编号 (N 默认值为 1)。

-i N              设定编号间隔为 N (默认值为 1)。例如，-i2 表示只用奇数编号，而 -v2 -i2 表示偶数编号。

-n [ln|rn|rz]    编码数字左侧对齐 (ln)，右侧对齐 (rn) 或加上前导零而且右侧对齐 (rz) (默认值为 ln)。

-w N              设定编号的位数 (N 默认值为 6)。

-s S              在编号与文本行之间插入一个 S 字符串 (S 默认值为 \t)。

除了编号之外，nl 还有将文本文件编排成虚拟页面的功能，也就是以各种编号方式来标注每篇页面的页眉 (header)、正文 (body)、页脚 (footer)。不过，nl 没神通广大到能自动判断标注的位置，这部分还是需要靠人工来做，也就是你得事先将 nl 规定的分隔字符串放在文本文件里的适当位置：“\ : \ : \ :” (页眉)、“\ : \ :” (正文)、“\ :” (页脚)，这些分隔字符串本身要各占一行。然后，你可以使用额外的选项 (请自己看 manpage) 来影响标注的编码方式。

## **od [options] [files]**

coreutils

/usr/bin

stdin stdout -file --opt --help --version

当你想看二进制文件 (binary) 的内容时，应该考虑使用 od (Octal Dump)。

它能以 ASCII、八进制、十进制、十六进制、浮点数等格式来显示一个或多个文件的内容：

```
$ od -w8 /usr/bin/who
0000000 042577 043114 000401 000001
0000010 000000 000000 000000 000000
0000020 000002 000003 000001 000000
...
```

上述命令以八进制数字显示 `/usr/bin/who` 程序文件的内容，每行显示 8 个字节。最左栏是该行数据在文件中的相对位置，同样地，也是以八进制数字表示。

## 常用选项

- `-N B` 只显示每个文件的前 `B` 个字节。默认行为是显示整个文件。
- `-j B` 略过每个文件的前 `B` 个字节。默认值为 0。
- `-w [B]` 每一行只显示 `B` 个字节。不指定 `-w` 选项时，默认行为是每行显示 16 个字节；单独指定 `-w` 选项而不给 `B` 参数，其效果与 `-w32` 一样。

上述选项的 `B` 参数，可以表示成十进制或十六进制数字（前面加上 `0x` 或 `0X`），或是在数字后面加上 `b`、`k`、`m` 来分别表示 512-byte 块、千字节（KB）、兆字节（MB）。

- `-s [B]` 只显示文件中的 ASCII 字符串以及该字符串在文件里的相对位置。
- `-A (d|o|x|n)` 设定相对位置字段（最左栏）的数字格式：`d`（十进制）、`o`（八进制）、`x`（十六进制）、`n`（不显示）。默认格式是八进制。
- `-t (a|c)[z]` 以字符格式显示字节。`a` 与 `c` 指定非字母数字字符的表示法，前者代表名称表示法（例如，字节值 `0x00` 表示成 `nul`）；后者代表转义序列表示法（escape sequence，例如，字节值 `0x00` 表示成 `\0`）。

`-t (d|o|u|x) [SIZE] [z]` 以整数格式显示文件内容。`o`代表八进制, `d`代表十进制有符号数, `u`代表十进制无符号数, `x`代表十六进制 (对于二进制, 请改用 `xxd`)。 `SIZE` 表示每个整数的字节数, 它可以是任何正整数, 也可以是 `C`、`S`、`T` 或 `L` 四者的其中之一 (分别代表 `char`、`short`、`int`、`long` 四种数据类型的长度)。

`-t f [SIZE] [z]` 以浮点数显示文件内容。 `SIZE` 代表多少个字节作为一个数值解释, 它可以是任何正整数, 也可以是 `F`、`D` 或 `L` 的其中之一 (分别代表 `float`、`double` 和 `long double` 数据类型的长度)。

如果在 `-t` 选项的参数后面加上一个 `z`, 表示要在每一行的最右侧新增一栏, 以显示该行数据的可显示字符 (类似于 `xxd` 的默认输出格式)。

## **xxd [options] [files]**

**vim-common**

*/usr/bin*

**stdin stdout -file --opt --help --version**

如同 `od`, `xxd` 能以各种不同的二进制或十六进制格式来表示二进制文件的内容, 例如:

```
$ xxd /usr/bin/who
0000000: 7f45 4c4e c101 0100 0000 0000 0000 0000 .ELF.....
0000010: 0200 c30c 0100 0000 a08c 0408 3400 0000 .....4...
0000020: 6824 000c 0000 0000 3400 2000 0600 2800 h$. ....4. ...(.
0000030: 1900 180c 0600 0000 3400 0000 3480 0408 .....4...4...
...
```

上述命令以每行16个字节的十六进制格式显示 `/usr/bin/who` 文件的内容。最左栏是该行数据在文件里的相对位置, 中间的八栏是数据的十六进制格式 (每栏各两个字节), 最右栏是该行数据的可显示字符 (不可显示字符以 “.” 表示)。

`xxd` 不仅有显示二进制文件内容的能力, 还能将自己的输出信息转换成二进制文件。

## 常用选项

- `l N` 只显示前  $N$  个字节（默认行为是显示整个文件）。
- `-s N` 从文件开始处的第  $N$  个字节开始显示。
- `-s -N` 从文件末端倒数第  $N$  个字节开始显示（关于  $+N$  的用途，请参考在线说明）。
- `-c N` 每行显示  $N$  个字节（默认值为 16）。
- `-g N` 以  $N$  个字节为一组，两组之间以空格相隔。 $N$  的默认值为 2。
- `-b` 以二进制数表示数据。
- `-u` 使用大写的十六进制数字。
- `-p` 以普通的 hexdump 格式输出，也就是每行 60 个连续字节。
- `-i` 以 C 程序语言的“数据结构”语法来表示文件内容。当数据来源为文件时，会产生一个 `unsigned char` 数组（数据）以及一个 `unsigned int` 变量（代表数组长度）。如果数据来源是 `stdin`，它只会显示输入数据的十六进制数字（以逗号分隔）而已。
- `-r` 反向操作：将 `xxd` 的 hexdump 输出信息转换回原文件的格式。只接受默认的 hexdump 输出格式，或是普通的 plain hexdump 格式（需加上 `-p` 选项）。如果你不嫌无聊的话，那就试着在管道两端各放一个互为反操作的命令，例如：

```
$ xxd myfile | xxd -r
$ xxd -p myfile | xxd -r p
```

试验上述命令时，如果 `myfile` 是真的二进制文件，小心输出信息可能搞乱你的 shell。建议你用文本文件来做实验。

## gv [options] file

gv

/usr/X11R6/bin

stdin stdout **-file** --opt --help --version

GhostView 可在 X 窗口环境下显示 Adobe Postscript 或 PDF 文件。你可以用 `gv` 或 `ghostview` 来启动它。这个程序的基本操作很简单，只要单击你想查看的页码，它就会显示对应的页面。稍微摸索几分钟，相信你能很快得心应手。

GhostView 是 Linux 系统上最被广为使用的 Postscript 解释器，但这不是唯一选择，像 `acroread` (<http://www.adobe.com/>) 和 `xpdf` (<http://www.foolabs.com/xpdf/>) 也有众多拥护者。

## 常用选项

- `-page P`                    从第 *P* 页开始看（默认值为 1）。
- `-monochrome`            使用单色、灰度或彩色显示模式。
- `-grayscale`
- `-color`
- `-portrait`                指定页面方向。默认方向由 `gc` 依据文件版式而自动决定。
- `-landscape`
- `-seascape`
- `-upsidedown`
- `-scale N`                设定缩放比例。正值的 *N* 表示放大，负值表示缩小。
- `-watch`                  当 Postscript 文件内容有变时，要（不要）自动重新加载。
- `-nowatch`

## **xdvi [options] file**

**texex-xdvi**

`/usr/bin`

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

TeX 文本处理系统所输出的二进制文件格式称为 DVI，其扩展名为 *.dvi*。用于显示 DVI 文件的程序是 `xdvi`，正如其名称的暗示，此程序必须在 X 窗口环境下才能运行。

另一种显示 DVI 文件的方法是使用 `dvips` 将 DVI 文件转换成 Postscript，然后使用 GhostView (`gv`) 来显示：

```
$ dvips -o myfile.ps myfile.dvi
$ gv myfile.ps
```

在 `xdvi` 窗口画面的右侧，可看到一系列用来操控页面的按钮，可让你前后翻页（你可使用 `-expert` 选项来隐藏这些按钮）。除了使用按钮之外，`xdvi` 也支持下列按键：



按键	作用
<b>q</b>	结束
<b>n</b> <b>Spacebar</b> <b>Enter</b> <b>PageDown</b>	下一页。紧接着按一个数字，可以多跳几页
<b>p</b> <b>Backspace</b> <b>Delete</b> <b>PageUp</b>	前一页。紧接着按一个数字，可以多跳几页
<b>&lt;</b>	到第一页
<b>&gt;</b>	到最后一页
<b>^L</b>	重新显示页面
<b>R</b>	重新载入DVI文件（比如说，你修改了文件内容）
鼠标左键	放大鼠标光标下的方形区域

---

xdvi 还有一些命令行选项，可用来控制颜色、摆设、缩放比例等整体显示效果。

## 11. 文件的创建与编辑

emacs	来自 Free Software Foundation (FSF) 的文本编辑程序。
vim	Unix vi 的加强版。
umask	设定新文件与目录的默认访问模式。
soffice	可编辑 Microsoft Word、Excel 和 PowerPoint 文本的办公软件包。
abiword	可处理 Microsoft Word 文件的编辑程序。
gnnumeric	可处理 Edit Excel 电子表格的办公软件。

要能够顺利使用 Linux，你至少必须熟练一种文本编辑工具，而这个领域分成两个阵营，分别是来自 Free Software Foundation 的 emacs 和 Unix vi 的后继加强版 vim。以本指南有限的篇幅，实在不足以介绍这两套不太好学

的超强工具（译注6），幸好它们都有在线教材，所以我们只列出一些最常用的编辑指令。

要编辑一个文件，可使用下列两个命令的其中之一：

```
$ emacs myfile
$ vim myfile
```

如果 *myfile* 不存在，文本编辑工具会自动创建它。你也可以用 `touch` 命令（见第 62 页“12. 文件属性”）产生一个空文件，以便稍后编辑：

```
$ touch newfile
```

或者将某程序的输出信息导入新文件（第 26 页“1/O 重定向”）来写入初期数据：

```
$ echo anything at all > newfile
```

倘若你需要与 Microsoft Windows 系统共享文件，稍后会介绍如何在 Linux 系统中编辑 Microsoft Word、Excel 和 PowerPoint 文件。

## 11.1 默认编辑器

有不少 Linux 程序会在必要时自动启动编辑器，例如，当你使用 `less` 时按下 `[v]`，这时候会调出 `vim`；或者，当你使用 `pine` 编写新邮件时，提供写信功能的其实是 `pico`。除了少数例外，大多数程序是调用默认编辑器来提供编辑功能，也就是 `VISUAL` 或 `EDITOR` 这两个环境变量所定义的编辑器：

```
$ EDITOR emacs
$ VISUAL=emacs
$ export EDITOR VISUAL    (非必要)
```

这两个变量都是必要的，因为不同的程序可能会检查不同的变量，所以最好两个变量都设定。你可在自己的 `~/.bash_profile` 中设定 `EDITOR` 和 `VISUAL` 变量，以免每次登录系统都要重新设定一次。你不一定要选择 `emacs` 或 `vim`，任何可接受文件名参数的编辑器都可以成为默认编辑器。

不管如何设定这些变量，所有系统管理员至少都应该懂一些 `vim` 和 `emacs`

---

译注 6：初学者可从比较简单的工具开始，例如 `nano`、`joe`、`jed` 等。

的基本指令，因为有些系统工具在应付重要文件时，只会在 vim 或 emacs 之间选择。

### emacs [options] [files]

emacs

/usr/bin

stdin stdout -file --opt --help --version

emacs 是功能相当强的编辑环境，它的指令多到你无法想象，它甚至内置了一套完整的程序语言，让你可以定义自己的编辑功能。第一次使用 emacs 的新手，最好花点时间看一下它的教学范例：

```
$ emacs
```

然后按 **h** **t**。

emacs 的大多数快捷键都涉及 **Ctrl** 键（例如 **^F**）或元键（meta key，这通常是指 **ESC** 或 **ALT** 键）。在 emacs 自己的说明书中，以 M- 来表示元键（例如，M-F 代表“按住元键不放，然后按 **F** 键”），所以我们也延续此惯例。表 1 列出了 emacs 的基本按键。

### vim [options] [files]

vim-enhanced

/usr/bin

stdin stdout -file --opt --help --version

vim 是 vi（旧的 Unix 标准编辑器）的升级版。vim 有自己的教学程序：

```
$ vimtutor
```

vim 有两种操作模式：插入模式（insert）与命令模式（command）。前者让你插入文本到文本文件中，后者用于下达编辑命令，“删除一整行”、“复制/粘贴”、“搜索”等。表 1 是 vim 的基本操作。

表 1：emacs 与 vim 的基本按键操作

作用	emacs	vim
于当前窗口启动编辑器	\$ emacs -nw [file]	\$ vim [file]
于新的 X 窗口启动编辑器	\$ emacs [file]	\$ gvim [file]
输入文本	text	i text <b>ESC</b>
保存并退出	^x^s 然后 ^x^c	:wq

表 1: emacs 与 vim 的基本按键操作 (续)

作用	emacs	vim
不保存并退出	<code>^x^c</code> 然后回答 No	<code>:q!</code>
保存	<code>^x^s</code>	<code>:w</code>
另存为	<code>^x^w</code>	<code>:w filename</code>
撤消 (Undo)	<code>^_</code>	<code>u</code>
暂时挂起编辑器 (非 X 环境下)	<code>^z</code>	<code>^z</code>
切换到编辑模式	(无)	<code>[ESC]</code>
切换到命令模式	<code>M-x</code>	<code>:</code>
放弃进行中的命令	<code>^g</code>	<code>[ESC]</code>
光标前移	<code>^f</code> 或 <code>[→]</code>	<code>l</code> 或 <code>[→]</code>
光标后移	<code>^b</code> 或 <code>[←]</code>	<code>h</code> 或 <code>[←]</code>
光标上移	<code>^p</code> 或 <code>[↑]</code>	<code>k</code> 或 <code>[↑]</code>
光标下移	<code>^n</code> 或 <code>[↓]</code>	<code>j</code> 或 <code>[↓]</code>
光标移到下一个词	<code>M-f</code>	<code>w</code>
光标移到前一个词	<code>M-b</code>	<code>b</code>
光标移到行首	<code>^a</code>	<code>0</code>
光标移到行尾	<code>^e</code>	<code>\$</code>
移到下一页	<code>^v</code>	<code>^f</code>
移到前一页	<code>M-v</code>	<code>^b</code>
移到缓冲区起点	<code>M-&lt;</code>	<code>gg</code>
移到缓冲区终点	<code>M-&gt;</code>	<code>G</code>
删除下一个字符	<code>^d</code>	<code>x</code>
删除前一个字符	<code>[Backspace]</code>	<code>X</code>

表 1: emacs 与 vim 的基本按键操作 (续)

作用	emacs	vim
删除下一个词	M-d	de
删除前一个词	<b>M-Backspace</b>	db
删除当前行	^a^k^k	dd
删除当前位置到行尾	^k	d\$
定义区 (标示区开头, 然后 移动光标到区结尾处)	<b>^Space</b>	v
剪切块	^w	d
复制块	M-w	y
粘贴块	^y	p
帮助	^h	:help
显示说明手册	^h i	:help

## umask [options] [mask]

shell 内置

bash

stdin stdout -file --opt --help --version

umask 是用于显示、修改访问模式掩码 (access mode mask) 的命令。此掩码的二进制数据, 描述了用户创建新目录或文件时, 新文件或目录的默认访问模式, 也就是拥有者、同组成员、其他人员对于新文件或新目录的读、写与执行权限 (参阅第 19 页的“5.4 文件保护”)。

```
$ umask
0002
$ umask 0
u=rwx,g=rwx,o=rwx
```

新文件的默认访问模式等于 umask XOR 0666 的运算结果, 而新目录的默认访问模式等于 umask XOR 0777 的运算结果。举例来说, 若 umask 值为 0002, 则新文件的默认访问模式为 0664 (因为 0002 XOR 0666 = 0664), 而新目录的默认访问模式为 0775 (因为 0002 XOR 0777 = 0775)。

如果上述解释让你觉得像在看外星文字, 请记得以下几个要领就够了: 掩码值 0022 赋予你自己完整的权限, 同组人与其他人都只有读或执行权限:

```
$ umask 0022
$ touch newfile && mkdir newdir
$ ls -ld new{file,dir}
drwxr-xr-x  2 me users 4096 Dec 27 15:23 newdir
-rw-r--r--  1 me users   0 Dec 27 15:23 newfile
```

掩码值 0002 赋予你自己与同组人完整权限，其他人只有读或执行权限：

```
$ umask 0002
$ touch newfile && mkdir newdir
$ ls -ld new{file,dir}
drwxrwxr-x  2 me users 4096 Dec 27 15:25 newdir
-rw-rw-r--  1 me users   0 Dec 27 15:25 newfile
```

掩码值 0077 赋予你自己完整权限，同组人与其他人没有任何权限：

```
$ umask 0077
$ touch newfile && mkdir dir
$ ls -ld new*
drwx-----  2 me users 4096 Dec 27 15:28 newdir
-rw-----  1 me users   0 Dec 27 15:28 newfile
```

总之，0077 最严格，0022 次之，而 0002 最不严格。

## soffice [files]

openoffice.org

/usr/lib/openoffice/programs

stdin stdout -file --opt --help --version

OpenOffice.org（注 9）是一套综合性的集成式办公软件包，它可编辑 Microsoft Word、Excel 和 PowerPoint 文件。你只要运行：

```
$ soffice
```

然后就可以开始工作了。soffice 程序具有辨识文件格式的能力，它能自动判断文件类型，并启动相应的程序（注 10）。soffice 是个很大的程序，它需要占用较多内存与磁盘空间。

OpenOffice.org 还有绘图（sdraw）、传真（sfax）、邮件标签（slabel）等功能，详情请见 <http://www.openoffice.org/> 或 soffice 的 Help 菜单。

---

注 9：“.org”是软件包名称的一部分。

注 10：OpenOffice.org 包里，用于处理文字工作的软件是 Writer（swriter，相当于 MS Word），电子表格软件是 Calc（scalcalc，相当于 MS Excel），版面设计软件是 Impress（simpresent，相当于 MS PowerPoint）。

---

abiword是另一个可编辑Microsoft Word文件的程序，它的规模比soffice小，速度也比较快，虽然在功能上略逊一筹，但是特别适合同时编辑多个文件。如果你在命令行指定文件名参数，所指定的文件必须事先存在，因为abiword不会自动帮你创建新文件。

gnumeric是一个电子表格程序，它能够编辑Microsoft Excel的电子表格，而且速度很快，功能也足够所需。如果你已经是Excel老手，gnumeric将给你相似的感觉。如同abiword，当你在命令行指定文件名参数时，所指定的文件必须存在，因为gnumeric不会帮你自动创建新文件。

---

## 12. 文件属性

- stat        显示文件与目录的属性。
- wc        计算文本文件的字节数、单词数、行数。
- du        显示文件与目录所占磁盘空间。
- file       辨识（猜测）文件的类型。
- touch     改变文件与目录的时间戳。
- chown    改变文件与目录的拥有者。
- chgrp    改变文件与目录的从属组。
- chmod    改变文件与目录的访问模式。
- chattr   改变文件与目录的扩展属性。
- lsattr   列出文件与目录的扩展属性。

显示Linux文件的内容只是事情的一半。我们还需要能够查阅文件内容之外的信息，像拥有者、大小、访问模式等。虽然ls -l命令（见第41页“8.

文件基本操作”)能够显示文件的部分属性,但是 Linux 还提供了其他工具来显示更详尽的额外信息。

## **stat [options] files**

coreutils

/usr/bin

stdin stdout -file --opt --help --version

stat 命令能显示文件(默认行为)或文件系统(加上 -f 选项)的重要属性。文件信息的形式如下:

```
$ stat myfile
  File: `myfile'
  Size: 3280  Blocks: 8   IO Block: 4096   regular file
Device: 302h/770d Inode: 196612  Links: 1
Access: (0644/-rw-r--r--)  Uid: (500/me)   Gid: (500/me)
Access: 2004-12-27 17:48:17.755769360 +0800
Modify: 2004-12-27 17:48:17.756769208 +0800
Change: 2004-12-27 17:48:17.756769208 +0800
```

在此例中,你可看到文件的名称、字节数(3280)、所占用的块数(8)、I/O 单位(4096)、文件类型(regular file)、该文件所在的设备类型与编号(302h/770d)、Inode 编号(196612)、硬链接的数量(1)、访问模式(0644/-rw-r--r--)、拥有者的 UID 与名称(500/me)、所属组的 GID 与名称(500/me)、最后一次访问时间、内容修改时间、状态改变时间(译注 7)。

文件系统的信息类似下面这样:

```
File: "myfile"
  ID: 0      Namelen: 255      type: ext2/ext3
Blocks: Total: 1032108 Free: 514738
      Available: 462310      Size: 4096
Inodes: Total: 524288 Free: 487317
```

你可看到文件名(myfile)、文件系统的标识符(0)、该文件系统可接受的文件名长度(255)、文件系统的类型(ext2/ext3)、块的总数(1032108)、剩余数(514738)、有效剩余数(462310)与单位大小(4096)、Inodes 的总数(524288)与剩余数(487317)。

---

译注 7: 时间格式随当前的 LANG 环境变量而定,精确度随硬件时钟的精确度和文件系统的类型而定。



-t 选项可使 stat 将所有数据都显示在同一行，并且省略掉标题，如此可使得 shell scripts 或其他程序方便分析。

```
$ stat -t /etc/passwd
/etc/passwd 1955 8 81a4 0 0 303 16571 1 0 0 ... (略)
$ stat -f -t /etc/group
/etc/group 0 255 * 515884 497661 471453 4096 ... (略)
```

## 常用选项

- l 跟随符号链接，并汇报链接对象。
- f 汇报文件所在的文件系统，而非文件本身。
- t 精简模式 (terse mode)：将所有信息都显示在同一行。

## wc [options] [files]

coreutils

/usr/bin

stdin stdout -file --opt --help --version

wc (word count, 字数计算) 程序可算出文本文件所含的字节数、单词数、行数。

```
$ wc myfile
 24 62 428 myfile
```

此文件共有 24 行、62 个以空格隔开的单词以及 428 个字节。

## 常用选项

- l 只显示行数。
- w 只显示单词数。
- c 只显示字节 (字符) 数。
- L 找出每个文件中最长的行并显示该行的长度 (以 byte 为单位)。

## du [options] [files|directories]

coreutils

/usr/bin

stdin stdout -file --opt --help --version

du (disk usage) 命令可计算文件或目录所占的磁盘空间。没有指定任何选项时，它会测量当前工作目录与其所有子目录，分别显示各目录所占的块数，最后才显示工作目录所占的总块数。

```
$ du
8    ./Notes
36   ./Mail
340  ./Files/mine
40   ./Files/bob
416  ./Files
216  ./PC
2404 .
```

此外，wc 也可以测量个别文件所占的磁盘空间：

```
$ du myfile myfile2
4    ./myfile
16   ./myfile2
```

## 常用选项

-b -k -m 以字节（-b）、千字节（-k）或兆字节（-m）为计算单位。

-B N 以 N（默认值 1024）为块大小的计算单位。

-h -H 输出适合用户阅读的单位。举例来说，若有两个目录的大小分别为 1 gigabyte 与 25 kilobytes，则 du -h 会输出 1G 与 25K。  
-h 选项使用的单位基数是 1024，而 -H 选项使用的单位基数是 1000。

-c 于最后输出合计占用容量。这是测量目录时的默认行为，但是 -c 选项让你在测量文件时也有相同效果。

-L 跟随符号链接，并测量链接目标的大小。

-s 只显示工作目录所占总空间。

## file [options] files

file

/usr/bin

stdin stdout -file --opt --help --version

file 命令可依据文件内容来猜测文件的类型：

```
$ file /etc/hosts /usr/bin/who letter.doc
/etc/hosts: ASCII text
/usr/bin/who: ELF 32-bit LSB executable, Intel 80386 ...
letter.doc: Microsoft Office Document
```

不同于其他某些操作系统，Linux 不记录文件类型。虽然很多文件都有扩展

名,但即便是应用程序本身,也不见得依据扩展名来辨识文件类型。基本上,Linux 文件的扩展名只是给用户的提示,而非内容类型的决定性因素。

## 常用选项

- |                                 |   |
|---------------------------------|---|
| <code>-b</code>                 | 省略文件名(最左栏)。   |
| <code>-i</code>                 | 以 MIME 类型名称(例如“text/plain”、“audio/mpeg”)来代替正常的输出信息。               |
| <code>-f file_withfnlist</code> | 读取 <i>file_withfnlist</i> 文件,将该文件中的每一行视为一个文件名,然后汇报这些文件名所指文件的类型。   |
| <code>-L</code>                 | 跟随符号链接,汇报链接目标的文件类型,而不是告诉你那是符号链接。                                  |
| <code>-z</code>                 | 如果受测文件为压缩文件(参阅 91 页“15. 文件压缩”),则检验解压缩后的内容,然后汇报其内容类型,而不是告诉你那是压缩文件。 |

## **touch [options] files**

coreutils

/bin

stdin stdout -file --opt --help --version

`touch` 命令可改变文件的两项时间戳属性:最后一次修改时间(上次修改文件内容的时间)以及最后一次访问时间(上次是何时被读取)。

```
$ touch myfile
```

你也可以指定任意的时间值:

```
$ touch -d "November 18 1975" myfile
```

如果目标文件不存在,则 `touch` 会自动创建一个新文件。这种方法常用来创建新的空文件。

## 常用选项

- |                 |          |
|-----------------|----------|
| <code>-a</code> | 只改变访问时间。 |
| <code>-m</code> | 只改变修改时间。 |

- c                    不自动产生新文件。
- d *timestamp*      设定文件的时间戳。*touch* 接受多种 *timestamp* 格式，从“12/28/2001 3pm”到“28-May”（当年指定日期的午夜时刻）到“next tuesday 13:59”到“0”（今日午夜）。使用 *stat* 可看到 *touch* 的修改结果。完整的 *timestamp* 格式说明，请参阅 *info touch*。
- t *timestamp*      效果类似于 -d 选项，但是 *timestamp* 的格式限定为 `[[CC]YY]MMDDhhmm[.ss]`。其中的 *CC* 是两位数的世纪（译注 8），*YY* 是两位数的年份，*MM* 是两位数的月份，*DD* 是两位数的日期，*hh* 是两位数的小时，*mm* 是两位数的分钟，*ss* 是两位数的秒钟。举例来说，-t 20041207150047 代表 2004 年 12 月 7 日 15:00:47。

## **chown [options] user\_spec files**

**coreutils**

*/bin*

*stdin stdout -file --opt --help --version*

*chown* (change owner) 命令可设定文件或目录的拥有权。

```
$ chown smith myfile myfile2 mydir
```

*user\_spec* 参数有下列几种可能性：

- 一个用户名称（或 UID 数值）。只设定拥有者。
- 一个用户名称（或 UID 数值），其后跟着一个冒号（:）与一个组名称（或 GID 数值）。设定拥有者与所属组。
- 一个用户名称（或 UID 数值），其后跟着一个冒号（:）。设定拥有者，以该用户的登录组为本文件的所属组。
- 一个冒号后面接着一个组名称（或 GID 数值）。只设定所属组。
- `--reference=file`。以 *file* 的拥有者与所属组作为本文件的拥有者与所属组。

---

译注 8： 正确说法是世纪数减 1。例如，21 世纪是写成 20。

## 常用选项

- `--dereference` 跟随符号链接，并作用在链接目标。
- `-R` 递归作用在整个子目录的每一个文件与子目录上。

### **chgrp [options] group\_spec files**

coreutils

/bin                      stdin stdout -file --opt --help --version

chgrp (change group) 命令可设定文件与目录的组拥有权。

```
$ chgrp smith myfile myfile2 mydir
```

*group\_spec* 参数有下列几种形式：

- 一个组名称或 GID 数值。
- `--reference=file`，以另一个文件的所属组作为本文件的所属组。

关于组的概念，请参阅第 134 页“26. 组管理”。

## 常用选项

- `--dereference` 跟随符号链接，作用在链接对象，而非链接文件本身。
- `-R` 递归改变整个子目录的组拥有权。

### **chmod [options] permissions files**

coreutils

/bin                      stdin stdout -file --opt --help --version

chmod (change mode) 命令可设定文件与目录的访问模式。并非每个文件都可被任何人访问（你知道，这不是 Windows 95），而 chmod 就是用来确保这点的工具。访问权限分为读、写、执行三种，用户分成拥有者、同组、其他人三类。三种用户各有三种权限，构成 9-bit 访问模式。

*permissions* 参数有三种形式：

- `--reference=file`，引用另一个文件的访问模式。
- 四位数的八进制数字，以位形式指出文件的绝对权限（absolute permissions）：最左侧的数字表示特殊属性，往右的数字依次表示拥有

