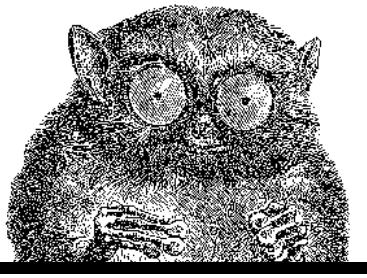


UNIX 技术手册

A Desktop Quick Reference for SVR4 and Solaris 7

第三版

Arnold Robbins 著

张龙卿 欧洋

张令军 张占军 译

O'REILLY®

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

O'Reilly & Associates, Inc. 授权中国电力出版社出版

中国电力出版社

图书在版编目（CIP）数据

UNIX 技术手册（第三版）/（美）罗宾斯（Robbins，A.）著；张龙卿等译．
－北京：中国电力出版社，2002.7
书名原文：UNIX in a Nutshell, Third Edition
ISBN 7-5083-0982-0

I. U... II.①罗 ...②张 ... III.UNIX 操作系统－技术手册 IV. TP316.81-62
中国版本图书馆 CIP 数据核字（2002）第 036838 号

北京市版权局著作权合同登记
图字：01-2002-2275 号

书　　名 / UNIX 技术手册（第三版）
书　　号 / ISBN 7-5083-0982-0
责任编辑 / 宋宏
封面设计 / Edie Freedman，张健
出版发行 / 中国电力出版社（www.infopower.com.cn）
地　　址 / 北京三里河路 6 号（邮政编码 100044）
经　　销 / 全国新华书店
印　　刷 / 北京市地矿印刷厂
开　　本 / 787 毫米×1092 毫米　16 开本　41.5 印张　614 千字
版　　次 / 2002 年 11 月第一版　2002 年 11 月第一次印刷
印　　数 / 0001-5000 册
定　　价 / 69.00 元（册）

作者简介

生于亚特兰大的**Arnold Robbins**是一名专业程序员及技术书籍的作者，他也是一位幸福的丈夫，是四个非常伶俐的孩子的父亲，还是一位业余的犹太法典编著者（负责巴比伦和耶路撒冷部分）。从1997年开始，他与他的家人一起幸福地居住在以色列。

Arnold从1980年开始使用Unix操作系统，当时他使用的是运行着第6版Unix系统的PDP-11。从1987年开始，他已经成为一名忠实的*awk*用户，并且开始使用*gawk*，即GNU项目的*awk*版本。作为POSIX 1003.2主席团的一名成员，他帮助制定了*awk*的POSIX标准。目前，他还负责维护*gawk*及其文档，这些文档可以从自由软件基金会获得（*http://www.gnu.org*），并且已经由SSC（*http://www.ssc.com*）出版，书名是《Effective AWK Programming》。

O'Reilly一直在向他约稿，他是《sed & awk》第二版的作者之一，也是《Learning the vi Editor》一书第6版的作者之一。

封面介绍

本书封面上的动物是眼镜猴，它是与狐猴有血缘关系的夜行哺乳动物。它的属名Tarsius来源于它极长的脚踝骨tarsus一词。眼镜猴居住在东印度群岛的丛林中，分布在印尼苏门答腊岛到菲律宾及印尼苏拉威西岛等地区，它们居住在树上，可以迅速敏捷地在树枝间跳来跳去。

眼镜猴是一种小动物，其身长只有6英寸，但是后面却拖着一条10英寸长的多毛的尾巴。它有着丝般的淡褐色或灰色毛发，圆形的脸，大大的眼睛。其前臂和后腿非常细长，爪的前端呈圆形，多肉的爪心可以很容易地抓住树枝。眼镜猴只有在夜晚才出来行动，白天它会藏在藤蔓植物的乱枝中或大树的顶上，主要以捕食昆虫为生。虽然它们是一种很好奇的动物，但是喜欢独来独往。

O'Reilly & Associates 公司介绍

为了满足读者对网络和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 & Associates公司授权中国电力出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly & Associates 公司是世界上在 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 & Associates 一直处于Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly & Associates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 & Associates公司具有深厚的计算机专业背景，这使得O'Reilly & Associates形成了一个非常不同于其他出版商的出版方针。O'Reilly & Associates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 & Associates 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 & Associates 依靠他们及时地推出图书。因为O'Reilly & Associates紧密地与计算机业界联系着，所以O'Reilly & Associates 知道市场上真正需要什么图书。

前言

本书第三版遵循这样的原则：“ 如果没有突破，就不进行修改 ”。该版本具有以下新特点：

- 改正了许多内容及印刷错误。
- 讲述了 Solaris 7，即 Sun 公司的基于 SVR4 操作系统的较新版本（注 1）。
- 添加了 60 个新命令，主要在第二章中讲述。
- 第四章现在讲述了 ksh 的 1988 和 1993 两个版本。
- 第七章现在讲述了 GNU `emacs` 的第 20 版本。
- 第十六章是新增加的一章，主要讲述了 `troff` *man* 宏。
- 从第十三章到十六章讲述了 `troff` 宏包，并使用简单的示例说明了使用宏的顺序。
- 第十七章现在讲述了 `refer` 及其相关的程序。
- 第十九章现在讲述了 RCS 的版本 5.7。
- SVR4 或 Solaris 中不常用的一些命令都移到附录二中讲述。
- 参考文献中列出了每个 Unix 专业人士应该参考的一些书籍。在正文中引用的所有书籍都列在了这里。

注 1： 本书所用版本基于 Intel x86 系统。

本书适合的读者

本书适合于Unix用户及Unix程序员，以及可能直接向用户及程序员提供技术支持的任何人（如系统管理员）。本书最适合那些对Unix系统已经熟悉的人员，这些人知道自己希望做什么事情，甚至知道如何去做，只需要对一些细节进行提示。例如，如果你希望从一个数据库中删除第三个字段，你可能会想：“我知道应该使用 `cut` 命令，但是该命令有哪些选项呢？”在大多数情况下，本书会提供特定的例子说明如何使用一条命令。

本参考手册也可以帮助那些对Unix系统的某些内容比较熟悉，但是对其他内容不熟悉的人员。本书许多章中都有关于特定主题的概述，尽管它并不全面，但通常已足够使你开始进入一个不熟悉的领域。

有些读者可能以前使用的是运行 BSD 或 SunOS 4.1 版本的 Unix 系统，为了帮助这些用户进行系统的转换，SVR4 和 Solaris 中提供了一组“兼容”命令，本书讲述了大部分兼容命令。

最后，如果你是使用Unix操作系统的新手，而且有足够的勇气，那么可以快速浏览一下本书，以便掌握Unix提供的内容。第一章中的“初学者指南”一节，说明了Unix中最有用的一些命令，该节有如何使用这些命令的一些示例，但是应注意：不要使用本书来替代一本好的Unix初学者指南。（你可以选择O'Reilly公司出版的《Learning the Unix Operating System》一书作为初学者指南。）本书可以作为 Unix 初学者指南的补充内容，而不是替代书籍。（本书全文中将多次引用其他相关的O'Reilly书籍，它们可以帮助你学习正在讨论的主题。）

本书覆盖的范围

本书划分为五部分：

- 第一部分（第一到五章）讲述了 Unix 命令和 Bourne、Korn 及 C shell 的语法和选项。
- 第二部分（第六到十一章）讲述了不同的编辑工具并说明了它们的命令设置（按照字母顺序或按组排列）。第二部分首先回顾了模式匹配等内容，包括用于特定编辑器的例子。
- 第三部分（第十二到十七章）讲述了 `nroff` 和 `troff` 文本格式化程序、相关的宏包和预处理程序 `tbl`、`eqn`、`pic` 及 `refer`。

- 第四部分（第十八到二十章）讲述了用于软件开发的Unix实用程序 —— SCCS、RCS 和 make。
- 第五部分（附录一、附录二和参考文献）包含了ACSII字符及对应值的一个列表（附录一）、SVR4和（或）Solaris中已经淘汰的命令（附录二）、Unix书籍的参考文献列表等。

本书英文排版约定

本书使用下列排版约定：

等宽字体（`Constant width`）
: 用于目录名、文件名、命令、程序名、函数和选项等。所有使用固定宽度显示的术语应按照字面内容输入，它也用于显示文件的内容或命令的输出。

等宽斜体（*`Constant width italic`*）
: 用于语法或命令摘要，以便说明一般的文本，应该使用用户提供的值来替换这些内容。

等宽粗体（Constant width bold）
: 用于示例，说明这些文本应该由用户按照字面内容输入。

斜体（*Italic*）
: 用于说明一般的参数和选项，应该使用用户提供的值替换这些斜体内容。斜体也用于表示URL、宏包名和示例中的注释。

`%`，`$`，`#`
: 在一些示例中作为C shell提示符（`%`）、Bourne shell或Korn shell提示符（`$`）。`#`是超级用户（root）的提示符。

`?`，`>`
: 在一些示例中作为C shell的第二个提示符（?）、Bourne shell或Korn shell的第二个提示符（>）。

program(N)
: 指明程序的参考页（manpage）在联机手册的第N部分。在示例中，*echo*(1)表示 echo 命令的项。

`[]` 在一个语法描述中将一些可选元素包括起来（不要输入括号）。注意，许多命令使用了参数[*files*]，如果省略了文件名，则假设使用标准输入（通常是键盘）。键盘输入结束时使用一个文件结束符。

EOF

表示文件结束符（通常是 CTRL-D）。

^x，CTRL-x

表示一个控制字符。可以在按下 Ctrl 键的同时再按下 *x* 键进行输入。

| 在命令的语法描述中，用来分隔多个并列的可选项。

总而言之，在大多数情况下，一个选项及其参数之间的空格可以省略；在其他情况下，空格（或缺少的空格）必须严格按照实际数量放在参数或选项的后面。例如，-w*n*（中间没有空格）与 -w *n*的意义可能是不同的。注意到选项语法中使用的空格是很重要的。

建议与评论

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们：

美国：

O'Reilly & Associates,Inc.
101 Morris Street
Sebastopol, CA 95472

中国：

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室
奥莱理软件（北京）有限公司

询问技术问题或对本书的评论，请发电子邮件到：

info@mail.oreilly.com.cn

最后，您可以在 WWW 上找到我们：

http://www.oreilly.com
http://www.oreilly.com.cn

致谢

感谢Yosef Gold让我使用他的办公室，使我能够进行有效而有成果的工作。感谢Deb Cameron 修订了第七章。也感谢 Gigi Estabrook 在我与 O'Reilly 公司的合作中给予的帮助和支持。感谢 Frank Willison 管理本项目。

有出色的审校者才有可能编写出精品图书，即使他们使作者付出了更多的劳动。我非常感谢 Glenn Barry（Sun 公司）提供了大量有益的注释。Nelson H. F. Beebe（犹他大学数学系）仔细审查了本书，由于他的辛勤劳动，本书的质量大大提高。特别感谢 Brian Kernighan（贝尔实验室）的复审及意见。与 `troff` 相关的几章特别得益于他的权威著作及专业知识，正像本书其余部分那样（不必提及更多的Unix）。Nelson H. F.Beebe、Dennis Ritchie （贝尔实验室）和 Peter H. Salus（Unix 史学家和作者）为参考文献提供了非常大的帮助。

最后，非常感谢我特别好的妻子 Miriam，没有她的爱与支持，是不可能完成本项目的。

Arnold Robbins
Nof Ayalon，以色列
1999.4

目录

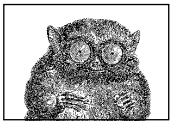
第二部分 文本编辑和处理

第一部分

命令和 shell

第一部分总结了用户和程序员感兴趣的 Unix 命令。它还描述了三个主要的 Unix shell，包括特殊语法和内置命令。

- 第一章，Unix 入门
- 第二章，Unix 命令
- 第三章，Unix shell 概述
- 第四章，Bourne shell 和 Korn shell
- 第五章，C shell



第一章

Unix 入门

Unix 操作系统在 20 世纪 70 年代早期创建于 AT&T Bell 实验室，在 20 世纪 80 年代末期由 USL（Unix 系统实验室）开发出了 System V Release 4（SystemV 版本 4）。现在 SCO 公司拥有 Unix 的源代码。由于 Unix 可以运行在多个厂商提供的不同硬件上，各个厂商被鼓励对 Unix 进行修改并发布自己的增值版本，从而导致如下多个版本的 Unix：USL 的 System V、BSD（Berkeley Software Distribution，出自加大伯克利分校）、Xenix 等等。

各种传统的融合

现在，Unix 开发者将各种不同的传统融合成一种更标准的版本。（仍在制定中的 POSIX，是一种基于 SystemV 和 BSD 的国际标准，它直接影响了这种融合。）这本参考手册主要讲述的正是许多人认为“更标准的”Unix 版本：System V Release 4（SVR4）和 Solaris 7（注 1）。

由 USL（AT&T 的子公司）和 Sun 公司联合开发的 SVR4，融合了 BSD 和 SVR3 的功能，并在基本的 Unix 命令集中添加了 24 个 BSD 命令（加上一些新的 SVR4 命令）。

注 1： 许多类 Unix 操作系统，比如 Linux 和那些基于 4.4BSD-Lite 的操作系统，也提供了与 SVR4 和 BSD 早期版本的标准的一致性和兼容性，但这些内容已经超出了本书的范围。

另外，SVR4 提供了一个 BSD 兼容包，该包是一个“替补”命令组。这个包中包括一些最基本的 BSD 命令，其目的是帮助那些源自 BSD 的系统的用户转换到 SVR4。

Solaris 7 是由 Sun 公司开发的分布式计算环境，其历史非常复杂。

Solaris 7 包括 SunOS 5.7 操作系统，并增加了另外一些功能，比如 CDE（Common Desktop Environment）和 Java 工具。SunOS 5.7 又融合了 SunOS 4.1 和 SVR4。另外，内核也进行了非常大的增强，以支持多 CPU、多线程进程、内核级线程、设备驱动程序的动态加载和其他内核模块。大多数用户级（和系统管理级）的内容直接来自 SVR4。因此，Solaris 7 基于 SVR4，但是包含一些附加的 BSD/SunOS 功能。为了帮助用户从以前的 SunOS 操作系统（大部分基于 BSD）移植到现在的操作系统，Solaris 提供了 BSD/SunOS 兼容包和二进制兼容包。

非商业用户可以“免费”获得 Sun 公司提供的用于 SPARC 和 Intel 体系结构的二进制版本的 Solaris，只需要支付媒介、运输和包装费。要了解更多的信息，可访问如下站点：*http://www.sun.com/developer*。

打包

影响 Unix 系统的另一个方面是打包（bundling）的概念。Unix 有许多功能，有时你只能使用其中的一部分。现在，Unix 系统经常分成或打包成多个组件包。你购买的 Unix 系统中会包括其中的一些组件；另一些组件是可选的，要购买这些组件需要支付额外的费用。打包允许你只选择需要的组件。典型的打包方式如下：

基本系统
: 基本的命令和实用程序。

编程
: 编译器、调试器和库。

文本处理
: `troff`、宏和相关工具。

窗口系统
: 一些图形用户界面，比如 OPEN LOOK、Motif 和 CDE。

打包的方式取决于厂商，比如 Solaris 公司在 Unix 基本系统中提供文本处理工具，而其他一些厂商则要求单独购买这些工具；另外，有一些厂商将编译器打包到基本系统中，而其他厂商却不提供编译器。

Solaris 安装级别和打包

当你（或你的系统管理员）第一次安装 Solaris 时，可以选择三种级别的安装方式：

终端用户系统支持
　该选项会安装最简单的系统。

开发者系统支持
　该选项除了安装基本系统以外，还会安装用于软件开发的库和头文件。

完全版本
　该选项会额外安装许多可选的实用工具，包括支持多种非英语语言及字符集。

注意，如果你选择了终端用户系统的安装，则本书讨论的许多命令在你的系统中都无法找到（如 `make` 和 SCCS 套件）；如果你有足够的磁盘空间，至少应该选择开发者系统的安装。

要获得 Solaris 的技术支持和公开发布的补丁，可以访问如下站点：*http://sunsolve.sun.com*。

Solaris 不提供 C 或 C++ 编译器，但是可以从 Sun 公司单独购买这些编译器，GNU C 编译器（其中包括了 C++）和其他专门为 Solaris 开发的免费编译器，可以到如下站点下载：*http://www.sunfreeware.com*。尽管没有提供 `pic`，但是 Solaris 还是包括了一个新版本的 `troff` 和它的伴随工具。

快速参考的主要内容

本书讲述了通用SVR4的主要功能，并讲述了兼容包和Solaris 7中新增加的功能。另外，本书还用一些章节讲述了 `emacs` 和 RCS，尽管这些内容不是标准 SVR4 发行版的一部分，但是由于它们是非常有用的附加软件包，所以许多Unix 操作系统中都提供了。

但是一定要记住 :如果你的操作系统中没有包括所有的组件包，则操作系统中将无法找到本书中讲述的有些命令。

本书第二章概要讲述了 Unix 命令，它构成了本书的主要内容，该章只讲述了用户 / 程序员命令，忽略了管理命令。第二章讲述了下面几部分内容：

- SVR4 中的命令和选项。

- 从兼容包和 Solaris 7 中选择的命令，比如与 Java 相关的工具。
- 可通过 Internet 获得源代码及（或）二进制代码的主要工具。

Solaris 用户应该注意，下面的命令或者没有打包，或者无法得到：

```
cb          cc          cflow       cof2elf
cscope      ctrace      cxref       lprof
pic
```

附录二中讲述了已经淘汰的SVR4命令。这些命令仍随SVR4或Solaris一起提供，但是这些命令的功能已经被其他命令或技术替代。

初学者指南

如果你刚刚开始使用一种Unix操作系统，大量的命令可能使你感到恐惧。为了帮助你克服这种恐惧，下面将列表说明用于各个方面的一小部分命令。

通信

`ftp` 文件传输协议。

`login` 登录到 Unix。

`mailx` 读取或发送邮件。

`rlogin` 登录到远程 Unix。

`talk` 写入到其他终端。

`telnet` 连接到另一个系统。

`vacation` 自动响应邮件。

比较

`cmp` 逐字节比较两个文件。

`comm` 比较两个排序文件中的条目。

`diff` 逐行比较两个文件。

`diff3` 比较 3 个文件。

`dircmp` 比较目录。

`sdiff` 并排比较两个文件。

文件管理

`cat` 连接多个文件或显示它们。

`cd` 改变目录。

`chmod` 改变文件的访问模式。

`cp` 拷贝文件。

`csplit` 在文件特定位置将文件分开。

`file` 确定一个文件的类型。

`head` 显示文件开头的几行。

`ln` 建立文件的别名。

`ls` 列表显示文件或目录。

`mkdir` 创建一个目录。

`more` 逐屏显示文件。

`mv` 移动或重命名文件或目录。

`pwd` 输出工作目录。

`rcp` 将文件拷贝到远程系统。

`rm` 删除文件。

`rmdir` 删除目录。

`split` 将多个文件平均分开。

`tail` 显示文件的最后几行。

`wc` 计算文件的行数、单词及字符数。

其他

`banner` 显示广告词。

`bc` 设置任意精度的计算器。

`cal` 显示日历。

`calendar` 检查备忘录。

`clear` 清屏。

`man` 获得一条命令的帮助信息。

`nice` 降低一项作业的优先权。

`nohup` 在退出系统后保存一项正在运行的作业。

`passwd` 设置登录系统的密码。

`script` 产生登录会话的一个抄本。

`spell` 报告拼写错误的单词。

`su` 变为一个超级用户。

打印

`cancel` 取消一个打印机请求。

`lp` 发送到打印机。

`lpstat` 获得打印机的状态。

`pr` 对打印内容进行格式化并编页码。

编程

`cb` C 语言源代码优化器。

`cc` C 语言编译器。

`cflow` C 语言函数流程图。

`ctags` C 语言函数参考信息（用于 vi）。

`ctrace` 使用函数调用跟踪的 C 语言调试器。

`cxref` C 语言交叉引用。

`lint` C 语言程序分析器。

`ld` 装载器。

`lex` 词法分析生成器。

`make` 按照指定的顺序执行命令。

`od` 以不同的方式转储输入。

`strip` 从一个目标文件中删除数据。

`truss` 跟踪信号和系统调用。

`yacc` 分析程序生成器，可以与 `lex` 命令一起使用。

搜索

`egrep` `grep` 命令的扩充版本。

`fgrep` 根据单词搜索文件。

`find` 根据文件名搜索系统。

`grep` 根据文本格式搜索文件。

`strings` 根据文本格式搜索二进制文件。

shell 编程

`echo` 在输出中重复命令行参数。

`expr` 执行算术运算和比较。

`line` 读取输入的一行信息。

`printf` 格式化并打印命令行参数。

`sleep` 暂停处理。

`test` 测试一个条件。

存储

`compress` 压缩文件以释放空间。

`cpio` 将档案文件拷贝到磁盘上或从磁盘上拷贝出。

`gunzip` 释放压缩（`.gz` 和 `.Z` 格式）的文件（首选命令）。

`gzcat` 显示压缩文件的内容（可以与 `zcat` 命令一起使用）。

`gzip` 压缩文件以释放空间（首选命令）。

`tar` 磁带归档。

`uncompress` 释放压缩的文件（`.Z` 格式）。

`zcat` 显示压缩文件的内容。

系统状态

`at` 随后运行命令。

`chgrp` 修改文件所在的组。

chown	修改文件的属主。
crontab	自动执行命令。
date	显示或设置日期。
df	显示空闲磁盘空间。
du	显示磁盘空间的使用情况。
env	显示环境变量。
finger	显示用户信息。
kill	终止一个运行的命令。
ps	显示进程。
stty	设置或显示终端设置。
who	显示登录的用户。

文本处理

cut	选择要显示的列。
ex	vi 的行编辑器。
fmt	使文本各行的长度大体一致。
join	将不同的列合并到一个数据库中。
nawk	awk 命令的新版本（用于文本数据库文件的模式匹配语言）。
paste	合并列或改变列的顺序。
sed	非交互的文本编辑器。
sort	对文件进行排序或合并。
tr	转换（重新定义）字符。
uniq	搜索一个文件中的重复或惟一的行。
vi	可视化的文本编辑器。
xargs	在可管理部分处理多个参数。

nroff 和 troff

在 SVR4 中，只有 deroff 命令没有包含在兼容包中。Solaris 包中包含了一个新版本的 troff 命令和它的预处理程序（没有包括 pic 命令）。

`deroff` 删除 `troff` 代码。

`eqn` 用于方程式的预处理程序。

`nroff` 用于终端显示的格式化程序。

`pic` 用于线图的预处理程序。

`refer` 用于文献引用的预处理程序。

`tbl` 用于表格的预处理程序。

`troff` 用于排版的格式化程序（包括 PostScript 打印机）。

源自 BSD 的系统的用户指南

如果要从源自 BSD 的系统转换到 SVR4 系统，应注意 BSD 命令保存在系统的 `/usr/ucb` 目录中。在使用特定的命令时，了解这一点尤其重要，因为兼容包中有一些命令与 SVR4 中的命令非常相似，但是两个版本中相同命令的使用方法通常不一样。如果你的 PATH 变量在 SVR4 命令目录（比如 `/usr/bin`）前面指定 `/usr/ucb`，将会终止运行 BSD 版本的命令。使用前应检查你的 PATH 变量（使用 `echo $PATH`）以确保设置正确，BSD 和 SVR4 都有的命令如下：