者、所属组、其他人的权限。图 3 以 0640 为例，分别指出各数字表示的意义。

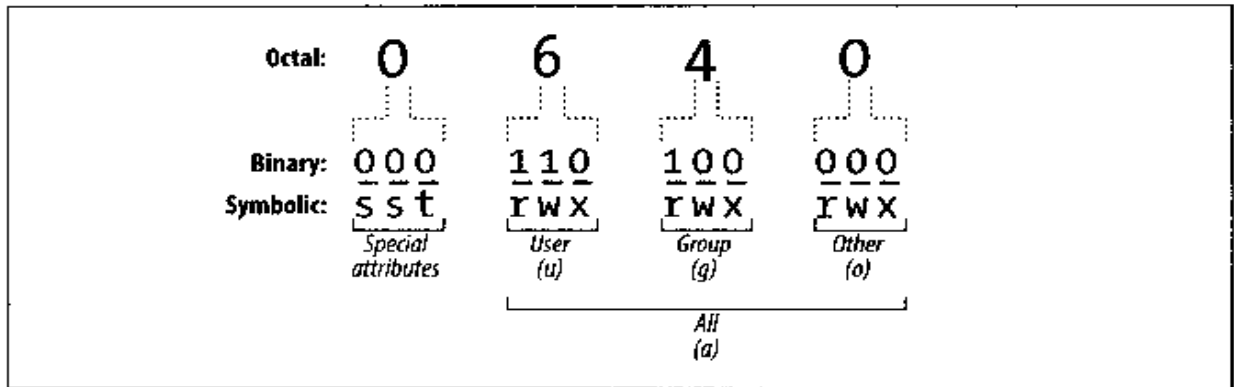


图 3：文件权限位

- 一个或多个以逗号相隔的字符串，这些字符串描述绝对权限或相对权限（相对于文件原有的权限）。

对于第三种形式，每个字符串可包含三个部分，分别是作用域（scope）、命令（command）、权限（permission），分述如下：

**作用域（可省略）**

u (user) 表示拥有者，g (group) 表示所属组，o (other) 表示非拥有者也非组成员的任何其他人，a (all) 表示全体用户（默认）。

**命令**

+ 表示增加权限，- 表示移除权限，= 表示设定绝对权限（忽略原本的权限）。

**权限**

r 表示可读（文件）或可显示内容（目录）；w 表示可修改文件或是在目录下新增文件；x 代表可执行（对于文件）或是可进入（对于目录）；X 代表有条件执行（稍后解释）；u 继承“拥有者”的权限；g 继承“所属组”的权限；o 继承“其他人”的权限；s 用于 setuid 或 setgid；t 代表粘贴位（sticky bit）。

举例来说，ug+rw 将使得拥有者与所属组成员获得可读可写的权限，而 a-x（或只有 -x）将移除所有人的可执行权限。u=r 将撤消原本的任何权限，然后使拥有者具有可读权限。你可以用逗号将权限描述字符串结合在一起，例如：ug+rw,a-x。

setuid和setgid仅适用于可执行文件（程序或脚本程序）。假设我们有一个可执行文件 *F*，其拥有者是 smith，所属组为 friends。假设 *F* 文件的 setuid (set user ID) 有效，则任何人在运行 *F* 程序的这段期间，将会暂时“成为” smith，并继承该身份的所有权限。

同理，若 *F* 文件的 setgid (set group ID) 权限位被设定，则任何能够运行 *F* 程序代码的人在该程序的运行期间，将暂时成为 *F* 所属组的成员。正如你的想象，setuid 和 setgid 对系统安全的影响甚巨，所以，除非你真的知道自己在做什么，否则不应该贸然设定 setuid 或 setgid 位。一次失误运行 chmod +s，就有可能给整个系统带来莫大的风险。

条件性执行权限 (x) 的作用类似于 x，唯一差别在于它只对可执行文件与目录有效，否则没有作用。

## 常用选项

-R 递归改变整个子目录的访问模式。

**chattr [options] [+ -=]attributes [files]**

**e2fsprogs**

/usr/bin

stdin stdout -file --opt --help --version

如果你曾经使用过其他 Unix 系统，或许会惊讶 Linux 文件竟然有访问模式之外的扩展属性。对于存放在 ext2 或 ext3 文件系统（Fedora 默认使用）中的文件，你可以用 chattr (change attribute) 来改变它们的扩展属性，用 lsattr 来显示它们。

如同 chmod，chattr 提供三种属性操作，分别是增加 (+)、撤销 (-) 以及设定 (=)。

属性	意义
----	----

a	仅供附加 (Append-only)：容许增添内容到文件末端，但是不容许修改其原有内容。仅有 root 有权限设定此属性
A	不记录访问时间。读取文件时，不自动更新其访问时间 (atime)
c	自动压缩：写入数据时自动压缩，读取数据时自动解压缩

属性	意义
d	不转储 (dump): 要求 dump 程序在制作备份时, 忽略此文件 (参阅 106 页 “18. 备份”)
i	不可变 (Immutable): 不能删除文件, 也不能改变文件内容。仅有 root 有权限设定此属性
j	日志记录数据 (Journaled data), 仅 ext3 文件系统有此属性
s	安全删除 (secure deletion): 文件被删除时, 将磁盘上的文件数据都归零
S	同步更新 (synchronous update): 发生变动时, 立刻将数据写回磁盘, 其效果就像是保存之后手动运行一次 sync (见第 101 页 “17. 磁盘与文件系统”)
u	禁删 (Undeletable): 文件不可被删除

---

## 常用选项

-R 递归处理整个子目录。

### lsattr [options] [files]

e2fsprogs

/usr/bin

stdin stdout -file --opt --help --version

使用 chattr 设定扩展属性之后, 可用 lsattr (list attributes) 来显示扩展属性的设定值。lsattr 的输出列表使用与 chattr 相同的属性字母。例如, 下列文件不可变 (immutable), 而且不容许删除 (undeletable):

```
$ lsattr myfile
-u--i--- myfile
```

## 常用选项

-R 递归列出整个子目录。

-a 列出包含隐藏文件 (文件名第一个字符为 “.”) 在内的所有文件。

-d 遇到目录时不显示其内容, 而显示目录本身的扩展属性。

没指定文件时, lsattr 列出当前工作目录下所有文件的扩展属性。



## 13. 文件位置

<b>find</b>	在目录树中找出特定条件的文件，并处理找到的文件。
<b>slocate</b>	产生文件索引表，于索引表中搜索特定字符串。
<b>which</b>	于搜索路径中找出特定的可执行文件，并显示该文件的位置（外部命令）。
<b>type</b>	bash 的内置命令，其效果类似 which。
<b>whereis</b>	找出可执行文件、文档、源程序等。

Linux 系统往往含有几千几百个文件，当你需要特定文件时，如何能迅速找出它们？第一步是组织一个有条理的目录树，将文件放在合乎逻辑的适当目录下，以便于记忆。然而，并非人人都能养成整理文件的习惯，也不是人人都是分类高手，为此，Linux 提供了一系列方便的查找文件的工具，以满足各种文件搜索需求。

**find** 可用于找出任何文件，但是它的查找方法很原始：沿着目录树逐层往下找，逐一比对每个文件是否符合搜索条件。**slocate** 的搜索方法比较快，但是在遇到最近频繁变化的文件系统时，可能会发生找不到新建文件的结果，因为 **slocate** 是查找一个预建的索引数据库（Fedora 会在每天半夜自动重建索引数据库）。

如果要搜索程序文件，**which** 和 **type** 命令可帮你检查 shell 搜索路径（PATH 环境变量）中的所有目录。**type** 是 bash shell 的内置命令（所以只能用于 bash 环境下），而 **which** 是一个外部程序（通常位于 `/usr/bin/which` 中）。**type** 的速度比较快，而且能检测命令别名（aliases）（注 12）；相对地，**whereis** 只搜索一组已知的目录，而不是搜索路径。

### **find [dir] [exp] [action]**

**findutils**

`/usr/bin`

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

**find** 从指定的一个或多个目录（**dir**）往下逐级搜索，找出符合搜索条件（**exp**）的每个文件，然后对找出的文件执行指定的动作（**action**）。这是一个

---

注 12：tcsh shell 会行使一些技巧，使得 **which** 也能检测命令别名。

非常有用的工具，它的选项超过 50 个，不过，其语法颇不寻常，你要学着调整自己的思路，才可有效运用此工具。以下是两个简单的例子：

从根目录下的整个文件系统中，找出名称为 *myfile* 的文件：

```
$ find / -type f -name myfile -print
```

显示所有目录的名称：

```
$ find / -type d -print
```

## 常用的搜索条件

<code>-name pattern</code>	找出名称 ( <code>-name</code> )、路径名称 ( <code>-path</code> )、符号链接的目标 ( <code>-lname</code> ) 符合 <code>pattern</code> 模式的文件。 <code>pattern</code> 可包含 shell 的文件名通配符 ( <code>*</code> 、 <code>?</code> 和 <code>[]</code> )，路径是相对于搜索起点。 <code>-iname</code> 、 <code>-ipath</code> 和 <code>-ilname</code> 选项的效果相当于 <code>-name</code> 、 <code>-path</code> 和 <code>-lname</code> ，但是不区分大小写 (case-insensitive)。
<code>-path pattern</code>	
<code>-lname pattern</code>	
<code>-iname pattern</code>	
<code>-ipath pattern</code>	
<code>-ilname pattern</code>	
<code>-regex regexp</code>	路径 (相对于搜索起点) 符合 <code>regexp</code> 正则表达式的文件。
<code>-type f d l b c p s</code>	只搜索普通文件 ( <code>f</code> )、目录 ( <code>d</code> )、符号链接 ( <code>l</code> )、块设备 ( <code>b</code> )、字符设备 ( <code>c</code> )、具名管道 (named pipes, <code>p</code> ) 或 socket ( <code>s</code> )。
<code>-atime N</code>	找出最近访问时间 ( <code>-atime</code> )、最近修改时间 ( <code>-mtime</code> ) 或状态改变时间 ( <code>-ctime</code> ) 刚好在 $N \times 24$ 小时之前的文件。使用 $+N$ 表示“超过 $N$ 天”， $-N$ 表示“少于 $N$ 天”。
<code>-ctime N</code>	
<code>-mtime N</code>	
<code>-amin N</code>	找出最近访问时间 ( <code>-amin</code> )、最近修改时间 ( <code>-mmin</code> ) 或状态改变时间 ( <code>-cmin</code> ) 刚好在 $N$ 分钟之前的文件。使用 $+N$ 表示“超过 $N$ 分钟”， $-N$ 表示“不到 $N$ 分钟”。
<code>-cmin N</code>	
<code>-mmin N</code>	
<code>-anewer other_file</code>	找出最近访问时间 ( <code>-anewer</code> )、最近状态改变时间 ( <code>-cnewer</code> ) 或内容修改时间 ( <code>-newer</code> ) 比另一个文件 ( <code>other_file</code> ) 更近的文件。
<code>-cnewer other_file</code>	
<code>-newer other_file</code>	

<code>-maxdepth N</code>	限制搜索深度（相对于搜索起点的子目录深度）
<code>-mindepth N</code>	至少（ <code>-mindepth</code> ）或最深（ <code>-maxdepth</code> ）为 $N$ 层。
<code>-follow</code>	跟随符号链接。
<code>-depth</code>	优先搜索深度较深者：优先（递归）查找目录的内容，然后才比对目录本身。如不指定此选项，当目录与较深层的文件都符合搜索条件时，目录会被优先处理，然后才轮到深层文件。
<code>-xdev</code>	限制搜索范围在单一文件系统，不跨越设备边界。
<code>-size N[bckw]</code>	找出单位大小为 $N$ 的文件。b、c、k、w 分别代表块、单字节字符、千字节（1024 bytes）、双字节词组。 $+N$ 表示“大于 $N$ ”， $-N$ 表示“小于 $N$ ”。
<code>-empty</code>	找出长度为 0 的普通文件或目录。
<code>-user username</code>	拥有者为 <code>username</code> 用户或 <code>gname</code> 组。
<code>-group gname</code>	
<code>-perm mode</code>	找出访问模式等于 <code>mode</code> 的文件或目录。使用
<code>-perm -mode</code>	<code>-mode</code> 表示指定位必须全部被设定，而 <code>+mode</code>
<code>-perm +mode</code>	表示任何指定位之一被设定就算符合条件。

上述搜索条件还可以进一步组合成更复杂的复合条件：

`exp1 -a exp2`

交集（AND），当 `exp1` 与 `exp2` 同时成立时，才算符合搜索条件。当两个表达式都出现在同一边时，这是默认行为，所以 `-a` 其实是可有可无的。

`exp1 -o exp2`

并集（OR），`exp1` 与 `exp2` 只要其中有一个成立，就算成立。

`! exp`

`-not exp`

当 `exp` 不成立时，条件成立。

(*exp*)

标示优先运算的部分，就像代数中的圆括号一样。不过，由于()也是shell的特殊字符，所以必须前置\符号（即\(*exp*\)）来避免shell误解。

*exp1* , *exp2*

如同C程序语言中的逗号运算，*exp1*和*exp2*会被依次运算，但只有*exp2*的运算结果会被当成比对条件。

搜索条件只是完整find命令的一半，你还得让find知道，要对符合条件的文件做什么动作。

## 动作

- print                    输出文件的相对路径（相对于搜索起点）。
- printf *fmtstr*        输出 *fmtstr* 字符串，此字符串可包含内嵌格式，就像C函数库中的printf()函数一样。详细说明请见info printf。
- print0                  效果类似 -print，但是将每行输出信息末端的\n字符换成\0 (ASCII 0) 字符。如果find找出的文件名列表中含有空格，而你又需要将文件名列表导入其他程序时，就有必要用 -print0来取代 -print。当然，接受文件名列表的程序，本身要有辨识\0字符的能力。具体来说，-print0是为了搭配xargs -0而设计的。
- exec *cmd* \;  
-ok *cmd* \;              运行指定的shell命令 (*cmd*)。你所描述的*cmd*，包括命令末端的分号在内，其中的任何shell特殊字符之前都必须加上\符号，以免shell立刻运行它们。在*cmd*里，你可以用“{ }”符号（包括双引号在内）来表示find所找出的文件。-ok和-exec的差别在于后者会直接运行*cmd*，而前者在每次运行*cmd*之前，都会先询问用户是否要运行。
- ls                      对找出的文件运行ls -dils。

find与xargs是绝佳搭档，因为前者会输出文件名列表到stdout，而后者从

stdin 读入文件名列表，然后对个别文件运行参数所指定的命令（参阅 man xargs）。举例来说：

```
$ find . -print0 | xargs -0 grep myxomatosis
```

上述命令可在当前工作目录下（连同所有子目录），找出任何含有“myxomatosis”字样的文件，并且输出含有该字样的文件内容。

另一个例子示范如何在 -exec 的 cmd 参数中描述 find 所找到的文件：

```
find . -name '*.zip' -exec unzip "{}" \;
```

上述命令找出工作目录下的所有 ZIP 压缩文件，然后使用 -exec 来运行 unzip 程序，借此解开所找到的 ZIP 压缩文件。

## **slocate [options]**

**slocate**

/usr/bin

stdin stdout -file --opt --help --version

slocate (secure locate) 是一个高效率的文件搜索工具，第一次运行它时，它会建立一个索引数据库，往后再次运行 slocate 时，它便可从索引数据库中迅速找出文件的位置。slocate 很适合用来反复搜索很少变动的目录树。不过，如果只是临时想到要查找某个文件或是需要进行一些复杂的处理，find 才是比较理想的选择。

Fedora Linux 每天会自动对整个文件系统重新建立索引数据库，不过，你也可以自己动手并指定索引数据库的位置：

```
$ slocate -u -o /tmp/myindex
```

或者，你也可以限制索引范围：

```
$ slocate -U directory -o /tmp/myindex
```

然后于该索引数据库中搜索特定字符串：

```
$ slocate -d /tmp/myindex string
```

slocate 的“secure”（安全）又是从何说起？在搜索期间，它不会显示一般用户身份无权查看的内容。所以，若 superuser 对某个保护目录建立了一个索引数据库，则 superuser 之外的其他人就不会看到保护文件的内容（但是能够搜索）。

## 索引选项

- u 从根目录开始建立整个文件系统的索引。
- U *directory* 建立 *directory* 目录树的索引。
- t (0|1) 使安全措施失效 (0) 或生效 (1)。默认值为 1。
- e *directories* 建立索引时, 排除特定目录。目录名称之间以冒号分隔。
- o *outfile* 写出索引数据到 *outfile* 文件。

## 搜索选项

- d *indexfile* 使用指定的索引文件。
- i 不区分大小写。
- r *regexp* 搜索文件名符合 *regexp* 正则表达式的文件。

## which file

which

/usr/bin

stdin stdout -file --opt --help --version

which 命令从 shell 的搜索路径 (PATH) 中找出某个可执行文件。对于能够直接以程序名称来运行的命令 (例如: who), which 可告诉你该命令所对应的程序文件在哪里:

```
$ which who
/usr/bin/who
```

你甚至可用 which 找出它自己:

```
$ which which
/usr/bin/which
```

如果搜索路径中有多个同名程序 (比如说 /usr/local/bin/who 和 /usr/bin/who), 则 which 只汇报第一个 (也就是会被 shell 运行的那一个)。这项特性使得 which 常被用来检查 shell 所运行的程序是否为我们预期的版本。

## type [options] commands

bash

shell 内置

stdin stdout -file --opt --help --version

type 命令的作用与 which 完全一样, 都是用来找出特定可执行文件在搜索路径中的位置:

```
$ type grep who
grep is /bin/grep
who is /usr/bin/who
```

然而，`type`是bash shell的内置命令，而`which`是位于磁盘上的外部程序：

```
$ type which type rm if
which is /usr/bin/which
type is a shell builtin
rm is aliased to `/bin/rm .'
if is a shell keyword
```

既然是内置命令，`type`的优点就是速度比`which`快些，而缺点是只能用于bash环境中。

## **whereis [options] files**

util-linux

/usr/bin

stdin stdout -file --opt --help --version

`whereis`命令从一组固定的目录中找出指定的文件。它能查找可执行文件、说明文件以及程序源文件。`whereis`或许有些可笑，因为你要找的文件不一定在它的搜索范围内。

### 常用选项（译注 9）

<code>-b</code>	只查找可执行文件（ <code>-b</code> ）、manpages（ <code>-m</code> ）
<code>-m</code>	或源程序（ <code>-s</code> ）。
<code>-s</code>	
<code>-B dirs... -f files...</code>	于指定的搜索范围（ <code>dirs</code> ）内找出可执行文
<code>-M dirs... -f files...</code>	件（ <code>-B</code> ）、manpages（ <code>-M</code> ）或源程序（ <code>-S</code> ）。
<code>-S dirs... -f files...</code>	你必须将 <code>-f</code> 选项放在搜索范围之后，在要
	找的目标文件之前。

---

## 14. 文件文本操作

`grep` 于文件中找出符合正则表达式条件的文本行。

---

译注 9：这几个选项正是 `whereis` 的可笑之处：如果你知道搜索范围，那又何必搜索？

cut        截取文件中的特定字段。

paste     附加字段。

tr        字符转换或压缩。

sort      调整文本行的顺序，使其符合特定准则。

uniq      找出重复的文本行。

tee       将 stdin 抄写到 stdout 的同时复制一个文件。

文本操作 (text manipulation) 是 Linux 的拿手好戏：通过一系列转换，将文本文件 (或 stdin) 转换成我们想要的形式。能够读取 stdin 并写信息到 stdout 的任何程序都不超出此范畴，不过，这里只介绍其中最常用、最有用的部分工具。

## **grep [options] pattern [files]**

**grep**

*/bin*

**stdin stdout -file --opt --help --version**

正则表达式 (regular expression) 常被简称为 regex、regexp 或模式 (pattern)。在 Linux 世界里，正则表达式常被用来描述文本模式，或表达复杂的字符串操作 (替换、删除、转换等)。有许多 Linux 工具都支持正则表达式，而 grep 是其中最常用的搜索工具，它常被用来找出文件中符合特定模式的文本行。举例来说，若某文件的内容是以下三行文字：

```
The quick brown fox jumped over the lazy dogs!
My very eager mother just served us nine pancakes.
Film at eleven.
```

而我们想知道 “pancake” 一词出现在哪一行，则可用 grep 像下面这样搜索此文件：

```
$ grep pancake myfile
My very eager mother just served us nine pancakes.
```

在此例中，“pancake” 就是一个简单的正则表达式 (或模板)；而 grep 会找出含有符合模板的字符串的文本行，然后输出它们 (译注 10)。

---

译注 10：为了方便说明，后文将“含有符合模板的文本行” (matching line) 简称为“合模文本行”。

---



grep支持两种正则表达式语法,分别是基本型(basic)与扩充型(extended)。这两种正则表达式同样都很有用,不分高下,它们纯粹只是语法不同而已,而非功能上有什么差异。你可以依照自己的经验或惯用的grep版本使用特定一种正则表达式语法。表2与表3分别列举了两种正则表达式的基本语法。

## 常用选项

- v 只输出合模文本行之外的其他文本行(也就是不符合正则表达式的文本行)。
- l 只输出含有合模文本行的文件的文件名,而非合模文本行本身。
- L 只输出完全不含合模文本行的文件的文件名。
- c 只输出合模文本行的数量。
- n 输出合模文本行之前,先输出该文本行于原文件中的行号。
- b 输出合模文本行之前,先输出该文本行距离文件首的相对位置(合模文本行第一个字符与原文件第一个字符之间相隔的字节数量)。
- i 匹配时不区分大小写。
- w 匹配完整词(必须整个词都符合模板)。
- x 只匹配完整文本行(必须整行文本都符合模板)。本选项的效力高于-w。
- A N 先输出合模文本行,然后输出原文件中紧接在合模文本行之后的N行文本。
- B N 先输出原文件中出现在合模文本行之前的N行文本,然后才输出合模文本行。
- C N 效果相当于同时使用-A N -B N,也就是输出原文件中出现在合模文本行之前与之后的N行文本。
- r 将整个子目录的所有文件都纳入搜索范围。
- E 使用扩充型正则表达式语法。效果等同于使用egrep。
- F 将给定的模板视为固定字符串,而非正则表达式。效果等同于使用fgrep。

表2是基本型与扩充型正则表达式的共同语法。

表2：正则表达式语法（基本型与扩充型的共同部分）

模板	意义
	任何单一字符
[...]	于所列字符中的任何单一字符
[^...]	除了所列字符之外的任何单一字符
(...)	分组
^	行首
\$	行尾
\<	单词开始处
\>	单词结尾处
[:alnum:]	任何字母与数字
[:alpha:]	任何字母
[:cntrl:]	任何控制字符
[:digit:]	任何数字
[:graph:]	任何图像字符
[:lower:]	任何小写字母
[:print:]	任何可输出字符
[:punct:]	任何标点符号
[:space:]	任何空格字符
[:upper:]	任何大写字母
[:xdigit:]	任何十六进制数字
*	量词，代表“零次或以上”
\c	照字面意义的c字符，即使c本身是正则表达式语法里的特殊字符。例如：“\*”模板代表“*”字符，而不是解释为量词；“\\”模板代表“\”字符本身，而不是作为转义符号。另一种等效语法是将常义字符放在一对方括号里，例如：[*]、[\]

**egrep [options] pattern [files]**

**grep**

**/bin**

**stdin stdout -file --opt --help --version**

egrep 的用途与 grep 一样，但是采用扩充型的正则表达式语法。效果如同 grep -E。表 3 是两者之间的差异部分：

表 3：正则表达式语法（基本型与扩充型的差异部分）

基本型	扩充型	意义
\		或（并集）
\+	+	量词，代表一次或多次
\?	?	量词，代表零次或一次
\{n\}	{n}	量词，代表 $n$ 次
\{n,\}	{n,}	量词，代表至少 $n$ 次
\{n,m\}	{n,m}	量词，代表至少 $n$ 次，至多 $m$ 次 ( $n < m$ )

表 2 与表 3 中的量词必须放在其他模板之后，用以表示该模板可连续重复出现的次数。例如，“.”代表“任意单一字符出现零次或多次”；“abc{1,3}”代表“字符串 abc 至少出现一次，至多连续出现三次”，也就是说“abc”、“abcabc”、“abcabcabc”这三个字符串都符合模板。

**fgrep [options] [fixed\_strings] [files]**

**grep**

**/bin**

**stdin stdout -file --opt --help --version**

fgrep 的用途类似 grep，但它接受的是固定字符串，而非正则表达式（相当于 grep -F 的效果）。fgrep 可接受多个固定字符串，字符串之间以换行符（\n）隔开。举例来说，若要找出文件中的“one”、“two”、“three”字符串，应该像下面这样输入 fgrep 命令：

```
$ fgrep 'one 注意这里有\n符号
> two
> three' myfile
```

使用 \n 作为字符串分隔符似乎有违一般惯例，但其实这是因为 fgrep 的典型用法是从文件（而不是命令行）读取模板字符串，而文件里的字符串主要是以 \n 为分隔符。举例来说，如果你有一个字典文件，每行各有一个字符串，像下面这样：

```
$ cat my_dictionary_file
aardvark
aback
abandon
...
```

这时候就可以用下列方法从一组输入文件中找出这些字符串(加上-f选项,表示从文件中读取模板字符串):

```
$ fgrep -f my_dictionary_file *
```

fgrep 也很适合搜索非字母数字字符,像\*、{等,因为fgrep将它们视为普通字符,而不是作为正则表达式中的模板字符来解释。

## cut -(b|c|f) range [options] [files]

coreutils

/usr/bin

stdin stdout -file --opt --help --version

cut 可从文本文件中截取出某栏数据。栏(column)的定义有多种形式,第一种是字符位置(例如,每行的第19个字符):

```
$ cut -c19 myfile
```

另一种定义是依据字节位置(这应该与字符定义一样,除非在多字节字符的环境下):

```
$ cut -b19 myfile
```

最常用的定义形式是分隔字段,例如,以每行的第5个字段(以逗号分隔):

```
$ cut -d, -f5 myfile
```

cut 可同时截取多个字段,你可以提供一段范围(3-16)、以逗号分隔的字段编号(3,4,5,6,8,16)或两种格式混合使用(3,4,8-16)。描述范围时,若没指定范围起点(例如-16),则假设从第一栏开始(即1-16);若没指定范围终点(例如5-),则假设到最后一栏。

## 常用选项

-d C

搭配-f选项时,将C视为输入字段之间的分隔符。默认的分隔符为\t(tab)。

`--output delimiter=C` 搭配 `-f` 选项时，以 `C` 作为输出字段之间的分隔符。默认的分隔符为 `\t` (tab)。

`-s` 抑制（不输出）不含分隔符的文本行。

## **paste [options] [files]**

**coreutils**

**/usr/bin**

**stdin stdout -file --opt --help --version**

`paste` 是 `cut` 的反操作：将多个文件视为栏，将它们组合到输出：

```
$ cat letters
A
B
C
$ cat numbers
1
2
3
4
5
$ paste numbers letters
1 A
2 B
3 C
4
5
$ paste letters numbers
A 1
B 2
C 3
4
5
```

## **常用选项**

`-d delimiters` 以 `delimiters` 为输出栏之间的分隔符。默认值为 `\t` (tab) 符。当你提供多个字符时（例如 `-dxyz`），`paste` 会依循环顺序将它们分别插入到每行的各栏之间（第一个分隔符是 `x`，接着是 `y`，然后是 `z`，再回到 `x` ……）。

`-s` 交换输出的行与列：

```
$ paste -s letters numbers
A B C
1 2 3 4 5
```

**tr** 命令可对输入数据中的某些特定字符进行一些简单、有用的操作（删除、删除重复字符等），或是转换成另一组字符。例如，我们可将所有元音字母都换成星号：

```
$ cat myfile
This is a very wonderful file.
$ cat myfile | tr aeiouAEIOU '*'
Th*s *s * v*ry w*nd*r*f*l f*l*.
```

或是删除所有元音字母：

```
$ cat myfile | tr -d aeiouAEIOU
Ths s vry wndrft fl.
```

或是将所有小写字母都换成大写：

```
$ cat myfile | tr 'a-z' 'A-Z'
THIS IS A VERY WONDERFUL FILE.
```

**tr** 将 *charset1* 中的第一个字符转换成 *charset2* 中的第一个字符，第二个对第二个，第三个对第三个……依此类推。如果 *charset1* 的长度为 *N*，则 *charset2* 里只有前 *N* 个字符会被用到（若 *charset1* 的长度超过 *charset2*，请见 **-t** 选项的说明）。

*charset* 有下列几种表达形式：

形式	意义
ABCD	字符序列 A、B、C、D
A B	范围从 A 到 B 之间的字符序列
[x*y]	连续 y 个 x 字符（例如 [K*6] 相当于 “KKKKKK”）
[[:class:]]	如同 <b>grep</b> 所接受的字符类（[:alnum:],[:digit:]等）