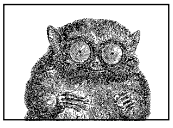
`basename`	`du`	`ls`	`tr`
`cc`	`echo`	`ps`	`vacation`
`chown`	`groups`	`stty`	
`deroff`	`ld`	`sum`	
`df`	`ln`	`test`	

本书只讲述这些命令的 SVR4 版本（通常情况下，标准 Solaris 版本的命令包含 BSD 版本的命令的功能及选项）。

Solaris：标准兼容程序

在一个 Unix 环境中，有许多不同的标准规定可移植程序的行为。POSIX 1003.2 和 XPG4 是两个比较著名的标准。其中，标准规定的行为与一个命令提供的历史行为是不同的，Solaris 在 `/usr/xpg4/bin` 目录中提供了这些命令的不同版本。下面将列出这些命令，但是在本书的其他部分并没有讲述这些命令，因为大部分用户的搜索目录中没有 `/usr/xpg4/bin`。用户手册上针对各个命令的条目讨论了 `/usr/bin` 版本和 `/usr/xpg4/bin` 版本的区别。

```
ar              ed          make        rm
awk             edit        more        sccs
basename        env         nice        sed
cp              expr        nl          sort
ctags           get         nm          sty
date            grep        nohup       tail
delta           id          od          tr
df              ls          pr          who
du              m4
```



第二章

Unix 命令

本章将讲述用户和程序员感兴趣的Unix命令, 其中大部分命令可以在SVR4的User's Reference Manual（用户参考手册）和 Programmer's Reference Manual（程序员参考手册）中找到。本章还将介绍兼容包中的多个附加命令, 这些命令使用/usr/ucb（即保存这些命令的目录名称）作为前缀。本章也介绍了一些 Solaris 7 专有的命令，比如那些使用 Java 的命令及有时编写 Internet 程序所必需的命令。

在 Solaris 中，有用的命令都保存在多个不同的 bin 目录中，比如 `/usr/ccs/bin`、`/usr/dt/bin`、`/usr/java/bin`和`/usr/openwin/bin`等, 而不只是保存在`/usr/bin`和 `/usr/ucb` 目录中。在这种情况下，本书将提供保存这些命令的完整路径，比如 `/usr/ccs/bin/make`。在一些实例中, 只将一个命令的符号链接保存在 `/usr/bin`目录中，而将实际命令保存在其他位置。

本章中的各个项都使用放在页面外边的命令名称作为标签。语法行后面是简要的描述, 然后是所有可用选项的列表。许多命令的最后还带有示例。如果你只需要命令的提示或建议，可以直接跳到示例部分。

有些命令选项只允许有特殊系统特权的用户使用, 该用户通常称为超级用户, 本书使用“特权用户”来代替超级用户。

描述命令语法的排版约定已在前言中列出。查找命令方面的附加帮助，可以参考索引。

按字母顺序排列的命令

addbib

addbib [*options*] *database*

程序的 refer 套件的一部分，参见第十七章。

admin

/usr/ccs/bin/admin [*options*] *files*

一个 SCCS 命令，参见第十八章。

appletviewer

/usr/java/bin/appletviewer [*options*] *urls*

只用于 Solaris。连接到指定的 *url*，并在自己的窗口中（在 Web 浏览器之外）运行指定的任何 Java applet。

options（选项）

-debug
: 在 Java 调试器（即 jdb）的内部运行 applet 浏览程序。

-encoding *name*
: 指定输入 HTML 文件的编码。

-J *opt*
: 将 *opt* 传递给 java 命令，*opt* 中不应该包含空格，如果需要可使用多个 -J 选项。

apropos

apropos *keywords*

在联机参考页中查找一个以上的关键字，与 man -k 命令相同，可参见 **whatis** 命令。

ar

/usr/ccs/bin/ar [-V] *key* [*args*] [*posname*] *archive* [*files*]

将一组文件 *files* 合并到一个档案文件 *archive* 中，加载器（使用 ld 命令）经常使用它来创建及更新库文件。该命令只能使用一个 key 字母，但是各个 Key 字母可以与附加的参数 *args*（参数之间没有间隔）进行合并。*posname* 是档案文件 *archive* 中一个文件的名字，如果要移动或替换文件 *files*，可以指定将它们放置在 *posname* 的前面或后面。另一个例子可参见附录二中的 **lorder** 命令。-V 选项在标准错误中输出 ar 的版本号。

ar

在 Solaris 中，可以在 *key* 和 *args* 前面添加一个中划线（-），尽管它们是规则选项。

key

d 从 *archive* 中删除文件 *files*。

m 将文件 *files* 移动到 *archive* 的最后。

p 输出 *archive* 中的所有文件 *files*。

q 向 *archive* 中追加文件 *files*。

r 替换 *archive* 中的文件 *files*。

t 列出 *archive* 的内容或指定的文件 *files*。

x 从 *archive* 或从指定的文件 *files* 中摘录内容。

args

a 与 key 选项中的 r 或 m 一起使用，将 *files* 放在档案的 *posname* 后面。

b 与 a 选项作用相同，只是将 *files* 放在 *posname* 的前面。

c 默默地建立一个档案文件 *archive*。

C 不使用档案中的文件名来替换已经存在的具有相同文件名的文件，与 T 选项一起使用时比较有用。只能用于 Solaris 中。

i 与 b 选项的作用相同。

s 强制重新建立档案 *archive* 的符号表（通常在运行了 strip 或 mcs 命令后使用它）。

T 如果一个文件系统不支持长文件名，则将长文件名截短。如果不使用该选项，则使用带有长文件名的文件时会出错。该选项只适用于 Solaris。

u 与 r 一起使用，来替换档案 *archive* 中已经修改的文件 *files*。

v 输出命令的详细描述。

示例

使用当前目录中的目标文件更新mylib.a中目标文件的版本,如果一

ar

些文件保存在mylib.a中而没有保存在当前目录中,则不会替换这些文件。

```
ar r mylib.a *.o
```

as

/usr/ccs/bin/as [*options*] *files*

从每个指定的汇编语言源文件*file*中生成一个目标文件。目标文件与源文件具有相同的根目录名，但是它使用.o后缀代替.s。该命令也可能有附加的特定专有选项，可参考 **dis**。

options

-m 在文件 *file* 上运行 m4。

-n 关闭对长 / 短地址的优化。

-o *objfile*
将输出放入目标文件 *objfile* 中 （默认是 *file*.o）。

-Q*c* 将汇编程序的版本号放入目标文件中 （当 *c*=y 时）；默认情况下是不放入目标文件中（当 *c*=n 时）。

-R 生成目标文件后删除源文件 *file*。

-T 强制遵从淘汰的汇编程序指令。

-V 显示汇编程序的版本号。

-Y [*key*,] *dir*
搜索用于 m4 预处理程序的目录 *dir*（如果 *key* 是 m）或包含预定义宏的文件（如果 *key* 是 d），或者两者都搜索（如果省略 *key*）。

at

at *options1* time [*date*] [*+increment*]
at *options2* [*jobs*]

在一个可选的日期 *date* 及特定时间 *time* 中，运行在标准输入中输入的命令（参见 **batch** 和 **crontab**）。使用 *EOF* 终止输入。*time* 可以使用数值小时（可以选择使用分钟和修饰符）形式或作为一个关键字；*date* 可以表示成一个月、日期、一星期中的一天或一个特殊的关键字。*increment* 是一个正整数，后面跟一个关键字。详细内容可以参考下面的列表。

at

options1

`-c` 使用 C shell 执行作业，只用于 Solaris。

`-f` *file*
运行 *file* 中列出的命令。

`-k` 使用 Korn shell 执行作业，只用于 Solaris。

`-m` 在完成作业后向用户发送邮件。

`-q` *queuename*
规划 *queuename* 中的作业的进度。*queuename* 的值使用 a 到 z 的小写字母表示。队列 a 是 at 作业的默认值；队列 b 是 batch 作业队列；队列 c 用于 cron 作业。该选项只用于 Solaris 。

`-s` 使用 Bourne shell 来执行作业，只用于 Solaris。

`-t` *time*
在时间 *time* 执行作业，使用 touch 允许的格式。只用于 Solaris。

options2

`-l` 输出正在为调用用户调度的所有作业；如果指定了作业，则只输出对这些指定作业的调度。参见 **atq**。

`-r` 删除以前调度的指定作业 *jobs*。要删除一个作业，你必须是一个特权用户或是作业的属主。要查看已调度作业的列表，应先使用 `-l`。参见 **atrm**。

time

hh:mm [*modifiers*]
用一个或两个数字表示的时间（默认为 24 小时的时钟）；可以使用一个或两个数字来表示可选的分钟；如果格式是 *h*、*hh* 或 *hhmm*，则可以省略中间的冒号 (:)。比如，下面都是有效时间：5、5:30、0530、19:45。如果添加了 am 或 pm 修饰符，则使用 12 小时的钟表来表示时间。如果添加了关键字 zulu，则相应地使用格林威治标准时间（UTC）。

`midnight | noon | now`
可使用上面的任何一个关键字来代替数字时间，now 后面必须紧跟一个 *increment* 参数。

at

date

month num[, *year*]

month 是 12 个月中的任意一个月，使用月份的完整名称或前三个字母的缩写形式；*num* 表示一个月中的某一天；*year* 是使用四位数字表示的年；如果给定的 *month* 在当前月份之前，则 at 使用下一年的这个月。

day 可以是一星期中的任意一天，可以使用该天的完整名称或者前三个字母的缩写形式。

today | tomorrow

表示当天或第二天。如果省略了 *date*，则当指定的时间 *time* 在当前时间后面时，at 使用的时间为 today；否则，将使用 tomorrow。

increment

如果希望指定一个执行时间或与当前时间相对的日期，可以使用该参数提供一个数字增量，该数字应该放置在 minute、hour、day、week、month 或 year 等关键字（或它们的复数形式）的前面。关键字 next 可以当作 +1 符号使用。

示例：

注意前两条命令是等价的。

```
at 1945 pm December 9
at 7:45pm Dec 9
at 3 am Saturday
at now + 5 hours
at noon next day
```

atq

atq [*options*] [*users*]

列出仍在队列中的由 at 命令建立的作业。通常情况下，作业按照执行的顺序排序。可指定 *users* 参数来检查其作业。如果没有指定 *users* 参数，那么在默认情况下，当你是特权用户时，会显示所有的作业；否则，只会显示你的作业。

options

-c 按照 at 命令的时间对队列进行排序。

atq

`-n` 只输出队列中作业的总数。

atrm

`atrm [`*options*`] [`*users* | *jobIDs*`]`

删除使用`at`排队并匹配指定的*jobIDs*参数的作业。特权用户也可以指定*users*，该用户的作业将被删除。

options

`-a` 删除属于当前用户的所有作业（特权用户可以删除所有作业）。

`-f` 无条件地删除所有作业，而不考虑关于删除作业的所有信息。

`-i` 提示为`y`（删除所有作业）或`n`（不删除作业）。

awk

`awk [`*options*`] [`*program*`] [`*var=value* `...] [`*files*`]`

使用模式匹配程序来处理指定的文件*files*。`awk`已经被`nawk`替代（还有一个GNU版本的命令`gawk`）。程序指令的通用格式如下:

pattern { *procedure* }

*pattern*和*procedure*都是可选项。当在命令行使用该格式时，必须使用单引号将程序包括起来，以避免shell将其当作特殊字符。程序中使用的任何变量都可以使用命令行参数的赋值格式`var=value`来设置一个初始值。要了解`awk`命令的详细使用方法及示例，可参见第十一章。

options

`-f` *file*

使用*file*中包含的程序指令来代替命令中指定的程序。

`-Fc` 将输入文件*file*作为由字母C分隔的字段。默认情况下，输入字段使用空格或（和）制表符分隔。

banner

`banner` *characters*

将*characters*当作广告在标准输出中输出，所提供的每个单词必须包含10个或10个以内的字符。

basename

basename *pathname* [*suffix*]

从给定参数 *pathname* 中删除路径前缀而只保留文件名，并在标准输出中输出。如果已经指定了文件名后缀 *suffix*，则文件名后缀（比如 .c）也会被删除。basename 通常是通过命令替换符（`…`）进行调用来产生一个文件名的。参见 **dirname**。

basename 的 Solaris 版本允许后缀是 expr 可以接受的模式。详细内容可参见对 **expr** 的说明。

示例：

下面的代码来自一个 Bourne shell 脚本片段:

```
ofile=output_file
myname="`basename $0`"
echo "$myname: QUITTING: can't open $ofile" 1>&2
exit 1
```

如果该脚本叫做 do_it，则会输出下面的标准错误消息:

```
do_it: QUITTING: can't open output_file
```

batch

batch

执行在标准输入中输入的命令，使用 *EOF* 结束。与只在特定时间执行命令的 at 不一样，batch 会依次执行各个命令（等到上个命令执行完后再执行下一个命令），这样可以避免因为同时运行多个后台作业而导致的系统负载过重。参见 **at**。

batch 与 at -q b -m now 等同。

示例：

```
$ batch
sort in > out
troff -ms bigfile > bigfile.ps
EOF
```

bc

bc [*options*] [*files*]

从一个基数到另一个基数之间交互执行任意精度的算术运算或数字转换。输入可以来自文件*files*, 也可以从标准输入中读取。如果要退出, 则键入 quit 或 *EOF*。

options

-c 不调用 dc，并只进行编译（因为 bc 是 dc 的预处理程序，所以 bc 通常调用 dc）。

-l 从数学库中获得可用的函数。

bc 是语法类似 C 的一种语言（和编译器）。bc 由标识符、关键字和符号组成，下面对这些成分进行简单的描述，示例见后面的内容。

标识符

标识符是一个单字符，由 a 至 z 的小写字母组成。标识符用作变量、数组或函数的名字。在同一个程序中, 可以使用相同的字母来命名一个变量、数组或函数。下面的标识符并不会混淆：

x 表示变量 x。

x[*i*] 表示有 *i* 个元素的数组 x。*i* 的范围在 0 到 2047 之间。也可以是一个表达式。

x(y,z) 表示具有 y 和 z 两个参数的函数 x。

输入 / 输出关键字

ibase、obase 和 scale 各自保存一个值。在同一行键入它们会显示出它们的当前值。通常情况下, 可以通过赋值来改变它们的值。字母 A~F 相当于数字值 10~15。

ibase = *n*

输入（比如键入）的数字按照基数 *n*（默认是 10）进行读取。

obase = *n*

按照基数 *n*（默认是 10）显示数字。注意，一旦 ibase 从 10 起进行改变，则用数字“ A ”将 ibase 或 obase 恢复到十进制。

bc

`scale = n`

使用 *n* 位小数来显示计算结果（默认是0，表示将结果截取为整数）。`scale` 通常指用于基数为 10 的计算。

语句关键字

分号或换行符用来将一个语句与其他语句区分开。只有将多个语句组合到一起时才使用大括号（`{}`）。

`if (`*rel-expr*`) {`*statements*`}`

如果表达式 *rel-expr* 是 true，则执行一个或多个 *statements*。例如：

```
if (x == y) i = i + 1
```

`while (`*rel-expr*`) {`*statements*`}`

如果表达式 *rel-expr* 是 true，则重复执行一个或多个 *statements*。例如：

```
while (i > 0) {p = p*n; q = a/b; i = i-1}
```

`for (`*expr1*`; `*rel-expr*`; `*expr2*`) {`*statements*`}`

与 `while` 类似。例如，要输出从 1 到 10 乘上 5，则可以输入如下内容：

```
for (i = 1; i <= 10; i++) i*5
```

`break`

终止一个 `while` 或 `for` 语句。

`quit`

退出 `bc` 命令。

函数关键字

`define `*j*`(`*k*`) {`

开始定义只有一个参数 *k* 的函数 *j*。允许使用额外的参数，并使用逗号间隔多个参数。语句以连续的行出现，每条语句带有一个 `}`。

`auto `*x, y*

设置 *x* 和 *y* 作为一个函数定义的局部变量，初始值为 0，在函数以外无意义。并且必须首先出现。

bc

`return`（*expr*）
将表达式*expr*的值传递回程序。如果（*expr*）停止则返回0；用于函数的定义。

`sqrt`（*expr*）
计算表达式*expr*的平方根。

`length`（*expr*）
计算*expr*中有多少数字。

`scale`（*expr*）
计算小数点右边的位数。

数学库函数
当`bc`命令中使用`-l`选项时，可以使用这些数学库函数。库函数将`scale`（比例）设置为20。

`s`（*angle*）
计算角*angle*的正弦值，该值是使用弧度表示的常量或表达式。

`c`（*angle*）
计算角*angle*的余弦值，该值是使用弧度表示的常量或表达式。

`a`（*n*）
计算*n*的反正切值，并返回使用弧度表示的一个角。

`e`（*expr*）
计算表达式*expr*的*e*次幂。

`l`（*expr*）
计算表达式*expr*的自然对数。

`j`（*n, x*）
计算整数阶为*n*的贝塞尔函数。

运算符
包括运算符和其他符号。运算符可以是算术运算符、一元运算符、赋值运算符或关系运算符。

算术运算符为： `+ - * / % ^`

一元运算符为： `- ++ --`

bc

赋值运算符为： =+ =- =* =/ =% =^ =

关系运算符为： < <= > >= == !=

其他符号

/* */

封装一段注释。

() 控制表达式运算的先后顺序（改变运算符的优先权）。也可以用于赋值语句来强制输出结果。

{} 用于将多个语句划为一组。

[] 数组下标

"*text*"

作为一个语句使用，用于输出文本 *text*。

示例：

注意，当你键入一些内容（一个数字或表达式）时，会对其进行运算并输出运算的结果，但是赋值语句不产生任何输出。

```
ibase = 8          八进制输入
20                 计算该八进制数的值
16                 终端显示十进制的值
obase = 2          使用基数 2 代替基数 10 来显示输出
20                 八进制输入
10000              终端显示二进制的值
ibase = A          恢复基数为 10 的输入
scale = 3          将结果截短为 3 位小数
8/7                求除法的值
1.001001000        忘记重新将输出设置为基于 10 的数字
obase = 10         现在输入是十进制，所以不需要“A”
8/7
1.142              在终端上显示结果（将数字截短了）
```

下面的行说明了函数的用法：

```
define p (r,n){    函数 p 使用了两个参数
    auto v         是一个局部变量
    v = r^n        计算 r 的 n 次幂
     return (v)}   获得返回值
scale = 5
x = p (2.5,2)      x = 2.5 ^ 2
x                  显示 x 的值
6.25
```

bc

```
length（x）        计算x中的数字个数
3
scale（x）         设置小数点右边的位数
2
```

bdiff

bdiff *file1 file2* [*options*]

对 *file1* 和 *file2* 两个文件进行比较，并报告两个文件中不同的行。bdiff 分开多个文件后运行 diff，可以使用它来处理因为太大而难于处理的文件。如果其中的一个文件是 -，则 bdiff 读取标准输入。参见 **diff**。

options

n 将各个文件分成具有 *n* 行的段（默认是 3500 行）。该选项必须首先列出。

-s 禁止来自 bdiff 的错误消息（但是可以显示来自 diff 的错误消息）。

biff

/usr/ucb/biff [y | n]

打开或关闭邮件通知。不使用任何参数时，biff 指示当前的状态。

当打开邮件通知时，每次有邮件发来时就会响铃，并且显示各条消息的开头几行。

cal

cal [[*month*] *year*]

不使用任何参数时，只会输出当前月份的日历。否则，会按照参数 *year* 输出12个月的日历（从1月份开始），或者按照参数 *month* 和 *year* 输出一个月的日历。*month* 的取值范围是 1~12，*year* 的取值范围是 1~9999。

示例

```
cal 12 1999
cal 1999 > year_file
```

calendar

calendar [*option*]

读取 calendar 文件并显示包含当前日期的所有行。calendar 文件类似于一个备忘录。可以按照下面的方法建立文件并添加条目：

```
5/4     在下午 2 点会见设计组
may 6   在回家路上买周年纪念卡
```

当日历为 5 月 4 日时，显示第一行。通过使用 crontab 或 at 可使 calendar 自动执行。也可以通过在 .profile 或 .login 等启动文件中包括它来使 calendar 自动执行。

option

- 允许一个特权用户为所有用户调用 calendar，从各个用户的登录目录中搜索名字为 calendar 的文件，匹配的项将通过邮件发送给用户。该功能主要通过 cron 使用。建议不要在网络环境中使用大的用户基数。

cancel

cancel [*options*] [*printer*]

取消使用 lp 进行的打印请求。该请求可以使用其 ID、当前使用的打印机 *printer* 或与请求关联的用户名（只有特权用户才能取消其他用户的打印请求）进行指定。使用 lpstat 可以确定要取消的 *id* 或打印机。

options

id 取消打印请求 *id*。

-u user
取消与 *user* 关联的请求。

cat

cat [*options*] [*files*]

读取一个或多个 *files* 并在标准输出中输出。如果没有指定 *files* 或者其中的一个文件指定为 -，则它会读取标准输入，并使用 *EOF* 终止输入。使用 >shell 操作符可以将多个文件合并到一个新文件中；使用 >> 操作符可以向一个已经存在的文件中追加多个文件。

cat

options

-b 与 -n 作用相似，但是不计算空行。只适用于 Solaris。

-e 输出一个 $ 来标记各行的结束。必须与 -v 一起使用。

-n 计算行数。只适用于 Solaris。

-s 禁止不存在的文件的消息。（注意：在一些系统中，-s会将多余的空行删除。）

-t 各个制表符输出为 ^I，各个换页符输出为 ^L。必须与 -v 一起使用。

-u 输出没有缓冲的输出（默认情况下使用块或屏幕行进行缓冲）。

-v 显示控制字符或其他非打印字符。

示例

```
cat ch1                  显示一个文件
cat ch1 ch2 ch3 > all    合并多个文件
cat note5 >> notes       向一个文件中追加内容
cat > temp1              在终端上建立文件，并使用 EOF 终止文件
cat > temp2 << STOP      在终端上建立文件，并使用 STOP 终止文件
```

cb

cb [*options*] [*files*]

使用合适的 C 编程结构来格式化文件 *files* 的 C 程序“优化器”。

options

-j 连接分开的行。

-l *length*
将长度超过 *length* 的行分开。

-s 将代码标准化为《The C Programming Language》一书中 Kernighan 和 Ritchie 指定的风格。

-V 在标准错误中输出 cb 的版本。

示例

```
cb -l 70 calc.c > calc_new.c
```

cc

/usr/ccs/bin/cc [*options*] *files*

编译一个以上的C源文件（*file*.c）、汇编源文件（*file*.s）或预处理C源文件（*file*.i）。cc会自动调用装载器ld（除非提供了-c选项）。有时，cc会产生具有.o后缀及一个相应根名称的目标文件。默认情况下，将输出放置到a.out中。cc可以接受附加的系统专有选项。