**tr** 也认识 **printf**（参阅 176 页）所能够接受的一系列控制符：**\a** (bell)、**\b** (backspace)、**\f** (formfeed)、**\n** (newline)、**\r** (return)、**\t** (tab)、**\v** (vertical tab)，以及表示成 **\ooo** 形式的任何字符（ooo 是该字符的八进制编码值）。

`tr` 适合用于简易的转换，但是对于比较复杂的转换，请考虑使用 `sed`、`awk` 或 `perl`。

## 常用选项

- d 删除任何出现于 `charset1` 中的字符。
- s 缩减连续出现的 `charset1` 字符。例如：`tr -s aeiouAEIOU` 会删除相邻的重复元音字母，使其成为单元音（“reeeeeeally”会变成“really”）。
- c 作用在 `charset1` 没列出的任何字符上。
- t 若 `charset1` 的长度超过 `charset2`，则截掉 `charset1` 末端多余的部分，使两者等长。若没指定 `-t` 选项，`charset2` 的最后一个字符会被（暗中）重复多次，直到 `charset2` 的长度等于 `charset1` 为止。

## `sort [options] [files]`

coreutils

/bin

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

`sort` 命令以字母顺序（或你指定的其他顺序）<sup>\*</sup>输出文本行。所有输入文件会先被串接在一起，然后一起参与排序。

```
$ cat myfile
def
xyz
abc
$ sort myfile
abc
def
xyz
```

## 常用选项

- f 不区分大小写。
- n 以数值顺序排序（也就是“9”会出现在“10”之前）。默认行为是以字母顺序排序（“10”会出现在“9”之前，因为“10”的第一个字符是“1”）。
- g 数值排序，但是能辨识科学计数法。例如：

```
$ cat numbers
743
```

```

7.4e3          (7.4 x 103 = 7400)
74.3
7.43
$ sort  n numbers
7.4e3          (这个数字其实最大)
7.43
74.3
743
$ sort  g numbers
7.43
74.3
743
7.4e3          (这次顺序对了)

```

关于科学计数法的技术细节，请参阅 `info sort`。

- u                    忽略重复的文本行。如果搭配 `-c` 选项来检查排序文件，在发现任何连续的相同文本行时，会造成检查失败。
- c                    只检查不排序。如果输入文件已经排序，则输出原文件，否则输出错误信息。
- b                    忽略前导空格。
- r                    变换输出顺序（变成降序）。
- k *pos1[,pos2]*    定义排序键（sorting key）的位置。
- t *X*                使用 *X* 作为字段分隔符（需配合 `-k` 选项）。

“排序键”是文本行中一段特定部位（例如，每行的“第五个字符”），用来与其他行的对应部位相比较。举例来说，若某文件的内容是下面这样：

```

aaaaaz
bbbby

```

以 `sort` 的默认行为排序上述文件时，会得到“aaaaaz”出现在“bbbby”之前的结果，因为 `sort` 是以整段文本行为比较单位。但如果以“第五个字符”为排序键，则“bbbby”会出现在“aaaaaz”之前，因为“y”在“z”之前。

`-k` 选项的 *pos1* 与 *pos2* 参数可以 *F.C* 形式来表示，其中，*F* 是字段的编号，而 *C* 是排序键第一个字符（或最后一个字符）在字段里的位置（*F* 和 *C* 都是从 1 起算）。省略 *pos2* 时，表示排序键的范围是到达行尾；省略 *C* 时，表示以字段的第一个字符为排序键的第一个字符（或最后一个字符）。



所以, `sort -k1,5` 是以第一个字段的第五个字符为排序键, 而 `sort -k2,8,5` 是以“第二个字段的第八个字符到第五个字段的第一个字符”为排序键。

你可用多个 `-k` 选项来定义多个排序键, 排序键的效力高低等同于命令行定义的顺序 (最先定义的排序键的效力最高)。

## **uniq [options] [files]**

coreutils

/usr/bin

stdin stdout -file --opt --help --version

`uniq` 命令作用在连续重复的文本行。举例来说, 若 *myfile* 的内容是下面这样:

```
$ cat myfile
axyz
bwij
bwij
clmn
bwij
```

则 `uniq` 将检测并处理 (或你指定的操作) 那两行连续的 `bwij`, 但是第三个 `bwij` 不受影响:

```
$ uniq myfile
axyz
bwij
clmn
bwij
```

`uniq` 通常用于处理 `sort` 排序后的数据:

```
$ sort myfile | uniq
axyz
bwij
clmn
```

在此例中, 只会剩下一行 `bwij`, 因为 `sort` 将它们排在一起了。此外, 你可以要求 `uniq` 计算重复次数, 而不是消除它们:

```
$ sort myfile | uniq -c
  1 axyz
  3 bwij
  1 clmn
```

## 常用选项

- c 计算相邻的重复文本行。
- i 不区分大小写。
- u 只输出不重复的文本行。
- d 只输出重复的文本行。
- s *N* 检测重复文本行时, 忽略每行的前 *N* 个字符。
- f *N* 检测重复文本行时, 忽略每行的前 *N* 个字段(字段之间以空格隔开)。
- w *N* 检测重复文本行时, 只考虑每行的前 *N* 个字符。如果搭配 -s 或 -f 选项, `uniq` 会先忽略指定数量的字符或字段, 然后才考虑其后的 *N* 个字符。

## **tee [options] files**

coreutils

/usr/bin

**stdin stdout -file --opt --help --version**

`tee` 的基本功能很像 `cat`, 两者同样都能够将 `stdin` 输出到 `stdout`, 不同之处在于 `tee` 还能够同时将输出写到你指定的文件里。`tee` 的典型用法是放在管道中间, 将拦截前端命令的 `stdout` 写入某文件, 同时又将同样的 `stdout` 传给下一个命令。例如:

```
$ who | tee original_who | sort
```

在此例中, 你会在 `stdout` 看到经过排序的 `who` 输出信息, 而在 `original_who` 文件中得到未经过排序的原始信息。

## 常用选项

- a 将信息附加到文件末端, 而不是改写原文件。
- i 不理睬中断信号。

---

## 14.1. 高级文件操作

虽然 `grep`、`cut`、`paste`、`tr`、`sort`、`uniq`、`tee` 与它们的各种排列组合已足以满足许多应用需求, 但它们其实只是全部文本操作工具中的冰山一角,

Linux 还提供了一系列功能更强大（也比较难学）的工具。这些工具的每一个几乎都值得以专书来讨论，以本手册有限的篇幅，只能粗略举几个典型范例，让你对它们的用途与用法有个大概的认识。

## awk

awk 是一种模式匹配语言，它能找出符合正则表达式描述的数据，然后对这些数据进行操作。举两个简单的例子来说明，假设 *myfile* 是一个文本文件：

```
$ awk '{print $2, $4}' myfile
```

上述命令可输出该文件每一行的第二个与第四个字。

```
$ awk '{length($0) < 60}' myfile
```

上述命令可找出长度小于 60 个字符的文本行，然后输出它们。

## sed

sed 也是一个模式匹配引擎，可用于操作文本行。它的语法相当接近于 vim 与 ed 这两个文本编辑工具。以下是两个简单的例子：

找出文件中的所有“red”字符串，然后改成“hat”：

```
$ sed 's/red/hat/g' myfile
```

输出文件内容，但是跳过前十行：

```
$ sed '1,10d' myfile
```

## m4

m4 是一种宏处理语言，它可找出文件中的关键字，然后替换成对应字符串。举例来说，若 *myfile* 文件的内容是下面这样：

```
$ cat myfile
My name is NAME and I am AGE years old
ifelse(QUOTE,yes,No matter where you go... there you are)
```

看看 m4 分别对 NAME、AGE 和 QUOTE 这三个关键字进行替换的效果：

```
$ m4 -DNAME=Sandy myfile
My name is Sandy and I am AGE years old
```

```
$ m4 -DNAME=Sandy -DAGE=25 myfile
My name is Sandy and I am 25 years old

$ m4 -DNAME=Sandy -DAGE=25 -DQUOTE=yes myfile
My name is Sandy and I am 25 years old
No matter where you go... there you are
```

## Perl、Python

Perl 与 Python 是功能齐全的脚本语言 (scripting language)，它们强大到足以建立完整的、牢固的应用系统。

## 15. 文件压缩

gzip	将文件压缩成 GNU Zip 格式的压缩文件
gunzip	还原 GNU Zip 格式的压缩文件
compress	将文件压缩成 Unix 的传统格式
uncompress	还原 Unix 传统格式的压缩文件
zcat	通过 stdin/stdout 压缩或还原文件 (支持 GNU Zip 与 Unix 传统格式)。
bzip2	将文件压缩成 BZip 格式的压缩文件
bunzip2	还原 Bzip 格式的压缩文件
zip	将文件压缩成 WinZip 格式的压缩文件
unzip	还原 WinZip 格式的压缩文件
uencode	将文件转换成 uuencoded 格式
uudecode	还原 uuencoded 格式的文件

Linux 支持多种压缩格式，包括最常用的 GNU Zip (扩展名为 .gz)、Unix 传统的 Lempel Ziv 格式 (.Z)、Windows 平台上最普遍的 WinZip 格式 (.zip) 以及压缩比最高的 Bzip 格式 (.bz2)。

与压缩文件有关的后续操作主要是封装与转换，比如说，将多个压缩文件集

合到一个封装文件（或是用 tar 将多个普通文件集合到一个封装文件，然后再予以压缩），以方便通过网络传输。

本节也会介绍如何将压缩文件（binary 文件）转换成适合 E-mail 传递的格式（注 11）。虽然现在已有标准协议与 MIME 工具能够自动处理电子邮件附件，但是至今仍有人使用老式的 uuencode 与 uudecode 程序，所以本节也会介绍它们。

如果你遇到我们没提到的压缩格式，像 Macintosh 计算机上常见的 hqx/sit 文件，或是 MS-DOS 时代的 Arc、Zoo、ARJ 等，你可以在 <http://www.faqs.org/faqs/compression-faq/part1/section-2.html> 和 <http://www-106.ibm.com/developerworks/library/l-lw-comp.html> 找到相关信息。

## gzip [options] [files]

gzip

/bin

stdin stdout -file --opt --help --version

gzip 可将文件压缩成 GNU Zip 格式，而 gunzip 可还原（解压缩）GNU Zip 格式的压缩文件。这类压缩文件的扩展名为 .gz。

### 典型用法

gzip file	压缩 file 成为 file.gz。原本的 file 会消失。
gzip -c file	将产生的压缩数据送到 stdout。
cat file   gzip	从管道中取得原始数据，并产生压缩数据。
gunzip file.gz	解开 file.gz 以产生 file。原本的 file.gz 会消失。
gunzip -c file.gz	将解压缩后的数据送到 stdout。
cat file.gz   gunzip	从管道中取得压缩数据，并予以解压缩。
zcat file.gz	解开 file.gz 压缩文件，将还原后的数据送到 stdout（相当于 gunzip -c）

---

译注 11：用于传递 E-mail 的 SMTP 协议只能接受 ASCII 字符，所以不能直接传递 binary 文件。

---

## 典型的 tarball (.tar.gz) 处理方法

`tar czf myfile.tar.gz dirname` 封装整个 *dirname* 目录，然后压缩封装文件。

`tar tzf myfile.tar.gz` 显示压缩文件的内容。

`tar xzf myfile.tar.gz` 先解开压缩文件，然后还原封装文件。

上述三个tar命令都可以再加上v选项，以使tar输出正在处理的文件名称。

## compress [options] [files]

**ncompress**

`/usr/bin`

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

compress 和 uncompress 能处理 Unix 的标准压缩格式 (Lempel Ziv)。这类压缩文件的扩展名为 .Z。

## 典型用法

`compress file` 压缩 *file* 成为 *file.Z*。原本的 *file* 会消失。

`compress -c file` 将产生的压缩数据送到 stdout。

`cat file | compress` 从管道中取得原始数据，并产生压缩数据。

`uncompress file.Z` 解开 *file.Z* 以产生 *file*。原本的 *file.Z* 会消失。

`uncompress -c file.Z` 将解压缩后的数据送到 stdout。

`cat file.Z | uncompress` 从管道取得压缩数据，并予以解压缩。

`zcat file.Z` 解开 *file.Z* 压缩文件，将还原后的数据送到 stdout (相当于 `uncompress -c`)

## 典型的 tarball (.tar.Z) 处理方法

`tar cZf myfile.tar.gz dirname` 封装整个 *dirname* 目录，然后压缩封装文件。

`tar tZf myfile.tar.gz` 显示压缩文件的内容。

```
tar xzf myfile.tar.gz
```

先解开压缩文件，然后还原封装文件。

在 `tar` 命令加上 `v` 选项，可使 `tar` 输出正在处理的文件的名称。

## **bzip2 [options] [files]**

**bzip2**

`/usr/bin`

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

`bzip2` 和 `bunzip2` 可处理 Burrows-Wheeler 格式的压缩文件。这是一种高压缩比的算法，但是需要较强的运算能力。Burrows-Wheeler 格式压缩文件的扩展名为 `.bz2`。

### **典型用法**

<code>bzip2 file</code>	压缩 <code>file</code> 成为 <code>file.bz2</code> 。原本的 <code>file</code> 会消失。
<code>bzip2 -c file</code>	将产生的压缩数据送到 <code>stdout</code> 。
<code>cat file   bzip2</code>	从管道取得原始数据，并产生压缩数据。
<code>bunzip2 file.bz2</code>	解开 <code>file.bz2</code> 以产生 <code>file</code> 。原本的 <code>file.bz2</code> 会消失。
<code>bunzip2 -c file.bz2</code>	将解压缩后的数据送到 <code>stdout</code> 。
<code>cat file.bz2   bunzip2</code>	从管道取得压缩数据，并予以解压缩。
<code>bzcat file.bz2</code>	解开 <code>file.bz2</code> 压缩文件，将还原后的数据送到 <code>stdout</code> （相当于 <code>bunzip2 -c</code> ）

### **典型的 tarball（.tar.bz2）处理方法**

<code>tar cjf myfile.tar.bz2 dirname</code>	封装整个 <code>dirname</code> 目录，然后压缩封装文件。
<code>tar tjf myfile.tar.bz2</code>	显示压缩文件的内容。
<code>tar xjf myfile.tar.bz2</code>	先解开压缩文件，然后还原封装文件。

加上 `v` 选项，可使 `tar` 输出正在处理的文件的名称。

zip和unzip可用于处理WinZip（或称为PKZip）压缩格式的文件。这类压缩文件的扩展名为.zip。在Linux的压缩工具中，zip和unzip算是很特别的了，因为它们是从Windows平台移植而来的工具。它们不仅是选项名称与其他压缩工具不一致，而且也不会删除原本的文件，更重要的差异是，unzip会自己解释文件名参数（这其实是Windows平台的惯例）。虽然它们也支持shell的通配符(\*、?)，但是你不能让shell代为展开，而必须让unzip收到含有通配符的参数。因此，如果你要解压缩一组\*.zip文件，依照直觉应运行如下命令：

```
unzip *.zip
```

其实是行不通的，正确的命令应该是下面这样：

```
unzip \*.zip
```

请注意，zip没有同样的毛病，它可以接受shell展开的文件名。

以下是常用的操作方法：

```
zip myfile.zip file1 file2 file3.....
```

将一系列文件压缩成myfile.zip。请注意，你可用带用通配符的文件名来表示一系列要被压缩的文件（file1 file2 file3.....），也就是说，zip mypic.zip \*.jpg是有效的命令（zip mypic.zip \\*.jpg反而是错误的！）。

```
zip -r myfile.zip dir1 dir2 dir3.....
```

递归压缩每个目录，产生一个myfile.zip文件。

```
unzip -l myfile.zip
```

显示压缩文件的内容。如果要显示多个压缩文件的内容，正确的命令是unzip -l \\*.zip。

```
unzip myfile.zip
```

解开整个.zip压缩文件。如果一次解开多个压缩文件，正确的命令是unzip \\*.zip。



在 E-mail 附件与 MIME 问世之前, 要通过 E-mail 来传递 binary 文件需要费一番工夫。你必须先用 uencode 将二进制文件编码成一种特殊的 ASCII 格式, 类似下面这样:

```
begin 644 myfile
M (R`N8F%S:~]P<F]F:6QE"B,@4G5N<R!F:7) S
  -"!W:&5N (&QO9V=I;F<@:6X@
M: 6YD97 (@!TY/344*"G1R87`@) PH@
  ('1E<W0@+6X@ (B134TA?04`83E1?4$E$
...
end
```

收信方得到这种奇怪的数据后, 可用 udecode 来还原。

举例来说, 若要产生 mybin 文件的 uencode 格式的输出生, 使用的命令如下:

```
$ uencode mybin biname > myfile.uu
```

在此例中, 第一个参数 (mybin) 是要被处理的源文件, 第二个参数 (biname) 是作为解码时所要产生的文件的名称。所以, 当译码者使用 udecode 解开 myfile.uu 文件之后:

```
$ udecode myfile.uu
```

将会得到一个名为 biname 的新文件, 该文件的内容与编码方的 mybin 完全一样。

---

## 16. 文件比较

diff           逐行比较两个文件 (或目录)。

comm           逐行比较两个有序文件。

cmp            逐字节比较两个文件。

md5sum         计算文件内容的 MD5 校验码

Linux 提供了三种比较文件内容的方法:

- 逐行比较 (diff、diff3、sdiff、comm), 适用于文本文件。

- 逐字节比较 (cmp)，通常用于二进制文件。
- 比较校验码 (md5sum、sum、cksum)。

上述工具都是针对命令行环境而设计的，如果你需要图像文件比较工具，不妨试试 `xxdiff` (<http://xxdiff.sourceforge.net>)。

<b>diff [options] file1 file2</b>	<b>diffutils</b>
<i>/usr/bin</i>	<b>stdin stdout -file --opt --help --version</b>

`diff` 可逐行比较两个文件（或两个目录）。当比较文本文件时，`diff` 可产生一份详细的差异报告。对于二进制文件，`diff` 只汇报它们是否有差异。如果相比的两个文件没有差异（无论是文本文件还是二进制文件），`diff` 不会产生输出信息。

`diff` 比较两个文本文件时所产生的差异报表，其格式如下：

```

标示符号（行号，差异类型）
< file1 的对应文本段（如果有）

> file2 的对应文本段（如果有）

```

举例来说，假设 *fileA* 的内容原本是下面这样：

```

Hello, this is a wonderful file.
The quick brown fox jumped over
the lazy dogs.
Goodbye for now.

```

现在我们删掉第一行，将第二行的“brown”改成“blue”，在最后一行多加一段文本，形成 *fileB* 文件：

```

The quick blue fox jumped over
the lazy dogs.
Goodbye for now.
Linux r00lz!

```

则 `diff fileA fileB` 以默认格式所产生的输出信息如下：

```

1,2c1      (fileA 的 line 1, 2, 成为 fileB 的 line 1)
< Hello, this is a wonderful file.  (fileA 的 line 1, 2)
< The quick brown fox jumped over
---                                (diff 的分隔符)
> The quick blue fox jumped over    (fileB 的 line 1)

```

```

4a4                                     (Line 4 是新增的)
> Linux r00lz!                         (新增的文本行)

```

在此例中，前导的<和>符号，分别代表`fileA`（旧文件）与`fileB`（新文件）。初次使用`diff`的用户可能会觉得上述格式很不人性化，事实上，`diff`的输出本来就不是给“人”看，而是给`patch`之类的修补工具用来把旧文件“修补”成新文件，许多程序设计师就是用这种方式来发布新版软件的修补文件。`diff`还提供了多种不同的输出格式，可分别支持多种修补工具。

选项	输出格式
<code>-n</code>	RCS 版本控制格式，与 <code>rcsdiff</code> 的输出效果一样
<code>-c</code>	上下文差异格式 (context diff format)，即 <code>patch</code> 可接受的格式
<code>-D macro</code>	C 语言的预处理器格式（使用 <code>#ifdef macro ...#else ... #endif</code> ）
<code>-u</code>	统一格式，将文件合并，在被删除的行之前加上“-”，新增行之前加上“+”。
<code>-y</code>	对比格式，使用 <code>-W</code> 可调整输出的宽度
<code>-e</code>	产生一个能将 <code>fileA</code> 改成 <code>fileB</code> 的 <code>ed script</code>
<code>-q</code>	不汇报差异处，只说明两者内容是否有异即可

`diff`也可以用来比较目录：

```
$ diff dir1 dir2
```

它会比较两个目录下的任何同名文件，并列出某目录有但另一个目录却没有的文件。加上`-r`选项可比较两个完整的子目录：

```
$ diff -r dir1 dir2
```

这可能会产生非常多的差异报告。

## 常用选项

- `-b` 不理睬空格的差异。
- `-B` 不理睬空白行。
- `-i` 大小写视为相同。
- `-r` 比较目录时，连同所有子目录也一并比较。

diff 不是唯一的文件差异比较工具，它还有其他同性质的伙伴，像 diff3（一次可比较三个文件）和 sdiff（能依照你的指示来整合两个文件差异处，产生第三个文件）。

**comm [options] file1 file2**

**coreutils**

**/usr/bin**

**stdin stdout -file --opt --help --version**

comm 命令比较两个有序文本文件，然后产生三栏输出（字段之间以 \t 隔开）：

1. 只出现在 *file1*，而 *file2* 没有的文本行。
2. 只出现在 *file2*，而 *file1* 没有的文本行。
3. 在两个文件中都出现的文本行。

举例来说，若 *file1* 和 *file2* 的内容分别为：

<i>file1:</i>	<i>file2:</i>
apple	baker
baker	charlie
charliedark	

则 comm 会产生这样的输出：

```
$ comm file1 file2
apple
    baker
    charlie
    dark
```

## 常用选项

- 1 省略第一栏。
- 2 省略第二栏。
- 3 省略第三栏。

**cmp [options] file1 file2 [offset1 [offset2]]**

**diffutils**

**/usr/bin**

**stdin stdout -file --opt --help --version**

cmp 比较两个文件的内容，如果完全相同，则不汇报任何信息；否则列出第一个差异处：

```
$ cmp myfile yourfile
myfile yourfile differ: char 494, line 17
```

cmp 的默认行为只告诉你差异在哪里，所以特别适合用来比较二进制文件。相对地，diff 只适合用来比较文本文件。

通常，cmp 从文件的开头处开始比对，但是你也可以改变比对起点：

```
$ cmp myfile yourfile 10 20
```

上述命令表示从 *myfile* 的第 10 个字节、*yourfile* 的第 20 个字节开始比较。

## 常用选项

-l 长格式输出：输出每个不同的字节：

```
$ cmp -l myfile yourfile
494 164 172
```

这表示距离文件开头的第 494 个字节，在 *myfile* 是“t”（八进制 164），而在 *yourfile* 是“z”（八进制 172）。

-s 不输出任何信息到 stdout，只返回适当的结束代码：0 代表两文件相同，1 代表有差异，若有某种原因造成比较失败（比如说，权限不足以打开某文件），则返回其他代表错误原因的代码。

## md5sum files --check file

coreutils

/usr/bin

stdin stdout -file --opt --help --version

md5sum 可计算文件的 32-byte MD5 校验码（关于技术细节，请参阅 <http://www.faqs.org/rfcs/rfc1321.html>）：

```
$ md5sum myfile
dd63602df1cccb57966d085524c3980f myfile
```

对于不同内容的文件，不管差异多么细微，都不太可能算出相同的 MD5 校验码，所以这是用来判断文件内容是否相同的可靠方法：

```
$ md5sum myfile1 > sum1
$ md5sum myfile2 > sum2
$ diff -q sum1 sum2
Files sum1 and sum2 differ
```

使用 --check 可检查一组文件中的哪些文件被改变了：

```

$ md5sum file1 file2 file3 > mysum
$ md5sum --check mysum
file1: OK
file2: OK
file3: OK
$ echo "new data" > file2
$ md5sum --check mysum
file1: OK
file2: FAILED
file3: OK
md5sum: WARNING: 1 of 3 computed checksums did NOT match

```

还有两个与md5sum性质类似的程序是sum和cksum，它们也是用于计算文件的校验码，不过使用的算法不一样而已。sum兼容于其他Unix系统，特别是BSD Unix（默认）或System V Unix（-s选项），而cksum可产生CRC校验码：

```

$ sum myfile
12410 3
$ sum -s myfile
47909 5 myfile
$ cksum myfile
1204634076 2863 myfile

```

第一个数值是校验码，第二个数值是块数。如你所见，校验码的数值不大，这表示很可能会发生冲突（collision）（译注 12），所以sum和cksum都不是很可靠的方法。md5sum是目前为止最可靠的方法。

使用diff或cmp时，你的系统上要同时存有要被比较的所有文件。然而，md5sum就没有这些缺点。所以，许多提供光盘镜像文件（.iso）下载服务的网站，都会另外提供一个含有MD5校验码的小文件（文件名通常是MD5SUM），让用户可用来检查自己下载的.iso文件是否与远程系统上的版本完全相同。

---

## 17. 磁盘与文件系统

**df**          显示每个已挂载的文件系统的可用容量。

**mount**      将磁盘分区挂载到目录树下。

译注 12：冲突：不同数据却得到相同校验码。校验码本身的长度大约反比于冲突机率（长度越长，机会越小）。

`umount` 卸载某磁盘分区。  
`fsck` 检查磁盘分区是否有错误。  
`sync` 将留滞在内存里的高速缓存数据写回磁盘。

Linux 系统可管理多个磁盘或磁盘分区。描述存储单位的术语很多，磁盘 (disk)、分区 (partition)、文件系统 (filesystem)、卷 (volume)，甚至连目录 (directory) 都有人说，而且各操作系统对这些术语的定义也不一致。为了避免误解，有必要先对 Linux 系统的相关术语进行解释。

磁盘 (disk) 是硬件存储设备，可被划分成若干个分区 (partition)，各分区被视为逻辑上独立的存储设备。Linux 以 `/dev` 目录下的特殊文件来代表分区。例如，`/dev/hda7` 是第一个 IDE 接口上的主硬盘 (master) 的某个分区，`/dev/sda` 代表系统中的第一个 SCSI 硬盘，而 `/dev/sda2` 是该硬盘上的第二个分区。以下是 `/dev` 下一些常见的特殊文件：

`hda` 第一个 IDE 接口上的主硬盘 (master)；分区为 `hda1`、`hda2` 等。  
`hdb` 第一个 IDE 接口上的次硬盘 (slave)；分区为 `hdb1`、`hdb2` 等。  
`hdc` 第二个 IDE 接口上的主硬盘；分区为 `hdc1`、`hdc2` 等。  
`hdd` 第二个 IDE 接口上的次硬盘；分区为 `hdd1`、`hdd2` 等。  
`sda` 第一个 SCSI 设备；分区为 `sda1`、`sda2` 等。  
`sdb` 第二个 SCSI 设备；分区为 `sdb1`、`sdb2`……。 `sdc`、`sdd` ……依此类推。  
`ht0` 能自动倒带的第一个 IDE 磁带机 (然后依序是 `ht1`、`ht2` 等)。  
`nht0` 不能自动倒带的第一个 IDE 磁带机 (然后依序是 `nht1`、`nht2` 等)。  
`st0` 第一个 SCSI 磁带机 (然后是 `st1`、`st2` 等)。  
`scd0` 第一个 SCSI 光驱 (然后是 `scd1`、`scd2` 等)。  
`fd0` 第一个软驱 (然后是 `fd1`、`fd2` 等)，通常挂载于 `/mnt/floppy`。

分区必须先被格式化 (formatted)，之后才可存储文件。所谓的格式化，就是写入一个文件系统 (filesystem) 到分区中。文件系统决定如何组织目录，如何记录文件名称、属性、权限以及文件数据的分布方式等。每个分区都只能有一个文件系统，但是 Linux 容许不同的分区别使用不同格式的文件系

统。Fedora默认的文件系统是ext3,其他常见的文件系统形式包括ext2(ext3的前一代)、vfat(MS Windows的文件系统)、ISO-9660(光盘上的文件系统)。硬盘的格式化程序通常是在安装Linux的过程中完成的,除非你在安装好Linux之后又安装新硬盘,或是想使用任何新的插入式存储媒介(软盘、USB等)。

建好文件系统(格式化)之后,还必须将文件系统挂载(mount)到目录树中的某节点下(通常在/mnt目录下,但是常有例外)才能使用它们。举例来说,若将某个Windows VFAT文件系统挂载到/mnt/win目录下,则该文件系统将成为目录树的一部分,而你在/mnt/win目录下所创建的文件将存储在该VFAT文件系统中。对于不在使用中的文件系统,你可以予以卸载(unmount),使得一般文件操作命令(cd、cp、rm、mkdir等)不能访问它们。系统管理员在进行维护时,就经常需要卸载文件系统。Linux系统在正常开机期间,会自动挂载硬盘上的文件系统。

## **df [options] [disk devices| files| directories]**

coreutils

/bin

stdin stdout -file --opt --help --version

df(disk free)程序可显示特定分区(文件系统)的总容量、已用空间、可用空间、挂载点。如果你指定某文件或目录,df就显示该文件或目录所属的文件系统信息;若不指定任何参数,df则直接报告所有已挂载的文件系统:

```
s df
Filesystem 1K-blocks      Used Available Use% Mounted on
/dev/sda1    1922892      860844    964368   48% /
/dev/sda2    15713888    10093176    4822472   68% /usr
/dev/sdb1     3936936     2022384    1714568   55% /home
/dev/sdb2    12668616     5311640    6713444   45% /var
```

## **常用选项**

- k 以千字节(默认值)或兆字节为容量单位。
- m
- B N 以你定义的块单位来显示容量。N默认值为1024,也就是假设每一块的单位容量是1KB。
- h 自动选择合适的容量单位。举例来说,若你有两个分区,分别剩下1GB与25KB的空间,则df -h会输出“1G”与“25K”。
- H



-h 选项使用的单位基数是 1024，而 -H 使用的单位基数是 1000（所以 -H 所显示的数值比较大）。

- l 只显示本地文件系统，不显示 NFS、SMB 之类的网络文件系统。
- T 额外显示文件系统类型（ext2、vfat 等）。
- t type 只显示特定类型的文件系统。
- x type 不显示特定类型的文件系统。
- i Inode 模式。显示每个文件系统的全部、已用、可用的 inode 数量，而非磁盘块。

<b>mount [options] device   directory</b>	<b>mount</b>
<i>/bin</i>	stdin stdout -file --opt --help --version

mount 命令使某硬件存储设备（通常是硬盘分区，例如 `/dev/hda1`）出现于系统目录树中的某个节点（挂载点）下，也就是某个现有目录下（例如 `/mnt/mydir`），让用户可通过该节点访问已有的存储设备：

```
# mkdir /mnt/mydir
# mount /dev/hda1 /mnt/mydir
# df /mnt/mydir
Filesystem 1K blocks    Used Available Use% Mounted on
/dev/hda1    5052032  138556   4656840    3% /mnt/mydir
```

mount 的选项与用法多到算不清，我们只探讨其中最基本的部分。

一般情况下，mount 会先读取 `/etc/fstab` 配置文件（filesystem table，文件系统表），借此取得如何挂载磁盘的信息。举例来说，当你下达 `mount /usr` 命令时，mount 会在 `/etc/fstab` 配置文件里找出含有 `/usr` 项目的记录，如下：

```
/dev/hdc1    /usr    ext3    defaults    0 2
```

通过 `/etc/fstab` 里的记录，mount 可知道它应该将磁盘设备 `/dev/hdc1` 挂载到 `/usr` 节点下，并告诉操作系统应以 ext3 格式来解释该文件系统（注 13）。由于 mount 涉及硬件操作，所以通常只有 superuser 才能使用，但是对于软盘与光盘之类的设备，则开放给任何用户来挂载与卸载。

---

注 13： 或者，你也可以用 mount 的 -t 选项来直接指定文件系统格式，例如：`mount -t ext2 /dev/hdc1 /mnt/mydir`。详情请见 `info mount`。

**umount** 是 **mount** 的反操作：使某磁盘分区脱离操作系统的管制，而这样的过程称为卸载：

```
$ umount /mnt/cdrom
```

你不能卸载使用中的设备，比如说，假设 **/mnt/cdrom** 刚好是你（或其他用户）当前的工作目录，或是 **/mnt/cdrom** 目录下还存在尚未关闭的文件，则上述命令就会失败。对于插入式存储媒介（光盘、软盘），在未卸载之前，不应该擅自抽离存储媒介；否则，有可能会损毁该存储媒介上的文件系统。

使用 **-a** 选项可一次卸载所有已挂载的文件系统：

```
# umount -a
```

当你根据正常步骤关机时，上述命令会被自动运行一次。

**fsck** (filesystem check) 命令可用于检查 Linux 磁盘分区上的文件系统是否有问题，甚至修复所找到的错误。当你的系统刚开机时，**fsck** 会自动运行；不过，必要时，你也可以自己动手运行它。一般而言，你必须先卸载要被检查的设备才可使用 **fsck** 进行检查，因为检查过程中不应该受到其他程序的干扰：

```
# umount /dev/hda3
# fsck -f /dev/hda3
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/home: 172/1281696 files (11.6% non-contiguous),
      1405555/2562359 blocks
```

其实 **fsck** 本身只是个前台代理程序，真正执行检查工作的一组位于 **/sbin** 下的 **fsck.\*** 工具程序（例如 **fsck.ext2**、**fsck.ext3**）。并非每种文件系统都有对应的检查工具，使用下列命令可查出你的系统有哪些工具可用：

```
$ ls /sbin/fsck.* | cut -d. -f2
```

## 常用选项

- A 依次检查 */etc/fstab* 所列的每一个文件系统。
- N 只显示检查应该执行的工作，而不进行任何实质检查。
- r 每次修正错误之前询问用户一次。
- a 自动修复错误（不建议使用，除非你真的知道你在做什么；否则，有可能严重损坏文件系统）。

## sync

coreutils

/bin

stdin stdout -file --opt --help --version

`sync` 命令要求系统内核将应该写回磁盘，但是仍留滞在内存里的数据，全数写回磁盘，让磁盘上的数据现况符合实际情况。通常，涉及磁盘活动的相关操作，像读、写、改变 inode 等，多半是发生在系统的缓冲存储器中，等到系统闲置时，内核会自动将缓冲存储器里的数据写回磁盘，这样可以提升系统运作效率。通常，你不必刻意下达 `sync` 指令，但是如果（假设）你想做一些有可能造成系统死机的危险操作，这时候就有必要先运行一次 `sync`，将可能的损害降到最低。

## 磁盘分区与格式化

磁盘的分区与格式化是相当复杂的操作。这里只提出一些你可能需要的程序，从它们的 manpage 开始看起吧。

<code>parted</code> 、 <code>fdisk</code> 或 <code>sfdisk</code>	用于磁盘分区的工具。它们的作用都一样，差异在于使用在不同的用户界面中。
<code>mkfs</code>	将磁盘分区或格式化（产生新的文件系统）。
<code>floppy</code>	格式化软盘。

## 18. 备份

<code>mt</code>	控制磁带机。
<code>dump</code>	将整个磁盘分区写入磁带。

**restore**      使用 **dump** 所产生的备份数据来恢复系统。

**tar**          读写磁带备份文件。

**cdrecord**     刻录光盘。

**rsync**        将一组文件镜像到另一个设备或其他主机中。

Linux 提供了多种方法让你备份重要文件：

- 存储到磁带机。
- 制作成光盘。
- 镜像到远程系统。

IDE接口的磁带备份机通常位于 `/dev/ht0` 下，SCSI接口的磁带机（注14）则可能在 `/dev/st0` 下。习惯上，我们会制作一个 `/dev/tape` 符号链接，使其链接到实际的磁带机设备文件：

```
$ ln -s /dev/ht0 /dev/tape
```

可用在Linux系统上的备份工具多如繁星，我们无法一一展示，而且个人习惯不同，我们也无法面面俱到。有人偏好 **cpio**，也有人惯用 **tar**，甚至迄今仍有许多人坚持使用 **dd** 进行低级备份。如果你打算使用它们，先花点时间看完它们的 **manpages** 吧。

## **mt [-f device] command**

**mt-st**

**/bin**

**stdin stdout -file --opt --help --version**

**mt** (magnetic tape, 磁带) 用来对磁带机进行简单的操作，像倒带、快进、快退、拉紧。以下列出部分常用的操作：

**status**          显示磁带机状态。

**rewind**         倒带。

**retension**      拉紧磁带。

**erase**          删除磁带中的数据。

注14： 有些系统使用 **ide-scsi** 模块将IDE磁带机仿真成SCSI设备，在这种情况下，也是通过 `/dev/st0` 来访问磁带机。

offline          使磁带机离线。  
eod              将磁带向前转到数据结尾处。

范例：

```
$ mt -f /dev/tape rewind
```

mt 也可以将磁带转动到下一个文件或下一个记录的开头处，不过，对于诸如此类的操作，通常会使用 tar 或 restore 来做。

## **dump [options] partition\_or\_files**

**dump**

/sbin

stdin stdout -file --opt --help --version

dump 将整个磁盘分区（或被选中的文件）写入备份媒介（磁带）。它支持完全备份（full backup）与增量备份（incremental backup），而且能自动判定哪些文件需要备份（也就是判断文件在上次备份之后是否有变动）。restore 可将 dump 存入备份媒体中的数据还原到系统中。

假设你想将整个 /usr 文件系统完全备份到 /dev/tape 磁带中，应该使用 -0（数字零）与 -u 选项：

```
# dump -0 -u -f /dev/tape /usr
```

-0 代表“0 级转储”（level zero dump），也就是完全备份。-u 选项在 /etc/dumpdates 文件中留下记录，注明这次备份的文件系统、级数与时间。

增量备份的级数从 1 到 9， $n$  级备份存储操作必须在  $n-1$  级备份之后才能备份新增或改动的文件。

```
# dump -1 -u -f /dev/tape /usr
```

千万别对仍在“活动中”的文件系统运行 dump，应该尽可能先予以卸载。

## **restore [options] [files]**

**dump**

/sbin

stdin stdout -file --opt --help --version

restore 能读取 dump 所产生的备份数据，然后恢复备份文件到磁盘，或是比较备份文件与磁盘文件的差异，也可以进行其他操作。restore 的最友好的操作模式是使用 -i 标记进行交互操作，这种模式让你能像浏览文件系统

一样来查看磁带内容,并选择要恢复的文件与目录,然后才开始进行恢复操作:

```
# restore -i -f /dev/tape
```

运行上述命令后,restore 会等待你下达后续操作命令,以下是你可用的命令:

<code>help</code>	显示辅助信息。
<code>quit</code>	结束程序(不进行任何恢复操作)。
<code>cd <i>directory</i></code>	如同 shell 的 <code>cd</code> 命令,这可让你在转储文件里切换工作目录。
<code>ls</code>	如同 Linux 的 <code>ls</code> 命令,可查看转储文件里的工作目录内容。
<code>pwd</code>	如同 shell 的 <code>pwd</code> 命令,可显示当前在转储文件里的工作目录的名称。
<code>add</code>	将文件或目录列于“撷取列表”(extraction list)中,该列表记录你想要恢复的文件。没有提供参数时,add 自动将当前工作目录与其所有文件都列到该列表中。
<code>add <i>filename</i></code>	将特定文件列到撷取列表中。
<code>add <i>dir</i></code>	将特定目录列到撷取列表中。
<code>delete</code>	删除撷取列表中的特定文件或目录。没有提供参数时,当前工作目录(连同其内容)会被移出撷取列表。
<code>delete <i>filename</i></code>	从撷取列表中删除特定文件。
<code>delete <i>dir</i></code>	从撷取列表中删除特定目录。
<code>extract</code>	从转储文件读出撷取列表所列的全部文件,然后将其恢复到磁盘文件系统中(诀窍:如果你的备份数据跨越多个磁带,则从最后一卷开始恢复)。

restore 也提供非交互操作模式:

```
restore -x  将磁带里的全部内容恢复到现有的文件系统中(请先 cd 到要恢复的文件系统的顶层)。
```

`restore -r` 将磁带里的全部内容恢复到一个全新格式化的文件系统中  
(请先 `cd` 到要恢复的文件系统的顶层)。

`restore -t` 列出转储文件的内容。

`restore -C` 比较转储文件内容与原本的文件系统。

## **tar [options] [files]**

**tar**

**/bin**

**stdin stdout -file --opt --help --version**

`tar` (tape archive) 是 Linux 与 Unix 的标准压缩工具。它的功能很多, 不仅能将多个文件 (连同目录结构) 压缩成适合传输或备份的单一文件, 也能解开压缩文件, 甚至能读写磁带机中的文件。例如, 下列命令将两个文件压缩在一起, 然后写入磁带机:

```
$ tar -cf /dev/tape myfile1 myfile2
```

`tar` 本身没有压缩功能, 但是它能调用其他压缩程序 (`gzip`、`bzip2`、`compress`) 来处理压缩文件。例如:

```
$ tar -czvf myarchive.tar.gz mydir 压缩整个 mydir 目录
```

```
$ tar -tzvf myarchive.tar.gz 列出压缩文件的内容
```

```
$ tar -xzvf myarchive.tar.gz 解开压缩文件
```

你也可以在命令行指定要处理的文件:

```
$ tar -xvf /dev/tape file1 file2 file3
```

如不指定, 则处理整个压缩文件。

## **常用选项**

`-c` 产生压缩文件。你必须在命令行中输入文件名与目录名。

`-r` 添加文件到现有的压缩文件。

`-u` 添加新增与改变的文件到现有的压缩文件。

`-A` 添加一个压缩文件到另一个压缩文件。例如:

```
tar -A -f /dev/tape myfile.tar.
```

`-t` 列出压缩文件的内容。

`-x` 从压缩文件撷取出文件。

- f *file* 从压缩文件读出指定的文件或是将文件写入压缩文件。*file*可以是磁带机设备文件 (*/dev/tape*) 或是普通的 tar 文件。
- d 比较封装文件与文件系统的差异。
- z 使用 gzip 压缩要写入的数据或是用 gunzip 解压缩要读取的数据。
- j 使用 bzip2p 压缩要写入的数据或是用 bunzip2 解压缩要读取的数据。
- Z 使用 compress 压缩要写入的数据或是用 uncompress 解压缩要读取的数据。
- b *N* 以  $N * 512$  bytes 为块的单位容量。
- v 输出操作状态。
- h 跟随符号链接。
- l 不超越文件系统边界。
- p 撷取文件时, 恢复它们原本的访问模式与拥有权。

### **cdrecord [options] tracks**

**cdrecord**

*/usr/bin*

**stdin stdout -file --opt --help --version**

cdrecord 可将 .iso 文件所描述的文件系统刻录进空白光盘 (CD-R 或 CD-RW)。cdrecord 只支持 SCSI 光盘刻录机, 但是通过 Linux 内核提供的 ide-scsi 仿真能力, 对于 IDE 接口的光盘刻录机也不成问题。

要将某目录的内容刻录成一张 Linux、Windows 和 Macintosh 系统都可读取的光盘 (注 15), 步骤如下:

#### **1. 先找出光盘刻录机的位置:**

```
$ cdrecord --scanbus
...
0,0,0 0) *
0,1,0 1) *
```

---

注 15: ISO9660 with Rock Ridge extension 是这三种系统都支持的格式。mkisofs 也能产生其他格式的 .iso 文件给 cdrecord 使用, 详情请参阅 man mkisofs.

---



```
0,2,0 2) *
0,3,0 3) 'YAMAHA ' 'CRW6416S ' '1.0d'
        Removable CD ROM
...
```

在此例中，刻录机位于 0,3,0（这是 SCSI 设备的标识符）。

2. 决定刻录速度。你应该选择刻录机与空白光盘两者之间的最小值。假设你的刻录机是 8 倍速的，但是空白光盘的质量只容许以 6 倍速刻录，则必须选择 6 倍速。
3. 将要刻录的文件集中于某目录下（比如说，*mydir*），将它们整理成你希望它们出现在光盘上的样子。*mydir* 本身不会被刻录进光盘。
4. 开始刻录：

```
$ DEVICE="0,3,0"
$ SPEED=6
$ mkisofs -R -l mydir > mydisk.iso
$ cdrecord -v dev=${DEVICE} speed=${SPEED}
  mydisk.iso
```

或者，如果你的系统够快，那么一行指令也可以解决（风险自负）：

```
$ mkisofs -R -l dir \
  | cdrecord -v dev=${DEVICE} speed=${SPEED} -
```

cdrecord 也可以刻录音乐光盘（Audio CD），不过，你有更好的选择。诸如 xcdroast（参阅 182 页“音频、视频”）之类的 GUI 程序，让你轻松完成这类工作。提醒你，xcdroast 只是提供一个 GUI 画面帮你代为操作 cdrecord 而已，实际的刻录工作仍然是由 cdrecord 负责运行。

## **rsync [options] source destination**

**rsync**

**/usr/bin**

**stdin stdout -file --opt --help --version**

rsync 可用来让分别位于不同位置的两组文件保持同步。你可以决定要同步到什么程度：只保持文件数据一致，还是连同文件属性（访问模式、拥有权限）也一并同步（这称为镜像）。两组文件不一定要在同一部机器上，也可以位于网络两端的不同机器。rsync 的用途很广泛，其选项超过 50 个，这里只能介绍少数几个有关备份的选项。

将 *D1* 目录镜像到同一机器上的 *D2* 目录下（*D1* 与 *D2* 不可以是对方的子目录）：

```
$ rsync -a D1 D2
```

若要将 *D1* 目录镜像到网络另一端的 *server.example.com* 主机上，你必须有远程主机的登录账号（假设是 *smith*），而且远程主机必须要事先启动 *sshd* server（以免通信内容外泄或遭到篡改）：

```
$ rsync -a -e ssh D1 smith@server.example.com:
```

## 常用选项

- o            维持文件拥有权（你可能需要远程主机的 *superuser* 权限）。
  - g            维持文件组拥有权（你可能需要远程主机的 *superuser* 权限）。
  - p            维持访问模式。
  - t            维持文件的时间戳。
  - r            同步范围扩及整个子目录树。
  - l            直接传输符号链接文件（而不是链接对象）。
  - D            容许传输设备文件（需要 *root* 权限）。
  - a            镜像：保持 *destination* 的所有文件属性都与 *source* 一样。相当于 *-Dgloprt*。
  - v            在同步过程中，显示当前所进行的操作。加上 *--progress* 选项可显示文件的传输进度。
  - e *command* 指定一个不同的远程 shell 程序（通常使用 *ssh* 来确保安全性）。
- 

## 19. 文件打印

- lpr*        打印文件内容。
- lpq*        查看打印队列。
- lprm*      删除队列中的打印工作。

Linux 有两套热门的打印系统：CUPS 与 LPRng。Fedora 随附的系统是 CUPS。两种系统所提供的工具程序被刻意取了同样的名称（*lpr*、*lpq* 和 *lprm*），甚至一些常用的选项也都刻意保持一致。然而，两种系统的相似度也仅止于

此，它们在实际运作机制与高级选项方面有着很大的差异。本手册仅介绍两种系统都通用的部分（同时也是大多数人只会用到的部分）。

在开始使用CUPS或LPRng之前，必须先让系统认识你的打印机。Fedora提供了一个方便而且容易使用的设置工具，可帮你产生适当的打印机配置文件：

```
# redhat-config-printer (FC1 & FC2)
```

或

```
# system-config-printer (FC3)
```

一般而言，只要依照system-config-printer的指示，应该不难完成打印机的设置。比较可能遇到困难的是USB接口的打印机，因为Linux对这类打印机的支持尚不齐全（因为某些厂商只提供Windows驱动程序，而又不愿公布技术细节）。如果有困难请查看<http://www.linuxprinting.org/>，它是个相当好的信息来源。

## **lpr [options] [files]**

**cups**

/usr/bin

stdin stdout -file --opt --help --version

lpr (line printer) 可将文件内容传送到打印机：

```
$ lpr -P myprinter myfile
```

## **常用选项**

-P *printername* 指定打印队列。*printername*是事先以system-config-printer定义的打印队列的名称（或别名）。

-# *N* 重复打印 *N* 份。

-J *name* 设定打印工作名称。*name*会被打印在封面（如果你的系统可以设定打印封面的话）。

## **lpq [options]**

**cups**

/usr/bin

stdin stdout -file --opt --help --version

lpq (line printer queue) 可列出停滞在打印队列中等待打印的工作。

## 常用选项

- P *printername* 只显示 *printername* 队列中的工作。
- a 显示所有打印队列。
- l 以较长的格式显示队列信息。

**lprm [options] [job\_IDs]**

**cups**

*/usr/bin*

**stdin stdout -file --opt --help --version**

**lprm** (line printer remove) 可取消留滞在队列里的一个或多个工作。每个打印工作都有一个专属代号, 使用 **lpq** 可查出打印工作的代号。假设你想删除 61 与 78 号工作, 命令如下:

```
$ lprm -P printername 61 78
```

如果没指定任何工作代号, 则用户当前的打印工作会被取消。一般用户只能取消自己的打印工作, 只有 **superuser** 才有权取消其他人的打印工作。-P 选项指出打印工作所留滞的队列。

---

## 20. 拼写检查

- look** 迅速查出某单词的正确拼写方式。
- aspell** 交互式的拼写检查程序。
- spell** 批量式的拼写检查程序。

Linux 内置了几组拼写检查工具。如果你已惯用 GUI 拼写检查工具, 你或许会觉得 Linux 的工具相当原始, 然而, 这些工具可用于管道中, 进而发挥许多意想不到的效果。

**look [options] prefix [dictionary\_file]**

**util-linux**

*/usr/bin*

**stdin stdout -file --opt --help --version**

**look** 是一个简单的英文字典查找工具, 你只要给它一个前缀 (prefix), 它就可从字典文件 (*/usr/share/dict/words*) 中找出具有该前缀的所有单词。例如, **look bigg** 可查出:

```
$ look bigg
Bigg
Biggs
```

如果你提供自己的字典文件，look就可从该文件中找出任何具有相关前缀的单词。任何以字母顺序排序的文本文件都可作为字典文件。

## 常用选项

- f 忽略大小写。
- t X 以X字符之前的部分 prefix 字符串（包括X字符本身）作为要查找的真正前缀。例如，look -t i big 会查出所有前缀为“bi”的单词。

<b>aspell [options] file  command</b>	<b>aspell</b>
<i>/usr/bin</i>	<b>stdin stdout -file --opt --help --version</b>

aspell 是一个有几打选项的超级检查工具。以下是少数几种典型的用法：

```
aspell -c file
```

交互式检查 *file* 中的所有单词的拼写，并选择性地更正错误。

```
aspell -l < file
```

输出 *file* 中所有拼错的单词到 stdout。

```
aspell dump master
```

将 aspell 的主字典输出到 stdout。

```
aspell help
```

显示一份精简扼要的帮助信息。在 <http://aspell.net> 上可找到更详尽的信息。

<b>spell [files]</b>	<b>aspell</b>
<i>/usr/bin</i>	<b>stdin stdout -file --opt --help --version</b>

spell 可找出文件中的所有错误拼写（依据它的字典文件），其效果相当于：

```
$ cat files | aspell -l | sort -u
```

若不指定文件，spell 从 stdin 读取输入数据。

## 21. 查看进程

- ps**            列出进程。
- uptime**       显示系统负载程度。
- w**            显示所有用户的活动中的进程。
- top**          监视进程的资源占用现况。
- xload**        以 X 窗口显示系统负载情况。
- free**         显示可用的内存空间。

进程 (process) 是 Linux 操作系统的工作单位, 你所运行的每一个程序都会引发一个或多个进程。Linux 提供了一系列可以观察并操控进程的工具。每个进程都有自己的专属代码, 称为 Process ID 或 PID。

进程不同于“6.3 任务控制”(第 31 页) 中所提到的任务 (job): 进程受操作系统内核的控制, 而任务受它们所属的 shell 控制。一个程序可能触发一个或多个进程, 而一个任务由一个 shell 命令所运行的一个或多个程序构成。

### **ps [options]**

**procps**

**/bin**

**stdin stdout -file --opt --help --version**

**ps** 可显示关于运行中进程的信息, 甚至也可显示其他用户的进程。

```
$ ps
  PID TTY          TIME CMD
 23216 pts/1        00:00:01 bash
 23219 pts/1        00:00:00 emacs
 23232 pts/1        00:00:00 ps
```

**ps** 的选项超过 80 个, 这里只列举其中少数几种最有用的组合。如果你觉得 **ps** 的选项名称似乎杂乱无章, 甚至很难与其功能联想在一起, 那是因为 GNU 版的 **ps** (Fedora 所提供的版本) 刻意集成了其他 Unix 系统的 **ps** 命令的功能, 所以同时要兼容于它们。

查看自己的进程:

```
$ ps ux
```

指定用户的所有进程:

```
$ ps -U username
```

特定程序引发的所有进程：

```
$ ps -C program_name
```

位于 *N* 号终端机的进程：

```
$ ps -tN
```

特定进程 (PID 1、2、3505)：

```
$ ps -p1,2,3505
```

所有进程，连同命令行（截掉超出屏幕宽度的部分）：

```
$ ps -cf
```

所有进程，连同完整命令行：

```
$ ps -cfww
```

呈现所有进程的层次关系（以缩排方式呈现子进程）：

```
$ ps -efH
```

由于 `ps` 所显示的信息很多，所以通常会搭配 `grep` 之类的过滤程序来筛选出特定信息。

## uptime

procps

/usr/bin

stdin stdout -file --opt --help --version

`uptime` 可告诉你系统从开机迄今已经连续运行的时间。

```
$ uptime
1:36pm up 9 days, 22:56,  2 user,  load average: 0.98,
1.04, 1.01
```

从左至右，各字段的意义分别是：当前时间 (1:36pm)、系统启动时间 (9 天 22 小时 56 分钟)、当前登录系统的人数 (2)、1 分钟内的负载均值 (0.98)、5 分钟内的负载均值 (1.04)、15 分钟内的负载均值 (1.01)。负载均值是单位时间内的活动进程的平均数量。就此例而言，可合理猜测该系统有一个至少已经连续运行了 15 分钟的进程。

w 可显示每个已登录的用户的当前进程，或者，更精确地说，应该是每位用户的每个 shell 的当前进程：

```
$ w
06:20:59 up 10 days,  5:04,  4 users,
                load average: 0.33, 0.21, 0.19
USER  TTY      FROM          LOGIN@  IDLE   JCPU   PCPU   WHAT
lin   vc/2    -             02:39   3:40m  1:08   0.02s  /bin/sh /usr/X11
lin   pts/1    :0.0         06:13   47.00s  0.82s  0.77s  ssh bill@venus
lin   pts/0    :0.0         02:41   20.00s  1.25s  0.06s  -bash
me    pts/2    dhcp183      06:20   0.00s  0.05s  0.02s  w
```

第一行是 uptime 显示的，其后是各栏信息的标题，各栏的意义分别是：登录用户的名称 (USER)、该用户所占的终端机 (TTY)、来源主机或 X 画面的编号 (FROM)、登录累积时间 (LOGIN@)、闲置时间 (IDLE)、两种 CPU 时间的测量值 (JCPU 与 PCPU，技术细节请参阅 man w) 以及当前的进程 (译注 13)。如果只想知道特定用户的进程信息，只要将该用户的名称提供给 w 即可。

若要最精简地输出信息，试试看 w -hfs。

## 常用选项

- h 不显示标题行。
- f 不显示 FROM 字段。
- s 不显示 JCPU 和 PCPU 字段。

top 可用来监视最活跃的进程，大约每隔一秒它便会自动更新一次信息。

```
44 processes: 43 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  cpu  user  nice  system  irq  softirq  iowait  idle
              total 0.0%  0.0%  1.9%   0.0%   0.0%   0.0%   98.0%
```

译注 13：应该是 shell job。



```

Mem: 256864k av, 246368k used, 10496k free, 0k shrd, 8616k buff
      14276k active, 31100k inactive
Swap: 433712k av, 468k used, 433244k free          25932k cached
PID  USER  PRI  NI  SIZE  RSS  SHARE STAT %CPU %MEM  TIME  CPU  COMMAND
3219  me    17   0   1068 1068  900   R    0.9  0.4   0:00  0    top
      1  root   8   0    432  432  384   S    0.0  0.1   0:04  0    init
      2  root   9   0      0   0  0     SW   0.0  0.0   0:00  0    keventd
...