注意

- 将/usr/ccs/bin添加到PATH中，则可以使用C编译器和其他的C编译系统工具。该命令会运行ANSI C编译器。如果希望运行ANSI C以前的编译器，可以使用/usr/bin/cc。
- Solaris 7没有提供C编译器，必须从Sun公司单独购买一个，或者从 *http://www.sunfreeware.com* 上下载一个GNU C编译器（GCC）。
- 在不同的Unix系统中，cc的选项会大不相同，我们这里只记录了一些比较通用的选项。要了解完整的信息还需要查看本机的文档。
- 通常情况下，cc会向装载器ld传递任何未被承认的选项。

options

-c 禁止装载并保持已经产生的任何目标文件。

-*Dname*[*=def*]
提供一个#define指令，定义 *name*（名称）为 *def*。如果没有给出 *def*，则值为1。

-E 只运行宏预处理程序，并向标准输出中发送结果。

-g 产生调试器需要的多个符号表信息。

-I*dir*
在目录 *dir* 中查找包括文件（除了标准位置之外）。并为要查找的每个新的 *dir* 提供一个-I。

-l*name*
将源文件 *file* 链接到库文件 lib*name*.so 或 lib*name*.a 上。

-L*dir*
与-I相似，但是在目录 *dir* 中查找库档案。

cc

`-o` *file*
将目标文件发送到文件 *file* 中，而不是发送到 `a.out` 中。

`-O` 优化目标代码（来自 `.c` 或 `.i` 文件）。

`-p` 产生基准代码来计算调用每个例程的时间，此时会建立`mon.out`文件，所以以后可以使用 `prof` 来产生一个执行配置文件。

`-P` 只运行预处理程序并将结果放到文件 *file*`.i` 中。

`-S` 编译（及优化，前提是提供了 `-O` 选项），但是不进行汇编或加载，汇编程序输出的内容保存到文件 *file*`.s` 中。

`-U`*name*
删除 *name* 的定义，就好像使用一个 `#undef` 指令。

示例

编译 `xpop.c` 并使用 X 库加载它：

```
cc -o xpop xpop.c -lXaw -lXmu -lXt -lX11
```

cd

`cd [`*dir*`]`

改变目录。`cd` 是一个内置的 shell 命令，参见第四章和第五章。

cdc

`/usr/ccs/bin/cdc -`*rsid* `[`*option*`]` *files*

一个 SCCS 命令，参见第十八章。

cde

`CDE`

只用于 Solaris。CDE（Common Desktop Environment）是 Solaris 系统默认的图形用户界面（Graphical User Interface，GUI），Solaris 7 用户可能在 CDE 和 OpenWindows 之间选择其一，但是 OpenWindows 已经被淘汰，Solaris 7 以后的版本不再支持。

对 CDE 的介绍已经超出了本书的范围，可以参考专门介绍该内容的书籍。但是，这里列出了一些比较有用的单独的 CDE 命令，这些命令保存在 `/usr/dt/bin` 目录下（用于 **Desktop** 的命令）。另外，有许多 OpenWindows 命令仍然有用，参见附录二的 **openwin** 部分。

cde

有用的 CDE 程序

下面的 CDE 和 Sun Desktop 命令可能是大家感兴趣的，更多的信息可以参见参考页。

`answerbook2`	Sun 公司的超文本文档阅读器。
`dtaction`	从 shell 脚本中调用 CDE 动作。
`dtbuilder`	CDE 应用程序生成器。
`dtcalc`	可以在屏幕上使用的科学、逻辑和金融计算器。
`dtcm`	日历管理器。
`dterror.ds`	进行错误通知和对话的 `dtksh` 脚本。
`dtfile_error`	用于错误对话的 `dtksh` 脚本。
`dticon`	图标编辑器。
`dtksh`	即 Desktop Korn shell，`ksh93` 的一个版本。
`dtmail`	邮件阅读器。
`dtpad`	简单文本编辑器。
`dtprintinfo`	打印作业管理器。
`dtscreen`	屏幕保护程序。
`dtterm`	终端仿真程序。
`fdl`	用于 PostScript 打印机的字体下载实用程序。
`hotjava`	基于 Java 的 Web 浏览器。
`sdtconvtool`	`iconv` 的图形用户界面。
`sdtfind`	文件搜索器。
`sdtimage`	图像浏览器（查看 PostScript、GIF、JPEG 等类型的图像）。
`sdtperfmeter`	系统性能测量计。
`sdtprocess`	进程管理器。

cflow

cflow [*options*] *files*

生成用于C、lex、yacc、汇编程序或目标文件*files*的外部函数调用的一个描述图（outline，又称流程图）。cflow也会接受-D、-I和-U等cc选项。

options

-dn 如果达到嵌套等级*n*，则停止绘制描述图。

-i_ 包括名字开头使用_的函数。

-ix 包括外部及静态的数据符号。

-r 颠倒显示的内容，显示各个函数的调用者并按照被调用者的字母排列顺序排列。

checkeq

checkeq [*files*]

只用于Solaris。检查nroff/troff输入文件中缺少或不平衡的eqn定界符。checkeq会检查.EQ/.EN对和由delim语句指明的内部定界符。

checknr

checknr [*options*] [*files*]

只适用于Solaris。检查nroff/troff源文件中不匹配的定界符和未知的命令。它也会检查来自open/close对的宏，比如.TS和.TE。如果没有*files*参数，则检查标准输入。

当按照其约定进行输入时，checknr可以非常正确地实现其功能：\fP总是终止对一种字体的改变；\s0总是恢复一个点大小的改变。checknr也知道*me*和*ms*两个宏。

options

-amacros

添加来自open/close对的宏对。表示新的宏的6个字符必须紧跟在-a的后面，比如-a.PS.PE表示pic宏。

-ccommands

不要抱怨给出的命令*commands*没有定义。将命令名称按照字

checknr

符串组合在一起，像在 `-a` 中那样。如果你有自己的宏包，该选项很有用。

`-f` 忽略内部的字体改变（`/f`）。

`-s` 忽略内部点大小的改变（`/s`）。

chgrp

chgrp [*options*] *newgroup files*

将一个或多个文件*files*的属主改为*newgroup*。*newgroup*既可以是一个组 ID 号，也可以是保存在 `/etc/group` 目录下的一个组名。要使用该命令，则必须拥有该文件，或者是一个特权用户。

options

`-f` 强行禁止错误消息。

`-h` 改变符号链接的组。通常情况下，`chgrp` 作用于符号链接引用的文件，而不是链接本身（该选项只用于部分 Unix 系统）。

`-R` 通过目录递归向下，包括子目录和符号链接，在其开始时会设置指定的组 ID。

chkey

chkey [*options*]

只适用于 Solaris。提示用户输入登录密码并使用该密码来加密一个新的密钥。参见 **keylogin** 和 **keylogout**。

options

`-p` 使用用户登录密码来重新加密已经存在的密钥。

`-m` *mechanism*

为特定的机制［即 *nisauthconf*（1）允许使用的机制］改变或重新加密一个密钥。

`-s` *database*

更新给出的数据库，该数据库可以是 `files`、`nis` 或 `nisplus` 三者之一。

chmod

chmod [*option*] *mode files*

改变一个或多个文件的访问模式。只有文件的属主或特权用户可以改

chmod

变其模式。可以通过连接来自 *who*、*opcode* 和 *permission* 中的字符来建立模式。*who* 是可选项（如果省略，默认是 a）。只能选择一个 *opcode*。

options

`-f` 禁止由于改变文件模式失败而导致的错误消息。

`-R` 在设置模式时递归降低目录参数。

who

`u` 用户

`g` 组

`o` 其他用户

`a` 所有用户（默认选项）

opcode

`+` 增加权限

`-` 删除权限

`=` 分配权限（并且删除指定字段的权限）

permission

`r` 读

`w` 写

`x` 执行

`s` 设置用户（或组）ID

`t` 粘滞位（sticky bit），保存文本模式（文件）或防止非属主用户删除文件（目录）

`u` 用户的存在权限

`g` 组的存在权限

`o` 其他用户的存在权限

`l` 强行锁定

另外，可以使用一个具有三个数字的序列来指定权限。其中，第一个

chmod

数字表示属主的权限 ;第二个数字表示组的权限 ;第三个数字表示其他用户的权限。通过下面列出的八进制值，可以计算出权限：

4 读

2 写

1 执行

注意：第四个数字可能在该序列的前面，该数字分配下列模式：

4 在执行时设置用户 ID

2 在执行时设置组 ID 或设置强制锁定

1 粘滞位

示例

向 *file* 添加用户执行权限：

```
chmod u+x file
```

下面的两个命令按照属主分配读－写－执行权限（7）;按照组分配读－执行权限（5）；按照其他用户分配只执行权限（1）：

```
chmod 751 file
chmod u=rwx,g=rx,o=x file
```

下面的命令可以向任何用户分配 *file* 的只读权限：

```
chmod =r file
chmod 444 file
chmod a-wx,a+r file
```

下面的命令设置用户 ID，并按照属主分配读－写－执行权限；按照组和其他用户分配读－执行权限：

```
chmod 4755 file
```

chown

chown [*options*] *newowner*[:*newgroup*] *files*

将一个或多个 *files* 的属主修改为 *newowner*。*newowner* 可以是一个用户 ID 号，也可以是 /etc/passwd 目录下的一个登录名称。可选项 *newgroup* 既可以是一个组 ID 号（GID），也可以是 /etc/group 目录

chown

下的文件中的一个组名。如果提供了 *newgroup*，则命令的行为是将一个或多个文件的属主修改为 *newowner*，并且使之属于 *newgroup*。

注意：一些系统接受一个句点或一个冒号作为*newowner*和*newgroup*的分隔符。POSIX 系统要求使用冒号，句点则用于兼容旧的BSD 系统。

options

`-f` 强制禁止错误消息。

`-h` 改变符号链接上的属主。通常情况下，chown 作用于符号链接引用的文件，而不是链接本身（该选项只用于部分 Unix 系统）。

`-R` 沿着目录递归下降，包括子目录和符号链接，并重新设置属主 ID。

cksum

`cksum [`*`files`*`]`

只用于 Solaris。计算并输出各个文件的循环冗余检验（Cyclic Redundancy Check, CRC）。CRC算法基于用于以太网信息包中的多项式。对于每个文件，cksum 会输出一个表格行：

`sum count filename`

其中，*sum* 是CRC；*count* 是文件中的字节数；*filename* 是文件名。如果使用了标准输入，则可以省略文件名。

clear

`clear`

清除终端显示。

cmp

`cmp [`*`options`*`]` *`file1 file2`*

比较*file1*和*file2*。如果*file1*或*file2*是-，则使用标准输入。参见**comm**和 **diff**。退出代码意义如下：

`0` 两个文件相同。

`1` 两个文件不同。

`2` 文件不可访问。

cmp

options

`-l` 对于每处区别，会以十进制形式输出字节数，并以八进制输出不同的字节。

`-s` 默默工作。不输出任何内容，但是返回退出代码。

示例

如果两个文件相同（退出代码是 0），则会输出一条消息：

```
cmp -s old new && echo 'no changes'
```

col

`col [`*`options`*`]`

一个处理反转的换行符和转义字符的后处理过滤器，允许来自 `tbl`（或者偶尔来自 `nroff`）的输出以合理的形式显示在终端上。

options

`-b` 忽略退格字符。当输出参考页时有用。

`-f` 处理半行垂直移动，但不是反转行的移动（通常情况下，半行的输入动作会显示在下一个整行上）。

`-p` 当作规则字符来输出未知的转义序列（通常被忽略），该选项会混淆输出，所以不推荐使用。

`-x` 通常情况下，`col` 通过将空格序列转换为制表符来节约输出时间。使用 `-x` 可以禁止这种转换。

示例

通过 `tbl` 和 `nroff` 运行 *file*，然后通过 `col` 和 `more` 进行过滤后捕获屏幕的输出：

```
tbl file | nroff | col | more
```

将参考页的输出保存到 *file*`.print` 中，并删除输出中的退格字符（否则会显示为 `^H`）：

```
man file | col -b > file.print
```

comb

`/usr/ccs/bin/comb [`*`options`*`]` *`files`*

一条 SCCS 命令，参见第十八章。

comm

comm [*options*] *file1 file2*

比较排序文件*file1*和*file2*共有的行，会产生三列输出：*file1*中独有的行；*file2*中独有的行；两个文件共有的行。comm与diff的相似之处是它们都用来比较两个文件。另外，comm的使用方法类似于uniq，comm在两个排序文件中选择重复或独有的行，而uniq只是在同一个排序文件中选择重复或独有的行。

options

- 读取标准输入。

-1 禁止输出第 1 列。

-2 禁止输出第 2 列。

-3 禁止输出第 3 列。

-12 只输出第 3 列中的行（即 *file1* 和 *file2* 中共有的行）。

-13 只输出第 2 列中的行（即 *file2* 中独有的行）。

-23 只输出第 1 列中的行（即 *file1* 中独有的行）。

示例

比较电影排行榜前 10 名的两个列表，并显示两个列表中都有的项：

```
comm -12 shalit_top10 maltin_top10
```

compress

compress [*options*] [*files*]

减小使用自适应 Lempel-Ziv 编码的一个或多个*files*的大小，并将其移动到文件*file*.z中。可以使用 uncompress 或 zcat 恢复原文件。

如果文件名为 -，或者没有文件名，则 compress 读取标准输入。

注意：Unisys公司声称拥有compress压缩算法的专利权。目前，通常优先使用 gzip 进行文件的压缩。

options

-b*n* 将编码中的位数限制到 *n*，*n* 是 9~16，16 是默认值。*n* 值越小，产生的文件越大，对文件进行的压缩程度越低。

-c 写入到标准输出中（没有对文件进行修改）。

compress

`-f` 进行无条件的压缩。也就是说，在覆盖文件以前不进行提示，即使压缩后文件比原来的文件大也不提示。

`-v` 输出文件压缩后减小的百分比。

cp

cp [*options*] *file1 file2*
cp [*options*] *files directory*

将 *file1* 拷贝到 *file2* 中，或者将一个或多个文件拷贝到相同目录 *directory* 下的同名文件中。如果目标是一个已经存在的文件，则会覆盖该文件；如果目标是一个已经存在的目录，则文件会拷贝到该目录下（拷贝时不会覆盖目录）。如果其中的一个输入是目录，则使用 `-r` 选项。

options

`-i` 在覆盖已经存在的文件以前进行提示，以便进行确认（`y` 即为 yes）。

`-p` 保存修改时间和拷贝文件的权限模式（通常 `cp` 提供调用用户的权限）。

`-r` 递归地将一个目录、该目录下的文件和其子目录拷贝到一个目标目录 *directory* 中，并复制目录树结构。（当至少有一个源文件 *file* 的参数是一个目录时，该选项与第二个命令行参数一起使用）。一定要记住，拷贝时符号和固定链接也会像真实文件一样拷贝，并且不会保存原始目录树的链接结构。

示例

将两个文件拷贝到它们的父目录中（仍使用以前的名字）：

```
cp outline memo ..
```

cpio

cpio *control_options* [*options*]

将文件档案拷贝到磁带、磁盘或本地计算机的另一个位置；或者从这些地方拷贝文件。`-i`、`-o` 或 `-p` 三个控制选项会接受不同的选项（参见 **pax** 和 **tar**）。

cpio

cpio -i [*options*] [*patterns*]

拷贝（选取）名字与选择的模式相匹配的文件。每种模式可以包括来自 Bourne shell 的文件名元字符（应该对模式加引号或转义，以便它们可以由 cpio 命令而不是 shell 进行解释）。如果没有使用任何模式，则会拷贝所有的文件。在选取文件时，不会使用保存在文档中的旧版本覆盖已经存在的文件（除非使用了 -u 选项）。

cpio -o [*options*]

拷贝出在标准输入中给出名字的一系列文件。

cpio -p [*options*] *directory*

将文件拷贝到相同系统的另一个目录中，目标路径使用 *directory* 的相对路径。

有效选项的比较

-i、-o 和 -p 选项可使用的值分别对应下面的第一行、第二行和第三行内容［为了更清晰一些，选项前面没有使用中划线（-）］。

```
i: 6  b B c C d E f H I k  m M r R s S t u v V
o: a A  B c C       H     L  M O          v V
p: a          d         l L m  P R       u v V
```

options

-a 重新设置输入文件的访问次数。

-A 向一个档案中追加文件（必须与 -O 一起使用）。

-b 交换字节和半字，字是 4 个字节的。

-B 每条记录使用 5120 个字节的块输入或输出（默认情况下每条记录是 512 字节）。

-c 读或写头信息并当作 ASCII 字符。当源和目标计算机是不同的类型时，该选项很有用。

-C *n*

与 -B 相似，但是块大小可以是任何正整数 *n*。

-d 根据需要建立目录。

cpio

`-E` *file*
从档案中的文件列表中提取文件名。

`-f` 颠倒拷贝的意义。拷贝除了匹配模式文件以外的所有文件。

`-H` *format*
按照 *format* 格式读或写文件头信息。格式的值是 `bar`（bar 格式的头或文件，只读的，只适用于 Solaris）、`crc`（包含扩展设备号的 ASCII 头）、`odc`（包含较小设备号的 ASCII 头）、`ustar`（IEEE/P1003 数据交换标准头）或 `tar`（tar 头）。Solaris 也允许使用 `CRC`、`TAR` 和 `USTAR`。

`-I` *file*
读取文件 *file* 作为一个输入档案。

`-k` 跳过损坏的文件头及 I/O 错误。

`-l` 链接文件代替拷贝文件，只能与 `-p` 一起使用。

`-L` 紧跟符号链接。

`-m` 保留以前的文件修改时间。

`-M` *msg*
当转换介质时输出 *msg*。在消息中可以使用变量 `%d` 作为下一种介质的数字 ID。`-M` 只有与 `-I` 和 `-O` 一起使用时才有效。

`-O` *file*
指示将输出保存到文件 *file* 中。

`-P` 用于保存 ACL，只能与 `-p` 一起使用。只适用于 Solaris。

`-r` 交互修改文件的名字。

`-R` *ID*
重新向用户分配其登录标识符是 *ID* 的文件所有权和组信息（只适用于特权用户）。

`-s` 交换字节。

`-S` 交换半字。

`-t` 以表格形式输出输入的内容（没有建立文件）。当与 `-v` 选项一起使用时，类似于 `ls -l` 的输出。

`-u` 无条件进行拷贝，旧文件可以覆盖新文件。

cpio

`-v` 输出一系列文件名。

`-V` 为读写文件输出一个点（这样可以说明`cpio`在运行，但是不会使屏幕输出混乱）。

`-6` 处理一个PWB Unix第6版档案格式的文件。只能与`-i`选项一起使用，与`-c`和`-H`选项是互斥的。

示例

产生使用`find`命令的旧文件的列表，并将列表作为`cpio`的输入：

```
find . -name "*.old" -print | cpio -ocBv > /dev/rmt/0
```

恢复磁带驱动器上所有文件名中包含了“ save ”的文件（如果需要会建立子目录）：

```
cpio -icdv "*save*" < /dev/rmt/0
```

移动一个目录树：

```
find . -depth -print | cpio -padml /mydir
```

crontab

`crontab` [*file*]
`crontab` *options* [*user*]

在当前的crontab文件上执行`crontab`命令，或者将一个指定的crontab文件*file*添加到crontab目录中。特权用户可以通过在任何选项后面提供一个*user*，为另一个用户执行`crontab`命令。

一个crontab文件是一系列命令的列表，每个命令一行，每一行会在给定时间自动执行。在指定各条命令执行时间以前必须提供它们的号码，这些号码使用5个字段，如下所示：

```
Minute          0～59
Hour            0～23
Day of month    1～31
Month           1～12
Day of week     0～6, 0 = Sunday
```

多个值之间使用逗号（,）间隔，连字符（-）表示一个范围，星号（*）则表示所有可能的值。例如，假设crontab的内容如下：

```
59 3 * * 5       find / -print | backup_program
0 0 1,15 * *     echo "Timesheets due" | mail user
```

crontab

第一条命令将在每周 5 上午 3:59 对系统文件进行备份，第二条命令会在每个月的 1～15 号通过邮件向用户发送一个提示。

options

`-e` 编辑该用户当前的 crontab 文件（或者建立一个 crontab 文件）。

`-l` 列出该用户 crontab 目录中的文件。

`-r` 删除该用户 crontab 目录中的文件。

cscope

cscope [*options*] *files*

用于在一个或多个 C、lex 或 yacc 源文件 *files* 中查找代码段的交互工具，生成一个符号交叉引用（默认名称是 `cscope.out`），然后调用一个菜单。该菜单会提示用户搜索函数、宏、变量或预处理程序指令等。键入`?`来显示交互的命令。如果需要（也就是说，如果文件名或文件的内容已经改变），接下来会调用 `cscope` 命令重新建立交叉引用。源文件名可以保存到文件`cscope.files`中。可以指定该文件来代替 *files*。当选项 `-I`、`-p`和`-T`放置在`cscope.files`中时也被识别。

options

`-b` 只建立一个符号交叉引用。

`-c` 以 ASCII 符号形式生成输出（不压缩数据）。

`-C` 查找时忽略大小写的区别。

`-d` 不更新符号交叉引用。

`-e` 在文件之间不显示 `^E` 提示符。

`-f` *out*
将交叉引用文件命名为 *out* 而不是 `cscope.out`。

`-i` *in*
检查名字列在 *in* 中而不是列在 `cscope.files` 中的源文件。

`-I` *dir*
在搜索默认目录（`/usr/include`）以前先搜索目录 *dir* 中包括的文件。`cscope` 会首先搜索当前目录，然后是 *dir*，最后是默认目录。

cscope

`-l` 按照行模式运行，主要用于屏幕编辑器中。

`-L` 与 *-n pat* 一起使用来进行单个的搜索。

`-p` *n*

显示文件名路径中的最后 *n* 部分。默认是 `1`（文件名），`0` 表示不显示文件名。

`-P` *path*

与 `-d` 选项一起使用，在存在的交叉引用中在文件名前添加 *path*，从而可以在不修改已经生成交叉引用的目录的情况下运行 cscope。

`-s` *dir*

在目录 *dir* 中而不是当前的目录中搜索源文件。

`-T` 只匹配 C 语言符号中的前 8 个字符。

`-u` 无条件地生成交叉引用（假设修改了所有文件）。

`-U` 忽略文件的时间信息（假设没有修改任何文件）。

`-V` 在屏幕第一行输出 cscope 的版本。

`-n` *pat*

进入输入的字段 *n*（从 0 开始），然后找到 *pat*。

csh

csh [*options*] [*arguments*]

使用语法类似于 C 语言的命令解释程序。csh（C shell）执行来自终端或文件的命令，参见第五章中的信息，包括命令行选项。

csplit

csplit [*options*] *file arguments*

将一个文件分成多个段，将各个段放置到名字为 `xx00` 到 `xx`*n*（*n*<100）的一系列文件中，按参数 *arguments* 指定的模式将文件断开。参见 **split**。

options

`-f` *file*

将新文件命名为 *file*`00` 到 *fileN*（默认值是 `xx00` 到 `xx`*n*）。

`-k` 保存最新建立的文件，即使出现了一个错误（通常情况下会删

csplit

除出现错误的文件)。当需要指定任意大的重复参数{*n*}且不希望“ out of range ”错误删除新文件时，该选项非常有用。

`-s` 禁止所有的字符计数。

arguments

参数可以是下面列出的任意一个表达式或多个表达式的组合,它可以包含空格或其他使用单引号括起来的字符。

/expr/

从当前行开始，到包含正则表达式*expr*的行结束，建立包括这些行的文件。该参数可以使用可选的后缀 *+n* 或 *-n*，其中 *n* 是 *expr* 所在行的下面或上面一行的行号。

%expr%

与*/expr/*相同，只是不会建立包括*expr*所在行前面的行的文件。

num 建立一个包括当前行到行号为 *num* 的行之间所有行的文件。

{*n*} 重复使用参数 *n* 次，可以紧跟上面的任何参数。文件将从包含 *expr* 的行或行号为 *num* 的块处分开。

示例

将文件 `novel` 分割成以章为单位的 20 个文件：

```
csplit -k -f chap. novel '%CHAPTER%' '{20}'
```