```

top是一个交互程序,在它运行的过程中,你可用几个单键指令来改变它的行为,像设定画面更新间隔([s])、隐藏闲置进程([i])、删除某进程([k])等。按[h]可显示完整的单键指令;要结束top,按[q]。

## 常用选项

- n *N*                    更新 *N* 次画面后自动结束。
- d *N*                    每隔 *N* 秒更新一次画面。
- p*N* -p*M*...           只显示 PID *N*、*M*……进程 (最多 20 个)。
- c                       显示进程的命令行参数。
- b                       直接将信息送到 stdout 而不进行任何画面控制。使用 top  
-b -nl > outfile 可将一个画面快照存入文件。

## xload

XFree86-tools

/usr/X11R6/bin

stdin stdout -file --opt --help --version

xload是X窗口环境下的一个小程序,它能显示系统负载曲线图(Y轴代表负载程度,X轴代表时间)。

## 常用选项

- update *N*              每隔 *N* 秒更新一次曲线图 (默认值为 10 秒)。
- scale *N*                将 Y 轴均分为 *N* 等分 (画 *N*-1 条小横线)。N 的默认值是 1 (不划分)。当负载程度提高时, xload 会自动增加划分 (并改变曲线图的 Y 轴比例)。
- hl *color*               以 *color* 为比例划分线的颜色。
- label *X*                在曲线图上方显示 *X* 字符串 (默认值是你的主机名称)。

- nolabel**            曲线图上方不显示任何标题。
- jumpscroll N**    当曲线到达右边缘时，往左方卷动 *N* 个像素（默认值是半个窗口的宽度）。

## free [options]

procps

/usr/bin

stdin stdout -file --opt --help --version

free 命令可显示内存使用情况。默认的容量计算单位是千字节。

```
$ free
      total        used        free   shared  buffers   cached
Mem:   256864      246152      10712         0     11736     22576
-/+ buffers/cache:  211840      45024
Swap:   433712         468     433244
```

未被任何进程占用的剩余内存，Linux 内核会尽量将它们作为高速缓存区。所以，在上述例子中，真正可用的剩余内存空间是 45024 KB。

## 常用选项

- s N**    持续运行，每隔 *N* 秒就更新一次。
- b**      以字节为计算单位。
- m**      以兆字节为计算单位。
- t**      在最后增列加总信息。
- o**      不显示“buffers/cache”整行。

# 22. 控制进程

**kill**      传送信号到（或结束）某进程。

**nice**      以特定优先级启动程序。

**renice**    在进程运行过程中改变其优先级。

进程被启动之后，它们可以被暂停、重新开始、终结、改变优先级。第 31 页“6.3 任务控制”已经讨论过部分操作，本节补足“终结”与“改变优先级”这两项操作。

shell 内置  
/bin/kill

stdin stdout -file --opt --help --version  
stdin stdout -file --opt --help --version

kill 的作用是传送信号 (signal) 给进程。信号有很多种，不同的信号会对进程造成不同的影响。kill 默认传送的是终结信号，其他信号的作用包括挂起 (suspend)、中断 (interrupt)、销毁 (crash) 等。你只能影响自己的进程，只有 root 才能影响其他用户的进程。

```
$ kill 13243
```

上述命令会传送一个“自我了结”的信号给进程。不过，并非所有程序都会乖乖照办，有些程序就是不理睬此信号。在这种情况下，你可以试着加上 -KILL 选项：

```
$ kill -KILL 13243
```

上述手法几乎可保证一定有效，但是可能留下后遗症。因为程序是被迫结束的，所以可能会有一些资源没有还给系统，或是在磁盘上留下一些奇怪的临时文件。

如果不清楚进程的 PID，试试看 pidof：

```
$ /sbin/pidof emacs
```

或是使用 grep 过滤 ps 的输出：

```
$ ps -efww | grep emacs
```

除了在文件系统中的 /bin/kill 程序，大多数 shell 都内置了 kill 命令，不过，两者的语法和行为并不一致。还好，两者都支持下列两种用法：

```
kill -N PID  
kill -NAME PID
```

其中的 *N* 是一个代表信号的数值，而 *NAME* 是信号的简名（没有前导的“SIG”前缀）。例如，若要送出 SIGHUP 信号：

```
$ kill HUP pid
```

使用 kill -l 可列出 kill 所能传送的所有信号。/bin/kill 程序与 bash 内置的 kill 命令都支持 -l 选项，但是两者列出的信号表有很大的差异：

```

$ /bin/kill 1
HUP INT QUIT ILL ABRT FPE KILL SEGV PIPE ALRM TERM
USR1 USR2 CHLD CONT
STOP TSTP TTIN TTOU TRAP IOT BUS SYS STKFLT URG IO
POLLD CLD XCPU XFSZ
VTALRM PROF PWR WINCH UNUSED
$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
 5) SIGTRAP     6) SIGABRT     7) SIGBUS      8) SIGFPE
 9) SIGKILL    10) SIGUSR1    11) SIGSEGV    12) SIGUSR2
13) SIGPIPE    14) SIGALRM    15) SIGTERM    17) SIGCHLD
18) SIGCONT    19) SIGSTOP    20) SIGTSTP    21) SIGTTIN
22) SIGTTOU    23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO
30) SIGPWR     31) SIGSYS     33) SIGRTMIN   34) SIGRTMIN+1
35) SIGRTMIN+2 36) SIGRTMIN+3 37) SIGRTMIN+4 38) SIGRTMIN+5
39) SIGRTMIN+6 40) SIGRTMIN+7 41) SIGRTMIN+8 42) SIGRTMIN+9
43) SIGRTMIN+10 44) SIGRTMIN+11 45) SIGRTMIN+12 46) SIGRTMIN+13
47) SIGRTMIN+14 48) SIGRTMIN+15 49) SIGRTMAX-15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX

```

关于各种信号的意义请查阅 `man 7 signal`。

## ***nice [-priority] command\_line***

**coreutils**

**/bin**

**stdin stdout -file --opt --help --version**

当你启动一个需要大量系统资源（CPU 运算能力、内存等）的程序时，将对其他进程（与用户）造成排挤效应。不过，你可以使用 `nice` 来降低程序的优先级，而达到“善待”（`nice`）他人的效果。举例来说，下面的例子要求系统以优先级 7 来排序一个超大文件：

```
$ nice -7 sort VeryLargeFile > outfile
```

若没指定优先级，`nice` 默认的优先级是 10。若想知道内核默认的优先级（也就是没使用 `nice` 来运行程序时，内核默认给进程的优先级），运行一次 `nice` 而不给任何参数，它就会告诉你：

```
$ nice
0
```

如果你是 `superuser`，你可以提高优先级（降低数值）：

```
$ nice --10 （没错，这是“dash 负10”）
```

要想知道你的进程当前的 `nice` 级别，请观察下列 `ps` 命令的输出信息中的“NI”字段：

```
$ ps -o pid,user,args,nice
  PID USER      COMMAND          NI
 10249 me        -bash            0
 10393 me        jed px           0
 10394 me        ps -o pid,user,a  0
```

## ***renice priority [options] PID***

**util-linux**

*/usr/bin*

**stdin stdout -file --opt --help --version**

`nice` 可以让你以特定优先级来启动程序，而 `renice` 可改变运行中进程的优先级。下面命令将 28734 号进程的友善程度（`nice level`）提高 5 级（降低优先级）：

```
$ renice +5 p 28734
```

一般用户只能降低优先级（提高数值），只有 `superuser` 能提升优先级（降低数值）。优先级的有效范围从 -20 到 +20，但是负太多的数值应予以避免，以免干扰关键的系统进程。

### **常用选项**

- `-p pid`            只影响 `pid` 所代表的进程。你可以省略 `-p` 选项而直接指定 `pid`，例如：`renice +5 28734`。
- `-u username`      影响特定用户拥有的全部进程。

## **23. 用户与其操作环境**

- `logname`    显示你的登录名称。
- `whoami`    显示当前的有效身份。
- `id`        显示某用户的 UID 与所有 GID。
- `who`       详细列出已登录的用户。
- `users`     简短列出已登录的用户。

**finger**        显示某用户的个人信息。

**last**         列出先前有谁在何时使用了这台计算机。

**printenv**     列出你的操作环境。

你是谁？这只有系统能确定。这些稀奇古怪的工具将告诉你关于用户的种种：他们的名称、登录时间、操作环境等。

---

**logname** coreutils

---

*/usr/bin* stdin stdout -file --opt --help --version

logname 可输出你的登录名称。或许它看似无聊，但其实在 shell script 里很有用。

```
$ logname
smith
```

---

**whoami** coreutils

---

*/usr/bin* stdin stdout -file --opt --help --version

whoami 输出用户当前的有效身份。如果你曾使用过 su 或 sudo 命令，你的有效身份有可能不同于登录名称，这就是 whoami 与 logname 之间最主要的差别：

```
$ logname
smith
$ whoami
smith

$ su
Password: *****
# logname
smith
# whoami
root
```

---

**id [options] [username]** coreutils

---

*/usr/bin* stdin stdout -file --opt --help --version

每位用户都有自己专属的数值标识符，称为 *User ID* 或 UID。每个组也有自己专属的数值标识符，称为 *Group ID* 或 GID。每位用户至少隶属于某一个默认组。id 可输出用户自己的 UID 以及每一个从属组的 GID：

```
$ id
uid=500(smith) gid=500(smith) groups=500(smith),
6(disk),490(src),501(cdwrite)
```

## 常用选项

- u 只输出有效用户的 UID。
- g 只输出有效组的 GID。
- G 只输出用户所属的每个组（有效组除外）的 GID。
- n 只输出名称（用户与组），而非 UID 与 GID 数值。此选项必须结合 -u、-g 或 -G 一起使用。例如，`id -Gn` 可产生与 `groups` 命令同样的输出。
- r 只输出 UID 或 GID 数值，而非名称。必须搭配 -u、-g 或 -G 的其中之一选项一起使用。

## who [options] [filename]

coreutils

/usr/bin

stdin stdout -file --opt --help --version

`who` 命令列出每个虚拟终端机上的用户，让你知道当前有哪些人正通过哪些虚拟终端机与你共享一台计算机：

```
$ who
smith      :0                Sep  6 17:09
barrett    pts/1            Sep  6 17:10
jones      pts/2            Sep  8 20:58
jones      pts/4                Sep  3 05:11
```

通常，`who` 的数据来源是 `/var/run/utmp`。它的 `filename` 参数让你能引用其他数据文件，像 `/var/log/wtmp`（过去的登录记录）或 `/var/log/btmp`（登录失败记录）（注 16）。

## 常用选项

- H 多输出一行标题栏。
- l 对于远程登录的用户，另外显示他们的来源主机。
- u 顺便显示每位用户在其终端机上的闲置时间。

---

注 16： 必须事先设定 `syslogd`，使其将两种登录状态分开记录才行。

- T 加注每位用户的终端机属性是否为“可写”（参阅161页的“mesg”命令）。加号（+）代表可写，减号（-）则否，问号（?）表示不清楚。
- m 只显示关于自己（当前终端机上的用户）的信息。
- q 只列出用户名称以及总人数。很像users命令的效果，但是多了一项人数统计。

---

**users [filename]** **coreutils**  
 /usr/bin stdin stdout -file --opt --help --version

---

users 简略列出当前登录的用户列表。如果某人同时启动了多个 shell，该用户的名字就会出现多次。

```
$ users
barrett jones smith smith smith
```

如同 who 命令，users 的数据来源也是 /var/log/utmp，而你可提供文件名使其改用其他数据来源。

---

**finger [options] [user[@host]]** **finger**  
 /usr/bin stdin stdout -file --opt --help --version

---

finger 可显示简短的个人信息：

```
$ finger
Login   Name           Tty      Idle Login Time
smith   Sandy Smith    :0                Sep 6 17:09
barrett Daniel Barrett :pts/1 24    Sep 6 17:10
jones   Jill   Jones    :pts/2          Sep 8 20:58

$ finger me
Login: me                               Name: Bill Lin
Directory: /home/me                     shell: /bin/bash
Office: Taipei, 02-27095678              Home Phone: 02-29801234
On since Fri Jan  7 09:48 (CST) on pts/0 from localhost
Mail last read Wed Nov 10 03:09 2004 (CST)
Plan:
Mistrust first impulses; they are always right.
```

user 参数可以是本机的用户名称，也可以是远程用户（以 user@host 形式表示）。不过，在讲究个人隐私的时代，现在几乎没有主机提供 finger 服务，所以别期待远程主机一定会响应 finger 的查询。



## 常用选项

- l 以长格式显示。
- s 以简短格式显示。
- p 不显示Project和Plan项目。这两项数据分别源自用户个人的`~/.project`和`~/.plan`文件。

**last [options] [users] [tty]**

**SysVinit**

*/usr/bin*

**stdin stdout -file --opt --help --version**

last 命令可显示过去的登录记录（以时间回溯顺序列出）：

```
$ last
bill pts/3    dhcp185.oreilly. Fri Jan  7 03:55    still logged in
root vc/1                                Thu Jan  6 07:49    still logged in
lin  pts/2    :0.0          Thu Jan  6 07:48    still logged in
me   pts/2    dhcp186.oreilly. Thu Jan  6 06:20 - 07:40 (01:19)
lin  pts/1    :0.0          Thu Jan  6 06:13    still logged in
...
```

你可以指定特定用户或 tty 名称来查看特定对象的登录记录。

## 常用选项

- N 只输出最近的 *N* 行记录（*N* 必须是一个正整数）。
- i 以 IP 地址代替主机名称。
- R 不显示主机名称。
- x 连同关机与 runlevel 变换事件（例如，从单用户模式进入多用户模式）也一并列出。
- f *filename* 改用 `/var/run/utmp` 之外的其他数据来源。

**printenv [environment\_variables]**

**coreutils**

*/usr/bin*

**stdin stdout -file --opt --help --version**

printenv 输出你的 shell 所知道的每一个环境变量以及对应的变量值：

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

```
MAIL=/var/spool/mail/smith
NAME=Sandy Smith
SHELL=/bin/bash
...
```

或者，你可以要求它只输出特定变量：

```
$ printenv HOME SHELL
/home/smith
/bin/bash
```

## 24. 管理用户账号

- useradd**      开设一个新账号。
- userdel**      撤销一个旧账号。
- usermod**      修改原有账号。
- passwd**      改变某账号的密码。
- chfn**        改变某用户的个人信息。
- chsh**        改变用户默认的 shell。

在安装 Fedora Linux 的过程中，安装程序会提示你创建两个账号，一个是 root（系统管理员），另一个是普通用户（应该就是你自己）。你可以在事后另外创建新账号，让其他人也可以使用你的 Linux 系统。

创建新账号是一件轻视不得的重要工作，因为每一个账号都可能带来潜在的危机，让人侵者有机可趁。所以，每位用户都应该要有够安全、难以猜测的密码，甚至应该定期变更密码。

---

<b>useradd [options] username</b>	<b>shadow-utils</b>
<i>/usr/sbin</i>	<b>stdin stdout -file --opt --help --version</b>

---

useradd 让系统管理员可为新用户创建新账号：

```
# useradd smith
```

不过，它的默认行为不是非常有用（用 `useradd -D` 查看）。所以，在实际使用中，我们通常会明确指定所有必要选项，例如：

```
# useradd -d /home/smith -s /bin/bash -g users smith
```

## 常用选项

<code>-d dir</code>	设定用户的个人目录。
<code>-s shell</code>	设定用户的 login shell 为 <i>shell</i> 。
<code>-u uid</code>	设定用户的 UID。除非你明确知道自己在做些什么，否则应该省略此选项，接受系统默认自动分配。
<code>-g group</code>	设定用户的初始组（即默认组）。 <i>group</i> 必须是一个已存在的组名称或 GID 值。
<code>-G group1,group2,...</code>	让用户同时成为 <i>group1</i> 、 <i>group2</i> 等现有组的成员。
<code>-m</code>	自动建立新账号的操作环境，包括创建个人目录、将系统框架目录（ <i>/etc/skel</i> ）的内容复制到新建的个人目录。框架目录中含有最基本的初始文件（例如 <i>~/.bash_profile</i> ），让用户可以顺利使用新环境。如果你有其他的框架目录，可以用 <code>-k</code> 选项（ <code>-k your_preferred_directory</code> ）来指定。

---

### **userdel [-r] username**

**shadow-utils**

*/usr/sbin*

**stdin stdout -file --opt --help --version**

`userdel` 命令可让管理员用来撤销旧账号：

```
# userdel smith
```

请注意，`userdel` 不会删掉用户个人目录下的文件，除非明确以 `-r` 选项要求 `userdel` 这样做。撤销账号之前，建议你考虑以“冻结”（使用 `usermod -L`）代替“撤销”的可能性。若真的要撤销账号，记得先做好文件备份。谁能保证你不再需要使用这些文件？

---

### **usermod [options] username**

**shadow-utils**

*/usr/sbin*

**stdin stdout -file --opt --help --version**

`usermod` 可让管理员修改特定用户账号的基本设定。比如说，改变个人目录：

```
# usermod -d 'home/another smith'
```

## 常用选项

<code>-d dir</code>	将用户的个人目录改为 <i>dir</i> 。
<code>-l username</code>	将用户的登录名称改为 <i>username</i> 。不过，如果系统上有任何东西会依据原本的登录名称（例如，shell script、分发邮件的规则等），擅改登录名称可能会有严重的后遗症。此外，对于系统账号（root、daemon、wheel等），更是千万不能随意变动！
<code>-s shell</code>	将用户的 login shell 改成 <i>shell</i> 。
<code>-g group</code>	将用户的初始组（默认组）改成 <i>group</i> （ <i>group</i> 必须是现有组的名称或 GID）。
<code>-G group1,group2,...</code>	让用户成为额外组的成员。如果列出的 <i>group1</i> 、 <i>group2</i> ……组里不含用户原本的从属组，则该用户将不再隶属于那些没列名的组。
<code>-L</code>	冻结账号，让用户无法登录。
<code>-U</code>	让 <code>-L</code> 所冻结的账号重新恢复效力。

**passwd [options] [username]**

**passwd**

*/usr/bin*

*stdin stdout -file --opt --help --version*

passwd 命令可修改你自己的登录密码：

```
$ passwd
```

只有 root 有权修改别人的密码：

```
# passwd smith
```

passwd 有选项，但大部分是关于密码过期的设定。除非你的系统安全政策是“不设防”，否则不应该用到这些选项。

**chfn** (change finger) 可用于更新少数几项由系统维护的个人信息，包括真实姓名、住宅电话、办公电话、办公室位置等，也就是 **finger** 命令会显示的那些信息。一般用户只能修改自己的个人信息(所以不必提供 **username** 参数)，只有 **root** 才有权修改自己以外的其他用户的个人信息。若不指定选项，**chfn** 会提示你输入各项数据：

```
$ chfn
Password: *****
Name [Shawn Smith]: Shawn E. Smith
Office [100 Barton Hall]:
Office Phone [212-555-1212]: 212-555-1234
Home Phone []:
```

### 常用选项

- f name**        设定真实名称。
- h phone**      设定住宅电话。
- p phone**      设定办公电话。
- o office**     设定办公室位置。

**chfn** 与 **finger** 已经很少用了，目前它们的作用几乎已被 **LDAP** 所取代。

**chsh** (change shell) 让用户可以改变自己的 login shell，或是让管理员改变某用户的 login shell。直接运行而不提供任何选项时，**chsh** 会提示你输入相关信息：

```
$ chsh
Changing shell for smith.
Password: *****
New shell [/bin/bash]: /bin/tcsh
```

下次再登录系统时，就会自动以 **/bin/tcsh** 返回 login shell。请注意，新的 shell 必须被列在 **/etc/shells** 文件里。

## 常用选项

- s *shell* 指定新的 shell。
  - l 列出所有可用的 shell。
- 

## 25. 改变身份

一般用户只能修改自己拥有的文件，只有管理员——superuser 或 root，可以完整访问整台机器并且做任何事，包括在无意中毁掉系统。所以，在一般情况下，管理员应该以平常身份登录系统，只有在需要管理系统时才切换到 root 身份。而 su 就是用于改变有效身份的命令：

```
$ su -l
Password: ***** (输入 root 的密码)
# (现在成为 root 了!)
```

当然，你得知道 root 的密码（如果是你的计算机，你应该知道！），才能切换到 root 身份。在你成功取得特权身份后，shell 会将提示符从 \$ 改成 #。完成管理工作之后，只要按 **^D** 或使用 exit 命令，就可回到原本的一般身份。

su 是获得管理特权的最简单的方法，事实上，你甚至可以用它来切换到另一个普通用户的身份——如果你知道密码的话：

```
$ su -l jones
Password: ***** (输入 jones 的密码)
```

## 常用选项

- i 运行 login shell（获得新身份的操作环境）。这是你每次都会想要使用的选项，因为这样才能自动获得正确的搜索路径。
- m 让新的 shell 继承当前的环境变量。
- c *command* 以新身份运行此 *command*，然后回到原本的身份。如果你经常需要这样做，请详读 man sudo。
- s *shell* 运行指定的 shell（即 /bin/bash）。

有时候，只为了临时运行一个简单的管理命令，而用 su 切换到特权身份，然后再切换回来，似乎有点小题大做（这也太麻烦了）。此外，如果你想将部

分特权开放给某些用户（比如说，让他们可以自己挂载光驱），但是又不想让他们知道root的密码，这时候该怎么办？sudo是针对此类情况而设计的工具，它可让你以自己的密码暂时切换到另一个身份，以该身份运行指定的命令之后再自动恢复到当前的身份。不过，使用sudo之前，必须先适当设定/etc/sudoers文件。关于这部分的内容，请参考系统管理方面的书籍。

## 26. 组管理

**groups**        列出用户所属的每一个组。

**groupadd**     成立一个新组。

**groupdel**     撤销一个旧组。

**groupmod**    修改组。

组（group）代表一群用户共同的身份（权限）。如果将某权限（譬如修改某文件）赋予某组，则该组的全体成员都将获得该权限。举例来说，若你有一个文件/tmp/sample，而你想让friends组的所有成员都可读、写、执行该文件，则程序如下：

```
$ groups
users smith friends
$ chgrp friends /tmp/sample
$ chmod 770 /tmp/sample
$ ls -l /tmp/sample
-rwxrwx--- 1 smith friends 2874 Oct 20 22:35 /tmp/sample
```

只有root可通过修改/etc/group来改变组成员（注17）。若要改变某文件的组拥有权，请复习第62页的“12. 文件属性”。

<b>groups [usernames]</b>	<b>coreutils</b>
<i>/usr/bin</i>	<b>stdin stdout -file --opt --help --version</b>

groups 可显示当前用户或特定用户所属的每一个组：

```
$ whoami
smith
```

---

注17： 不同的系统可能以其他方式来存储组成员列表。

```
$ groups
smith users
$ groups jones root
jones : jones users
root : root bin daemon sys adm disk wheel src
```

---

**groupadd [options] group****shadow-utils****/usr/sbin****stdin stdout -file --opt --help --version**

groupadd命令用于建立新组。大多数情况下,你应该使用-f选项来避免设重复的组:

```
# groupadd -f friends
```

**常用选项**

-g *gid* 指定新组的GID,而不让groupadd自己选择。

-f 若指定的组名称与现有组的重复,则显示错误信息,然后结束。

---

**groupdel group****shadow-utils****/usr/sbin****stdin stdout -file --opt --help --version**

groupdel命令可撤销现有的组。

```
# groupdel friends
```

在撤销组之前,最好先找出该组拥有的所有文件,以便事后处理:

```
# find / -group friends -print
```

groupdel并不会改变任何文件的组拥有权,它只是从系统记录中删掉该组的名称而已。

---

**groupmod [options] group****shadow-utils****/usr/sbin****stdin stdout -file --opt --help --version**

groupmod命令可修改特定组的名称或GID。

```
# groupmod -n newname friends
```

groupmod不会影响该组所拥有的任何文件,它只是从系统记录中改掉组的名称或GID而已。一般而言,修改组名称没有什么大碍(通常受影响的是用



户的习惯，而非系统)，但是修改组的 GID 则可能会造成原本属于该组的文件，在事后却隶属于一个不存在的组。

## 常用选项

`-g gid`      改变组的 GID（小心为之）。

`-n name`     将组的名称改成 *name*。

## 27. 基本的主机信息

`uname`            显示基本的系统信息。

`hostname`        显示或修改系统的主机名称。

`dnsdomainname`   如同 `hostname -d`。

`domainname`      如同 `hostname -y`。

`nisdomainname`   如同 `hostname -y`。

`ypdomainname`    如同 `hostname -y`。

`ifconfig`        设定或显示网络接口的信息。

每一台 Linux 机器（或“主机”，*host*）都有自己的名称、网络 IP 地址以及其他网络操作参数。本小节汇总了如何显示（或设定）这类信息的工具命令。

### **uname [options]**

**coreutils**

**/bin**

**stdin stdout -file --opt --help --version**

`uname` 命令可显示关于计算机的基本信息：

```
$ uname -a
Linux server.example.com 2.6.6-1.435.2.3 #1 Thu Jul 1
08:25:29 EDT 2004 i686 i686 i386 GNU/Linux
```

在此例中，你可见到主机名称（*server.example.com*）、Linux 内核的版本（2.6.6-1.435.2.3）、CPU 数量（#1）、内核编译日期（Thu Jul 1 08:25:29 EDT 2004）、硬件名称（i686）、CPU 型号（i686）、硬件平台（i386）以及操作系统名称（GNU/Linux）。

上述信息没有固定标准,在不同的硬件平台或内核版本中会有不同格式的信息。

## 常用选项

- a 所有信息。
- s 只显示内核名称 (默认)。
- n 只显示主机名称。
- r 只显示内核版本。
- m 只显示硬件名称。
- p 只显示 CPU 型号。
- i 只显示硬件平台。
- o 只显示操作系统名称。

### **hostname [options] [name]**

net-tools

/bin

stdin stdout -file --opt --help --version

hostname 可显示计算机在网络中的识别名称,也就是所谓的主机名称 (hostname)。这项信息与实际的网络环境配置有关,它可以显示完整主机名称 (FQDN):

```
$ hostname
myhost.example.com
```

也可以只显示简名:

```
$ hostname
myhost
```

如果你是 root, 你也可以用它设定主机名称:

```
# hostname orange
```

不过,hostname所做的改变不会持续到下次开机,除非你修改/etc/sysconfig/network 配置文件里的HOSTNAME 参数(对 RedHat 与 Fedora 系统而言)。然而,要让 Internet 认可你的主机名称,还必须要有 DNS server 提供名称解析

服务才行。这方面的内容超出本手册的范畴，所以就此打住。总而言之，不要随意乱改主机名称！

## 常用选项

- i                    显示主机的 IP 地址。
- a                    显示主机的别名。
- s                    显示主机的简名。
- f                    显示主机的完整名称 (FQDN)。
- d                    显示主机的 DNS 域名。
- y                    显示主机的 NIS 或 YP 域名。
- F *hostfile*        依据 *hostfile* 来设定主机名称。

## ifconfig interface

net-tools

/sbin

stdin stdout -file --opt --help --version

`ifconfig` 是用于观察或设定网络接口参数的工具。这方面的内容超出本手册的范畴，这里只示范几项小技巧。

查看默认网络接口（通常是 *eth0*）的信息：

```
eth1  Link encap:Ethernet  HWaddr 00:D0:B7:6C:47:20
       inet addr:192.168.1.3 Bcast:192.168.1.255 Mask:255.255.255.0
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:10305565 errors:0 dropped:0 overruns:0 frame:0
       TX packets:9518856 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:2622150752 (2500.6 Mb)  TX bytes:733967465 (699.9 Mb)
       Interrupt:18 Base address:0xef00 Memory:f3feb000-f3feb038
```

这里包含了网卡的硬件地址 (00:D0:B7:6C:47:20)、IP 地址 (192.168.1.3)、局域网的广播地址 (192.168.1.255)、网络掩码 (255.255.255.0)、启动状态与信息框大小 (UP...)、接收 (RX) 与发送 (TX) 数据量统计。

-a 选项可列出所有已启动的网络接口：

```
$ ifconfig -a
```

设定网络接口的 IP 参数，并启动该接口：

```
# ifconfig eth1 192.168.20.18 netmask 255.255.255.0
```

关闭特定网络接口：

```
# ifconfig eth1 down
```

重新启动特定网络接口：

```
# ifconfig eth1 up
```

关于细节，请参阅 `man ifconfig`。

---

## 28. 检查远程主机

**host**            查询主机名称、IP 地址以及 DNS 信息。

**whois**          查询 Internet 网域或 IP 地址的注册数据。

**ping**           检查本地与远程主机之间的通连状态。

**traceroute**    追踪本地到远程主机的网络路径。

面对远程计算机时，你或许会想要多知道些关于对方的信息：他们是谁？IP 地址是多少？在哪里？Linux 提供了一组方便的查询工具，让你对远程计算机情况有个初步的认识。

<b>host [options] name [server]</b>	<b>bind-utils</b>
<i>/usr/bin</i>	<i>stdin stdout -file --opt --help --version</i>

**host** 可从 DNS server 查出远程机器的主机名称或 IP 地址：

```
$ host www.redhat.com
www.redhat.com has address 209.132.177.50
$ host 209.132.177.50
50.177.132.209.in-addr.arpa domain name pointer www.redhat.com.
```

它还能查出更多信息：

```
$ host -a www.redhat.com
Trying "www.redhat.com"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19768
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 0
```

```
;; QUESTION SECTION:
;www.redhat.com.          IN      ANY

;; ANSWER SECTION:
www.redhat.com.          60      IN      A      209.132.177.50

;; AUTHORITY SECTION:
redhat.com.              517     IN      NS      ns3.redhat.com.
redhat.com.              517     IN      NS      ns1.redhat.com.
redhat.com.              517     IN      NS      ns2.redhat.com.
Received 102 bytes from 127.0.0.1#53 in 381 ms
```

在此例中，你可看到RedHat公司设置了三台DNS server来提供名称查询服务，而这些数据是位于168.95.1.1的DNS server所提供的。如果你不相信该服务器的权威性，你可以要求host向你所相信的DNS server查询：

```
$ host www.redhat.com ns1.redhat.com
Using domain server:
Name: ns1.redhat.com.
Address: 66.187.233.210#53
Aliases:
www.redhat.com has address 209.132.177.50
```

直接输入host而不加任何选项，可见到完整的选项列表。

## 常用选项

- a 显示所有能查出来的信息。
- t 选择要查询的资源记录类型：A、AXFR、CNAME、HINFO、KEY、MX、NS、PTR、SIG、SOA等。例如，查询redhat.com网域的邮件交换机（MX记录）：

```
$ host -t MX redhat.com
redhat.com mail is handled by 10 mx3.redhat.com.
redhat.com mail is handled by 20 mx2.redhat.com.
redhat.com mail is handled by 10 mx1.redhat.com.
```