使用 `address_list` 数据库中的数据建立 100 个地址文件（从 `xx00` 到 `xx99`），每个文件包括 4 行：

```
csplit -k address_list 4 {99}
```

ctags

`ctags [`*options*`]` *files*

为在指定的 C、Pascal、FORTRAN、`yacc` 或 `lex` 源文件中定义的函数或宏建立一个名称列表，Solaris 中的 `ctags` 命令还可以处理 C++ 源文件。输出列表（默认命名为 tags）包含的行形式如下：

```
name    file    context
```

其中 *name* 是函数或宏的名称；*file* 是定义 *name* 的源文件；*context* 是

ctags

一种搜索模式，该模式用于显示代码中包含了 *name* 的行。在建立了标签列表后，就可以在任何文件及类型中调用 `vi`：

```
:set tags=tagsfile
:tag  name
```

这些命令将会把 `vi` 编辑器切换为与 *tagsfile*（与 `-f` 一起指定）中列出的与 *name* 关联的源文件。

options

`-a` 向已经存在的标签列表中添加新的标签输出。

`-B` *context* 使用反向搜索模式。

`-`*`ftagsfile`*
将输出放置到文件 *tagsfile*（默认是 `tags`）中。

`-F` *context* 使用前向搜索模式（默认值）。

`-t` 包括 C `typedefs` 作为标签。

`-u` 更新标签文件来反映函数的新位置（比如，当函数被移动到一个不同的源文件中时）。该选项会删除旧的标签，并添加新的标签。

`-v` 建立各个函数、源文件和页号（1 页 =64 行）的列表（索引）。`-v` 会建立一个与 `vgrind` 一起使用的文件。

`-w` 禁止警告消息。

`-x` 产生各个函数、它的行号、源文件和上下文的一个列表。

示例

在用于所有 C 程序的 `Taglist` 中保存标签：

```
ctags -f Taglist *.c
```

更新标签并保存到文件 `Newlist` 中：

```
ctags -u -f Newlist *.c
```

ctrace

`ctrace [`*`options`*`] [`*`file`*`]`

调试一个 C 程序。`ctrace` 会读取 C 源文件 *file* 并向标准输出中输出

ctrace

一个修改的版本。通用的选项是 `-f` 和 `-v`。`ctrace` 也接受 `cc` 选项 `-D`、`-I` 和 `-U`。

options

`-e` 按照浮点数输出变量。

`-f` *functions*
只跟踪指定的函数 *functions*。

`-l` *n*
跟随一个语句循环 *n* 次（默认是 20 次）。

`-o` 以八进制的形式输出变量。

`-p` *s*
通过函数 *s*（默认是 `printf`）打印跟踪输出。

`-P` 在跟踪以前运行 C 预处理程序。

`-Q`*c* 在输出中输出有关 `ctrace` 的信息（如果 *c*=`y`），或者禁止信息（如果 *c*=`n`，默认值）。

`-r`*file*
将跟踪函数包修改为 *file*（默认是 `runtime.c`）

`-s` 禁止某些冗余代码。

`-t`*n* 跟踪每个语句中的 *n* 个变量（默认是 10，最大是 20）。

`-u` 按照无符号数输出变量。

`-v` *functions*
不跟踪指定的函数 *functions*。

`-V` 在标准错误中输出版本信息。

`-x` 按照浮点数输出变量。

cut

`cut` *options* [*files*]

从一个或多个文件 *files* 中选择列或字段的一个列表。必须指定 `-c` 或 `-f`，该列表包括一系列整数，可以使用逗号（,）来分隔值或使用中划线（`-`）来指定一个范围（比如 `1-10`、`15`、`20` 或 `50-`）。参见 **paste** 和 **join**。

cut

options

-b *list*

*list*指定了字节的位置，而不是字符的位置。当使用多字节字符时，该选项很重要。使用该选项时，文件中的每一行应该小于或等于1023个字节。该选项只用于Solaris。

-*clist*

剪切在*list*中标识的字符位置。

-d*c* 与-f一起来指定字段的定界符是*c*（默认是制表符）。特殊字符（例如空格）必须用引号引起来。

-*flist*

剪切在*list*中标识的字段。

-n 不分割字符。当与-b一起使用时，cut不会分隔多字节的字符。只适用于Solaris。

-s 与-f一起使用，来禁止没有定界符的行。

示例

从/etc/passwd目录中提取用户名和真正的名字：

```
cut -d: -f1,5 /etc/passwd
```

找出登录的用户，但是只列出登录用户名称：

```
who | cut -d" " -f1
```

剪切文件*file*的第4列字符，将它们粘贴到同一个文件中作为第一列，并将结果发送到标准输出上显示：

```
cut -c4 file | paste - file
```

cxref

cxref [*options*] *files*

对各个C源文件*files*建立一个交叉引用表格，该表格列出了所有的符号，表格的各个列分别用于显示各个文件的名称、相关的函数、文件及行。在表格中，赋值符号用=标记，声明符号用-标记，定义符号用*标记。cxref也接受cc选项-D、-I和-U。

cxref

options

`-c` 报告一个表格中的所有文件。

`-C` 不执行第二次传递的 `cxref` 命令，并将第一次传递的输出保存到 `.cx` 文件中（与 `lint` 和 `cc` 中的 `-c` 选项相似）。

`-d` 通过省略输出声明来简化报告。

`-F` 使用完整的路径名而不只是文件名来输出文件。

`-l` 不输出局部变量。

`-L[`*n*`]`
将 LINE 字段限制为 *n* 列（默认是 5 列）。

`-o` *file*
将输出发送到文件 *file* 中。

`-s` 安静模式，不输出输入的文件名。

`-t` 格式化为 80 列的列表。

`-V` 在标准错误上输出版本信息。

`-w[`*n*`]`
格式化 *n* 列的最大宽度（默认是 80，*n* 必须大于 50）。

-Wn1,n2,n3,n4
设置各（或任何）列的宽度为 *n1*、*n2*、*n3* 或 *n4*（默认值分别是 15、13、15 和 20），列标题分别为 NAME、FILE、FUNCTION 和 LINE。

date

`date [`*option*`] [+`*format*`]`
`date [`*options*`] [`*string*`]`

第一种格式会输出当前的日期和时间，并指定一种可选择的显示格式 *format*；第二种格式中，特权用户通过提供一个数字形式的字符串 *string* 来设置当前的日期。*format* 可以包含文字正文字符串（空格必须用引号括起来）以及字段描述符，可设置值如下（该列表显示了一些合理的分组）：

format

`%n` 插入一个换行符。

date

%t	插入一个制表符。
%m	一年中的月份（01~12）。
%d	一个月中的日（01~31）。
%y	使用最后两位数字表示一年（00~99）。
%D	使用 %m/%d/%y 格式的日期。
%b	简写的月份名称。
%e	一个月中的日（1~31），单个数字前面填充一个空格。
%Y	四位数字表示的年（比如 1996）。
%g	在一个世纪内基于星期的年（00~99）。只适用于 Solaris。
%G	基于星期的年，包括世纪（0000~9999）。只适用于 Solaris。
%h	与 %b 选项相同。
%B	完整的月份名称。
%H	使用 24 小时格式表示的小时（00~23）。
%M	表示分钟（00~59）。
%S	表示秒（00~61）。61 允许闰秒和双闰秒。
%R	使用 %H:%M 格式表示的时间。
%T	使用 %H:%M:%S 格式表示的时间。
%k	表示小时（24 小时的时钟，0~23），单个数字前面要加一个空格，只适用于 Solaris。
%l	表示小时（12 小时的时钟，1~12），单个数字的时间前面要加一个空格。只适用于 Solaris。
%I	使用 12 小时格式表示的小时（01~12）。
%p	表示上午或下午的字符串（默认是 AM 或 PM）。
%r	以 %I:%M:%S %p 格式表示的时间。
%a	简写的工作日。
%A	完整的工作日。
%w	表示星期几（星期天 = 0）。
%u	十进制数表示的工作日（1~7），星期天 =1，只用于 Solaris。

date

%U 一年中的星期号（00~53），星期天是一周的开始。

%W 一年中的星期号（00~53），星期一是一周的开始。

%V ISO-8601 星期号（01~53）。按照 ISO-8601 标准，一周从星期一开始，每年的第一周必须同时包含一月的第 4 天和该年的第二个星期四。如果一月的星期一是 2、3、4 号，则前面的天数属于前一年的最后一周。该选项只适用于 Solaris。

%j 一年的儒略日（001~366）。

%Z 时区名称。

%x 特定于国家的日期格式。

%X 特定于国家的时间格式。

%c 特定于国家的时间和日期格式（默认是 %a %b %e %T %Z %Y，例如，Mon Feb 1 14:30:59 EST 1993）。

实际的格式由 *strftime(3)* 库程序决定。在 Solaris 中，特定于国家的格式依赖于 LC_CTYPE、LC_TIME、LC_MESSAGES 和 NLSPATH 环境变量的设置。

options

-a *s.f*

（只用于特权用户。）逐渐调整系统时钟，直到它从它所认为的“当前”时间移动 *s* 秒（这允许在系统运行过程中连续对时钟进行微调）。*f* 表示每次移动几分之一秒。默认情况下，时钟会加速移动。在 *s* 前面加 *a*- 来降低速度。

-u 使用格林威治标准时间（UTC）来显示或设置时间。

设置日期的字符串

特权用户可以通过提供一个数字形式的字符串设置日期。字符串包含的时间、天和年可以使用如下三种方式进行连接：*time*、[*day*]*time* 或者 [*day*]*time*[*year*]。注意，不要输入括号。

time

有两个数字表示小时，另外两个数字表示分钟（*HHMM*）。*HH* 使用 24 小时的格式。

date

day 有两个数字表示月份，另外两个数字表示月份中的某一天（*mmdd*），默认是当前的日期和月份。

year

可以使用4位数字表示一年，也可以使用后2位数字表示。默认是当前的年。

示例

设置日期为1999（99）年7月1日（0701）上午4点钟（0400）的格式如下：

```
date 0701040099
```

下面的命令：

```
date +"Hello%t Date is %D %n%t Time is %T"
```

会产生如下格式的日期：

```
Hello      Date is 05/09/93
           Time is 17:53:39
```

dc

dc [*file*]

一个交互的桌面计算器程序，可以执行任意精度的整数计算（输入可以来自一个文件*file*）。通常情况下，不能直接运行dc命令，因为它是通过bc（参见**bc**）进行调用的。dc提供了执行各种算术运算的单字符命令和操作符。dc的工作原理类似于一个反向波兰计算器，因此，操作符和命令会紧跟在受其影响的数字后面。操作符包括+ - / * % ^（正如在C语言中一样，尽管^表示求幂）。一些简单的命令如下：

p 输出当前的结果。

q 退出dc。

c 清除堆栈中所有的值。

v 获得平方根。

i 改变输入基数，类似于bc命令的ibase。

o 改变输出基数，类似于bc命令的obase。

dc

k 设置比例因子（小数后数字的个数）。类似于 bc 的命令 scale。

! 行的剩余部分是 Unix 命令。

示例

```
3 2 ^ p          求 3 的平方，然后输出结果
9
8 * p            当前的值（9）乘上 8，然后输出结果
72
47 - p           72 减去 47，然后输出结果
25
v p              求 25 的平方根，然后输出结果
5
2 o p            按照基数 2 输出当前的结果
101
```

注意，除了数字之间需要空格外，其他位置不需要空格。

dd

dd [*option=value*]

建立输入文件的一个副本（if=），如果没有指定输入文件则使用标准输入，并使用限定的条件，然后将结果发送给输出文件（如果没有指定 of，则使用标准输出）。可以使用任何数量的选项，尽管 if 和 of 是最常用的选项，并且一般要首先指定这些选项。由于 dd 可以处理任意大小的块，当在原始的物理设备之间转换时，该命令是非常有用的。

options

bs=*n*

表示将输入和输出块的大小设置为 *n* 个字节。该选项可以取代 ibs 和 obs。

cbs=*n*

设置转换缓冲区（逻辑的记录长度）的大小是 *n* 字节。只用于转换标志是 ascii、asciib、ebcdic、ebcdicb、ibm、ibmb、block 或 unblock 的情况。

conv=*flags*

按照下面列出的一个或多个标志 *flags* 转换输入（使用逗号间隔）。最前面的6个标志是互斥的，接下来的两个标志是互斥的，再接下来两个也一样。

dd

`ascii` EBCDIC 到 ASCII 的转换。

`asciib` EBCDIC到ASCII的转换，使用与BSD兼容的转换，只适用于 Solaris。

`ebcdic` ASCII 到 EBCDIC 的转换。

`ebcdicb` ASCII到EBCDIC的转换，使用与BSD兼容的转换，只适用于 Solaris。

`ibm` 按照 IBM 约定进行 ASCII 到 EBCDIC 的转换。

`ibmb` 按照IBM约定进行ASCII到EBCDIC的转换，使用与 BSD 兼容的转换，只适用于 Solaris。

`block` 长度可变的记录（也就是说，使用换行符终止）到固定长度记录的转换。

`unblock` 固定长度的记录到可变长度记录的转换。

`lcase` 大写字母到小写字母的转换。

`ucase` 小写字母到大写字母的转换。

`noerror` 当出现错误时继续进行处理（在一行中最多为 5 处）。

`notrunc` 不截短输出文件。这会保存在输出文件中dd调用未覆写的块。只适用于 Solaris。

`swab` 交换所有的字节对。

`sync` 将所有的输入块填充到 ibs 中。

`count=`*n*

只拷贝 *n* 个输入块。

`files=`*n*

拷贝 *n* 个输入文件（比如，来自磁带），然后退出。

`ibs=`*n*

设置输入块的大小是 *n* 字节（默认是 512）。

`if=`*file*

从 *file* 中读取输入（默认是标准输入）。

`obs=`*n*

设置输出块的大小是 *n* 个字节（默认是 512）。

dd

of=*file*
将输出写入到文件 *file* 中（默认是标准输出）。

iseek=*n*
从输入文件的开头位置查找 *n* 个块（与 skip 相似，但是对于磁盘文件的输入更有效）。

oseek=*n*
从输出文件的开头位置查找 *n* 个块。

seek=*n*
与 oseek 一样（为了兼容性而保留）。

skip=*n*
跳过 *n* 个输入块，主要用于磁带。

通过分别添加字母 k、b 和 w，可以将表示大小的值（*n*）乘上因数 1024、512 或 2。也可以使用字母 x 作为两个数字的乘法运算符。

示例

将一个输入文件中的所有字母转换为小写：

```
dd if=caps_file of=small_file conv=lcase
```

提取变长的数据，并按照固定长度将其写入到输出 out 中：

```
data_retrieval_cmd | dd of=out conv=sync,block
```

delta

/usr/ccs/bin/delta [*options*] *files*

一条 SCCS 命令，参见第十八章。

deroff

deroff [*options*] [*files*]

删除所有的 nroff/troff 请求和宏、反斜线转义序列、来自指定文件 *files* 的 tbl 和 eqn 构造器等。

options

-i 忽略 .so 和 .nx 请求，只用于 Solaris。

-mm 禁止出现在 *mm* 宏行中的文本（也就是说，在输出段落时不输出标题）。

deroff

`-ml` 与`-mm`作用相同，但是它还会删除*mm*宏创建的列表，例如`.BL/.LE`、`.VL/.LE`构造器（一般不处理嵌套列表）。

`-ms` 禁止出现在*ms*宏中的文本（也就是说，输出段落时不输出标题）。只用于Solaris。

`-w` 输出的文本以列表形式显示，每个单词占一行。参见**xargs**命令中的示例。

df

`df [`*`options`*`] [`*`name`*`]`

报告在所有安装的文件系统或给出的*name*中可用的空闲磁盘块和信息节点的数量（可使用`-F`检查未安装的文件系统）。*name*可以是设备名称（比如`/dev/dsk/0s9`）、安装点的目录名称（比如`/usr`）、目录名称或远程文件系统（比如一个NFS文件系统）的名称。除了列出的选项外，不同的文件系统类型或`df`模块中还有一些特定的附加选项。

options

`-a` 提供有关所有文件系统的信息，甚至包括那些通常在`/etc/mnttab`中标记为忽略的信息。该选项只用于Solaris。

`-b` 只输出空闲的磁盘空间数量，以千字节作为计量单位。

`-e` 只输出空闲文件的数量。

`-F` *`type`*
报告*type*指定的未安装的文件系统。可以通过文件`/etc/vfstab`查看可用的类型。

`-g` 输出整个`statvfs`结构（覆盖其他的输出选项）。

`-i` 只有`/usr/ucb/df`，使用与`df -k`相似的格式来输出已经使用及可以使用的信息节点数量。

`-k` 输出分配的空间，以千字节表示（通常不与其他选项一起使用）。该选项按照传统上由`df`的BSD版本使用的格式产生输出。

`-l` 该选项只报告本地文件系统。

`-n` 只输出文件系统类型的名字。不使用其他参数时，`-n`会列出安装的所有文件系统的类型。

df

`-o` *suboptions*
提供使用逗号间隔的所有特定于类型的子选项 *suboptions* 的一个列表。

`-t` 报告总的已分配空间及空闲空间。

`-V` 回显命令行但是不执行命令。

diff

`diff [`*options*`] [`*diroptions*`]` *file1 file2*

`diff`报告*file1*和*file2*两个文件中不同的行。输出内容由来自各个文件中的上下文的多个行组成，并使用符号 < 标记 *file1* 中的文本，使用符号 > 标记 *file2* 中的文本。上下文行的前面会放置一个将 *file1* 转换到 *file2* 的`ed`命令（`a`、`c`或`d`）。如果有一个文件的名字为 `-`，则会读取标准输入；如果其中的一个文件是目录，则 `diff`会在该目录中查找对应于另一个参数的文件名（比如`diff my_dir junk`与`diff my_dir/junk junk`作用相同）。如果两个参数都是目录，则 `diff`会报告所有名称相同的文件对（比如`olddir/program`和`newdir/program`）中不同的行；另外，`diff`命令会列出一个目录中独有的文件名及两个目录共有的子目录。参见**bdiff**、**cmp**、**comm**、**diff3**、**dircmp** 和 **sdiff** 等命令。

options

`-c`、`-C`、`-D`、`-e`、`-f`、`-h`和`-n`中的任何两个选项都不能在一起使用（它们都是互斥的）。

`-b` 忽略重复的空格和行结尾处的空格，多个空格会当作一个空格看待。

`-c` 以交替的格式产生输出，具有三行上下文（通常称为“context diff”）。

`-C`*n* 与 `-c` 选项相似，但是产生 *n* 行的上下文。

`-D` *def*
将 *file1* 和 *file2* 两个文件合并进一个文件中，该文件包括了C预处理程序条件指令（`#ifdef`）。先定义 *def*，然后再编译产生文件 *file2*；编译时没有定义 *def* 会产生文件 *file1*。

diff

`-e` 产生命令（a、c、d）的一个脚本，以便使用 ed 编辑器从文件 *file1* 重新建立文件 *file2*。

`-f` 产生一个脚本以便从 *file2* 重新建立文件 *file1*，该脚本使用与 `-e` 选项相反的顺序，所以不能用于 ed。

`-h` 实现非完全的比较（但是有希望速度快一些），可能无法显示复杂的区别（比如，许多改变的长扩展），不能与 `-e` 和 `-f` 一起使用。

`-i` 忽略大小写的区别。

`-n` 与 `-f` 功能相似，但是只计算已改变的行数。`rcsdiff` 也以这种方式工作。

`-t` 在输出行中扩展制表符，主要用于保留 `-c` 格式改变的缩进。

`-w` 与 `-b` 作用相似，但是会忽略所有的空格和制表符，比如，`a + b` 与 `a+b` 相等。

只有当两个文件参数都是目录时，下面的 *diroptions* 才是有效的：

diroptions

`-l` 长格式，输出会使用 `pr` 标注页数，以便 `diff` 为每个文件列出的信息从新的一页开始显示，其他比较列在后面。

`-r` 在共同的子目录中为各个文件递归运行 `diff`。

`-s` 报告相同的文件。

`-Sfile`
从 *file* 文件开始进行目录比较，并跳过按字母顺序排在 *file* 前面的文件。

diff3

`diff3 [options] file1 file2 file3`

比较三个文件并使用下面的代码报告它们的区别：

`====` 三个文件全不同。

`===1` *file1* 与其他两个文件不同。

`===2` *file2* 与其他两个文件不同。

`===3` *file3* 与其他两个文件不同。

diff3

options

`-e` 建立一个 ed 脚本来将 *file2* 和 *file3* 两个文件的所有区别都保存到文件 *file1* 中。

`-E` 与 `-e`作用相同，但是它会使用尖括号标记三个文件中所有不同的行。

`-x` 建立一个 ed 脚本来将三个文件的所有区别都保存到文件 *file1* 中。

`-X` 与 `-x`作用相同，但是它会使用尖括号标记三个文件中所有不同的行。

`-3` 建立一个 ed 脚本来将 *file1* 和 *file3* 两个文件的所有区别都保存到文件 *file1* 中。

diffmk

`diffmk` *`oldfile newfile markedfile`*

该程序主要用于检查一个文档的多个版本之间修改的内容。`diffmk` 会比较一个文件的两个版本（旧文件 *oldfile* 和新文件 *newfile*）并建立包含 `troff` “change mark（改变标记）”请求的第三个文件（*markedfile*）。当 *markedfile* 使用 `nroff` 或 `troff` 格式化时，两个文件的区别会在页面边缘的空白部分进行标记（通过 `.mc` 请求）。`diffmk` 使用竖线符号（`|`）来标记已经改变的行，使用星号（`*`）来标记已经删除的行。注意，即使修改是不合理的（比如，多余的空格、不同的输入行长度等），也会产生修改标记。

示例

如果要使`diffmk`运行在多个文件上，简便的方法是建立目录来保存这些文件的新、旧版本，然后再建立一个目录来保存标记文件：

```
$ mkdir OLD NEW CHANGED
```

将旧文件移动到 OLD 目录中，将新文件移动到 NEW 目录中，然后使用下面的 Bourne shell 脚本：

```
$  cat do.mark
for file
do
   echo "Running diffmk on $file ..."
   diffmk ../OLD/$file $file ../CHANGED/$file
```

diffmk

```
done
```

你必须在保存新文件的目录中运行该脚本：

```
$ cd NEW
$  do.mark Ch*
```

dircmp

dircmp [*options*] *dir1 dir2*

比较 *dir1* 和 *dir2* 的内容，参见 **diff** 和 **cmp**。

options

-d 在不一样的多个文件上面执行 diff。

-s 不报告相同的文件。

-wn 将输出行的长度修改为 *n*（默认是 72）。

dirname

dirname *pathname*

输出 *pathname*，不包括目录的最后一级。主要用于从路径名中提取实际的文件名。参见 **basename**。

dis

/usr/ccs/bin/dis [*options*] *files*

反汇编对象或档案文件 *files*。参见 **as**。

options

-C 显示反改编的 C++ 符号名称，只用于 Solaris。

-d *section*
只对指定段的数据进行反汇编，并输出其偏移量。

-D *section*
与 -d 作用相同，但是会输出数据的实际地址。

-F *func*
只对指定的函数进行反汇编，对于附加的函数则重新使用 -F。

-l *string*
只对库文件 *string*（比如使用 malloc 函数为 libmalloc.a 分配的 *string*）进行反汇编。

dis

`-L` 在包含调试信息的文件（比如，使用 `cc -g` 编译的文件）中查找 C 源标签。

`-o` 以八进制形式输出（默认是十六进制）。

`-t` *section*
与 `-d` 作用相同，但是该选项会输出文本。

`-V` 在标准错误中输出版本信息。

dos2unix

`dos2unix [`*options*`] `*dosfile unixfile*

只用于 Solaris。将使用 DOS 扩展字符集的文件转换为符合 ISO 标准的对应内容。如果 *dosfile* 和 *unixfile* 是同一个文件，则完成转换后两个文件会合并成一个文件。参见 **unix2dos**。

options

`-ascii`
删除多余的回车符，并将 DOS 文件结尾字符转换为 Unix 文件结尾字符（或删除）。

`-iso`
与默认的动作相同。

`-7` 将 8 位的 DOS 图形字符转换为空格字符。

download

`/usr/lib/lp/postscript/download [`*options*`] [`*files*`]`

将一种字体添加到一个或多个 PostScript 文件 *files* 的开头位置。该命令通过直接向 PostScript 规范中添加字体名称，使得在打印 PostScript 文件时可以使用附加的字体。`download` 通过处理以 `%%DocumentFonts:` 开头、后面紧跟一个 PostScript 字体名称列表的 PostScript 注释来决定添加哪种字体。`download` 会装载其名字列在一个映射表中的字体，该映射表链接字体定义的系统文件和 PostScript 名称。Times 系列字体映射表的内容可能如下所示：

```
Times-Bold      times/bold
Times-Italic    times/italic
Times-Roman     times/roman
```

download

开头为一条斜线的文件名被一字不差地使用；否则，会使用与主机字体目录相对的名称。

options

- 读取标准输入。

-f 搜索整个PostScript文件而不仅仅是头注释。头注释如%%DocumentFonts：(atend)将download重定向到文件的结尾。当这样的注释不存在时使用该选项。

-H *fontdir*
使用*fontdir*作为搜索定义字体的文件的目录（默认是/usr/lib/lp/postscript）。

-m *table*
使用文件*table*指定的映射表。*table*中的前导符“/”表示绝对路径；否则（正如在前面的选项中一样），文件名会追加在-H指定的*fontdir*的后面。没有-H选项时，默认是/usr/lib/lp/postscript。

-p *printer*
通常情况下，download会装载驻留在主机上的字体。使用该选项，download首先会检查驻留在打印机*printer*上的字体（查找/etc/lp/printers/printer/residentfonts目录）。

dpost

/usr/lib/lp/postscript/dpost [*options*] [*files*]

用于将troff格式的文件*files*转换为可以打印的PostScript的后处理程序。

options

- 读取标准输入。

-c *n*
将各个页面打印*n*份（默认是1）。

-e 0 | 1 | 2
设置文本的编码为0（默认值）、1或2。较大的编码值会减小输出的大小并提高打印速度，但是可靠性会降低。

dpost

`-F` *dir*
设置字体目录为 *dir*（默认为 `/usr/lib/font`）。

`-H` *dir*
设置主机的字体目录是 *dir*。该目录下的文件必须用于描述 PostScript 字体，并且其文件名对应两个字符的 `troff` 字体。

`-L` *file*
设置 PostScript 序言为 *file*（默认是 `/usr/lib/postscript/dpost.ps`，在 Solaris 上是 `/usr/lib/lp/postscript/dpost.ps`）。

`-m` *scale*
通过因子 `scale` 来（成倍）增加逻辑页面的大小（默认是 1.0）。

`-n` *n*
在每个输出页面上打印 *n* 个逻辑页面（默认是 1 个）。

`-o` *list*
只显示使用逗号分隔的列表 *list* 中包含的页面。页面的范围由 *n*-*m* 指定。

`-O` 从输出中省略 PostScript 图像。主要用于网络环境中。

`-p` *layout*
将 *layout*（页面布局）指定为 `portrait`（页面的长边是垂直的，是默认值）或 `landscape`（页面的长边是水平的）。*Layout* 可以简写为 `p` 或 `l`。

`-T` *device*
使用 *device* 很好地描述可用的 PostScript 字体，默认是 `post`，使用 `dpost` 可以读取 `/usr/lib/font/devpost` 中的二进制文件。一般情况下不要使用 `-T` 选项，通常情况下，如果系统中有 PostScript 字体，则最好使用该字体。

`-w` *n*
使用 *n* 个点宽（默认是 0.3）的行来绘制 `troff` 图形（比如 `pic`、`tbl`）。

`-x` *n*
在 x 坐标轴上的坐标是 *n* 英寸（如果 *n* 是正数，则在右边）。

dpost

`-y` *n*

在 y 坐标轴上的坐标是 *n* 英寸（如果 *n* 是正数，则在下边）。默认原点是页面的左上角。

示例

```
pic file | tbl | eqn | troff -ms -Tpost | dpost -c2 | lp
```

du

du [*options*] [*directories*]

显示磁盘的使用情况，也就是说，显示各个指定目录和其子目录（默认是当前目录）使用的 512 字节大小的磁盘块的数量。

options

`-a` 输出所有文件而不仅仅是子目录使用的磁盘空间的数量。

`-d` 不能同时用于多个文件系统，只用于 Solaris。

`-k` 以千字节为单位输出磁盘信息。

`-L` 用于符号链接，处理链接引用的文件或目录，而不是链接本身。只用于 Solaris。

`-o` 不将子目录的统计信息添加到父目录的汇总中，如果还使用`-s`，也不会产生影响。只用于 Solaris。

`-r` 如果无法访问一个文件或目录，则输出一条“ cannot open（无法打开）”消息。

`-s` 只输出各个指定目录的主要汇总信息。

echo

echo [`-n`] [*string*]

在标准输出中回显参数，通常用于产生来自 shell 脚本的提示。它是一条`/bin/echo`命令。在`/usr/ucb`目录中也有`echo`命令，并且该命令也内置在 Bourne、C 和 Korn shell 中（参见第四章和第五章）。

尽管`echo`在概念上是所有 Unix 命令中最简单的命令，但是在实际使用该命令时却比较复杂，因为该命令可以移植和存在多个不同的版本（考虑使用`printf`替代）。下面概括了这些不同：

echo

版本的区别

`/bin/echo`

不能使用 `-n` 选项，用于解释下面描述的转义序列。

`/usr/ucb/echo`

如果它是第一条命令，则接受 `-n` 选项。不解释转义序列。

Bourne shell `echo`

不能使用 `-n` 选项，用于解释下面描述的转义序列，不包括 `\a`。

C shell `echo`

如果它是第一条命令，则接受 `-n` 选项。不解释转义序列。

Korn shell `echo`

在 $PATH 目录中进行搜索，然后执行它找到的第一个版本的 `echo` 的功能。

转义序列

`\a`	报警（ASCII BEL）。（`/bin/sh` 的 `echo` 不提供该转义序列。）
`\b`	退格。
`\c`	禁止终结的换行符（与 `-n` 相同）。
`\f`	进纸。
`\n`	换行。
`\r`	回车。
`\t`	制表符。
`\v`	垂直制表符。
`\\`	反斜线。
`\0`*nnn*	使用八进制数字 *nnn* 表示 ASCII 字符。其中 *nnn* 是 1、2 或 3 等数字，前面需要添加一个 0。

示例

```
echo "testing printer" | lp
echo "TITLE\nTITLE" > file ; cat doc1 doc2 >> file
echo "Warning: ringing bell \07"
```

ed

ed [*options*] [*file*]

标准文本编辑器。如果指定的文件*file*不存在，则ed会建立该文件；否则，会打开该文件进行编辑。一般不再使用行编辑器ed，因为可以使用vi和ex文本编辑器代替它。一些实用程序，比如diff，仍继续使用ed命令的语法。该命令的加密版本（具有-x选项）只能在美国使用。

options

-C 与-x作用相同，但是假设对文件*file*的开始位置已经进行了加密。

-p *string*
将*string*设置为命令的提示符（默认是*）。P命令可以显示或禁止显示提示符。

-s 禁止字符的计数、诊断和用于shell命令的!提示符。早期版本的ed使用简化的-，现在仍可以使用该选项。

-x 提供一个密钥来加密或解密使用crypt的文件*file*。

edit

edit [*options*] [*files*]

一个行文本编辑器，可以使新用户运行一个简化版本的ex。set变量report、 showmode和magic被预置，以便用来报告改变的编辑内容、显示编辑模式（处于:vi模式），并要求字面上的搜索模式（不允许使用元字符）。（在美国以外的国家或地区不支持加密。）edit与ex接受一样的选项，选项列表可以参见**ex**。更多的信息参见第八章和第九章。

egrep

egrep [*options*] [*regexp*] [*files*]

搜索文件中的行与正则表达式*regexp*匹配的一个或多个文件。egrep不支持元字符\(、\)、*n*、\<、\>、\{及\}，但是支持其他的元字符，以及扩展字符集+、?、|和()。应记住使用引号将这些字符引起来。正则表达式在第六章中讲述。如果有匹配的行，退出状态为0；如果没有匹配的行，退出状态为1；如果出现错误，退出状态为2。参见**grep**和**fgrep**。

egrep

options

`-b` 将行所在的块号码放在各行的前面（不是特别有用）。

`-c` 只输出匹配行的数量。

`-e` *regexp*
如果 *regexp* 以 - 开始，则使用该选项。

`-f` *file*
从 *file* 中提取表达式。

`-h` 只列出匹配的行而不是文件名（与 `-l` 相反）。

`-i` 忽略字母的大小写。

`-l` 只列出文件名而不是匹配的行。

`-n` 显示行和它们的行号。

`-s` 安静模式：只输出错误消息并返回退出状态。SVR4 中没有提供该选项，但是通常大多数商业 Unix 系统都提供该选项。

`-v` 输出不匹配 *regexp* 的所有行。

示例

在文件 *file* 中搜索 *Victor* 或 *Victoria* 出现的位置：

```
egrep 'Victor（ia)?' file
egrep '（Victor|Victoria）' file
```

在文件 *files* 中查找并输出字符串，比如 *old.doc1* 或 *new.doc2* 等，并且包括它们的行号：

```
egrep -n '（old|new）\.doc?' files
```

eject

`eject [`*options*`] [`*media*`]`

只用于 Solaris。弹出可移动的媒体，比如软盘或 CD-ROM。该命令对于通过 `vold` 进行管理的媒体或没有弹出按钮的媒体（比如 Sim SPARC 系统中的软驱）是必需的。*media* 可以是一个设备的名字或者其别名，比如 `floppy` 或 `cdrom`。

通过有效的卷管理，`eject` 可以卸装在指定的 *media* 中安装的任何文件系统。在这种情况下，如果一个窗口系统正在运行，则会弹出一个

eject

对话框；如果没有卷管理，则只是简单地将一个“ eject ”命令发送到指定的设备上。

options

`-d` 输出将要弹出的默认设备的名称。

`-f` 当卷管理不起作用时，强制弹出可移动的媒体，即使正在使用该设备。

`-n` 显示别名和与其对应的真实设备的列表。

`-p` 不使用窗口式的弹出对话框。

`-q` 查询设备中是否具有媒体，并使用退出状态来决定结果。

elfdump

`elfdump [`*`options`*`] `*`filename`*` ...`

只用于 Solaris。象征性地转储目标文件的一部分。*files* 可以是单独的文件，也可以是目标文件的 `ar` 档案（库）。

options

`-c` 输出节标题。

`-d` 输出 `.dynamic` 节。

`-e` 输出 ELF 标题。

`-i` 输出 `.interp` 节。

`-G` 输出 `.got` 节。

`-h` 输出 `.hash` 节。

`-n` 输出 `.note` 节。

`-N` *name*
使用指定的名称 *name* 限定一个选项（比如，选择一个带有 `-s` 的特定符号表）。

`-p` 输出程序的标题。

`-r` 输出重新分配的节。

`-s` 输出符号表节。

`-v` 输出版本节。

elfdump

`-w` *file*
将指定节写到文件 *file*。

env

env [*options*] [*variable=value ...*] [*command*]

显示当前的环境，如果指定了环境变量，则将它们设置为一个新的值并显示修改的环境变量；如果指定了命令 *command*，则可以在修改后的环境中执行它。

options

`-` 完全忽略当前的环境。

`-i` 与“ - ”作用相同，只用于 Solaris。

eqn

eqn [*options*] [*files*]

与 `troff` 等同的预处理程序，参见第十七章。

error

/usr/ccs/bin/error [*options*] [*files*]

读取编译器的错误消息，并将它们插入到产生它们的源文件中。这样，在典型的编辑－编译－调试循环中更容易处理错误。其典型用法如下：

```
cc -O -c  files  2>&1 | error
```

options

`-n` 不编辑任何文件；在标准输出中输出错误。

`-q` 进行查询。将错误消息插入到一个文件前会提示用户输入`y`或`n`来做出响应。

`-s` 输出不同类型错误的统计信息。

`-v` 将错误消息插入到源文件后，在文件上运行 `vi`。

`-t` *list*
只处理后缀出现在 *list* 中的文件，并使用小点间隔后缀。该选项允许使用通配符，但是一般应使用引号将其引起来，以免 shell 将其当作命令使用。

ex

ex [*options*] *files*

一个行文本编辑器，是ed的超集和vi的根，更多的信息参见第八章和第九章。

options

-c *command*
通过执行给定的ex命令（通常是一个搜索模式或行地址）开始编辑会话。如果*command*中包含空格或特殊的字符，则应使用单引号将其括起来，以防止被shell解释。例如，*command*可以是'**:set list**'（显示制表符及换行符）或*/word*（查找*word*）或'**$**'（显示最后一行）。（注意，-c *command*以前的格式为+*command*，旧版本仍在使用。）

-l 运行在LISP模式以便编辑LISP程序。

-L 列表输出由于编辑器或系统崩溃而保存的文件名。

-r *file*
在一个编辑器或系统崩溃后恢复及编辑*file*。

-R 使用只读模式编辑文件，以防止对文件的意外修改。

-s 禁止状态消息（比如，错误提示），主要用于运行一个ex脚本（-s是原来的 - 选项，旧版本仍在使用）。

-t *tag*
编辑包含*tag*的文件，并在其定义中决定编辑器的位置（更多的信息可参见 **ctags**）。

-v 调用vi。直接运行vi是一种简单方式。

-V 即Verbose，在标准错误中输出非终端的输入。主要用于跟踪正在运行ex的shell脚本。

-w*n* 设置窗口的大小为*n*。主要用于速度较慢的拨号（或较慢的Internet）连接。

-x 提供一个密钥来加密或解密使用crypt的文件*file*。

-C 与-x作用相同，但是假设文件*file*以加密的形式开始。

ex

示例

下面的两个示例会将exscript中的ex命令应用到文本文件doc中：

```
ex -s doc < exscript
cat exscript | ex -s doc
```

expand

expand [*options*] [*files*]

将制表符扩展为合适数量的空格。expand会读取指定的文件*files*，如果没有提供*files*参数，则读入标准输入，参见**unexpand**。

options

`-t` *tablist*

按照*tablist*解释制表符，*tablist*是一个使用空格或逗号分隔并按照升序排列的一系列数字的列表，用来描述输入数据的制表位（tabstop）。

-n 每隔*n*个字符设置一个制表符。默认值是8。

-tablist

按照*tablist*解释制表符，*tablist*是一个使用空格或逗号分隔并按照升序排列的一系列数字的列表，用来描述输入数据的制表位。

示例

剪切输入数据的第10~12列，即使使用了制表符：

```
expand data | cut -c 10-12 > data.col2
```

expr

expr *arg1 operator arg2* [*operator arg3* ...]

按照表达式计算参数的值并输出结果。可以用于比较及查找字符串。参数和操作符必须使用空格间隔。大多数情况下，参数是一个整数，此时可以逐字输入或使用一个shell变量表示。有三种类型的操作符：算术、关系和逻辑。expr的退出状态是0（表达式非零或非空）、1（表达式是0或空值）、2（表达式是无效的）。

expr通常用于shell脚本中，用来执行简单的数学运算，比如加法或

expr

减法。由于Korn shell在程序中已经内置了算术运算的能力，所以已经不再使用该命令。

算术运算符

使用下面的运算符可以生成将结果显示出来的数学表达式：

+ 将 *arg2* 与 *arg1* 相加。

- 从 *arg1* 中减去 *arg2*。

* 乘上参数。

/ *arg1* 除以 *arg2*。

% 取 *arg1* 除以 *arg2* 后的余数。

如果没有使用圆括号将加法和减法括起来，则将其计算放在最后。符号 *、左括号（和右括号）在 shell 中是有意义的，所以在使用这些符号时必须转义（在符号前面添加一个反斜线，或者使用单引号或双引号将符号引起来）。

关系运算符

使用关系运算符可以比较两个参数。参数可以是多个单词，这种比较先假设 a < z 及 A < Z。如果比较语句是 true，则结果是 1；如果比较语句是false，则结果是0。左尖括号<和右尖括号>必须进行转义。

= 参数相等吗？

!= 参数不相等吗？

> *arg1* 比 *arg2* 大吗?

>= *arg1* 大于等于 *arg2* 吗?

< *arg1* 小于 *arg2* 吗?

<= *arg1* 小于等于 *arg2* 吗?

逻辑运算符

逻辑运算符可用于比较两个参数。根据值的不同，结果可以是 *arg1*（或其中的一部分）、*arg2* 或 0。符号 | 和 & 必须进行转义。

expr

| 表示逻辑或。如果 *arg1* 是一个非零的值（或非空），则结果是 *arg1*；否则，结果是 *arg2*。

& 表示逻辑与。如果 *arg1* 和 *arg2* 都是非零（或非空）的值，则结果是 *arg1*；否则，结果是 0。

: 与 grep 命令相似。*arg2* 是在 *arg1* 中进行搜索的一种模式。在这种情况下，*arg2* 必须是一个正则表达式。如果 *arg2* 模式使用 \(\)括起来，则结果是匹配 *arg1* 的一部分；否则，结果只是匹配字符的编号。默认情况下，一种模式匹配总是用在第一个参数的开头（搜索字符串在开始位置隐式使用一个 ^）。要匹配字符串的其他部分，可以使用 .* 开始搜索字符串。

示例

先进行除法运算，结果是 10：

```
expr 5 + 10 / 2
```

先进行加法运算，结果是 7（对 7.5 进行了截短）：

```
expr \( 5 + 10 \) / 2
```

对变量 i 加 1，这就是变量在 shell 脚本中是如何增加的：

```
i=`expr $i + 1`
```

如果变量 a 是字符串 “ hello ”，则输出 1（true）：

```
expr $a = hello
```

如果变量 b 加上 5 后大于等于 10，则输出 1：

```
expr $b + 5 \>= 10
```

下面的示例中，变量 p 是字符串 "version.100"。

下面的命令会输出 p 中字符的数量：

```
expr $p : '.*'                  结果是11
```

匹配所有的字符并输出这些字符：

```
expr $p : '\(.*\)'              结果是 "version.100"
```

输出 p 开头的小写字母的数量：

expr

```
expr $p : '[a-z]*'          结果是7
```

在 p 的开头匹配小写字母：

```
expr $p : '\([a-z]*\)'      结果是 "version"
```

如果 $x 包含了 5 个以上的字符，则会将其截短；否则，会输出 $x。（当第一个参数是 0 或空时，则使用第二个参数进行逻辑或运算；也就是说，当匹配失败时会这样做。）如果 $x 包含了空白字符，使用双引号是一个好主意。

```
expr "$x" : '\(.....\)' \| "$x"
```

在一个 shell 脚本中，使用文件名的前 5 个字母表示该文件：

```
mv "$x" `expr "$x" : '\(.....\)' \| "$x"`
```

（为了避免使用相同的名字覆盖原来的文件，应使用 mv -i.）

exstr

exstr [*options*] *file*

从C源文件中提取字符串，以便将它们保存到一个数据库中，并在应用程序运行时使用 gettxt 库函数检索它们。不使用任何选项时，exstr会产生一个grep类型的列表，只显示文件名及字符串。exstr命令是可以在定制国际上通用的应用程序时使用的几个命令之一。

典型的应用包括三个步骤：

1. 指定 -e 和 C 源文件，并且将输出重新定向到一个文件中。这将建立一个文本字符串和标识信息的数据库。
2. 通过在数据库中添加mkmsgs命令以前返回的信息来编辑该数据库。
3. 使用已编辑的数据库作为输入，指定 -r 和 C 源文件。通过调用 gettxt，可以替换硬编码的文本字符串。gettxt可以使你访问文本字符串的转换版本（该字符串驻留在环境变量 LC_MESSAGES 指定的目录中）。

options

-d 与 -r 一起使用，为 gettxt 给出第二个参数，即原始的文本字符串。如果 gettxt 调用失败，则会输出该字符串。

exstr

`-e` 从 *file* 中提取文本字符串（`-e`不能与其他选项一起使用）。该信息的显示格式如下：

`file:line:field:msg_file:msg_num:string`

file	来自命令行的 C 源文件。
line	在文件 *file* 中发现的字符串所在的行号。
field	字符串开始的内部编号位置。
msg_file	初始值为空, 但是在稍后编辑数据库时会填充新值。*msg_file* 是运行 `mkmsgs` 命令时产生的消息字符串列表的名字。
msg_num	初始值为空，但是稍后会填充新值。它对应于 *msg_file* 中字符串的顺序。

`-r` 通过调用 `gettxt`，替换源文件中的字符串。

示例

假设一个 C 源文件的名字为 `proverbs.c`：

```
main() {
     printf ("Haste makes waste\n");
     printf ("A stitch in time\n");
 }
```

1. 首先发出命令：

   ```
   exstr -e proverbs.c > proverb.list
   ```

   `proverb.list` 可能类似于下面的内容:

   ```
   proverbs.c:3:8:::Haste makes waste\n
   proverbs.c:4:8:::A stitch in time\n
   ```

2. 运行 `mkmsgs` 建立一个可以通过 `gettxt` 调用读取的消息文件（比如 `prov.US`）。如果前面的两个 proverb 字符串在 `prov.US` 中列在第9行和第10行, 则可以按照下面的内容编辑`proverb.list`：

   ```
   proverbs.c:3:8:prov.US:9:Haste makes waste\n
   proverbs.c:4:8:prov.US:10:A stitch in time\n
   ```