若host还不能满足你，不妨试试dig或nslookup。这两个同样都是很好用的DNS查询工具，前者比较新，它所产生的数据格式可以直接提供给DNS server使用；而后者提供交互式的操作，虽然已经有点过时，但是在许多Linux和Unix系统中还可以找到它。

whois 可向 Whois server (多半由 TWNIC 之类的域名管理机构提供) 查询某网域的注册信息。例如:

```
$ whois redhat.com
[Querying whois.internic.net]
[Redirected to whois.networksolutions.com]
[Querying whois.networksolutions.com]
[whois.networksolutions.com]
NOTICE AND TERMS OF USE: You are not authorized to access or
... (一段繁琐的免责声明) ...
Registrant:
Red Hat, Inc. (REDHAT-DOM)
  P.O. Box 13588
  Research Triangle Park, NC 27709
  US

  Domain Name: REDHAT.COM
...
```

或者, 你可以查出 IP 地址的拥有者:

```
$ whois 208.201.239.36
[Querying whois.arin.net]
[whois.arin.net]
UUNET Technologies, Inc. UUNET1996B (NET-208-192-0-0-1)
  208.192.0.0 - 208.255.255.255
SONIC.NET, INC. UU-208-201-224 (NET-208-201-224-0-1)
  208.201.224.0 - 208.201.255.255
```

曾有一段时间, Whois 被视为泄露隐私的管道而恶名昭彰; 如今, 却因为有助于追查垃圾邮件来源, 反而大受管理人员的重视。

## 常用选项

- h registrar** 向指定的 whois server 查询数据, 例如: `whois -h whois.geektools.com`。Fedora 提供的 whois 版本很聪明, 它能自动判断应该向哪些 Whois server 查询数据, 或至少提供如何查出数据的线索。
- p portQuery** 使用指定的 TCP port (默认值是 TCP port 43)。

**ping [options] host****iputils****/bin****stdin stdout -file --opt --help --version**

ping 用于测试本机与远程主机之间的通连性。它会送出一些小封装包（更精确的说是 ICMP Echo request 封装包）到远程主机，然后等待响应。如果远程主机有响应（送回 ICMP Echo response 封装包），则表示两端之间的网络是通的。

```
$ ping google.com
PING google.com (216.239.57.99) 56(84) bytes of data.
64 bytes from 216.239.57.99: icmp_seq=0 ttl=244 time=186 ms
64 bytes from 216.239.57.99: icmp_seq=1 ttl=244 time=185 ms
64 bytes from 216.239.57.99: icmp_seq=2 ttl=244 time=185 ms
^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 200lms
rtt min/avg/max/mdev = 185.294/185.896/186.621/0.740 ms, pipe 2
```

ping 会持续反复进行“发送—接收”过程，你可按下 **^D** 来中断它，或是事先用 **-c** 选项限定测试次数。

**常用选项**

- c N**     测试 *N* 次后自动停止。
- i N**     每次测试之间等待 *N* 秒钟的间隔（默认值为 1 秒）。
- n**       不向 DNS 查询 IP 地址对应的主机名称。

**traceroute [options] host [packet\_length]****traceroute****/bin****stdin stdout -file --opt --help --version**

traceroute 是很有趣的工具，它能查出本地主机到远程主机之间的网络路径（封装包从本地到远程沿途所经过的中继站），以及各段路程所花的平均时间。

```
$ traceroute yahoo.com
 1 server.example.com (192.168.0.20) 1.397 ms 1.973ms 2.817 ms
 2 10.221.16.1 (10.221.16.1) 15.397 ms 15.973 ms 10.817 ms
 3 gbr2-pl0.cblma.ip.att.net (12.123.40.190) 11.952 ms 11.720 ms
11.705 ms
```

```
...  
16 p6.www.dcn.yahoo.com (216.109.118.69) 24.757 ms 22.659 ms *
```

对于路途中的每一段路程，tracert会检测三次，并汇报每次的往返速度。如果某一站在5秒内没有任何反应，则标示一个\*符号；连续三次都没反应，就放弃该站继续探测下一段路程。由于tracert探测过程中可能遭遇到防火墙的干扰或是遇到其他障碍，所以你可能见到如下一堆符号：

符号	意义
!F	需要将封装包分解成更小的单位 (Fragmentation needed)
!H	主机不可达 (Host unreachable)
!N	网络不可达 (Network unreachable)
!P	协议不可达 (Protocol unreachable)
!S	发送方选径失败 (Source route failed)
!X	通信被刻意阻断
!N	ICMP unreachable code <i>N</i>

默认的探路封装包大小是40 bytes，但是你可通过提供 *packet\_length* 参数来要求tracert改送其他大小的探路封装包（例如：tracert myhost 120）。

## 常用选项

- n 数值模式：只输出IP地址，不查主机名称（可节省不少测试时间！）。
- w *N* 将等待响应时间的上限改为*N*秒（默认值为5秒）。

## 29. 网络连接

- ssh 安全地登录远程主机或运行远程命令。
- telnet 登录远程主机（不安全！）。
- scp 安全地在本地与远程之间传输文件（批量操作）。



**sftp**       安全地在本地与远程之间传输文件（交互操作）。

**ftp**        在本地与远程之间传输文件（交互操作，不安全！）。

在Linux系统中，你可以轻易地建立网络连接，使你能够登录远程机器或是在两端之间传输文件。你只要确定连接方式是否安全即可。

**ssh [options] host [command]**

**openssh-clients**

*/usr/bin*

**stdin stdout -file --opt --help --version**

如果你有远程主机的 shell 账号，ssh（Secure shell）程序能让你安全地登录远程机器：

```
$ ssh remote.example.com
```

或者，不登录远程系统而直接运行远程命令，然后返回运行结果：

```
$ ssh remote.example.com who
```

ssh 会加密（encrypt）所有在网络连接中传输的数据，包括你的用户名称与密码。SSH 协议支持多种验证方法，包括公钥（public key）与主机标识符（host ID），详情请见 `man sshd`。

## 常用选项

**-l username**   以 `username` 账号登录远程系统。如不指定，ssh 使用你当前的本地账号。另一种等效语法是 `username@remotehost`，例如：

```
$ ssh smith@server.example.com
```

**-p port**        连接到远程机器的 TCP port（SSH 协议默认的服务器端口是 22）。

**-t**            要求远程系统配置一个 tty 终端机。当你运行交互式的远程程序（比如文本编辑器）时，这个选项特别有用。

**-v**            输出详尽的交互信息，适用于调试。

**telnet [options] host [port]**

**telnet**

**/usr/bin**

**stdin stdout -file --opt --help --version**

telnet 让你能登录远程机器 (如果你有远程机器的 shell 账号), 或建立 TCP 连接到远程服务器。

```
$ telnet remote.example.com
```

由于 telnet 没有加密能力, 所以你的用户名称与密码是以明文形式在网络中传输, 任何有能力拦截网络通信的人都有机会知道你的秘密。所以, 你应该尽可能以 ssh 代替 telnet, 除非是以下两种例外情况:

- 在 Kerberos 环境下, 在客户端与服务器端两边都使用加强版 (“kerberized”) 的 Telnet 软件。Fedora 提供的 telnet 能支持 Kerberos 环境。详情请参阅 <http://web.mit.edu/kerberos/>。
- 连接过程不涉及任何敏感信息。比如说, 想要检查远程机器上的 web server (位于 TCP port 80) 是否还在:

```
$ telnet remote.example.com 80
Trying 192.168.55.21...
Connected to remote.example.com (192.168.55.21) .
Escape character is '^]'.
xxx (随意输入一些无意义的字符, 然后按 Enter)
<HTML><HEAD>      (没错, 这是 web server)
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that
this server could not understand.<P>
</BODY></HTML>
Connection closed by foreign host.
```

由于我们不鼓励你继续使用 telnet, 所以就不解释它的选项了。

**scp local\_spec remote\_spec**

**openssh-clients**

**/usr/bin**

**stdin stdout -file --opt --help --version**

scp (secure copy) 让两部机器之间可安全地传输整批文件。两端之间的所有通信数据都被加密, 所以不怕泄露。scp 的用法与 cp 很像, 而且不提供交互操作模式 (如果需要交互, 请改用 sftp)。

```
$ scp myfile remote.example.com:newfile
```

```
$ scp -r mydir remote.example.com:
$ scp remote.example.com:myfile .
$ scp -r remote.example.com:mydir .
```

如果你在远程的用户名称不同于在本地的用户名称，则可以用 `username@host` 语法来表示你的远程身份：

```
$ scp myfile smith@remote.example.com:
```

描述 `remote_spec` 时，若不指定路径名称，则假设是远程账号的个人目录。

## 常用选项

- p 复制所有文件属性（权限、时间戳等）。
- r 复制整个子目录。
- v 产生详细的调试信息。

## **sftp (host | username@host)**

**openssh-clients**

**/usr/bin**

**stdin stdout -file --opt --help --version**

`sftp` 能够安全连接到远程机器，提供类似 `ftp` 的操作方式，让你在两端之间传输文件。操作示范如下：

```
$ sftp remote.example.com
Password: *****
sftp> cd MyFiles
sftp> ls
README
file1
file2
file3
sftp> get file2
Fetching /home/smith/MyFiles/file2 to file2
sftp> quit
```

若你在远程机器上的用户名称不同于在本机的用户名称，请以 `username@host` 形式的参数来描述：

```
$ sftp smith@remote.example.com
```

指令	作用
help	列出指令列表
ls	列出当前的远程目录的内容
lls	列出当前的本地目录的内容
pwd	显示当前的远程工作目录的名称
lpwd	显示当前的本地工作目录的名称
cd <i>dir</i>	改变远程工作目录
lcd <i>dir</i>	改变本地工作目录
get <i>file1</i> [ <i>file2</i> ]	下载远程的 <i>file1</i> 到本地的 <i>file2</i>
put <i>file1</i> [ <i>file2</i> ]	上传本地的 <i>file1</i> 到远程的 <i>file2</i>
mget <i>file*</i>	从远程下载多个文件（可用 * 和 ? 描述远程文件）
mput <i>file*</i>	从本地上传多个文件（可用 * 和 ? 描述本地文件）
quit	断线，然后结束 sftp

### ftp [*options*] *host*

ftp

/usr/bin

stdin stdout -file --opt --help --version

ftp (File Transfer Protocol) 可在两台已经连接好的计算机之间传输文件。它提供一个仿真 shell 的交互用户界面，但是没有加密功能，你的账号密码与文件数据是以明文形式在网络中传输，所以不安全。如果可以，应该以 sftp 代替此工具。

在 sftp 中列出的交互操作指令都可以用在 ftp 中，但是其他没列出来的指令可能会有些不同。

## 30. 电子邮件

- evolution    GUI 电子邮件客户端程序。
- mutt        文本模式的电子邮件客户端程序。
- mail        最简单的文本模式的电子邮件客户端程序。

Fedora提供了好几种电子邮件工具,包括了服务器端与客户端的。因为本手册有限的篇幅,不可能解释如何建立邮件服务器,本节将简单介绍X窗口与shell环境下分别有哪些常用的电子邮件收发工具。

观察 `/var/log/maillog` 日志文件可得知电子邮件消息的收发进度,如果你有root权限,则可利用 `mailq` 命令来列出尚未寄出邮件(仍滞留在你的机器上)。

## evolution

evolution

`/usr/bin`

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

Ximian Evolution是一个外观上有点类似Microsoft Outlook的电子邮件收发程序。在Gnome桌面环境下,你可通过主菜单的“因特网”→“电子邮件”来启动它,或是直接在终端机窗口下达 `evolution` 命令。

第一次启动Evolution时,会自动出现“Evolution配置帮助”,这是一系列能帮你设定电子邮件账号的对话框(就像Windows的“向导”一样),只要依照指示,应该不难完成设定工作。如果想要增加新的电子邮件账号或是修改现有的账号,可根据下列步骤进行:

1. 选择“工具”→“设定”,在出现的对话框中按“新增”或“编辑”按钮。
2. 新增账号时,“Evolution配置帮助”会带领你逐步完成设定。
3. 点击“编辑”按钮时,会出现一个具有6个标签页的对话窗口。其中的“身份”标签页可让你修改全名与电子邮件地址。
4. 在“接收邮件”标签页中,你可修改“服务器类型”(IMAP、POP、本地投递)。对于POP和IMAP服务器,你还必须提供服务器的主机名称或IP地址以及ISP提供的用户名称;对于本地投递,则必须提供你的本地邮箱的路径。
5. 在“发送邮件”标签页中,你可选择寄件服务器的类型:如果你的邮件是由远程服务器代为转送(比如说,统一由办公室的邮件网关传送,或是你的ISP提供转发服务),则“服务器类型”应该选择“SMTP”,并提供服务器的主机名称或IP地址。另一方面,如果你要让自己的系统直接传递邮件,则“服务器类型”应该选择“sendmail”。

6. 其余的标签页与选项可根据你的情况来决定。完成设定后，按“确定”按钮可回到 Evolution 的主窗口，现在你可以进行基本的邮件操作：

收件夹	查看新收邮件。
新增	编写新邮件。
发送 / 接收	收发新邮件。
恢复	回信给寄信人。
全部恢复	回信给寄信人与副本抄送 (cc) 名单上的每一个人。
转寄	将邮件转寄给第三方。

Evolution 的功能还不只这些，你自己体验吧！此外，Fedora Core 3 随附的 Mozilla Thunderbird 也是一个广受欢迎的电子邮件工具，其功能与 Evolution 不相上下，而且操作方法也符合一般的使用习惯，有兴趣者请自己摸索。

## mutt [options]

mutt

/usr/bin

stdin stdout -file --opt --help --version

mutt 是一个可在普通终端机（或终端机窗口）上运行的 TUI (Text User Interface) 电子邮件工具，所以，你可以通过 SSH 连接到远程机器上来运行它。mutt 的功能相当齐全，它有许多指令与选项。要启动它，只需输入：

```
$ mutt
```

当主画面出现时，它会列出你的邮箱中的所有消息（每封邮件一行）。表 4 是 mutt 主画面的按键指令。

表 4：mutt 主画面的按键指令



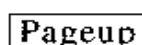
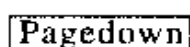
按键	作用
	移到前一封邮件
	移到下一封邮件
	显示前一页邮件列表
	显示下一页邮件列表

表 4: mutt 主画面的按键指令 (续)

按键	作用
<b>Home</b>	移到第一封邮件
<b>Ende</b>	移到最后一封邮件
<b>m</b>	编写新邮件。这会调用你的默认文本编辑器程序 (Editor 环境变量所指的编辑器), 当你编写完邮件并结束文本编辑工具之后, 按 <b>y</b> 可发出邮件, 按 <b>q</b> 可暂存新邮件
<b>r</b>	回复当前邮件。操作过程类似 <b>m</b>
<b>f</b>	转寄当前邮件给第三方。操作过程类似 <b>m</b>
<b>i</b>	当你查看个别邮件时, 此按键让你回到邮箱列表
<b>C</b>	将当前邮件复制到另一个邮箱
<b>d</b>	删除当前邮件

编写新邮件时, 在你离开文本编辑程序之后, 可使用如表 5 中所示的指令来修改邮件:

表 5: 用于处理新编写邮件的 mutt 按键指令

按键	作用
<b>a</b>	添加附件 (attachment)
<b>c</b>	设定副本抄送名单 (CC list)
<b>b</b>	设定密件抄送名单 (BCC list)
<b>e</b>	再次编辑邮件内容
<b>r</b>	编辑回信地址 (Reply-To) 字段
<b>s</b>	编辑主题栏 (subject)
<b>y</b>	寄出邮件
<b>C</b>	复制邮件到文件
<b>q</b>	存储邮件, 暂不寄送

不管在何种状态下, 表 6 中所示的指令都有效。

表 6: 通用的 mutt 指令

按键	作用
<code>[?]</code>	列出所有指令 (可换到下一页, 也可离开)
<code>[^G]</code>	取消进行中的未完指令
<code>[q]</code>	结束 mutt 程序

mutt 的官方网站是 <http://www.mutt.org>, 你可在 [http://www.cs.utk.edu/~help/mail/mutt\\_starting.php](http://www.cs.utk.edu/~help/mail/mutt_starting.php) 中找到关于 mutt 的教材。

还有一个与 mutt 性质相同的工具程序是 pine, 它提供的文本菜单界面甚至比 mutt 更方便、更容易学习。

### **mail [options] recipient**

**MailX**

**/bin** **stdin stdout -file --opt --help --version**

mail 程序 (或 Mail) (注 18) 是一个简易的寄件程序, 一般情况下, 你不会想要使用它, 但是当你设计 shell script 时, mail 可让你只用一个命令就寄出邮件, 相当方便。

在命令行环境下, 用 mail 寄送邮件的方法如下:

```
$ mail smith@example.com
Subject: my subject
I'm typing a message.
To end it, I type a period by itself on a line.
.
Cc: jones@example.com
$
```

由于 mail 将来自 stdin 的数据作为邮件内容, 所以, 利用管道, 只要一行命令就可送出邮件:

```
$ echo "Hello world" | mail -s "subject" smith@example.com
```

或者, 事先将长篇的邮件内容写在一个文件里, 然后传给 mail:

---

注 18: 对于较旧的 Unix 系统, mail 和 Mail 原本是不同的程序, 但是在 Linux 中, 这两者是相同的。事实上, /usr/bin/Mail 只是一个指向 /bin/mail 的符号链接而已。



```
$ mail -s "my subject" smith@example.com < filename
$ cat filename | mail -s "my subject" smith@example.com
```

使用管道与 I/O 重定向，你会发现“一行命令”（one liner）的威力真是无穷大。

## 常用选项

- `s subject`            设定邮件的主题栏。
- `-v`                    详细输出发送过程的交互信息（译注 14）。
- `-c addresses`        设定副本抄送名单。邮件地址之间用逗号隔开。
- `-b addresses`        设定密件抄送名单。邮件地址之间用逗号隔开。

## 31. 网络浏览

- `firefox`            功能强大的网络浏览器。
- `lynx`                文本模式的网络浏览器。
- `wget`                撷取网页并存入磁盘。
- `curl`                撷取网页并存入磁盘。

Linux 提供多种访问全球信息网的工具，有传统的浏览器，有基于文本的浏览器，还有撷取网页的工具。

### **firefox [options] [URL]**

**mozilla**

`/usr/bin`

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

Mozilla 曾经是 Linux 系统中最受欢迎的网络浏览器，它同时兼备了浏览功能与操作邮件的功能，但或许是功能太多了，以至于运行效率不高。Firefox 是使用与 Mozilla 相同的软件引擎（Gecko）写出来的“纯”浏览器，它没有操作邮件的功能，所以启动速度相当快；它注重安全性，而且提供许多贴心的功能，特别是自动阻挡弹出式广告窗口（pop-up ads）这项功能，更是为人称道。Firefox 可在多种平台上运行，包括 Linux、Windows、Mac OS X。

---

译注 14：使用本地的 Postfix MTA server 寄信时，`-v` 选项没有作用，因为 Postfix 的模块化结构，使得 `mail` 无法从 SMTP client 取得交互信息。

---

在Gnome桌面环境下,在主菜单的“因特网”子菜单中可以找到启动Firefox的选项;或者,你也可以在终端机窗口下,用下列方式启动Firefox:

```
$ firefox &
```

Firefox的画面应该不会让你感到陌生,MSIE的基本功能(浏览、前进、后退、书签、浏览记录等)Firefox都有,而你应该需要但MSIE却没提供的功能,像标签页浏览(按`[^T]`)、抑制广告窗口、字号缩放(`[^+]`、`[^-]`),Firefox也是一应俱全。

除了Mozilla与Firefox之外,Linux中还有许多浏览器可供选择,像Mozilla Netscape (<http://www.netscape.com>)、Opera (<http://www.opera.com>)、KDE的Konquerer (<http://www.konquerer.org>)、Gnome的Epiphany (<http://www.gnome.org>)和Galcon (<http://galeon.sourceforge.net>)。

### **lynx [options] [URL]**

**lynx**

**/usr/bin**




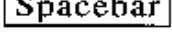













**stdin stdout -file --opt --help --version**

lynx是一个纯文本浏览器,虽然现在已经很少用了,但是当图像不重要或是网络连接速度缓慢时,lynx不失为一个应急的选择:

```
$ lynx http://www.yahoo.com
```

所有浏览动作都是通过键盘来操作的,而非鼠标。别期待lynx能够显示花哨的网页,特别是含有表格(table)或框架(frame)的网页会看起来特别不正常。尽管如此,通常还是能看出个大致端倪。

按键	作用
<code>[?]</code>	求助
<code>[k]</code>	列出所有按键与对应的作用
<code>[^G]</code>	取消运行中的命令
<code>[q]</code>	结束lynx
<code>[Enter]</code>	相当于鼠标左键:单击当前光标所在的链接,或是送出当前的窗体字段
<code>[←]</code>	回到前一页

按键	作用
	前进到下一页，或是单击当前光标所在的链接
	到新的网址 (lynx 会提示你输入 URL)
	存储、输出或寄出当前网页
	往下卷动
	往上卷动
	将光标移到下一个链接或是窗体的下一个字段
	将光标移到前一个链接或是窗体的前一个字段
	回到页首
	跳到页尾
	回到首页
	搜索网页中的文本
	将当前网页存入书签
	列出所有书签
	删除一个书签
	显示当前网页与链接的属性
	显示 HTML 源代码模式 (再按一次  可回到正常模式)

lynx 的命令行选项超过 100 个，所以它的 manpage 很值得一看。

## 常用选项

<code>-dump</code>	输出设计好的网页到 stdout (相对于 <code>-source</code> )。
<code>-source</code>	输出 HTML 源代码到 stdout (相对于 <code>-dump</code> ，类似于 <code>wget</code> 和 <code>curl</code> 命令)。
<code>-emacskeys</code>	要求 lynx 以 emacs 编辑器的按键习惯进行操作。
<code>-vikeys</code>	引用 vim (或 vi) 的按键惯例。
<code>-homepage=URL</code>	将 URL 设定为首页。

-color               以彩色文本显示网页。  
-nocolor             以单色文本显示网页。

## wget [options] URL

wget

/usr/bin

stdin stdout -file --opt --help --version

wget 可访问 URL 所指的 HTTP 或 FTP 站点，从该站点下载信息并存储到文件中或送到 stdout。wget 是一个绝佳的网页抓取工具，它能下载构成个别网页的所有文件，也能够抓取任意深度的网页组织结构。举例来说，我们可以抓取 Yahoo 的首页：

```
$ wget http://tw.yahoo.com
--20:31:42-- http://tw.yahoo.com/
      => `index.html'
Resolving tw.yahoo.com... 202.43.195.52
Connecting to tw.yahoo.com[202.43.195.52]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]

[  <-> ] 43,285      168.69K/s

20:31:42 (168.25 KB/s) - `index.html' saved [43285]
```

这会将 Yahoo 的首页存入当前工作目录下的 *index.html* 文件中。wget 提供了续传的功能，比如说，当你用 wget 下载一个超大文件，在还没下载完毕之前网络就断线了。你可以在网络恢复连接之后，以 wget -c 加上原本的 URL，从断点继续下载文件的其余部分。

另一个和 wget 同享盛名的工具是 curl，但是它的默认行为是将数据下载并写到 stdout，而不像 wget 是写入文件：

```
$ curl http://www.yahoo.com > mypage.html
```

wget 的选项超过 70 个，我们只能介绍其中少数几个最常用的选项（curl 有另一组选项，请自己参阅它的在线说明）。

## 常用选项

-i filename           从指定的文件里读出 URL，然后分别抓取它们。  
-O filename           将取得的 HTML 全部写入指定的文件，一页接着一页。

<code>-c</code>	续传模式：若前次下载过程被中断，只留下部分的下载结果，则接着从断点下载其余的部分。也就是说，假设你用 <code>wget</code> 下载一个 150 KB 的文件，但是在取得 100 KB 时中断了，则 <code>-c</code> 选项可让你接着下载其余的 50 KB，并且将后来取得的部分并入原本不完全的 100 KB 的文件中。不过， <code>wget</code> 有可能被“愚弄”，如果远程文件在你续传之前有所改变，则 <code>wget</code> 所取得的结果就是错误的。因此，只有在你确定远程文件没有改变时，才可以使用这个选项。
<code>-t N</code>	若连续尝试 $N$ 次都没法获得远程响应，则放弃。 $N=0$ 表示永不放弃。
<code>--progress=dot</code> <code>--progress=bar</code>	设定下载进度条的形状。
<code>--spider</code>	不进行任何实质下载，只检查远程网页是否存在。
<code>-r</code>	以指定的 URL 为基准点，递归撷取其下的整个目录树。
<code>-nd</code>	使用递归撷取时，不在本地复制相同的目录结构，而将所有文件都集中于当前的工作目录中。
<code>-l N</code>	使用递归撷取时，限定最多可深达 $N$ 层（默认值为 5）。
<code>-k</code>	在存储所取得的文件之前，先修改其中的 HTML 链接，使它们可直接在本地被观看。
<code>-p</code>	下载构成完整网页的所有必要文件，包括 CSS 样式表与图像文件。
<code>-L</code>	只跟随相对链接（于单一网页内），而不管绝对链接。
<code>-A pattern1,pattern2, pattern3,...</code>	接受模式：只下载匹配文件名模式的文件。 <code>pattern</code> 可含有 <code>?</code> 、 <code>*</code> 等通配符，就像 shell 一样。

`-R pattern1,pattern2,` 拒绝模式:遇到匹配名称模式的文件时,不予下载。  
`pattern3,...`

`-I pattern1,pattern2,` 目录涵盖范围:只下载位于特定目录下的文件。  
`pattern3,...`

`-X pattern1,pattern2,` 目录排除范围:位于特定目录下的文件,不予下载。  
`pattern3,...`

---

## 32. 新闻组

Usenet News 是迄今依然健在的古老在线社群。它由成千上万个新闻组 (newsgroup) 构成,各组有各自的讨论主题,每个组都像是公开的电子布告栏,每人都可以随意发表信息到特定组,也可以恢复别人发表的信息。Fedora 随附的新闻组阅读工具是 `slrn`,但是在 Internet 上还可以找到其他同类工具,诸如 `rn`、`lrn`、`tin`、`pine` 等。Mozilla 也内置了读取 Usenet News 的工具 (“窗口” → “邮件与新闻组”)。此外,Google Groups 也提供了搜索 Usenet News 的服务: <http://groups.google.com>。

很多 ISP 都提供了 NNTP 服务 (Usenet news 所用的通信协议),你可通过这些 NNTP server 来读取并发表新信息。Usenet news 采取订阅制度,第一次连接到 NNTP server 时,你必须先订阅你有兴趣的组,NNTP client 会自动将你的订阅记录存放于个人目录下的某个文件中,通常是 `~/.newsrc` 或 `~/.jnewsrc`,看你所用的 NNTP client 而定,

### **slrn [options]**

**slrn**

`/usr/bin`

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

`slrn` 是一个 Usenet 新闻阅读工具 (NNTP client)。第一次使用前,必须先设定 shell 的 `NNTPSERVER` 环境变量,指出你想要使用的新闻服务器:

```
$ export NNTPSERVER=netnews.hinet.net
```

然后产生一个新闻组文件 (只有第一次使用 `slrn` 时才需要):

```
$ slrn --create
```

然后在出现的画面中选择你想订阅的新闻组（译注15）。

以后，每当你想要阅读网络新闻时，只要输入：

```
$ slrn
```

它就会自动显示你所订阅的新闻组。表7中是可用于操作此画面的按键指令。

表7：slrn的新闻组列表的操作指令

按键指令	作用
<b>q</b>	离开 slrn
<b>j</b>	选择下一个组
<b>k</b>	选择前一个组
<b>Enter</b>	阅读所选的组
<b>p</b>	发表新信息到所选的组
<b>a</b>	订阅新组（你必须事先知道名称）
<b>a</b>	取消订阅所选的组（离开 slrn 时才会删除）。按 <b>s</b> 可重新订阅

当你按 **Enter** 进入特定新闻组时，slrn 会下载该组的最新信息的标题，并以树状结构呈现各信息的关联性。表8汇总了此页面可用的按键指令。

表8：slrn的个别组页面的操作指令

按键指令	作用
<b>q</b>	离开组，回到组列表
<b>j</b>	选择下一个主题
<b>k</b>	选择前一个主题
<b>Enter</b>	阅读所选的主题
<b>c</b>	将所有主题标示为已读。若反悔可按 <b>ESC u</b>

---

译注15：诀窍：按 **/** 然后输入组名称关键字（例如：“news.newsfan.net”），可迅速找到你要订阅的新闻组。在组名称上按 **s** 表示订阅，**U** 表示不订阅。

---

表 9 是阅读单篇文章时常用的指令。

表 9: slrn 的单篇文章页面的操作指令

按键指令	作用
<b>q</b>	结束阅读，回到个别组
<b>Spacebar</b>	下一篇文章
<b>b</b>	回到文章的前一页
<b>r</b>	以 E-mail 回复文章的作者
<b>f</b>	发表一篇追加文章
<b>p</b>	发表一篇新文章
<b>o</b>	保存文章到文件中
<b>n</b>	到下一篇未读文章
<b>p</b>	到前一篇未读文章

---

不管在哪一个页面，随时按下 **[?]** 可显示辅助页面。slrn 的指令与选项很多，多到需要写成配置文件 (`~/.slrnrc`)。这里只能介绍最基本的功能，其余的要靠自己到 `/usr/share/doc/slrn*` 和 `http://www.slrn.org` 去发掘 (译注 16)。

---

## 33. 实时消息

<b>gaim</b>	实时消息兼 IRC client。
<b>talk</b>	Linux/Unix 的聊天工具。
<b>write</b>	发送消息到另一个终端机。
<b>mesg</b>	管制 talk 和 write。
<b>tty</b>	输出你的终端机设备名称。

Linux 提供多种通信工具，其中有些可让同一台机器上的用户互相沟通 (例如 talk、write)，以及让网络两端的人们可以互相聊天 (例如 gaim)。

---

译注 16: 就我个人认为，pine 是比较符合一般操作习惯的新闻组阅读工具。

---



## **gaim [options]**

**gaim**

*/usr/bin*

**stdin stdout -file --opt --help --version**

**gaim**是一个支持多种实时消息协议的通信工具，包括AOL、MSN、Yahoo等。它同时也是一个IRC (Internet Relay Chat) client。你需要X窗口环境来运行它：

```
$ gaim &
```

如果你还没有任何IM服务的账号，你得先申请一个。比如说，在<http://www.aim.com>可申请AOL实时消息账号。取得IM账号之后，只要按下**gaim**的[Accounts]按钮，并输入账号资料（屏幕的名称与密码），接着就可以进行链接了。

### **常用选项**

- u screenname**    设定你的默认账号使用 *screenname*。
- l**                启动 **gaim**时自动登录（假设你存储过密码）。
- w [message]**    设定自己为离开（away）状态，你可以决定是否提供 *message* 给试图与你会话的人。

## **talk [user[@host]] [tty]**

**talk**

*/usr/bin*

**stdin stdout -file --opt --help --version**

早在ICQ、MSN、Yahoo、AOL问世几十年前，**talk**就已经实现了实时消息的概念，它能让登录Unix/Linux系统的两名用户在同一台机器或不同主机上，通过tty终端机（例如xterm）进行一对一交谈。**talk**原本只能用在Unix/Linux系统中，但后来也移植到其他平台上，无形中也扩展了其通信范围。使用**talk**时，你必须告诉它交谈对象是谁：

```
$ talk friend@example.com
```

如果你的交谈对象同时登录多个终端机，则必须另外指出会话对象的tty。如果连接成功，则可以见到一个分成上下两半的画面，你可看见自己输入的聊天内容，也可以同时看见对方输入的聊天内容。

---

**write user [tty]****util-linux****/usr/bin****stdin stdout -file --opt --help --version**

`write` 比 `talk` 更原始，它只能发送一段信息给登录同一台 Linux 机器的另一位用户：

```
$ write smith
Hi, how are you?
See you later.
^D
```

按下 `^D` 表示终止连接。`write` 的典型用法是通过管道发送一段消息给其他人：

```
$ echo 'Howdy!' | write smith
```

---

**mesg [yn]****SysVinit****/usr/bin****stdin stdout -file --opt --help --version**

`mesg` 可控制当前终端机是否要接受 `talk` 和 `write` 的连接。`mesg y` 表示容许，`mesg n` 表示拒绝；只有 `mesg` 而不注明 `y` 或 `n` 时，则输出当前的状态（`y` 或 `n`）。`mesg` 无法影响 `gain` 之类的实时消息工具。

```
$ mesg
is n
$ mesg y
```

---

**tty****coreutils****/usr/bin****stdin stdout -file --opt --help --version**

`tty` 显示当前 shell 所占用的终端机设备的名称。

```
$ tty
/dev/pts/4
```

## 34. 屏幕输出

`echo`      输出一段简单文本到 `stdout`。

`printf`    输出格式化文本到 `stdout`。

`yes`      重复输出同样的文本到 `stdout`。

**seq**        产生序号。

**clear**      清理屏幕或窗口。

Linux 提供一组可输出消息到 `stdout` 的工具，好让你能自言自语：

```
$ echo hello world
hello world
```

这类命令乍看之下好像很无聊，但其实都有妙用。它们对于 Linux 的学习、程序调试、shell scripts 的设计（参阅第 185 页“40. shell Script 程序设计”）有莫大的帮助。有人说，学会如何运用平凡无奇的工具来完成神奇的工作，才算熟练掌握了 Linux 的精髓。

### **echo [options] strings**

**bash**

shell 内置或 `/bin/echo`

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

`echo` 唯一的作用是输出它的每一个参数：

```
$ echo We are having fun
We are having fun
```

不幸地，Linux 环境下有两个 `echo`：一个是 `bash` 本身的内置命令，另一个是 `/bin/echo`，两者的行为略有差异。通常你运行的版本应该是 `bash` 的内置命令，使用 `type echo` 可判断你实际使用的版本。

### 常用选项

- n**    输出信息字符串后，不额外输出一个 `\n` 字符。
- e**    辨识并解释转义字符序列（Escape character sequence）。例如，`echo 'hello\a'` 原本会输出“hello\a”字符串，但是 `echo -e 'hello\a'` 则会输出“hello”字符串，然后发出“哔”一声响。
- E**    不解释转义字符序列（和 `-e` 相反）。

可用的转义字符序列如下：

- \a**        警告声（Alert）（“哔”一声）。
- \b**        回删（Backspace）
- \c**        不输出信息末端的 `\n` 字符（与 `-n` 等效）。

\f 换页 (Form feed)。  
\n 换行 (Line feed, newline)。  
\r 回车 (Carriage return)。  
\t 水平定位 (Horizontal tab)。  
\v 垂直定位 (Vertical tab)。  
\ 反斜线 (backslash)。  
\' 单引号。  
\" 双引号。  
\nnn ASCII 八进制编码值为 *nnn* 的字符。

## **printf template [args]**

**bash**

shell 内置

stdin stdout -file --opt --help --version

printf 能输出格式化字符串到 stdout，可以说是 echo 的加强版。printf 的用法很像 C 语言程序中的 printf() 函数，两者同样都能将各参数依次代入格式化字符串中的格式符中，从而产生新字符串。例如：

```
$ printf "User %s is %d years old.\n" sandy 29
User sandy is 29 years old.
```

第一个参数是格式化字符串，其中含有两个格式符：%s 和 %d；后续的两个参数，“sandy”和“29”，会被依次代入那两个格式符所在的位置，而成为你见到的输出字符串。或许你会觉得这有点像 m4 的宏处理效果，但是当你看到 printf 的浮点数处理能力时，或许就不会这么说了：

```
$ printf "That'll be $%.2f, sir.\n" 3
That'll be $3.00, sir.
```

其实，Linux 系统中有两个 printf 命令：一个内置于 bash shell，另一个是 /usr/bin/printf。这两个版本几乎一样，唯一的差异是 bash 内置的版本额外支持 %q 这个格式符，此符号会将输入字符串的特殊字符改成转义序列，使输出字符串可被顺利作为 shell 的输入。请留心其中的差异：

```
$ printf "This is a quote: %s\n" "\""
This is a quote: "
```

```
$ printf "This is a quote: %q\n" "\""  
This is a quote: \"
```

格式化字符串中有多少个格式符，就必须提供多少个代入参数。如果你给的参数太多，printf会忽略掉多余参数；反之，若给的参数太少，则printf会自行代入默认值（0代入数值格式符，“”代入字符串格式符）。毋庸置疑，即使printf可以自行处理，你也应该自己避免这类错误。如果shell script里出现这种错误，肯定会有一群难以发现的缺陷等着你。

详细的格式符说明，请参阅man 3 printf。这里只汇总了其中比较常用的格式符：

%d	十进制整数。
%ld	十进制长整数。
%o	八进制整数。
%x	十六进制整数。
%f	浮点数。
%lf	双精度浮点数。
%c	单个字符。
%s	字符串。
%q	任何shell元字符皆前置\符号的字符串。
%%	百分比符号本身。

在百分比符号之后格式字符之前，可以插入一段描述输出字符串长度的数值。例如，“%5d”表示输出长度为5的十进制数，“%6.2f”表示浮点数总长度为6个字符，其中小数点后面占两位。以下是一些常用的数值格式：

$n$  长度至少为  $n$ 。

$0n$  长度至少为  $n$ ，不足的位补0（例如0045）。

$n.m$  长度至少为  $n$ ，小数点后面占  $m$  位。

printf也能解释“\n”（换行）、“\a”（哔！）之类的转义字符（参阅echo命令）。

**yes [string]****coreutils****/usr/bin****stdin stdout -file --opt --help --version**

`yes` 命令持续输出给定的字符串（默认字符串为“y”），每行显示一个字符串。

```
$ yes again
again
again
again
...
```

乍看之下，这个命令似乎没有用，但其实 shell script 里时常利用 `yes` 来将交互式命令转变为批处理命令。假设有个交互程序会用“Are you SURE you want to do that”这样的问题来烦你，你可以将 `yes` 的输出导入该程序，让它自动代替你回答所有问题：

```
$ yes | my_interactive_command
```

当 `my_interactive_command` 结束时，`yes` 就跟着结束了。

**seq [options] specification****coreutils****/usr/bin****stdin stdout -file --opt --help --version**

`seq` 可产生一系列有固定间隔的整数或实数，适合用管道传送序号到其他的程序中。`seq` 有三种参数形式：

单一数值：上限

`seq` 从 1 开始数到上限值：

```
$ seq 3
1
2
3
```

两个数值：下限与上限

`seq` 从第一个数值正数或反数到第二个数值：

```
$ seq 5 2
5
4
3
2
```

```
$ seq 4 7
4
5
6
7
```

三个数值：下限、间隔、上限

`seq`从下限（第一个数值）开始正数，每次累加一个间隔值（第二个数值），直到不超过上限（第三个数值）：

```
$ seq 1 .3 2
1
1.3
1.6
1.9
```

## 常用选项

`-w` 输出足够多的前导0，让所有输出行都等宽：

```
$ seq -w 8 10
08
09
10
```

`-f format_string` 应用类似`printf`的格式化字符串。格式中至少要含有一个`%g`（默认的）、`%e`或`%f`：

```
$ seq -f '**%g**' 3
**1**
**2**
**3**
```

`-s string` 以指定的字符串为数值之间的分隔符。默认的字符是`\n`（所以每行一个数值）：

```
$ seq -s ':' 10
1:2:3:4:5:6:7:8:9:10
```

## clear

**neurses**

`/usr/bin`

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

此命令会清干净当前的 shell 画面或终端机窗口中的所有内容，然后将 shell 提示符移到画面顶端。

## 35. 算术运算

**xcalc** 图形计算器。

**expr** 在命令行进行简单计算。

**dc** 文本计算器。

需要计算器吗？Linux提供了你熟悉的图形计算器，也提供了在命令行环境能使用的运算工具。

### **xcalc [options]**

**XFree86-tools**

*/usr/X11R6/bin*

**stdin stdout -file --opt --help --version**

**xcalc** 是一个 X 窗口环境下的图形计算器，它提供四则运算、对数与指数运算、阶乘、三角函数、统计函数、径度量与度度量转换。如果你偏好reverse-polish notation (RPN) 计算器，**-rpn** 选项可如你所愿。

### **expr expression**

**coreutils**

*/usr/bin*

**stdin stdout -file --opt --help --version**

**expr** 是命令行工具，它提供简单的四则运算、统计字符串长度、比大小、逻辑（布尔）运算能力。

```
$ expr 7 + 3
10
$ expr '(' 7 + 3 ')' '*' 14    (shell 的特殊字符要以单引号标示)
140
$ expr length ABCDEFG
7
$ expr 15 '>' 16
0
```

使用**expr**时，比较麻烦的是每个参数之间都要有空格，而且任何对shell有特殊意义的字符（<>、()、\*）都必须以一对单引号标示。所以，在前述例子中，用来改变计算顺序的圆括号，也必须写成怪怪的样子（'(' ')'）。表10是**expr**所支持的运算符：



表 10: `expr` 的运算符

算符	数值运算	字符串运算
<code>+</code>	加法	
<code>-</code>	减法	
<code>*</code>	乘法	
<code>/</code>	除法	
<code>%</code>	余数	
<code>&lt;</code>	小于	字母顺序在前
<code>&lt;=</code>	小于等于	字母顺序在前或相等
<code>&gt;</code>	大于	字母顺序在后
<code>&gt;=</code>	大于等于	字母顺序在后或相等
<code>=</code>	等于	等于
<code>!=</code>	不等于	不等于
<code> </code>	逻辑“或”	逻辑“或”
<code>&amp;</code>	逻辑“且”	逻辑“且”
<code>s : regexp</code>		检查字符串 <i>s</i> 的模式是否匹配正则表达式 <i>regexp</i>
<code>substr s p n</code>		从 <i>s</i> 字符串的第 <i>p</i> 个字符开始输出 <i>n</i> 个字符 ( <i>p</i> =1 表示第一个字符)。
<code>index s chars</code>		找出 <i>chars</i> 在字符串 <i>s</i> 里首次出现的位置。0 表示没出现。此运算符的作用与 C 的 <code>index()</code> 函数一样。

对于逻辑表达式，数字 0 与空字符串皆被视为 `false`，除此之外的其他值一律被视为 `true`。对于逻辑结果，0 视为 `false`，1 视为 `true`。

老实说，`expr` 并不是非常有效率，对于较复杂的需求，建议考虑改用 Perl 或 Python 之类的语言。

**dc** (desk calculator) 是一个采用 RPN (reverse-polish notation) 语法的基于堆栈的交互式计算器 (译注 17)，它从 **stdin** 读入 RPN 运算表达式，将计算结果写到 **stdout**。如果你曾用过惠普公司 (Hewlett-Packard) 的 RPN 计算器，应该不难学会 **dc** 的用法。但如果你已习惯传统计算器，而且没有见过 RPN 语法，可能会觉得 **dc** 像是来自外星的玩意儿。既然如此，这里只介绍 **dc** 的最基本的用法。

堆栈与计算器操作：

- q**      结束 **dc**。
- f**      输出整个堆栈。
- c**      删除 (清空) 整个堆栈。
- p**      输出堆栈最顶端的值。
- P**      弹出 (pop, 或移除) 堆栈最顶端的值。
- nk**    设定后续运算要精确到小数点后面的第 *n* 位 (默认值是 0, 即整数运算)。

双目运算 (涉及两个数值的运算) 会从堆栈顶端弹出两个值，然后将计算结果推回堆栈：

- +**      加法。
- 减法。
- \***      乘法。
- /**      除法。
- %**      余数。
- ^**      指数 (栈顶为指数，其次为基数)。

---

译注 17: 给没有学过数据结构课程的读者: RPN 是一种“先算子后算符”语法的表达式; 堆栈是一种“先进后出” (FILO) 的数据结构, 可用于估算 RPN 表达式的结果。

---

单目运算（只涉及一个数值的运算）会弹出堆栈顶端的值，将计算结果推回堆栈：

v     开方。

范例：

```
$ dc
4 5 + p      (计算 4 + 5)
9
2 3 ^ p      (计算 23)
8
10 * p       (堆栈顶端的值 × 10)
80
f            (输出整个堆栈)
80
9
+p           (弹出堆栈顶端的两个值，输出它们的总和)
89
```

---

## 36. 日期与时间

xclock     图形时钟。

cal        输出月历。

date       显示或设定日期与时间。

ntpdate    使用远程的 NTP server 来设定系统时间。

想约会？来点乐子如何？（译注18）试试这些能显示或设定系统日期的工具。

**xclock [options]**

**XFree86-tools**

*/usr/X11R6/bin*

**stdin stdout -file --opt --help --version**

xclock 是一个可在 X 环境下运行的简单时钟。如果你偏好其他模式，你可以选择其他的，诸如 oclock（圆形时钟）、t3d（立体弹跳球，位于搜索路径之外的 */usr/X11R6/lib/xscreensaver/t3d*）以及 GNOME 和 KDE 面板上所显示的小时钟。

---

译注 18：这两句是双关语，原文是：“Need a date? How about a good time?”。

## 常用选项

- analog 有刻度的指针时钟（默认模式）。
- digital [-brief] 能显示日期与时间的数字钟。加上 -brief 只显示时间。
- update N 每隔 N 秒重新显示一次时间。

**cal [options] [month [year]]**

**util-linux**

**/usr/bin**

**stdin stdout -file --opt --help --version**

cal 可显示任何年月的月历。不加任何参数时，它输出当月的月历：

```
$ cal
    January 2005
Su Mo Tu We Th Fr Sa
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

或者，输出某年某月的月历：

```
$ cal 9 1970
    September 1970
Su Mo Tu We Th Fr Sa
                1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

如果省略月份，则输出一整年的年历。

## 常用选项

- y 输出今年的年历。
- m 以星期一为每周的第一天方式显示本月的月历。
- j 以“当年第几天”来显示日期。例如，2月1日是32，9月21日是265，依此类推。

**date** 能显示当前的本地时间，也可以设定系统时间：

```
$ date
Sun Sep 28 21:01:25 EDT 2003
```

**date** 所显示的日期格式，随系统当前的 **LANG** 环境变量而异。此外，你可另外提供一个格式化字符串，要求 **date** 以你指定的格式来显示日期：

```
$ date '+%D'
09/28/03
$ date '-The time is %l:%M %p on a beautiful %A in %B'
The time is 9:01 PM on a beautiful Sunday in September
```

格式化字符串的第一个字符必须是“+”，可用于字符串中的格式符号，见下表。

**格式符号      意义**

**范例**

**完整日期与时间：**

<b>%c</b>	24 小时制的完整日期与时间	Sun 28 Sep 2003, 09:01:25 PM EDT
<b>%D</b>	数字日期，两位数的年份	09/28/03
<b>%x</b>	数字日期，四位数的年份	09/28/2003
<b>%T</b>	24 小时制的时间	21:01:25
<b>%X</b>	12 小时制的时间	09:01:25 PM

**单位词：**

<b>%a</b>	星期名称（简写）	Sun
<b>%A</b>	星期名称（完整）	Sunday
<b>%b</b>	月份名称（简写）	Sep
<b>%B</b>	月份名称（完整）	September
<b>%Z</b>	时区	EDT
<b>%p</b>	上午或下午	PM

格式符号	意义	范例
数字：		
%w	周日期 (0-6, 0 代表星期天)	0
%u	周日期 (1-7, 1 代表星期一)	7
%d	月日期 (前导 0)	02
%e	月日期 (前导空格)	2
%j	年日期 (前导 0)	005
%m	月份编号 (前导 0)	09
%y	两位数的年份	03
%Y	四位数的年份	2003
%M	分钟 (前导 0)	09
%S	秒钟 (前导 0)	05
%l	12 小时制的小时 (前导空格)	9
%I	12 小时制的小时 (前导 0)	09
%k	24 小时制的小时 (前导空格)	9
%H	24 小时制的小时 (前导 0)	09
%N	纳秒	737418000
%s	从 Linux 纪元 (1970/1/1 0:0:0) 起算迄今所经过的秒数	1068583983
其他：		
%n	换行	
%t	跳格	
%%	百分比符号	%

## 常用选项

<code>-d date_or_time_string</code>	以你要求的格式来显示 <i>date_or_time_string</i> 所表示的时间。
<code>-r filename</code>	以你要求的格式来显示 <i>filename</i> 文件的最后修改时间。

`-s date_or_time_string` 将系统日期与时间设定成 `date_or_time_string` 所代表的时间。只有 root 才有权限做这件事。

---

**ntpdate timeserver****ntp**`/usr/sbin``stdin stdout -file --opt --help --version`

`ntpdate` 可从远程的 NTP server 取得准确的时间值，并设定本地主机的系统时间：

```
# ntpdate stdtime.sinica.edu.tw
11 Jan 15:42:43 ntpdate[25496]: adjust time server
140.109.1.4 offset 0.051177 sec
```

众所周知，个人计算机的硬件时钟不太可靠，长期运行下来，误差几个小时是常有的事。幸好 Internet 上有 NTP server 提供准确的时间值来源，让我们可轻松设定准确时间。定期手动运行 `ntpdate` 命令不是个好办法。你应该参考下一节“37. 调度工作”的方法，利用 `cron` 机制让系统定期自动运行 `ntpdate` 命令；或者，你也可以启动 `ntpd` daemon（参阅 <http://www.ntp.org>），使其自动将系统时间维持在一定误差内，甚至提供 NTP 服务给局域网。如果你不知道哪里有 NTP server，到 Google 去搜索“public ntp time server”。

---

## 37. 调度工作

`sleep` 休眠一段时间。

`watch` 每隔一段时间重复运行某程序。

`at` 安排某工作（job）于未来时间执行一次。

`crontab` 安排工作（job）于未来时间定期执行。

Linux 提供几种调度工具，让你可要求系统定期执行一些例行性工作。这些工具可分别应付复杂程度不等的工作。

---

**sleep time\_spec****coreutils**`/bin``stdin stdout -file --opt --help --version`

`sleep` 可让系统休眠一段时间。`time_spec` 可以是一个整数（秒），也可以

---

是一个整数后面接着一个计时单位：s、m、h、d，分别代表“秒”、“分”、“时”、“日”。

```
$ sleep 5m    (休眠5分钟)
```

sleep的用途是放在一组连续命令中，用以延迟后续命令的起始运行时间：

```
$ sleep 10 && echo 'Ten seconds have passed.'  
(10 秒之后)  
Ten seconds have passed.
```

## **watch [options] command**

**procs**

**/usr/bin**

**stdin stdout -file --opt --help --version**

watch能每隔一段时间重复运行另一个命令，默认的间隔时间是1秒。被运行的命令是直接传给shell，所以任何特殊字符都必须予以界定（以单引号）或转义（前置\符号）。watch会准备一个全屏环境来显示命令结果（其实是在运行之前先clear一次屏幕），让你方便观察运行结果的变化。举例来说，watch -n 60 date可以每分钟运行一次date命令。按`^D`可结束观察。

## **常用选项**

-n seconds 设定运行时间间隔。以秒为单位。

-d 强调两次输出之间的差异。

## **at [options] time\_spec**

**at**

**/usr/bin**

**stdin stdout -file --opt --help --version**

at命令在你指定的时间运行一次shell命令：

```
$ at 7am next sunday  
at> echo Remember to go shopping | Mail smith  
at> lpr $HOME/shopping-list  
at>  
<EOT>  
job 559 at 2003-09-14 21:30
```

at所接受的time\_spec格式并没有严格要求，一般而言，它可接受下列形式：

- 时间后面跟着日期（不是日期后面接着时间）。
- 只有一个日期（会假设是当前的时间）。



- 只有一个时间（会假设今天或明天的同一时间）。
- 某特殊时间形容词，像 now、midnight 或 teatime (16:00)。
- 上述任何形式后面接着一段时差，像 “+ 3 days”。

at 能接受的日期格式没有严格规定：

“december 25 2003”、“25 december 2003”、“december 25”、“25 december”、“12/25/2003”、“25.12.2003”、“20031225”、“today”、“thursday”、“next thursday”、“next month”、“next year”……你能够想到的，大概都可以。月份名称可以简写成三个字母 (jan、feb、mar …)。时间的表示法也很随意：“8pm”、“8 pm”、“8:00pm”、“8:00 pm”、“20:00”和“2000”都是同样的意义。时差是一个“+”或“-”符号后面跟着一个空格以及一段间隔时间，例如：“+ 3 seconds”、“+ 2 weeks”、“- 1 hour”等（注19）。

如果你描述的时间不完整，at 就从当前的系统日期与时间补足遗漏的信息。所以，“next year”表示从此刻算起的一年后；“thursday”代表即将到来的星期四的此时此刻；“december 25”表示即将来临的12月25日的此时此刻；“4:30pm”是今天或明天的下午4:30（根据当前时间是在4:30pm之前或之后而定）。

at 命令直到执行时才会交给 shell 来评估，所以，通配符、变量以及任何要靠 shell 处理的东西，都是到执行时才会被展开。此外，你的当前环境（见 printenv）也会被各个工作保留，让它们在运行时，就好像是自己登录系统后亲自下命令一样。不过，命令别名不会传给 at 作为工作，所以，别把它们算在内。

用于列出 at 工作的命令是 atq (“at queue”)：

```
$ atq
559 2003-09-14 07:00 a smith
```

在 at 工作还没执行之前，都可以用 atrm (“at remove”) 予以撤销：

```
$ atrm 559
```

---

注 19： /usr/share/doc/at-\*/timespec 可查到 at 的时间格式的精确语法。

## 常用选项

`-f filename` 从文件读入命令，而不是从 `stdin`。

`-c job_number` 输出 `job_number` 所代表的工作内容到 `stdout`。

### **crontab [options] [file]**

**vixie-cron**

`/usr/bin`

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

`crontab` 命令也是安排工作于未来执行，它不同于 `at` 之处是：`crontab` 安排的是例行性工作，像“每月的第二个周末的午夜运行某命令”。任务的内容与周期性不是在 `crontab` 命令行指定，而是描述于一个文件（称为个人的“`crontab` 文件”）中。编辑此文件的命令是：

```
$ crontab -e
```

保存之后，`crontab` 会自动将你的文件安装到一个系统目录（`/var/spool/cron`）下。往后，每隔一分钟，一个称为 `cron` 的 Linux 服务进程会检查你的 `crontab` 文件，并执行其中所有到期的工作。

```
$ crontab -e
```

以默认编辑器（`$EDITOR`）编辑用户个人的 `crontab` 文件。

```
$ crontab -l
```

输出个人的 `crontab` 文件内容到 `stdout`。

```
$ crontab -r
```

删除个人的 `crontab` 文件。

```
$ crontab myfile
```

将 `myfile` 安装到系统调度目录下。

系统管理员可用 `-u username` 来处理其他用户的 `crontab` 文件。

`Crontab` 文件的内容：每一行各描述一项工作（空白行与“#”为首的注释行除外）；每一行各有六个字段，字段之间以空格隔开；前五个字段描述时间，第六个字段描述工作命令本身。

分钟

介于 0 到 59 之间的正整数。可能的格式包括：单一数值（30）、以逗号分隔的一系列整数（0,15,30,45）、时段（20-30）、一系列时段（0-15,

---

周期性工作

50-59) 或是 \* (整个时段)。此外, 时段后面可以用 “/n” 来表示每隔 n 分钟, 例如, \*/12 和 0-59/12 同样都是代表 0, 12, 24, 36, 48 (也就是每隔 12 分钟)。

#### 小时

数值范围介于 0 至 23, 语法同 “分钟” 字段。

#### 日期

数值范围介于 1 至 31, 语法同 “分钟” 字段。

#### 月份

数值范围介于 1 至 12, 语法同 “分钟” 字段。容许使用三个字母的缩写来描述月份名称 (jan、feb、mar……), 但是月份名称不能用于时段范围与序列 (jan, mar 和 jan-mar 是错误的语法)。

#### 星期

介于 0 (星期日) 到 6 (星期六) 之间的整数, 语法同 “月份” 字段。可用三个字母的缩写来描述星期名称 (sun、mon、tue……), 但是星期名称不能用于时段范围与序列 (sun, mon 和 sun-mon 是错误的语法)。

#### 构成任务的 shell 命令

你指定的 shell 命令, 会在你的登录环境下运行, 这表示 \$HOME 之类的环境变量可以用于 shell 命令中, 搜索路径也有效。不过, 尽管如此, 基于安全考虑, 同时也为了避免命令别名、shell 内置命令混淆不清, 习惯上仍以绝对路径来表示 (例如, 以 /usr/bin/who 代替 who)。如果命令运行过程中会产生任何输出信息, 则 cron 会将这些信息以邮件形式发给你。

表 11: crontab 的时间字段范例

* * * * *	每分钟
45 * * * *	每小时的第 45 分钟 (1:45、2:45 ……)
45 9 * * *	每天上午 9:45
45 9 8 * *	每月第八天的上午 9:45
45 9 8 12 *	每年的 12 月 8 日, 上午 9:45
45 9 8 dec *	每年的 12 月 8 日, 上午 9:45
45 9 * * 6	每个星期六的上午 9:45

45 9 * * sat	每个星期六的上午 9:45
45 9 * 12 6	12 月的每个星期六的上午 9:45
45 9 8 12 6	每年的 12 月 8 日与每个星期六的上午 9:45

---

## 38. 图像、屏幕保护程序

eog	显示图像文件。
gqview	显示图像文件与幻灯片。
ksnapshot	截取屏幕画面。
gimp	编辑图像文件。
gnuplot	制作图表。
xscreensaver	运行屏幕保护程序。

对于图像的查看与编辑，Linux 有一堆方便的工具以及成吨的选项。详细介绍它们只会让你倒胃口，我们的目标是提起你的兴趣，让你知道有哪些程序值得你自己去探索。

### **eog [options] [files]**

**eog****/usr/bin****stdin stdout -file --opt --help --version**

eog (Eye of Gnome) 是一个用于观看图像的 X 窗口程序，它支持多种图像文件格式，而且可以同时观看多个图像文件——不管它们的格式是否相同：

```
$ eog file1.jpg file2.gif file3.pbm
```

eog 的大部分选项很有用。就此打住，我们只想让你知道 eog 有选项，好让你能够自己探索（提示：eog --help）。

### **gqview [options] [file]**

**gqview****/usr/bin****stdin stdout -file --opt --help --version**

gqview 也是支持多种图像文件格式的看图工具，而且能自动依次播放，就像看幻灯片一样。当你启动 gqview 时，它会自动找出工作目录下的所有图

像文件，并提供一个列表，当你点击列表中的图像文件名称时，右窗格便会显示该图像文件。运用你的直觉，画面上的菜单应该难不倒你，所以，自己试试看吧！按`^q`可结束程序。