3. 最后，指定 `-r` 来插入 `gettxt` 调用：

   ```
   exstr -rd proverbs.c < proverb.list > Prov.c
   ```

exstr

`Prov.c` 的国际化版本，现在看起来如下所示：

```
extern char *gettxt ();
main() {
  printf (gettxt ("prov.US:9",
  "Haste makes waste\n"));
  printf (gettxt ("prov.US:10",
  "A stitch in time\n"));
}
```

factor

`factor [`*num*`]`

产生 *num* 的素数因子或者从输入中读取数字。

false

`false`

该命令只返回一个没有成功（非0值）的退出状态，不执行任何操作。通常用于 Bourne shell 脚本中。参见 **true** 命令。

示例

```
# 该循环从来没有执行
while false
do
        commands
done

# 该循环一直运行
until false
do
        commands
done
```

fdformat

`fdformat [`*options*`] [`*device*`]`

只用于 Solaris。格式化软盘和 PCMCIA 存储卡。*device* 是要进行格式化的设备的名称，格式化的速度与磁盘介质的密度、磁盘驱动器的容量和卷管理是否有效有很大的关系。

options

`-b` *label*

为介质设置一个标签 *label*。SunOS 标签最多可使用 8 个字符；DOS 标签最多可使用 11 个大写字母。

fdformat

-B *file*
在 MS-DOS 磁盘上的 *file* 中安装引导文件，只可以与 -d 或 -t dos 一起使用。

-D 格式化容量为 720KB（3.5 英寸）或 360KB（5.25 英寸）的双密度磁盘（与 -l 或 -L 选项作用相同）。用于高密度或加强密度的磁盘。

-e 当磁盘格式化完毕后，弹出磁盘。

-E 格式化一张 2.88MB（3.5 英寸）加强密度的磁盘。

-f 强制格式化磁盘，在格式化以前不进行提示。

-H 格式化一张 1.44MB（3.5 英寸）或 1.2MB（5.25 英寸）的高密度磁盘。用于加强密度的驱动器。

-M 使用 1.2MB（3.5 英寸）的中密度格式化高密度磁盘。只能与 -t nec 选项一起使用。等同于 -m。

-U 卸装介质上的任何文件系统，然后格式化。

-q 安静模式，不输出状态消息。

-v 完成格式化后，校验介质中的每个数据块。

-x 不进行格式化，只向磁盘写入一个 SunOS 标签或 MS-DOS 文件系统。

-t dos
安装 MS-DOS 文件系统并引导扇区的格式化。与 DOS 的 format 命令或 -d 选项作用相同。

-t nec
在完成格式化后，安装一个 NEC-DOS 文件系统及引导扇区。只能与 -M 一起使用。

兼容选项

这些选项主要是为了与以前版本的 fdformat 命令兼容。建议不要使用这些选项。

-d 与 -t dos 相同。

-l 与 -D 或 -L 相同。

fdformat

`-L` 与 `-l` 或 `-D` 相同。

`-m` 与 `-M` 相同。

fgrep

`fgrep [`*`options`*`] [`*`pattern`*`] [`*`files`*`]`

在一个或多个 *files* 中查找匹配一个文字的、文本字符串模式 *pattern* 的行。因为 `fgrep` 不支持正则表达式，因此比 `grep` 命令执行速度快（`fgrep` 即快速的 `grep`）。如果查找到匹配内容，则退出状态为 0；退出状态为 1 表示没有匹配内容；退出状态为 2 表示出错。参见 **egrep** 和 **grep** 命令。

options

`-b` 在每一行的前面添加其块号 （不是特别有用）。

`-c` 输出匹配行的数量。

`-e` *pat*
如果 *pat* 的开头是 “ - ”，则使用该选项。

`-f`*file*
从文件 *file* 获得模式列表。

`-h` 输出匹配的行而不是文件名（与 `-l` 相反）。

`-i` 忽略字母的大小写。

`-l` 不输出匹配的行而只输出其所在的文件名。

`-n` 输出行和行号。

`-s` 安静模式: 只输出错误消息并返回退出状态。SVR4 不支持该选项，但是大多数商业 Unix 系统都支持该选项。

`-v` 输出所有与指定模式 *pattern* 不匹配的行。

`-x` 输出所有与指定模式 *pattern* 匹配的行。

示例

输出文件 *file* 中没有包含空格的行:

```
fgrep -v ' ' file
```

输出文件 *file* 中包含了 `spell_list` 中列出的单词的行:

fgrep

```
fgrep -f spell_list file
```

file

file [*options*] *files*

按照指定文件 *files* 中包含的数据类型对文件进行分类，该命令通过检查魔文件（magic file）（通常是 /etc/magic）来识别许多常用文件类型。

options

-c 检查魔文件的格式（与 -c 选项一起使用的 *files* 参数无效）。

-flist
在文件名列表 *list* 中列出的文件上运行 file。

-h 不紧跟符号链接。

-m *file*
使用 *file* 作为魔文件来代替 /etc/magic。

许多文件类型都是常用的，输出时会列出文件名和其类别。例如:

```
ascii text
c program text
c-shell commands
data
empty
iAPX 386 executable
directory
[nt]roff, tbl, or eqn input text
shell commands
symbolic link to ../usr/etc/arp
```

示例

列出作为 nroff/troff 输入的所有文件:

```
file * | grep roff
```

find

find *pathname(s) condition(s)*

该命令在查找特定组的文件方面特别有用(后面列出了大量的示例)。它沿着以各个*pathname*开始的目录树向下查找并定位符合指定条件 *conditions* 的文件。至少要指定一个 *pathname* 及一个 *condition*。最有用的条件包括 -print（必须显式给出该选项以显示输出）、-name

find

和 -type（用于一般情况）、-exec 和 -size（用于高级用户），以及 -mtime 和 -user（用于系统管理员）。在 Solaris（和其他较新的 Unix系统）中，如果没有提供其他条件，则 -print 被当作默认条件。

使用 \(\)（转义的圆括号）将条件括起来可以对其进行分组，使用 !（在 C shell 中使用 \!）可以对其取“非”，通过使用 -o 分隔它们来提供选择，或者进行重复（对匹配条件增加限制，通常只用于 -name、-type 和 -perm）。

如果在命令行中有太多的文件需要命名，则 find 命令经常与 xargs 命令一起使用（参见 **xargs**）。

conditions

-atime *+n* | *-n* | *n*

搜索在多于 *n* 天（+*n*）、少于 *n* 天（-*n*）或正好是 *n* 天以前最后访问过的文件。注意，find 会改变作为 *pathnames* 提供的目录的访问时间。

-cpio *dev*

使用 cpio 命令可获得匹配文件并将其写入到设备 *dev* 上，该命令已经废弃。

-ctime *+n* | *-n* | *n*

搜索多于 *n* 天（+*n*）、少于 *n* 天（-*n*）或正好 *n* 天以前改变的文件。这种改变指修改、文件权限的改变或属主的改变等等。因此，-ctime 比 -atime 或 -mtime 包含了更多的内容。

-depth

沿着目录树向下，跳过多个目录并首先对实际的文件进行操作（然后是文件的父目录）。该选项对于保存在不可写目录中的文件非常有用（比如，当使用带 cpio 选项的 find 命令时）。

-exec *command* {} \;

在 find 命令找到的匹配文件上运行 Unix 命令 *command*，假设提供的 *command* 会成功运行在各个文件上，即返回 0 退出状态。当运行 *command* 时，参数 {} 会替换当前的文件，整个序列后面紧跟一个转义的分号（\;）。

find

`-follow`
跟随符号链接并追踪访问过的目录（不要与`-type l`选项一起使用）。

`-fstype` *type*
搜索保存在 *type* 类型的文件系统中的文件。

`-group` *gname*
查找属于组 *gname* 的文件，*gname* 可以是一个组名字或一个组 ID 号。

`-inum` *n*
搜索信息节点号为 *n* 的文件。

`-links` *n*
搜索有 *n* 个链接的文件。

`-local`
搜索实际保存在本地计算机系统中的文件。

`-ls` 显示带有关联的统计信息的匹配文件（就好像运行`ls -lids`命令）。

`-mount`
搜索保存在与 *pathname* 相同的文件系统中的文件。

`-mtime` *+n* | *-n* | *n*
搜索最后修改日期为多于 *n* 天（+*n*）、少于 *n* 天（-*n*）或正好是 *n* 天的文件。

`-name` *pattern*
搜索名字与 *pattern* 匹配的文件，可以使用文件名元字符，但是应进行转义或者引用。

`-ncpio` *dev*
使用`cpio -c`获得匹配文件并将其写入设备 *dev* 中。该选项已经废弃。

`-newer` *file*
搜索比 *file* 参数指定的文件修改时间更晚的文件，与`-mtime`相似。

find

`-nogroup`

搜索属于一个组的文件，该组不在 `/etc/group` 目录下。

`-nouser`

搜索一个用户拥有的文件，该用户不在 `/etc/passwd` 目录下。

`-ok` *command* `{} \;`

与 `-exec` 作用相同。但是在执行 *command* 以前，用户必须进行响应（使用一个 `y`）。

`-perm` *nnn*

搜索权限设置（比如 `rwx`）精确匹配八进制数 *nnn* 的文件（比如，664 匹配 `-rw-rw-r--`）。可使用一个中划线（`-`）作为通配符来匹配任何指定的位（比如，`-perm -600` 匹配 `-rw******`，其中 * 可以是任何模式）。有些系统允许使用 +*nnn* 替代。

Solaris 允许 *nnn* 是与 `chmod` 所允许的形式一样的符号模式。

`-print`

使用完整的路径名来输出匹配的文件和目录。在 Solaris 中，这是默认的设置。

`-prune`

“剪除”不希望进行目录搜索的目录树，即跳过最近匹配的目录。

`-size` *n*`[c]`

搜索包含 *n* 个块的文件；如果指定了 `c`，则搜索文件长度为 *n* 个字符（字节）的文件（一个块为 512 个字节）。

`-type` *c*

搜索类型为 *c* 的文件，*c* 可以是如下值：

`b`	块特殊文件
`c`	字符特殊文件
`d`	目录
`D`	Door 特殊文件，只用于 Solaris
`f`	普通文件
`l`	符号链接
`p`	先进先出或命名管道

find

s 套接字

`-user` *user*

搜索属于一个 *user* 名或 ID 的文件。

`-xdev`

与 `-mount` 相同，只用于 Solaris（和一些 BSD 系统）。

示例

列出你的主目录中的所有文件（和子目录）：

```
find $HOME -print
```

列出在 `/work` 目录中名称为 `chapter1` 的所有文件：

```
find /work -name chapter1 -print
```

列出 ann 用户拥有的所有 memo 文件（注意多个起始路径的使用）：

```
find /work /usr -name 'memo*' -user ann -print
```

搜索文件系统中的参考页目录（从根目录开始）：

```
find / -type d -name 'man*' -print
```

搜索当前目录，查找不是以大写字母开头的文件名，并把它们发送到打印机中：

```
find . \! -name '[A-Z]*' -exec lp {} \;
```

搜索并比较文件名结尾不是 .Z 的文件:

```
compress `find . -type f \! -name '*.Z' -print`
```

删除系统中所有的空文件（删除前先提示）:

```
find / -size 0 -ok rm {} \;
```

跳过 RCS 目录，但是列出剩余的只读文件：

```
find . -name RCS -prune -o -perm 444 -print
```

在系统中搜索在最近两天之内修改过的文件（进行备份的好办法）：

```
find / -mtime -2 -print
```

按照一种模式沿着目录树递归使用 `grep`：

```
find /book -print | xargs grep '[Nn]utShell'
```

finger

finger [*options*] *users*

显示一个或多个用户*users*的数据，包括在*users*的主目录中的.plan和.project文件中包含的信息。可以使用登录名指定用户（精确匹配），也可以使用名或姓指定用户（显示所有匹配名称的信息）。网络环境认可*user@host*和*@host*形式的参数（现在，Internet上的许多系统不允许进行finger请求连接）。

options

-b 显示时省略用户的主目录和shell。

-f 使用-s可以省略通常在短格式中显示的标题。

-h 显示时省略.project文件。

-i 显示“空闲”格式，这是一种简短的格式（与-s相似）。

-l 强制使用长格式（默认选项）。

-m *users*必须精确匹配用户名，而不是查找是否匹配名和姓。

-p 显示时省略.plan文件。

-q 显示“快速”格式，进行全部简化（需要精确匹配用户名）。

-s 显示短格式。

-w 与-s一起使用，省略通常显示在短格式中的用户名全称。

fmt

fmt [*options*] [*files*]

填充和连接文本，产生长度基本相同的行（与nroff不同，这些行是不可调整的）。fmt会忽略空行和以小点（.）或“From:”开头的行。emacs编辑器使用ESC-q来连接段落，所以fmt对其他编辑器比较有用，比如vi编辑器。下面的vi命令会填充并连接当前段落中的剩余部分：

```
!}fmt
```

options

-c 不调整开始的两行，但是将第二行后面的行与第二行排为一行，对于使用悬挂标签开始的段落比较有用。

fmt

`-s` 将比较长的行分开，而只保留短行。对保留代码行的一部分比较有用。

`-w` *n*
新产生的行不比 *n* 列宽，默认是 72 列（要与 BSD 兼容，可以使用 *-n* 代替该选项）。

ftp

`ftp [`*options*`] [`*hostname*`]`

将文件传输到远程网络站点 *hostname* 上或进行相反操作。`ftp` 会向用户提示命令的使用方法，键入 `help` 可以看到已知命令的列表。

options

`-d` 启用调试功能。

`-g` 禁止文件名扩展（*globbing*）。

`-i` 关闭交互式提示。

`-n` 第一次连接不能自动登录。

`-v` 显示详细内容，显示来自远程服务器的所有响应。

gcore

`gcore [`*option*`]` *process_ids*

建立（“ 获得 ”）各个指定运行进程的核心映像（core image）。可以将核心映像和调试器一起使用。要使用该命令，你必须拥有该运行进程或者是一个特权用户。

option

`-o` *file*
建立一个名字为 *file.process_id*（默认是 *core.process_id*）的核心文件。

gencat

`gencat [`*option*`]` *database msgfiles*

增加（或合并）包含在一个或多个 *msgfiles* 中的消息，该 *msgfiles* 具有格式化的消息数据库文件。如果数据库 *database* 不存在，则会建立该数据库。*msgfiles* 中的各条消息前面都有一个数字标识符，可以在一行的开头使用美元符号（$）并且后面加一个空格或制表符来表示一个注释行。参见 **genmsg** 和 **mkmsgs** 命令。

gencat

option

`-m` 建立一个*database*, 该数据库向后兼容使用早期版本的`gencat`建立的数据库。只适用于 SVR4。

genmsg

`genmsg [`*`options`*`] `*`files`*` ...`

只适用于 Solaris。从使用 *catgets*（3C）以便用 `gencat` 进行进一步处理的源代码中提取消息字符串。该命令主要用来在国际化应用程序时建立可以由翻译器使用的初始数据。

options

`-a` 将输出追加（合并）到 `-o` 指定的文件中。

`-b` 将提取的注释放置在相应消息的后面，而不是放在消息前面。

`-c` *`tag`*
在输出文件的一个注释中提取包含标签*tag*的消息并使用`$`前缀将它们写出来。

`-d` 在输出文件中添加原始的消息作为注释。

`-f` 与 `-r` 一起使用，覆盖最初的输入文件；与 `-l` 一起使用，也可以覆盖项目文件。

`-g` *`file`*
建立一个*file*并当作一个项目文件, 列出设置号和它们的最大消息编号。

`-l` *`file`*
使用作为一个项目文件的 *file* 中的信息来计算新的消息编号。

`-m` *`prefix`*
使用 *prefix* 填充消息，主要用于测试。

`-M` *`suffix`*
使用 *suffix* 填充消息，主要用于测试。

`-n` 在输出中添加注释来说明用于消息的原始文件的名字和行号。

`-o` *`msgfile`*
将输入放入 *msgfile* 中。

genmsg

`-p` *preprocessor*
在提取消息以前通过预处理程序 *preprocessor* 运行源文件。

`-r` 使用 `-1`（负 1）替换消息号。与 `-l` 作用相反。

`-s` *tag*
从源文件中提取使用 `/* SET` *tag* `*/` 格式的注释。将它们作为注释写入到输出中，并使用 `$` 作为前缀，只提取第一个符合条件的注释。

`-t` 将提取的消息增长到原来的三倍，主要用于测试。

`-x` 没有警告消息，并且设置编号检查及冲突的范围。

get

`/usr/ccs/bin/get [`*options*`] `*files*

一条 SCCS 命令，参见第十八章。

getconf

`getconf [-v` *spec*`] `*system_var*
`getconf [-v` *spec*`] `*path_var path*
`getconf -a`

只用于Solaris。POSIX指定该命令为决定系统局限性的方便途径。第一种格式会输出系统配置变量的值；第二种格式会输出与文件系统相关的参数值；第三种格式会输出所有的系统配置变量的值。

options

`-a` 输出所有系统配置变量的名字和值。

`-v` *spec*
使用 *spec* 来控制配置变量值的选择。

getopts

`getopts` *string name* `[`*arg*`]`

与 Bourne shell 内置的命令 `getopts` 相同，参见第四章。

gettext

`gettext [`*domain*`] `*string*

只用于 Solaris。检索和输出字符串 *string* 的转换版本。提供了对 *gettext*(3C)实用程序的 shell 级别的访问。在 `/usr/lib/locale/`*lang*`/LC_MESSAGES/`*domain*`.mo`目录中对转换进行搜寻。*lang*是当前

gettext

场所（比如 en_US）。如果没有提供域 *domain*，则使用 $TEXT－DOMAIN 的值来代替它。如果没有域，或者没有发现任何转换，gettext只是简单地显示*string*。如果$TEXTDOMAINDIR存在，则可以使用它的值来代替 /usr/lib/locale/。

gettxt

gettxt *msgfile:msgnum* [*default_message*]

获得保存在文件 *msgfile* 并且其消息 ID 是 *msgnum* 的消息。*msgnum* 是 1 到 *n* 中的一个数字，*n* 是 *msgfile* 中的消息数量。gettxt 在目录 /usr/lib/locale/*locale*/LC_MESSAGES 中查找 *msgfile*，其场所（locale）是编写消息字符串使用的语言。*locale* 的值通过环境变量 LC_MESSAGES 进行设置，或者失败后使用 LANG 环境变量设置。如果两者都没有设置，*locale* 的默认位置是目录 C。如果 gettxt 操作失败，它会显示 *default_message* 或（如果什么也没有指定）字符串 “ Message not found!!(没有发现消息）”。

gprof

/usr/ccs/bin/gprof [*options*] [*objfile* [*pfile*]]

只用于 Solaris（许多较新的 Unix 系统也有该命令）。用于显示 C 程序调用图（call-graph）的描述数据。使用 cc 的 -xpg 选项编译的程序（在其他的编译器中使用 -pg）会产生一个调用图的描述文件（profile）*pfile*，其默认名字是 gmon.out。指定的目标文件 *objfile*（默认为a.out）包含了一个使用*pfile*读取并关联的符号表，参见**prof** 和 **lprof**。

options

-a 不输出静态声明的函数。

-b 简要形式，不输出配置文件中的字段描述。

-c 查找程序的静态调用图，调用的数量为 0 表明只有静态的父和子。

-C 在将 C++ 符号名输出出来以前对其进行反改编（demangle）。

-D 使用该选项，你可以提供一个以上的已经存在的*pfiles*。处理在所有指定的配置文件中的信息并产生一个名字为gmon.sum 的描述文件来显示运行的差别，参见下面的 -s 选项。

gprof

`-e` *name*
不输出例程 *name* 的图形描述文件项，`-e` 可能会重复。

`-E` *name*
与 `-e` 相似。但是该选项在计算时间时，会省略 *name* 占用的时间。

`-f` *name*
只对例程 *name* 输出图形描述文件项。`-f` 可以重复。

`-F` *name*
与 `-f` 相似。另外，在计算时间时，只使用输出例程使用的时间。`-F` 可以重复，它会覆盖 `-E`。

`-l` 不输出局部符号中的项。

`-s` 使用该选项，你可以提供一个或多个存在的 *pfiles*。它会汇总所有指定的描述文件中的信息，并将其发送到描述文件 `gmon.sum` 中。主要用于汇总几次运行的数据。

`-z` 输出没有使用的例程，主要与 `-c` 一起使用来找出从来没有使用的例程。

`-`*n* 只输出顶部的 *n* 个函数。

grep

`grep [`*options*`] `*regexp*` [`*files*`]`

按行搜索与正则表达式 *regexp* 匹配的一个或多个 *files*。正则表达式在第六章讲述。退出状态为0表示有匹配的行；1表示没有匹配的行；2表示出现错误。参见 **egrep** 和 **fgrep**。

options

`-b` 将其块号放在各个行的前面（不是特别有用）。

`-c` 只输出匹配行的数量。

`-e` *pat*
如果 *pat* 的开头使用 `-`，则使用该选项。在 Solaris 中，该选项只能用在 `/usr/xpg4/bin/grep` 中，不能用在 `/usr/bin/grep` 中。该选项通常用于一些较新的 Unix 系统中。

`-h` 输出匹配的行而不是文件名（与 `-l` 相反）。

grep

`-i` 忽略大小写。

`-l` 只列出文件名而不列出匹配的行。

`-n` 输出行及其行号。

`-s` 禁止不存在或不可读的文件中的错误消息。

`-v` 输出所有与 *regexp* 不匹配的行。

`-w` 限定*regexp*匹配一个完整的单词（类似在`vi`中使用`\<` 和`\>`）。SVR4 不支持该选项，但是许多商业 Unix 系统通常都支持该选项。

示例

列出使用 C shell 的用户数量:

```
grep -c /bin/csh /etc/passwd
```

列出至少有一个`#include`指令的头文件：

```
grep -l '^#include' /usr/include/*
```

列出没有包含模式 *pattern* 的文件：

```
grep -c pattern files | grep :0
```

groups

`groups [`*user*`]`

显示 *user* 隶属的组（默认是你所在的组）。

组都保存在`/etc/passwd and /etc/group`目录中。

gunzip

`gunzip [`*gzip options*`] [`*files*`]`

与`gzip -d`等同。通常作为`gzip`的硬链接（hard link）（译注 1）。`gunzip` 中没有`-1` ... `-9`和相应的 long 形式的选项；其他的`gzip`选项都可用，更多信息可参见 **gzip**。

译注 1： 硬链接在 Unix 中指的是指向同一文件的两个不同路径名。与 Windows 里的快捷方式类似。

gzcat

gzcat [*gzip options*] [*files*]

gzip 的一个链接，以替代使用名称 zcat，它用于保留 zcat 到 compress 的原始链接，其作用等同于 gunzip -c。

在一些系统中会将其当作 zcat 安装，更多的信息可参见 **gzip**。

gzip

gzip [*options*] [*files*]

GNU Zip。使用 Lempel-Ziv（LZ77）编码减小一个或多个文件的大小，并移动到 *file*.gz 中。可使用 gunzip 恢复这些文件。如果文件名是一个中划线（-），或者没有任何 *files*，gzip 会读取标准输入。通常情况下，gzip 的压缩效果比 compress 好得多，而且运算法也不受任何限制。

gzip 会忽略符号链接。原始文件的名字、权限和修改时间会保存到压缩文件中，当对文件进行解压缩时，会恢复这些信息。gzip 可以对使用 compress、pack 或 BSD compact 压缩的文件进行解压缩，默认选项保存在环境变量 GZIP 中。

gunzip 等同于 gzip -d。通常是到 gzip 命令的硬链接。gzcat 和 zcat 等同于 gunzip -c，通常也是到 gzip 的硬链接。

注意：尽管 gzip 没有随 SVR4 和 Solaris 发布，但是它是可以通过 Internet 获得的程序的实际标准文件压缩程序，可以从自由软件基金会（Free Software Foundation）获得其源代码（*http://www.gnu.org*）。可以从*http://www.sunfreeware.com*获得适用于Solaris的预编译的二进制代码。gzip 也有其自己的站点：*http://www.gzip.org*。

options

与大部分 GNU 程序相似，gzip 的命令行选项有短版本和长版本：

-a, --ascii
ASCII 文本模式：按照本地的惯例来转换行尾，只有部分系统支持它。

-c, --stdout, --to-stdout
将输出写入到标准输出中，并保持原来的文件不变。单个的输

gzip

入文件会分别压缩。要获得最好的压缩效果，应在压缩以前将所有的输入文件连接起来。

`-d, --decompress, --uncompress`
用于文件的解压缩。

`-f, --force`
强制执行文件的压缩及解压缩，即使目标文件已经存在或者该文件有多个链接。

`-h, --help`
显示一屏帮助信息后退出。

`-l, --list`
列出压缩文件压缩前后的大小、压缩比及原始的文件名。使用`--verbose`选项还会列出压缩方法，即 32 位 CRC，以及原始文件最后修改的时间；使用`--quiet`选项，则不会显示文件的标题及总行数。

`-L, --license`
显示`gzip`的许可证后退出。

`-n, --no-name`
如果使用的是`gzip`，则在压缩文件中不会保存原始的文件名及修改的时间；如果使用的是`gunzip`，则不会恢复原始的文件名及修改时间，而使用压缩文件的文件名及修改时间（这是默认设置）。

`-N, --name`
如果使用的是`gzip`，则在压缩文件中会保存原始的文件名及修改时间（这是默认设置）；如果使用的是`gunzip`，则会基于压缩文件中的信息恢复原始的文件名及修改的时间。

`-q, --quiet`
禁止所有的警告。

`-r, --recursive`
在当前的目录树中进行递归遍历并压缩（对于`gunzip`是解压缩）找到的所有文件。

gzip

`-S` *.suf*, `--suffix` *.suf*
使用 *.suf* 后缀代替 `.gz`。一个空后缀会使 `gunzip` 解压缩所有指定的文件，而不管它们的后缀是什么。

`-t, --test`
检查压缩文件的完整性。

`-v, --verbose`
显示压缩及解压缩文件的名称和减小的百分比。

`-V, --version`
显示版本号和编译选项，然后退出。

`-n, --fast, --best`
控制压缩方法，*n* 是 1～9 之间的一个数字。`-1`（与 `--fast` 作用相同）压缩速度比较快，但是压缩比最小；`-9`（与 `--best` 作用相同）压缩比最大，但是压缩速度比较慢。所以可以根据需要在 1~9 之间选择压缩比及压缩速度，默认的压缩级别是 `-6`，该选项通过降低压缩速度来实现较好的压缩。在实际应用中，默认选项是非常好的，所以一般不需要使用这些选项。

head

head [*options*] [*files*]

输出一个或多个 *files* 的开头几行（默认是 10 行）。

options

`-n` 输出文件开头的 *n* 行。

`-n` *n*
输出文件开头的 *n* 行，只适用于 Solaris。

示例

显示 `phone_list` 文件中开头的 20 行：

```
head -20 phone_list
```

显示区号为 202 的最前面的 10 个电话号码。

```
grep '(202)' phone_list | head
```

help	`/usr/ccs/bin/help [`*`commands`* `|` *`error_codes`*`]` 一个 SCCS 命令，参见第十八章。
hostid	`hostid` 输出主机的 16 进制 ID 号。
hostname	`hostname [`*`newhost`*`]` 输出主机名，一般与`uname`作用相同。特权用户可以将主机名修改为 *newhost*。
iconv	`iconv -f` *`from_encoding`* `-t` *`to_encoding`* `[`*`file`*`]` 将文件*file*的内容从一种字符集（以前使用的编码）转换为另一种字符集（将要使用的编码）。如果目标字符集中没有提供源字符集中一些字符的等价字符，则使用下划线（_）代替这些字符。支持的转换字符集保存在`/usr/lib/iconv`目录中。
id	`id [-a]` 列出用户和组ID，使用`-a`可以列出所有的组。当你作为另一个用户运行一个`su`会话时，`id`会显示该用户的信息。
indxbib	`indxbib` *`files`* 程序的`refer`套件的一部分，参见第十七章。
ipcrm	`ipcrm [`*`options`*`]` 删除由 *options* 指定的消息队列、信号集或共享内存标识符。`ipcrm`主要用于释放一些程序无法释放的共享内存。应先使用`ipcs`显示将要删除的条目。 options `-m` *`shmid`* 删除共享内存标识符 *shmid*。

ipcrm

`-M` *shmkey*
删除使用键 *shmkey* 建立的 *shmid*。

`-q` *msqid*
删除消息队列标识符 *msqid*。

`-Q` *msgkey*
删除使用键 *msgkey* 建立的 *msqid*。

`-s` *semid*
删除信号标识符 *semid*。

`-S` *semkey*
删除使用键 *semkey* 建立的 *semid*。

ipcs

`ipcs [`*options*`]`

输出活动的进程间通信程序的数据。

options

`-m` 报告活动的共享内存段。

`-q` 报告活动的消息队列。

`-s` 报告活动的信号。

使用 `-m`、`-q` 或 `-s` 选项，则只会报告指定的进程间程序。否则，所有这三个选项的信息都会输出出来。

`-a` 几乎使用所有的输出选项（`-bcopt` 的短格式）。

`-A` 使用所有的输出选项（`-bciopt` 的短格式）。只用于 Solaris。

`-b` 报告最多允许使用的消息的字节数、内存段大小及信号数量。

`-c` 报告创建者的登录名字及组。

`-C`*file*
从文件 *file* 而不是从 `/dev/kmem` 中读取状态信息。

`-i` 报告与内存段连接的共享内存的数量，只适用于 Solaris。

`-N`*list*
使用核心的“名称列表”（核心中函数和变量的列表）来代替目录 `/stand/unix`（Solaris:`/dev/ksyms`）。

ipcs

`-o` 报告突出的用法。

`-p` 报告进程的数量。

`-t` 报告时间信息。

jar

`/usr/java/bin/jar [`*`options`*`] [`*`manifest`*`] `*`dest files`*

只用于 Solaris。Java 档案工具。所有指定的对象和目录树（如果给定目录）被压缩进一个 Java 档案中，以方便下载。`jar` 基于 ZIP 和 ZLIB压缩格式；`zip`和`unzip`可以处理 `.jar` 文件而不会出现任何问题。如果没有提供 *manifest*，`jar` 会自动产生一个。*manifest* 会成为档案的第一项，它包含需要的任何档案元信息。

其使用方法与`tar`相似，`tar`前面的中划线“ - ”可以从选项中省略。

options

`-c` 在标准输出中建立一个新档案或空档案。

`-f` 第二个参数 *dest* 是需要处理的档案。

`-M` 使用指定的 *manifest* 来代替建立一个 *manifest* 文件。

`-m` 不建立 *manifest* 文件。

`-o` 不使用 ZIP 压缩格式压缩该文件。

`-t` 在标准输出中输出档案的内容列表。

`-v` 在标准错误中产生详细的输出。

`-x[`*`file`*`]`
提取指定的文件 *file*；如果没有指定 *file*，则提取所有的文件。

java

`/usr/java/bin/java [`*`options`*`] `*`classname`*` [`*`args`*`]`

只适用于 Solaris。编译并运行 Java 字节码类文件。默认情况下，编译器使用当前系统的JIT（Just In Time）编译器。*args* 会传递到 Java 程序的 `main` 方法。参见 **java_g**。

options

`-cs, -checksource`
比较源代码文件与编译的类文件的修改时间，如果源代码时间较新，则重新编译它。

java

`-classpath` *path*
使用 *path* 作为类文件的搜索路径，覆盖 $CLASSPATH。*path* 中的多个目录使用冒号（:）间隔。

`-debug`
输出必须用于调试并允许 `jdb` 连接到一个会话的密码（参见 **jdb**）。

`-D`*prop=val*
重新定义 *prop* 的值为 *val*。该选项可以使用多次。

`-fullversion`
输出完整的版本信息。

`-help`
输出一条用法消息。

`-ms` *size*
设置堆栈的初始大小为 *size*，按字节计算。可分别用 `k` 或 `mm` 来表示千字节或兆字节。堆栈的默认大小是 4MB。

`-mx` *size*
设置堆栈的最大值为 *size*，按字节计算。可分别用 `k` 或 `mm` 来表示千字节或兆字节。堆栈默认的最大值是 16MB。该值必须大于 1000 字节并大于等于初始堆栈的大小。

`-noasyncgc`
禁用异步的无用信息回收。

`-noclassgc`
禁用 Java 类的无用信息回收。

`-noverify`
禁用校验。

`-oss` *size*
设置一个 Java 线程中 Java 代码可使用的最大堆栈大小。添加 `k` 或 `m` 分别表示千字节或兆字节。默认的最大值是 400KB。

`-prof[:`*file*`]`
只适用于 `java_g`。启用 Java 运行时配置文件。如果已经提供，则在指定的文件 *file* 中放置跟踪程序；否则使用 `./java.prof`。

java

`-ss` *size*
设置一个 Java 线程中 C 代码可使用的最大堆栈的大小。添加 `k` 和 `m` 分别表示千字节或兆字节。默认的最大值是 128KB。

`-t` 只适用于 `java_g`，用于跟踪运行的指令。

`-v, -verbose`
每次装载一个类文件时，在标准输出中输出一条消息。

`-verbosegc`
每次回收程序释放内存时，都显示一条消息。

`-verify`
在所有代码上运行字节码校验程序。

`-verifyremote`
在通过一个类装载程序装入的所有代码上运行校验程序。当进行解释时，该选项是默认设置。

`-version`
显示 `java` 的版本信息。

java_g

`/usr/java/bin/java_g [`*options*`] `*classname*` [`*args*`]`

只适用于 Solaris。`java_g` 是 Java 解释器的非优化版本，它使用一个 Java 调试器，比如 `jdb`。否则，它与 `java` 具有相同的选项并实现相同的功能。更多的信息可以参见 **java** 的选项。

javac

`javac [`*options*`] `*files*

只适用于 Solaris。将 Java 源代码编译成 Java 字节码，以便在 Java 中运行。Java 源文件必须具有 `.java` 后缀，并且必须按该文件中包含代码的主类的名字命名。产生的字节码文件使用 `.class` 后缀。默认情况下，产生的类文件保存在与源文件相同的目录中。使用 CLASSPATH 变量可以列出 `javac` 将在其中查找类的目录及（或）ZIP 文件。

options

`-classpath` *path*
使用 *path* 中用冒号（:）分隔的目录列表代替 CLASSPATH 来

javac	搜索类文件。通常情况下，最好将当前目录（“.”）放置在搜索路径中。
	`-d` *dir*
	指定产生的类文件建立的位置。
	`-depend`
	重新编译丢失或过期的类文件，这些类文件由其他类文件引用，而不只是由源代码引用。
	`-deprecation`
	对一个淘汰成员或类的每次使用或覆盖都进行警告，而不是最后进行警告。
	`-encoding` *encoding*
	使用 *encoding* 编码的源文件，没有该选项，则使用系统的默认转换器。
	`-g` 生成带有行号的调试表格。使用 `-O` 选项，也会产生局部变量的信息。
	`-J`*option*
	将 *option* 传递给 `java`。*option* 不应该包含空格。如果需要，可以使用多个 `-J` 选项。
	`-nowarn`
	禁止所有警告。
	`-O` 执行优化，以便产生速度较快但是比较大的类文件，它也可能减慢编译的速度。该选项应该根据判断使用。
	`-verbose`
	输出文件编译及装载时的信息。

javadoc	`/usr/java/bin/javadoc [`*options*`] `*files*`	`*classes* 只适用于 Solaris。处理在 Java 源文件中的声明及文档注释，并产生一个 HTML 页面来描述公共或保护型类、接口、构造器、方法及字段。`javadoc` 还会在 `tree.html` 文件中产生一个类的层次以及在 `Allnames.html` 文件中产生一个成员的索引。

javadoc

options

`-author`
包括 `@author` 标签。

`-classpath` *path*
使用*path*作为类文件的搜索路径，会覆盖$CLASSPATH。*path*是一个使用冒号间隔的目录列表。最好使用 `-sourcepath` 代替 `-classpath`。

`-d` *dir*
在 *dir* 中建立产生的 HTML 文件。

`-docencoding` *encoding*
对产生的 HTML 文件使用 *encoding*。

`-encoding` *encoding*
使用 *encoding* 对 Java 源代码进行编码。

`-J` *opt*
将 *opt* 传递给运行时系统，更多的信息可参见 **java**。

`-nodeprecated`
排除用 `@deprecated` 标记的段落。

`-noindex`
不产生包索引。

`-notree`
不产生类和接口的层次。

`-package`
只包括包、保护型和公共的类及成员。

`-private`
包括所有的类和成员。

`-protected`
只包括保护型和公共的类和成员。这是默认选项。

`-public`
只包括公共的类和成员。

`-sourcepath` *path*
使用*path*作为类源文件的搜索路径。*path*是使用冒号间隔的目

javadoc

录列表。如果没有指定该选项，则默认使用当前的 `-classpath` 目录。在该目录中运行 `javadoc` 操作源文件时，可以省略该选项。

`-verbose`
输出用于分析源文件的时间的额外消息。

`-version`
包括 `@version` 标签。

不再使用 `-doctype` 选项。只可能产生 HTML 文档。

javah

`/usr/java/bin/javah [`*`options`*`] `*`classes`*` | `*`files`*

只适用于 Solaris。生成 C 文件头及（或）源文件，用于实现本机的方法，生成的 `.h` 文件定义了一种结构，其中的成员与相应的 Java 类中的成员相似。头文件名来自相应的 Java 类。如果该类在一个包中，则包名会添加到文件名及结构名前面，并使用下划线进行区分。

注意：Java 本地接口（Java Native Interface，JNI）不需要头文件和存根文件。使用 `-jni` 选项可以建立 JNI 本地方法的函数原型。

options

`-classpath` *`path`*
使用 *path* 作为类文件的搜索路径来代替 $CLASSPATH。*path* 是使用冒号间隔的目录列表。

`-d` *`dir`*
将产生的文件放置在 *dir* 中。

`-help`
输出一条帮助消息。

`-jni`
生成 JNI 本地方法的函数原型。

`-o` *`file`*
连接为所有类生成的文件头和源文件，并将它们写入文件 *file* 中。

`-stubs`
生成 C 语言声明，而不是文件头。

javah

`-td` *dir*
使用 *dir* 代替 `/tmp` 作为保存临时文件的目录。

`-trace`
向生成的存根中添加跟踪信息。

`-v` 显示详细信息。

`-version`
输出 `javah` 的版本号。

javakey

`/usr/java/bin/javakey [`*options*`]`

只用于 Solaris。Java 的安全工具。使用 `javakey` 可以生成档案文件的数字签名，并建立及管理实体、它们的密钥和证书的数据库，还标明它们的“受信任”（或不受信任）状态。

选项前面的符号（`-`）可以省略。每个 `javakey` 调用中只能指定一个选项。

options

在下面的选项参数中，*id_or_signer* 是已经保存在数据库中的一个安全 ID 或一个安全签名者。

`-c` *identity* `[true | false]`
产生一个新的数据库身份 *identity*。选项中的 `true` 或 `false` 用于指明是否可以信任 *identity*，默认选项是 `false`。

`-cs` *signer* `[true | false]`
在数据库中产生一个名字为 *signer* 的签名者，选项 `true` 和 `false` 用于指示是否可以信任该签名者。默认值是 `false`。

`-dc` *file*
显示 *file* 中的证书。

`-ec` *id_or_signer cnum cfile*
将来自 *id* 或签名者的证书 *cnum* 输出到 *cfile* 中，号码必须由 `javakey` 提前产生。

`-ek` *id_or_signer public* `[`*private*`]`
将用于 *id* 或 *signer* 的公钥（public key）输出到文件 *public* 中。

javakey

可选择将私钥（private key）输出到文件 *private* 中。密钥必须符合 X.509 格式。

`-g signer algorithm ksize [public] [private]`
是 `-gk` 的缩写形式，用于为 *signer* 产生一个密钥对。

`-gc` *file*
按照 *file* 中的指令产生一个证书。

`-gk` *signer algorithm ksize* [*public*] [*private*]
使用标准算法 *algorithm* 来产生一个密钥对，密钥大小为 *ksize* 位。公钥放置在文件 *public* 中，私钥放置在文件 *private* 中。输出私钥时应该给予警告。

`-gs` *dfile jarfile*
按照 *dfile* 中的指令对 Java 档案文件 *jarfile* 进行签名。

`-ic` *id_or_signer csrcfile*
将保存在 *csrcfile* 中的公钥证书与指定的 *id* 或 *signer* 进行关联。如果已经存在一个证书，则该证书必须与以前的证书匹配；否则，该证书会分配给 *id* 或 *signer*。

`-ii` *id_or_signer*
提供有关 *id* 或 *signer* 的信息。`javakey` 读取交互键入的信息。使用包含一个小点的行来结束该信息。

`-ik` *identity ksrcfile*
将保存在 *ksrcfile* 中的公钥与 *identity* 进行关联，该公钥必须符合 X.509 格式。

`-ikp` *signer public private*
从文件 *public* 和 *private* 中导入密钥对，并将它们与 *singer* 进行关联。这些密钥必须符合 X.509 格式。

`-l` 列出数据库中所有身份和签名者的用户名。

`-ld` 与 `-l` 相似，但是该选项能够提供详细的信息。

`-li` *id_or_signer*
仅提供指定的 *id* 和 *signer* 的详细信息。

`-r` *id_or_signer*
从数据库中删除 *id* 或 *signer*。

javakey

`-t` *`id_or_signer`* `[true | false]`

设置或重置 *id* 或 *signer* 的信任等级。

示例

建立一个新的、可以信任的身份 `arnold`：

```
javakey -c arnold true
```

列出 `arnold` 的详细信息：

```
javakey -li arnold
```

javald

`/usr/java/bin/javald [`*`options`*`] `*`class`*

只适用于Solaris。建立一个Java应用程序的包装。`javald`会产生一个程序，运行时，会在适当环境中运行指定的 Java 程序。对于只是希望运行该应用程序的用户，这会隐藏合适的 CLASSPATH 环境变量等方面的内容。

options

`-C` *`path`*

将 *path* 添加到运行应用程序的CLASSPATH中。该选项可以使用多次。

`-H` *`dir`*

将 JAVA_HOME 环境变量设置为 *dir*。

`-j` *`list`*

向 `java` 中传递 *list*。多个选项应该用引号引起来。

`-o` *`wrapper`*

在文件 *wrapper* 中放置已经产生的包装。

`-R` *`path`*

将 *path* 添加到运行应用程序时使用的LD_LIBRARY_PATH环境变量中。这允许 `java` 查找 `native` 方法。

javap

`/usr/java/bin/javap [`*`options`*`] `*`classfiles`*

只用于Solaris。反汇编Java类文件并输出结果。默认情况下，`javap` 会输出指定类的公共字段和方法。

javap

options

-b 被忽略。主要为了与 JDK 1.1 中的 javap 向后兼容。

-c 输出给定类中各个方法的反汇编生成的字节码。

-classpath *path*
使用 *path* 代替 $CLASSPATH 作为类文件的搜索路径。*path* 是使用冒号间隔的一个目录列表。

-h 产生可以用于 C 头文件中的代码。

-J *option*
将 *option* 直接传递给 java。

-l 显示行号和局部变量信息。

-package
只反汇编包、保护型及公共类和方法。该选项是默认设置。

-private
反汇编所有的类和成员。

-protected
只反汇编保护型及公共类和成员。

-public
只反汇编公共类和成员。

-s 显示内部的类型签名。

-verbose
输出各个方法的堆栈大小、参数个数及局部变量的个数。

-verify
运行 Java 校验程序。

-version
输出 javap 的版本。

jdb

/usr/java/bin/jdb [*options*] [*class*]

只用于 Solaris。jdb 是 Java 调试器（Java Debugger）。它是一个面向行的调试器，类似于传统的 Unix 调试器，它可以实现对本地或远程 Java 解释器的检查和调试。

jdb

jdb可用来代替java, 此时要运行的程序已经在调试器中启动起来；或者它可以用来连接一个已经运行的java会话。后一种情况，java必须使用-debug选项进行了启动，该选项会产生一个密码供你随后在 jdb 命令行中输入。

options

-host *host*

连接到主机上正在运行的 Java 解释器。

-password *password*

使用*password*连接到已经运行的Java解释器。该密码使用java -debug 提供。

join

join [*options*] *file1 file2*

连接排序文件 *file1* 和排序文件 *file2* 中共有的行，如果 *file1* 是“ - ”，则读取标准输入。输出包含 *file1* 和 *file2* 中的公共字段和各个行的剩余部分。下面的选项中，*n* 可以是 1 或 2，分别对应 *file1* 和 *file2*。

options

-a[*n*]

列出文件*n*（如果省略*n*，则指两个文件）中不成对的行。Solaris 中不允许省略 *n*。

-e *s*

使用字符串 *s* 来替换所有空的输出字段。

-jn *m*

将文件*n*（如果省略*n*，则指两个文件）中的第 *m* 个字段连接起来。

-o *n.m*

各个输出行都包含文件号为 *n* 而字段号为 *m* 的字段。如果没有要求，则会禁止公共字段。

-t*c* 使用字符 *c* 作为输入和输出的分隔符。

-v *n*

只输出文件 *n* 中不成对的行。使用 -v 1和 -v 2，则会输出所有不成对的行。该选项只用于 Solaris。

join

`-1` *m*

连接文件1中的字段*m*。字段从1开始。该选项只适用于Solaris。

`-2` *m*

连接文件2中的字段*m*。字段从1开始。该选项只适用于Solaris。

示例

假设使用下面的输入文件：

```
$  cat score
olga    81      91
rene    82      92
zack    83      93

$  cat grade
olga    B       A
rene    B       A
```

按照等级列出分数，包括不匹配的行：

```
$  join -a1 score grade
olga 81 91 B A
rene 82 92 B A
zack 83 93
```

将每个分数与其等级配对：

```
$  join -o 1.1 1.2 2.2 1.3 2.3 score grade
olga 81 B 91 A
rene 82 B 92 A
```

jre

`/usr/java/bin/jre [`*options*`] `*class*` [`*arguments*`]`

只适用于 Solaris。Java 运行时环境。该程序实际上会运行已经编译的 Java 文件。

options

`-classpath` *path*

使用*path*作为类文件的搜索路径，覆盖$CLASSPATH，*path*是一个使用冒号间隔的目录列表。

`-cp` *pathlist*

将一个或多个路径添加到$CLASSPATH值前面。当提供多个路

jre

径时，使用冒号间隔各个路径。组件可以是目录，也可以是将要执行的文件的完整名称。

`-D`*`prop=val`*

将 *prop* 的值重新定义为 *val*，该选项可以使用多次。

`-help`

输出一条帮助消息。

`-ms` *`size`*

设置堆的初始大小为 *size*，以字节作为计量单位。如果添加了 `k` 或 `m`，则分别表示千字节或兆字节。

`-mx` *`size`*

设置堆的最大值为 *size*，以字节作为计量单位。如果添加了 `k` 或 `m`，则分别表示千字节或兆字节。

`-noasyncgc`

禁止异步的无用信息回收。

`-noclassgc`

禁止 Java 类的无用信息回收。

`-nojit`

不进行 JIT 编译，而使用默认的解释器。

`-noverify`

禁止校验。

`-oss` *`size`*

在一个Java线程中设置Java代码的最大堆栈的大小，以字节表示，添加`k`或`m`分别表示千字节或兆字节。默认最大值是400KB。

`-ss` *`size`*

在一个Java线程中设置C代码的最大堆栈的大小，以字节表示，添加`k`或`m`分别表示千字节或兆字节。默认的最大值是128KB。

`-v, -verbose`

每装载一个类文件时，在标准输出中输出一条消息。

`-verbosegc`

无用信息回收程序每次释放内存时，输出一条消息。

jre

`-verify`
在所有代码上面运行字节码校验程序。注意，这样只会校验实际执行的字节码。

`-verifyremote`
在一个类装载器装载的所有代码上面运行校验程序。当进行解释时，这是默认设置。

jsh

`jsh [`*`options`*`] [`*`arguments`*`]`

`sh`（即 Bourne shell）的作业控制版本。这会提供标准 shell 的前台和后台进程的控制，参见第四章。

keylogin

`keylogin [-r]`

只适用于 Solaris。提示用户输入一个密码，然后使用它解密该用户的密钥。该密钥由安全网络服务（比如 Secure NFS、NIS+）使用。只有在用户登录时没有提示输入一个密码的情况下需要使用 keylogin。`-r` 选项会更新 `/etc/.rootkey`，只有特权用户才可以使用该选项。参见 **chkey** 和 **keylogout**。

keylogout

`keylogout [`*`option`*`]`

只适用于 Solaris。取消对安全网络服务（比如 Secure NFS、NIS+）使用的密钥的访问（删除）。参见 **chkey** 和 **keylogin**。

option

`-f` 忘记根密钥。如果在一台服务器上指定，NFS 的安全会被破坏。所以使用该选项时要当心。

kill

`kill [`*`options`*`] `*`IDs`*

终止一个或多个进程 *ID*，此时你必须拥有该进程或是一个特权用户。该命令与 Bourne、Korn 和 C shell 中内置的 kill 命令相似。*ID* 前面的中划线用于指定一个进程组 *ID*（内置版本不允许进程组 ID，但允许作业 ID）。

kill

options

`-l` 列出信号的名称（自身使用）。

`-s` *signal*

将信号 *signal* 发送到给定的进程或进程组，使用信号编号（来自 `/usr/include/sys/signal.h`）或名称（来自 `kill -l`）。使用一个编号为 9 的信号时，kill 是绝对的。只适用于 Solaris。