## 常用选项

- f 以全屏模式显示图像（按`v`键可在全屏与窗口模式之间转换）。
- s 模拟幻灯片的显示方式，依次显示每一个图像文件。按`s`键可在slideshow与普通模式之间切换。

### **ksnapshot [options]**

**kdegraphics**

*/usr/bin*

stdin stdout -file --opt --help --version

ksnapshot 是个相当贴心的屏幕截取工具，当你启动它时：

```
$ ksnapshot
```

它会拍摄屏幕画面，并显示一个缩图。现在，你可以将它存入图像文件中，或是再拍另一个画面。唯一需要稍微注意的是图像文件格式的选择。当你保存时，必须提供适当的扩展名，ksnapshot 才知道应该使用哪一种图像文件格式：*.jpg* 代表 JPEG 格式，*.bmp* 是 Windows bitmap，*.pbm* 是 portable bitmap，*.eps* 代表 Encapsulated Postscript，*.ico* 是 Windows icon 等。通过“Save Snapshot”→“Filter”选项，可查出 ksnapshot 支持的所有图像文件格式。

更详细的信息，可从 ksnapshot 窗口的“Help”按钮得知，或是在 shell 环境下运行 `ksnapshot --help-all`。

### **gimp [options] [files]**

**gimp**

*/usr/bin*

stdin stdout -file --opt --help --version

GIMP (GNU Image Manipulation Program) 是一个足以媲美 Adobe Photoshop 的全功能图像编辑软件包，它很复杂，但是效果令人赞叹。关于 GIMP 的详细信息，请参阅 <http://www.gimp.org>。启动 GIMP 的方法很简单：

```
$ gimp
```

若要编辑特定图像文件：

```
$ gimp filename
```

如果你认为 GIMP 太复杂了，超乎你的需求，建议你试试 xv（可从 <http://www.trilon.com/xv> 网站免费下载）：

```
$ xv myfile.jpg
```

在图像上点击鼠标右键，就会出现编辑工具菜单。

---

<b>gnuplot [options] [files]</b>	<b>gnuplot</b>
<b>/usr/bin</b>	<b>stdin stdout -file --opt --help --version</b>

---

gnuplot 是一个科学用途的程序，它能依照你定义的方程式与取样范围绘出点，将点连接成曲线。你可将绘制好的曲线图存储成绘图仪或打印机能接受的格式，例如 PostScript。

使用 gnuplot 之前，你得先学一套相当简单但是很有用的程序语言。举例来说，如果想画出  $y=x^2$  方程式当  $x$  的值在 1 至 10 之间的曲线图：

```
$ gnuplot
gnuplot> plot [1:10] x**2
```

这时，你应该会看到一个令股市投资人兴奋不已的曲线图。另一个例子如下：

```
gnuplot> plot [ -3.14:3.14] sin(x)
```

这会画出正弦函数在  $-\pi$  到  $+\pi$  之间的曲线。电子工程师时常在示波器上见到这种曲线。离开 gnuplot：

```
gnuplot> quit
```

使用类似下面的命令，可将 gnuplot 的输出存储成 Postscript 文件：

```
$ echo 'set terminal postscript; plot [1:10] x**2'
| gnuplot > output.ps
```

详情请参阅 <http://www.gnuplot.info>。

---

<b>xscreensaver</b>	<b>xscreensaver</b>
<b>/usr/X11R6/bin</b>	<b>stdin stdout -file --opt --help --version</b>

---

xscreensaver 是一个很灵活的屏幕保护系统，它含有上百个非常有趣的动画。当 xscreensaver 被启动时，它会在后台进行，而你可控制它的各种行为。

开始活动之前的等待时间。在默认情况下，Fedora的GUI桌面环境（Gnome或KDE）会在系统闲置5分钟之后，自动启动xscreensaver。如果你觉得这段时间太长或太短，你可通过主菜单进行修改。在KDE中：“Control Center” → “Appearance & Themes” → “Screen Saver”，或者在桌面空白处按鼠标右键，然后选择“Configure Desktop” → “Screen Saver”。对于GNOME环境，点击主菜单中的“Preferences” → “Screensaver”。

**屏幕锁定。**不管在GNOME还是KDE环境下，每当你想离开座位而又不希望别人看到你的屏幕时，你都可以通过主菜单的“Lock Screen”来锁定屏幕。当你回到座位后，可用自己的登录密码来解除锁定。

**命令行控制。**在命令行输入xscreensaver-demo命令，可预览许多动画，并设定你想要的运行方式，然后用xscreensaver-command来控制程序的行为：

\$ xscreensaver-command -activate	立刻黑屏
\$ xscreensaver-command -next	选择下一个动画
\$ xscreensaver-command -prev	选择前一个动画
\$ xscreensaver-command -cycle	随机循环播放
\$ xscreensaver-command -lock	立刻锁定屏幕
\$ xscreensaver-command exit	结束

---

## 39. 音频、视频

<b>grip</b>	播放CD、抓轨、MP3 编码
<b>xmms</b>	播放音频文件（MP3、WAV）
<b>cdparanoia</b>	从CD抓轨，产生WAV文件。
<b>audacity</b>	编辑音频文件。
<b>xcdroast</b>	GUI光盘刻录工具。

Linux系统所支持的动态影像效果，确实没有Windows系统的生动，不知道这是不是Fedora没包含任何影像播放工具的原因。但我们相信，情况正在逐渐改变，因为Internet中可找到的Linux视频播放工具越来越多。有几个比较热门的工具，像mpg123 (<http://www.mpg123.de/>)、smpeg (<http://>

[www.lokigames.com/development/](http://www.lokigames.com/development/)) 以及 `mplayer` (<http://www.mplayerhq.hu>), 都有越来越成熟的趋势。

在音频方面, Linux 系统的支持算是颇为齐全, 甚至比 Windows 系统更好。某些在 Windows 系统上要付费才能获得的功能, 例如抓轨、MP3 编码器, 在 Linux 中却有现成的免费工具。本小节介绍的工具, 都有迷人的 GUI 界面, 而且功能与说明文件都很齐全, 所以这里不会谈论太多细节。如果你需要信息来源, Internet 中有些专门讨论 Linux 音频和 MIDI 的网站, 例如 <http://linux-sound.org/> 和 <http://www.xdt.com/ar/linux-snd>。

### **`grip [options]`**

**`grip`**

`/usr/bin`

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

`grip` 能够播放音乐光盘 (Audio CD), 也能够读取光盘中的音轨, 存储成 WAV 文件, 并将文件转换成 MP3 格式。它内置了详尽的辅助说明, 而且操控方式也很直截了当。

### **`cdparanoia [options] span [outfile]`**

**`cdparanoia`**

`/usr/bin`

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

`cdparanoia` 能够读取音乐光盘中的音轨, 存储成 WAV 文件或是其他格式 (参阅它的 manpage)。典型的用法是:

```
$ cdparanoia N
```

截取第  $N$  个音轨到文件。

```
$ cdparanoia -B
```

截取整张音乐光盘上的每一个音轨, 并分别存储成文件。

```
$ cdparanoia -B 2-4
```

截取第 2、3、4 个音轨到不同的文件中。

```
$ cdparanoia 2-4
```

截取第 2、3、4 个音轨到单一文件中。

如果在抓轨的时候遇到困难, 可试着用 `cdparanoia -Qvs` (详细显示搜索光盘的过程), 然后看看问题出在哪里。若想要将 WAV 文件转换成 MP3 格



式，请参考 LAME (<http://lame.sourceforge.net/>) 或 NotLame ([http://www.idiap.ch/~sanders/not\\_lame/](http://www.idiap.ch/~sanders/not_lame/))。

## **xmms [options] [files]**

**xmms**

*/usr/bin*

stdin stdout -file --opt --help --version

xmms (X MultiMedia System) 是一组优秀的音乐播放软件，它支持 MP3、WAV、Ogg Vorbis 以及其他多种音乐格式 (译注 19)。它有一个友好的 GUI 界面，让你能用类似控制 CD 音响的方式来播放音乐。最容易掌握的方法就是直接试用看看：

```
$ xmms
```

或是在命令行提供音乐文件的名称：

这里汇总了一些有用的操控动作：

动作	作用
右击标题栏	显示主菜单
单击 PL 按钮	显示播放列表 (按 “Add” 可添加文件)
单击 EQ 按钮	显示均衡器
右击播放列表中的项目	播放该项目
右击播放列表	显示播放列表菜单

## **audacity [files]**

**audacity**

*Fedora 不提供*

stdin stdout -file --opt --help --version

audacity 是一个能修改 WAV、MP3、Ogg 格式的音频文件编辑工具，它有一个很人性化的 GUI 界面，会显示音频文件的波形，让你能够剪接音频数据，并提供滤波、回音、重低音、反转等特效。Fedora 没有提供 Audacity，但是我们建议你到 <http://audacity.sourceforge.net> 下载来运行试试。

---

译注 19: Fedora Core 2 提供的 xmms 不支持 MP3 格式，但是你可从 <http://www.xmms.org> 取得最新的、没缩减的版本。

xcdroast 是一个具有 GUI 界面的 CD 刻录工具，或者应该更精确地说，xcdroast 是 cdrecord 的 GUI 封装程序，因为实际的刻录工作是由 cdrecord 来进行的，xcdroast 只是提供一个容易操作的图形界面来整合刻录参数而已。所以，cdrecord 的刻录能力如何，xcdroast 也就跟着如何。

如你所知，刻录工具分成 SCSI 与 IDE 两种形式。cdrecord 只支持 SCSI 刻录机，但是 Linux 内核提供了 *ide-scsi* 仿真模块，使得 IDE 刻录机也可以作为 SCSI 刻录机来使用。曾经有一段时间，在 Linux 系统上刻录光盘是一件颇有难度的事，甚至被认为是初学者是否有资格晋升为高手的门槛，而 xcdroast 的出现显然降低了这一门槛。

详细解释如何刻录光盘已经超出本手册的范畴，但是，如果你的光盘刻录机出现在下列命令的输出信息中：

```
$ cdrecord -scanbus
```

那基本上表示你应该不会遇到太棘手的问题；否则，建议你花点时间研究“CD-Writing Howto”这篇文章（位于 <http://www.tldp.org/HOWTO/CD-Writing-HOWTO.html>）。此外，在开始使用 xcdroast 之前，最好先到 <http://www.xcdroast.org> 逛逛，仔细阅读相关说明文件，特别是 FAQ，因为设置步骤有点复杂。就绪之后，运行：

```
$ xcdroast
```

然后单击“Setup”，并确定所有设定值都是你想要的样子。单击“Save Configuration”，然后单击“OK”就回到主画面。现在，你可以选择“Duplicate CD”或“Create CD”，然后继续操作。提醒一下，你可能需要 root 的权限才能刻录光盘，这取决于你的系统是如何设置的。

---

## 40. shell script 程序设计

先前谈到 shell (bash) 的时候，我们曾说过它内置了一套程序语言。事实上，你可写出“程序”（或是“shell script”）来完成单一命令所不能完成的工作。如同任何好的程序语言，shell 也有“变量”、“条件判断”（if-then-else）、循

环、输入、输出等元素。单单就shell script这个主题，就足够写出一本书了，所以这里只提供刚好足以入门的基本参考信息。至于完整的说明信息，请参阅 `info bash` 或“bash shell 入门”([http://www.oreilly.com.tw/chinese/linux/learn\\_bashshell.html](http://www.oreilly.com.tw/chinese/linux/learn_bashshell.html))。

## 40.1 空格与换行

bash shell script 对于空格与换行非常敏感，因为这套程序语言中的“关键字”(keyword) 其实是由 shell 估算的命令，而命令的参数之间必须以空格隔开。类似地，出现于命令中间的换行符 (`\n`) 会使得 shell 误认为命令不完整。所以，学习 shell script 的第一课就是养成正确的语法习惯，以免造成日后不必要的调试麻烦。

## 40.2 变量

变量 (variable) 让你用一个名称来代表某种意义的数值或字符串：

```
$ MYNAME="smith"
$ MYAGE=35
$ echo $MYNAME $MYAGE
smith 35
```

存储于变量中的值，其实本质上都是字符串，即使它们全部都是数字。不过，在必要时，shell 也能将这种“纯数字字符串”当成数值来处理：

```
$ NUMBER="10"
$ expr $NUMBER + 5 (+符号的左右两侧至少要有个空格)
15
```

在 shell script 里表示变量值时，最好以双引号界定，以免造成运行时错误。如果没有双引号，当 shell 遇到没定义的变量（通常是因为输错变量名称造成的）或是变量值里含有空格时，就可能引发意想不到的后果，造成 script 的行为错乱。

<pre>\$ FILENAME="My Document" \$ ls \$FILENAME ls: My: No such file or directory ls: Document: No such file or directory \$ ls -l "\$FILENAME" My Document</pre>	<p>含有空格的文件名 列出来试试 糟了！ls 见到两个参数 这样才对 ls 只见到一个参数</p>
---	--

如果变量名称与另一个字符串紧接在一起,则必须以一对花括号界定,以免发生意料外的情况:

```
$ HAT="fedora"
$ echo "The plural of $HAT is $HATs"
The plural of fedora is          糟了! 没“HATs”这个变量
$ echo "The plural of $HAT is ${HAT}s"
The plural of fedora is fedoras   这才是我们要的结果
```

## 40.3 输入与输出

Script的输出主要是由echo与printf命令提供。161页的“34. 屏幕输出”已介绍过这两个命令。

```
$ echo "Hello world"
Hello world
$ printf "I am %d years old\n" `expr 20 + 20`
I am 40 years old
```

shell script的输入主要是靠read命令来取得,它每次从stdin读入一行数据,并将其存入一个变量中:

```
$ read name
Sandy Smith
$ echo "I read the name $name"
I read the name Sandy Smith
```

## 40.4 逻辑值与返回值

程序的精华在于“条件判断”与“循环”,而这其中的关键就在于“逻辑测试”(Boolean test),也就是分辨“真”(true)与“假”(false)。对于shell,数值0代表“真”或“成功”,除此之外的其他数值一律视为“假”或“失败”。

此外,任何Linux命令结束时,都会返回一个代表运行结果的整数值给shell,此值称为“返回值”(return value)或“结束状态”(exit status)。你可用特殊变量\$?来表示返回值:

```
$ cat myfile
My name is Sandy Smith and
I really like Fedora Linux
$ grep Smith myfile
My name is Sandy Smith and          找到一处相同
```

```

$ echo $?
0
$ grep aardvark myfile
$ echo $?
1

```

所以结束状态为“成功”

没找到

所以结束状态为“失败”。

许多Linux命令的返回值具有特殊含义，各命令的manpage里通常会说明返回值的真正意义。唯一可以确定的是，返回值0一定是代表成功，因为这是所有Linux命令的共识，同时也是POSIX标准的规定。

## test 和 “[”

对于只涉及数值和字符串的逻辑表达式，可用bash shell内置的test命令来计算其逻辑值。如果计算结果为“真”，则test返回0；否则返回1：

```

$ test 10 -lt 5
$ echo $?
1
$ test -n "hello"
$ echo $?
0

```

10 < 5 ?

当然不

“hello”字符串的长度不为0吗？

没错，长度不是0

表12列出test常见的参数，它们可用于检查整数、字符串、文件的性质。

test有一个不寻常的别名：[（左方括号），以便用于条件判断与循环中。当你使用这种写法时，必须在测试语句的末端补一个“]”符号（右方括号）。下列各项测试与前例是完全等效的：

```

$ { 10 -lt 5 }
$ echo $?
1
$ [ -n "hello" ]
$ echo $?
0

```

切记，“[”是一个命令，它和任何其他命令一样，在命令名称与各个参数之间至少要保持一个空格的间隔，如果你疏忽了，将发生奇怪的事：

```

$ [ 5 -lt 4 ]
bash: [: missing ']'

```

在“4”和“]”之间没有空格

在此例中，test认为它的最后一个参数是“4]”字符串，所以向你抱怨它找不到结尾的“]”符号。

表 12: test 命令的常用参数

---

文件测试

<code>-d name</code>	测试 <i>name</i> 是否为一个目录
<code>-f name</code>	测试 <i>name</i> 是否为普通文件
<code>-L name</code>	测试 <i>name</i> 是否为符号链接
<code>-r name</code>	测试 <i>name</i> 文件是否存在且为可读
<code>-w name</code>	测试 <i>name</i> 文件是否存在且为可写
<code>-x name</code>	测试 <i>name</i> 文件是否存在且为可执行
<code>-s name</code>	测试 <i>name</i> 文件是否存在且其长度不为 0
<code>f1 -nt f2</code>	测试 <i>f1</i> 是否比 <i>f2</i> 更新
<code>f1 -ot f2</code>	测试 <i>f1</i> 是否比 <i>f2</i> 更旧

---

字符串测试

<code>s1 = s2</code>	测试两个字符串的内容是否完全一样
<code>s1 != s2</code>	测试两个字符串的内容是否有差异
<code>-z s1</code>	测试 <i>s1</i> 字符串的长度是否为 0
<code>-n s1</code>	测试 <i>s1</i> 字符串的长度是否不为 0

---

整数测试

<code>a -eq b</code>	测试 <i>a</i> 与 <i>b</i> 是否相等
<code>a -ne b</code>	测试 <i>a</i> 与 <i>b</i> 是否不相等
<code>a -gt b</code>	测试 <i>a</i> 是否大于 <i>b</i>
<code>a -ge b</code>	测试 <i>a</i> 是否大于等于 <i>b</i>
<code>a -lt b</code>	测试 <i>a</i> 是否小于 <i>b</i>
<code>a -le b</code>	测试 <i>a</i> 是否小于等于 <i>b</i>

---

组合与否定测试

<code>t1 -a t2</code>	AND (交集): 当 <i>t1</i> 与 <i>t2</i> 条件同时成立时, 才算成立
<code>t1 -o t2</code>	OR (并集): 只要 <i>t1</i> 或 <i>t2</i> 任一条件成立, 就算成立
<code>! your_test</code>	否定测试: 当 <i>your_test</i> 失败时, 则条件成立
<code>\( your_test \)</code>	改变运算顺序 (与代数一样)

---

## true 与 false

bash 内置两个与逻辑值有关的命令：true 与 false，它们唯一的作用是分别返回 0 与 1 结束状态：

```
$ true
$ echo $?
0
$ false
$ echo $?
1
```

这两个命令主要是用于条件判断与循环中。

## 40.5 条件判断

if 语句依据条件判断的结果选择执行路径。条件可能是简单或复杂的命令。最简单的 if 语句形式是 if-then：

```
if command    若 command 的结束状态为 0
then
    body       条件成立时
fi
```

范例：

```
if [ `whoami` = "root" ]
then
    echo "You are the superuser"
fi
```

另一种形式是 if-then-else 语句：

```
if command
then
    body1       条件成立时
else
    body2       条件不成立时
fi
```

范例：

```
if [ `whoami` = "root" ]
then
    echo "You are the superuser"
else
```

```

    echo "You are an ordinary dude"
fi

```

最复杂的形式是 if-then-elif-else，这可让你判断许多条件：

```

if command1
then
    body1          当 command1 成立时
elif command2
then
    body2          当 command2 成立时
elif ...
...
else
    bodyN          当所有条件都不成立时
fi

```

范例：

```

if [ `whoami` = "root" ]
then
    echo "You are the superuser"
elif [ "$USER" = "root" ]
then
    echo "You might be the superuser"
elif [ "$bribe" -gt 10000 ]
then
    echo "You can pay to be the superuser"
else
    echo "You are still an ordinary dude"
fi

```

当需要判断的条件太多，但是受测对象都一样时，if-then-elif-else就显得繁琐。这时候case语句是比较好的选择，它依据单一判断条件的结果，选择最适合执行的命令分支：

```

echo 'What would you like to do?'
read answer
case "$answer" in    受测对象为 answer 变量
eat)
    echo "OK, have a hamburger"
    ;;
sleep)
    echo "Good night then"
    ;;
*)
    echo "I'm not sure what you want to do"

```



```

    echo "I guess I'll see you tomorrow"
    ;;
esac

```

case 语句的标准语法是：

```

case string in
    expr1)
        body1
        ;;
    expr2)
        body2
        ;;
    ...
    exprN)
        bodyN
        ;;
    *)
        bodyelse
        ;;
esac

```

其中的 *string* 可以是任何值，但通常是类似 `$myvar` 这样的变量值；*expr1*、*expr2*... *exprN* 是测试结果的模式（细节请参阅 `info bash reserved case`），而最后的 “\*” 代表前述模式都不匹配时，应选择的命令分支，它相当于 `if` 语句中最后的 `else`。每一组命令集合都必须以 `;;` 结束，就像下面这样：

```

case $letter in
    X)
        echo "$letter is an X"
        ;;
    [aeiou])
        echo "$letter is a vowel"
        ;;
    [0-9])
        echo "$letter is a digit, silly"
        ;;
    *)
        echo "I cannot handle that"
        ;;
esac

```

## 40.6 循环

while循环由一个判断条件与一组命令构成,只要判断条件持续成立,就重复运行循环体内的命令。

```
while command      当 command 的结束状态为 0 时
do
    body
done
```

举例来说,若myscript script的内容是:

```
i=0
while [ $i -lt 3 ]
do
    echo "again"
    i=`expr $i + 1`
done
```

运行结果:

```
$ ./myscript
0
1
2
```

通常,while循环的主体应该包含能够改变判断条件的命令,否则会造成死循环。

until循环也是由一个判断条件与一组命令构成,但是它与while循环相反:until循环会重复运行那一组命令,直到判断条件成立为止:

```
until command      当 command 的结束状态不是 0
do
    body
done
```

范例:

```
i=0
until [ $i -gt 3 ]
do
    echo "again"
    i=`expr $i + 1`
done
```

运行结果：

```
$ ./myscript
0
1
2
```

for 循环由一个变量、一组数据（变量值）与一组命令构成，数据值会被依次代入变量，然后运行一次循环主体，直到所有数据值都被处理过为止。

```
for variable in list
do
    body
done
```

范例：

```
for name in Tom Jack Harry
do
    echo "$name is my friend"
done
```

运行结果：

```
$ ./myscript
Tom is my friend
Jack is my friend
Harry is my friend
```

for 循环特别适用于处理一系列文件。例如，当前工作目录下的特定类型文件：

```
for file in *.doc
do
    echo "$file is a stinky Microsoft Word file"
done
```

某些情况下，你或许会需要无穷循环。while 与 until 都有无穷循环的效果，你只要提供一个永远成立（或永远不成立）的条件即可：

```
while true
do
    echo "forever"
done

until false
do
```

```
    echo "forever again"
done
```

通常，你会想在无穷循环内放一个判断条件，并以 `break` 或 `exit` 来结束循环。真正的“无穷”循环其实很少见。

## 40.7 break 与 continue

`break` 命令可跳出它所在的最内层循环。假设有一个 `myscript`：

```
for name in Tom Jack Harry
do
    echo $name
    echo "again"
done
echo "all done"
```

它的运行结果原本是：

```
$ ./myscript
Tom
again
Jack
again
Harry
again
all done
```

现在加上 `break`：

```
for name in Tom Jack Harry
do
    echo $name
    if [ "$name" = "Jack" ]
    then
        break
    fi
    echo "again"
done
echo "all done"
```

看看会发生什么事：

```
$ ./myscript
Tom
again
```

```
Jack          发生了 break
all done
```

continue 迫使循环立刻跳过本回合未完成的部分，直接进入下一回合。同样以先前的 myscript 为例：

```
for name in Tom Jack Harry
do
    echo $name
    if [ "$name" = "Jack" ]
    then
        continue
    fi
    echo "again"
done
echo "all done"
```

看看会怎样：

```
$ ./myscript
Tom
again
Jack          发生 continue
Harry
again
all done
```

break 和 continue 都可以接受一个数值参数 (break N、continue N)：对于 break，N 代表要跳出多少层循环；对于 continue，则代表要略过多少回合。不过，实际中很少这样做，因为那会导致你的 script 混乱，所以我们也建议你最好尽量避免使用。

## 40.8 shell script 的制作与运行

shell script 本质上只是普通文本文件，凡是可以在 bash 提示符后输入的命令，都可以出现在 script 文件里。要运行 script，你有三种选择：

### 标准方法

将下列文本加到 script 文件的顶端（第一行靠左对齐）：

```
#!/bin/bash
```

然后改变文件访问模式，使其成为可执行文件：

```
$ chmod +x myscript
```

为了方便，你可将写好的 script 放在搜索路径中（非必要步骤）。习惯上，个人写的 script 是放在 `~/bin` 目录下，若也要给其他用户使用，则是放在 `/usr/local/bin` 目录下。放在搜索路径中的 script，可被当成普通命令来运行：

```
$ myscript
```

若 script 不是放在搜索路径中而是位于工作目录下，而且搜索路径中也没包含“.”（工作目录）（注 20），则必须在 script 名称之前加上“./”，shell 才能找到你的 script：

```
$ ./myscript
```

### 以 subshell 运行

bash 会将它的参数视为 script 文件的名称，并予以运行：

```
$ bash myscript
```

请注意，由于 script 是在 subshell 的环境里运行的，所以，script 对于环境所做的任何改变（设定 shell 变量、改变工作目录等）仅止于 subshell，而不影响 login shell。

### 以 login shell 运行

对于会影响 shell 环境的 script，应该交给当前的 shell 去运行：

```
$ . myscript
```

应该采用哪种方法，取决于 script 本身的性质。一般而言，工具性的 script 应该运行 `#!/bin/bash` 命令来保护好。至于为了应付临时工作而写的一次性 script，那就要根据是否影响 shell 环境来决定了。

## 40.9 命令行参数

shell script 也都能够接受命令行参数，就像其他 Linux 命令一样（事实上，有许多 Linux 命令本身其实就是 script。）。bash shell 提供了一系列特殊变量，让你能够在 script 里处理参数。

首先，含有所有参数的特殊变量是 `$@`，而 `$1`、`$2`、`$3` 等则代表个别参数：

---

注 20： 将工作目录纳入搜索路径确实很方便，但是基于安全考虑，Fedora 和许多 Linux distribution 都没有这么做。

```

$ cat myscript
#!/bin/bash
echo "My name is $1 and I come from $2"
echo "Your info : $@"
$ ./myscript Johnson Wisconsin
My name is Johnson and I come from Wisconsin
Your info : Johnson Wisconsin

```

很显然，script 无法预先知道用户使用给几个参数。为此，bash 提供了另一个特殊变量 `$#` 来代表参数个数：

```

if [ $# -lt 2 ]
then
    echo "$0 error: you must supply two arguments"
else
    echo "My name is $1 and I come from $2"
fi

```

特殊变量 `$0` 代表 script 自己的名称。当 script 需要显示自己的用法或错误信息时，这个变量就可以派上用场：

```

$ ./myscript Bob
./myscript error: you must supply two args

```

用一个简单的 `for` 循环搭配 `$@` 特殊变量，就可以逐一处理每一个参数，不管实际有多少个：

```

for arg in $@
do
    echo "I found the argument $arg"
done

```

## 40.10 返回结束状态

`exit` 命令可用于结束 script，并返回指定的状态码给 shell。传统上，状态码 0 代表成功，1（或任何非零值）代表失败。若 script 结束之前没调用 `exit`，则 shell 会自动假设状态码为 0。

```

if [ $# -lt 2 ]
then
    echo "Error: you must supply two args"
    exit 1
else
    echo "My name is $1 and I come from $2"
fi
exit 0

```

```
$ ./myscript Bob
./myscript error: you must supply two args
$ echo $?
1
```

## 40.11 除了 shell Scripting 之外

shell script 的用途很广泛，但毕竟不是万能的。所以，Linux 中还有许多更强的脚本语言与编译语言。

语言	程序	信息来源
Perl	perl	<a href="http://www.perl.com/">http://www.perl.com/</a>
Python	python	<a href="http://www.python.org/">http://www.python.org/</a>
C/C++	gcc	<a href="http://www.gnu.org/software/gcc/">http://www.gnu.org/software/gcc/</a>
Java	javac、java <sup>注</sup>	<a href="http://java.sun.com/">http://java.sun.com/</a>
FORTRAN	g77	<a href="http://www.gnu.org/software/fortran/fortran.html">http://www.gnu.org/software/fortran/fortran.html</a>
Ada	gnat	<a href="http://www.gnu.org/software/gnat/gnat.html">http://www.gnu.org/software/gnat/gnat.html</a>

注： 必须另外安装，Fedora 与大多数 Linux distribution 都没随附。

## 后记

虽然本手册已经涵盖了 Linux 的许多命令与功能，但是这些都只是皮毛而已，Fedora 与其他 Linux 包所提供的程序超过上千个！我们鼓励你自己去探索，持续学习 Linux 系统所带来的强大功能。祝好运！

## 致谢

衷心感谢我的编辑 Mike Loukides，他是 O'Reilly 编辑群中的传奇人物。我也要感谢技术校阅人员，他们是 Ron Bellomo、Wesley Crossman、David Debonnaire、Tim Greer、Jacob Heider 和 Eric van Oorschot，以及 VistaPrint 的 Alex Schowtka 与 Robert Dulaney。最后，感激我的家庭成员 Lisa 和 Sophie。