`-`*signal*

将信号 *signal* 发送到给定的进程或进程组。

ksh

`ksh [`*options*`] [`*arguments*`]`

Korn shell命令解释器，更多的信息可参见第四章，包括命令行选项。

ld

`/usr/ccs/bin/ld [`*options*`]` *objfiles*

将几个目标文件*objfiles*按照指定的顺序合并进一个可执行目标模块中（默认是 `a.out`）。`ld` 是一个装载器，通常编译器命令（比如 `cc`）会自动调用它。

options

`-a` 强制静态链接使用默认行为（产生一个目标文件并列出未定义的引用）。不要与 `-r` 选项一起使用。

`-b` 忽略共享引用符号的特殊处理（只有动态链接）。输出变得非常有效，但是很少共享。

`-B` *directive*

遵守下面的指令之一：

`dynamic` 当装载时，使用动态库（*lib*`.so`）和静态库（*lib*`.a`）来解析未知的符号。

`eliminate` 删除没有分配一个版本定义的符号。只适用于 Solaris。

`group` 将一个共享对象及其从属内容看成是一个组。意味着 `-z defs`，只适用于 Solaris。

`local` 将没有分配一个版本定义的全局符号看成是局部符号，只适用于 Solaris。

ld

`reduce` 减少版本定义指定的符号信息。只适用于Solaris。

`static` 装载时，只使用静态库（*lib*`.a`）来解析未知的符号。

`symbolic` 在动态链接中，将一个符号绑定到其局部定义中，而不是绑定到其全局定义中。

`-d[`*c*`]`
动态（*c* 是 `y`）或静态（*c* 是 `n`）地链接。默认值是动态链接。

`-D`*token*`,...`
输出标记（token）指定的调试信息，使用 `help` 可以获得可能取值的列表。只适用于 Solaris。

`-e` *symbol*
设置 *symbol* 为输出文件入口点的地址。

`-f` *obj*
使用正在建立的共享对象的符号表作为共享对象*obj*上的辅助过滤器，不要与 `-F` 一起使用。只适用于 Solaris。

`-F` *obj*
使用正在建立的共享对象的符号表作为共享对象 *obj* 上的过滤器，不要与 `-f` 一起使用。只适用于 Solaris。

`-G` 在动态链接中，建立一个共享对象并允许使用未定义的符号。

`-h` *name*
使用*name*作为共享目标文件来在动态链接过程中搜索（默认是 Unix 目标文件）。

`-i` 忽略 LD_LIBRARY_PATH。主要用于避免对正在建立的可执行程序的运行时搜索造成不必要的影响。只适用于 Solaris。

`-I` *name*
使用*name*作为装载器（解释器）的路径名以便写入到程序头中，默认是 none（静态）或 `/usr/lib/libc.so.1`（动态）。

`-l`*x* 搜索名字为`lib`*x*`.so`或`lib`*x*`.a`的库（在行上放置该选项会影响开始库搜索的时间）。

`-L` *dir*
在标准搜索目录前搜索目录 *dir*（该选项必须放在 `-l` 之前）。

ld

`-m` 显示输入 / 输出节的内存描述情况。

`-M` *mapfile*
从 *mapfile* 中调用 `ld` 指令（`-M` 会搞乱输出，建议不要使用）。

`-N`*string*
将值为 *string* 的一个 `DT_NEEDED` 项添加到正在建立对象的 `.dynamic` 节中。只适用于 Solaris。

`-o` *file*
将输出发送到文件 *file* 中（默认是 `a.out`）。

`-Q`*c* 在输出中列出 `ld` 的版本信息（*c* = `y`，默认值），或者不列出（*c* = `n`）。

`-r` 允许输出服从另一个 `ld` 命令（保留重新分配的信息）。

`-R` *path*
将 *path* 中使用冒号间隔的目录列表记录到运行时装载器使用的目标文件。可以提供多个实例，所有的值会连接到一起。

`-s` 删除符号表并重新分配项。

`-t` 禁止对多个大小不同的定义符号的警告。

`-u` *symbol*
在符号表中输入符号 *symbol*。当从一个档案库装载时比较有用。*symbol* 必须放在定义该符号的库的前面（所以 `-u` 必须放在 `-l` 的前面）。

`-V` 输出 `ld` 的版本。

`-YP,`*dirlist*
指定一个使用逗号间隔的目录列表来代替默认的搜索目录（参见 `-L`）。

`-z defs | nodefs | text`
指定 `nodefs` 以便允许使用未定义的符号。默认值 `defs` 将未定义的符号当作致命错误。当存在非可写的重定位时，使用 `text` 可以产生一个错误。

`-z` *directive*
只适用于 Solaris，遵循下面的指令之一：

`allextract` 提取所有的档案成员。

ld	`combreloc`	合并多个重定位的节。
	`defaultextract`	返回默认的档案提取规则。
	`ignore`	忽略没有作为链接一部分引用的动态依赖关系。
	`initfirst`	只用于共享对象，该对象的初始化放在同时添加到进程中的其他对象的初始化前面。同样，它的结束部分在其他对象结束之后运行。
	`lazyload`	标记惰性装载的动态依赖关系。
	`loadfltr`	在运行时而不是在第一次绑定时标记即时处理的过滤对象。
	`muldefs`	允许符号的多次定义，并使用第一个出现的定义；否则，符号的多次定义是一个致命错误。
	`nodefs`	允许未定义的符号，这是共享对象的默认值，这种行为对于可执行程序是未定义的。
	`nodelete`	标记对象为在运行时不可删除。
	`nodlopen`	只用于共享对象。该对象不能从*dlopen*(3x)获得。
	`nolazyload`	不标记惰性装载的动态依赖关系。当第一次绑定到一个对象而不是在进程启动时装载惰性装载对象。
	`nopartial`	将输入可重定向对象中的部分初始化符号扩展到生成的输出文件中。
	`noversion`	不包括任何版本的节。
	`now`	强制对象进行非惰性运行时的绑定。
	`origin`	对象需要在运行时进行即时的 $ORIGIN 处理。
	`record`	记录没有作为链接的一部分进行引用的动态依赖关系。这是默认值。

ld

`redlocsym`	从 SHT_SYMTAB 符号表中删除 SECT 以外的所有局部符号。
`textoff`	在动态模式中，允许对所有节进行重新定位。包括那些不可写的节，这是共享对象的默认值。
`textwarn`	只用于动态模式。如果那里保留任何非可写、可分配的节的重定位，则会发出警告。这是可执行程序的默认设置。
`weakextract`	允许使用“弱”定义来触发档案提取。

ldd

`ldd [`*`option`*`] `*`file`*

列出动态依赖关系，即如果执行了文件 *file*，则会列出将要装载的共享对象。(如果一个有效的 *file* 不需要共享的对象，`ldd` 会执行成功，但是不产生任何输出)。另外，`ldd` 的选项可以显示运行 *file* 引起的未解析符号引用。

option

下列选项中一次只能使用一个：

`-d` 只检查对数据对象的引用。

`-r` 检查对数据对象和函数的引用。

Solaris 选项

下面这些附加的选项只能用于 Solaris 中：

`-f` 强制检查不安全的可执行文件。作为特权用户运行 `ldd`命令时，使用该选项是非常危险的。

`-i` 输出初始化节的执行顺序。

`-l` 为了列出所有的被过滤者和它们的依赖关系，对任何过滤器进行即时处理。

`-s` 显示共享对象依赖关系的搜索路径。

`-v` 显示所有的依赖关系和版本要求。

lex

/usr/ccs/bin/lex [*options*] [*files*]

产生一个基于包含在一个或多个输入 *files* 中的正则表达式和 C 语句的词法分析程序（名字为 lex.yy.c）。参见参考文献中的 **yacc** 和 *lex & yacc*。

options

-c C 语言（默认）中的 *file* 的程序语句。

-e 处理 EUC（Extended Unix Code，扩展的 Unix 代码，8 位）字符。该选项与 -w 选项不能同时使用。将 yytext[] 的类型设置为无符号字符。该选项只用于 Solaris。

-n 禁止输出概要。

-Q*c* 输出 lex.yy.c（如果 *c* = y）中的版本信息或禁止显示信息（如果 *c* = n，默认值）。

-t 将程序写入到标准输出中，而不是 lex.yy.c 中。

-v 输出计算机产生的统计信息的概要。

-V 在标准错误中输出版本信息。

-w 处理 EUC（8 位或更宽的）字符。不能与 -e 选项一起使用。将 yytext[] 的类型设置为 wchar_t。只用于 Solaris。

line

line

从标准输入中读取下一行内容并将其写入到标准输出中，退出状态是 *EOF* 之上的 1 。主要用于在 csh 脚本中读取终端上的内容。

示例

输出来自 who 的输出的前两行：

```
who | ( line ; line )
```

lint

/usr/ccs/bin/lint [*options*] *files*

在指定的 C 程序中检测 bug、可移植问题和其他可能的错误。默认情况下，lint 使用 C 库 llib-lc.ln 中的定义。如果需要，来自 .c 文件的输出可以保存到具有 .ln 后缀的 “ 目标文件 ” 中。第二个 lint

lint

命令可以在 .ln 文件或库中调用，以便进行进一步的检查。lint 接受 cc 选项 -D、-I 和 -U，也可以接受特定于系统的其他 cc 选项。参见参考文献中列出的《Checking C Programs with lint》一书。注意：该命令检查使用 ANSI C 编写的程序。如果希望检查使用 pre-ANSI C 编写的程序，应使用 /usr/ucb/lint。还应注意在 BSD 和 System V 中，-a、-b、-h 和 -x 选项的意义正好相反。

options

-a 忽略分配到不是 long 变量中的 long 值。

-b 忽略不会到达的 break 语句。

-c 不执行 lint 的第二次传递，将来自第一次传递的输出保存到 .ln 文件中（与 BSD -i 选项相同）。

-F 使用完整的路径名输出文件，而不只是文件名。

-h 不会对 bug、较差的风格或额外的信息进行测试。

-k 重新显示被 /* LINTED [*message*] */ 指令禁止的警告，并输出附加的消息 *message*（如果已经指定）。

-L*dir*
在搜索标准目录以前，先在目录 *dir* 中搜索 lint 库。

-l*x* 除了 llib-lc.ln 以外还使用 llib-l*x*.ln 库。

-m 忽略可能是静态的外部声明。

-n 不检查兼容性。

-o *lib*
从 lint 第一次传递的输出中建立名字为 llib-l.*lib* 的 lint 库。

-p 检查 C 变体的可移植性。

-R*file*
将 .ln 输出（来自 a.c 文件）放入 *file* 中，以便 cxref 可以使用。

-s 产生短（一行）的诊断。

-u 忽略未定义或未使用的函数或外部变量。

-v 忽略函数中未使用的参数，与指定 /* ARGSUSED */ 指令相同。

lint

`-V` 在标准错误中输出产品的名称和版本。

`-W`*file*
除了 *file* 是为 `cflow` 准备的以外，其他功能与 `-R` 选项相同。

`-x` 忽略被外部声明引用的未使用的变量。

`-y` 与 `/* LINTLIBRARY */` 指令作用相同，该指令与提供 `-v` 和 `-x` 选项作用相同。

listusers

`listusers [`*options*`]`

只用于Solaris。列出所有用户，可以选择按照组或按照特定的用户。

options

`-g` *grouplist*
在使用逗号间隔的组列表 *grouplist* 中列出所有的用户。

`-l` *users*
只列出指定的 *users*，并按照登录顺序排列。也可能会提供一个使用逗号间隔的列表。

ln

`ln [`*options*`]` *file1 file2*
`ln [`*options*`]` *files directory*

建立文件 *files* 的假名（链接），允许使用不同的名字访问这些文件。第一种格式会将 *file1* 链接到 *file2*，其中 *file2* 通常是一个新文件名。如果 *file2* 是一个已经存在的文件，则首先会删除该文件；如果 *file2* 是一个已经存在的目录，则在该目录中产生一个名字为*file1*的链接。第二种格式会在目录 *directory* 中建立多个链接，各个链接与指定的文件具有相同的名字。

options

`-f` 强制建立链接（在覆盖前不进行提示）。

`-n` 不会覆盖已经存在的文件。

`-s` 建立一个符号链接。这允许你跨文件系统建立链接，当运行 `ls -l` 时可以看到链接的名字（如果不使用该选项，则不得不使用 `find -inum` 命令来查找一个文件链接的其他名字）。

locale

locale [*options*] [*name* ...]

只用于Solaris，显示特定于场所的信息。不使用参数时，locale会汇总当前场所的信息。根据使用的参数，locale会输出整个场所的类别或一个场所内特定项的值等信息。一个公共场所是一个可以访问的应用程序。参见**localedef**。

options

-a 输出所有可用公共场所的信息，POSIX场所应该总是可用的。

-c 提供场所类别 *name* 的信息，有时与 -k 一起使用，有时不与 -k 一起使用。

-k 输出给定场所关键字的名称和值。

-m 输出可用的 charmap（字符映射）的名称。

localedef

localedef [*options*] *localename*

只用于Solaris。localedef从标准输入或使用 -i 选项指定的文件中读取一个场所的定义，格式记录在 *locale*(5)参考页中，它会产生一个临时C源文件，该文件被编译进一个共享目标库中。使用特定于场所的环境变量设置的程序会使用该库文件，以便返回给定场所的正确值。

生成文件的名字为 *localename*.so.*version*，默认的32位版本使用 /usr/lib/locale/*localename*/*localename*.so.*version*，SPARC系统的64位环境使用 /usr/lib/locale/*localename*/sparcv9/*localename*.so.*version*。

options

-c 建立一个共享的目标文件，即使出现警告也如此。

-C *options*
将 *options* 传递到C编译器。该选项一般与 -W cc 一起使用。

-f *mapfile*
文件*mapfile*提供字符符号的一种映射，并对元素符号和实际字符编码进行比较。如果使用符号名称定义场所，则一定要使用该选项。

localedef

`-i` *localefile*
从 *localefile* 而不是从标准输入中读取场所定义。

`-L` *options*
在C源文件名字的后面将 *options* 传递到C编译器。该选项一般不与 `-W cc` 一起使用。

`-m` *model*
指定 `-m ilp32` 来产生 32 位的目标文件（这是默认值）。使用 `-m lp64` 可以产生 64 位的目标文件（只用于SPARC）。

`-W cc,`*args*
将 *args* 传递到 C 编译器，各个参数通过逗号间隔。

`-x` *exfile*
从扩展文件 *exfile* 中读取附加的选项。

示例

为Klingonese建立一个64位的共享目标场所文件，忽略任何警告消息：

```
localedef -c -m lp64 -i klingon.def klingon
```

logger

`logger [`*options*`] [`*messages*`]`

只用于Solaris，把消息记录到系统的日志中，如果有命令行消息，也会记录；否则，消息会通过 `-f` 提供的文件逐行读取并记录到日志中。如果没有给出任何文件，则 `logger` 从标准输入中读取消息。

options

`-f` *file*
读取 *file* 中的消息并记录到日志中。

`-i` 使用各自的消息来记录 `logger` 进程的进程ID。

`-p` *priority*
使用给出的优先权 *priority* 将各条信息记录到日志中，优先权的形式为 *facility.level*，默认值是 `user.notice`。更多的信息可以参见 *syslog*(3)。

logger

`-t` *tag*

向各个消息行添加标签 *tag*。

示例

警告即将出现的问题：

```
logger -p user.emerg 'Incoming Klingon battleship!'
```

login

`login [`*options*`]`

注册到系统并标识自己的身份。在每个终端会话开始的时候，系统会提示用户输入用户名和密码（如果有关的话）。选项一般不使用。

Korn shell 和 C shell 有自己内置版本的 `login`，更多的信息参见第四章和第五章。

options

user

作为用户 *user* 进行注册（代替系统的提示）。

`-d`*tty*

指定 *tty* 的路径名作为登录端口。

`-h host [`*term*`]`

该选项用于通过 `telnet` 进行远程登录，以指明登录来自主机 *host*，用户的终端类型是 *term*。只用于 Solaris。

`-p` 将当前的环境传递给新登录的会话，只用于 Solaris。

`-r` *host*

该选项用于通过 `rlogin` 进行远程登录，以指明登录来自主机 *host*。只用于 Solaris。

`var=`*value*

在用户名后面指定该选项时，为一个或多个环境变量分配值。注意，PATH 和 SHELL 不能改变。

value

将多个值传递给环境变量。不包含等号（=）的值分配给形式为 `L`*n* 的变量，*n* 从 0 开始递增。该选项只用于 Solaris。

logname

logname

显示用户的登录名。SVR4会输出位于/etc/profile目录下的LOGNAME环境变量的值。Solaris在/var/adm/utmp目录中搜索用户，该目录下保存着登录用户的信息。参见**whoami**。

look

look [*options*] *string* [*file*]

只用于Solaris。遍历一个排序文件并输出所有开头为字符串*string*的行，该字符串最多能使用256个字符。该程序执行速度比fgrep快，因为它使用的*file*已经排序，它可以使用二分法搜索整个文件，而不是从头到尾按顺序进行查找。

如果没有提供*file*，look会使用-df选项来搜索/usr/share/lib/dict/words目录（拼写目录）。

options

-d 按照字典顺序进行搜索。只使用字母、数字、空格和制表符进行比较。

-f 在比较时忽略大小写。

-t *char*
使用字符*char*作为终止字符，也就是说，忽略*char*右边的所有字符。

lookbib

lookbib *database*

程序refer套件的一部分，参见第十七章。

lp

lp [*options*] [*files*]

将文件*files*发送到打印机。不使用参数时，打印标准输入。要与其他文件一起打印标准输入，可以指定中划线（-）为*files*之一。

options

-c 将文件*files*拷贝到打印机的假脱机系统中，如果在*file*仍在排队等待打印时对其进行了修改，则不会影响打印结果。

lp

`-d` *dest*
将输出发送到目标打印机 *dest*。

`-d any`
该选项放在 `-f` 和 `-s` 的后面，用于打印支持给定格式或字符集的任何打印机上的请求。

`-f` *name*
在预打印格式 *name* 上面打印请求，*name* 引用由管理命令 `lpforms` 设置的打印机属性集。

`-H` *action*
按照指定的动作 *action* 进行打印：`hold`（在打印前进行通知）、`resume`（恢复一个挂起的请求）、`immediate`（打印下一个，只适用于特权用户）。

`-i` *IDs*
覆盖当前在队列中的请求 ID 使用的 `lp` 选项，并在 `-i` 后面指定新的 `lp` 选项。例如，可以改变发送副本的数量。

`-m` 在打印 *files* 之后发送邮件。

`-n` *number*
指定要打印的副本的数量。

`-o` *options*
设置一个或多个特定于打印机的选项。标准选项如下：

`cpi=`*n*	每英寸打印几个字符。*n* 可以是 `pica`、`elite` 或压缩的字号。
`lpi=`*n*	每英寸打印 *n* 行。
`length=`*n*	打印的页面为 *n* 个单位长，比如，`11i`（英寸）、`66`（行）。
`nobanner`	从请求中删除标题页面（分隔符）。
`nofilebreak`	禁止打印不同的文件时进纸。
`width=`*n*	打印的页面为 *n* 个单位宽，比如，`8.5i`（英寸）、72（列）。
`stty=`*list*	指定 `stty` 选项的一个用括号引起来的列表。

lp

`-p` 打印作业完成后进行通知。只用于 Solaris。

`-P` *list*
只打印在 *list* 列表中指定的页号。

`-q` *n*
使用优先权级别 *n* 打印请求（39 是最低级别）。

`-r` 如果 *content* 不合适则不接受请求，拒绝使用其他内容替代（已经淘汰；只能与 `-T` 一起使用）。

`-s` 禁止消息。

`-S` *name*
使用指定的字型轮和字符集进行打印。

`-t` *title*
在打印输出的标题页上使用标题 *title*。

`-T` *content*
向支持 *content* 的一台打印机发送请求（默认是 `simple`，系统管理员可以通过 `lpadmin -I` 设置 *content*）。

`-w` 在打印完 *files* 后，将一条消息写入到用户的终端上（如果用户没有登录，则与 `-m` 选项相同）。

`-y` *modes*
按照本地定义的模式 *modes* 进行打印。

示例

在打印 5 份 `report` 后发送邮件：

```
lp -n 5 -m report
```

格式化及打印 `thesis`，并打印 `title`：

```
nroff -ms thesis | lp - title
```

lpq

`/usr/ucb/lpq` [*options*] [*job#s*] [*users*]

显示打印机队列。标准 SVR4 使用 `lpstat`。

lpr

`/usr/ucb/lpr` [*options*] [*files*]

将 *files* 发送到打印机。标准 SVR4 使用 `lp`。

lprm

/usr/ucb/lprm [*options*] [*job#s*] [*users*]

从打印机队列中删除请求。标准 SVR4 使用 cancel。

lprof

lprof [*options*]
lprof -m *files* [-T] -d *out*

只用于SVR4。逐行显示一个程序的描述文件数据，数据包括源文件的一个列表、各个源代码行（带有行号）、各个行执行的次数等。默认情况下，lprof解释描述文件 *prog*.cnt。当编译一个程序或产生一个名字为 *prog*（默认是 a.out）的共享对象时，则使用指定的 cc -ql来产生该文件。PROFOPTS 环境变量在运行时可以控制文件的配置，参见 **prof** 和 **gprof**。

options

-c *file*
读取输入的描述文件 *file*，而不是读取 *prog*.cnt。

-d *out*
在文件 *out* 中存储合并的描述文件数据。必须与 -m 一起使用。

-I *dir*
在 *dir* 及默认的目录（/usr/include）中搜索包括文件。

-m *files*
合并多个描述文件 *files* 并统计执行的总次数。*files* 使用的形式为 *f1*.cnt、*f2*.cnt、*f3*.cnt，其中各个文件包含的配置文件数据来自同一个程序的不同运行版本。该选项与 -d 一起使用。

-o *prog*
查看一个程序的描述文件，该程序使用名字 *prog* 来代替产生描述文件时使用的名字。如果文件已经改名或者移动到其他位置，则需要使用 -o。

-p 输出默认的列表。主要与 -r 和 -s 一起使用。

-r *list*
与 -p 一起使用，只输出 *list* 中列出的源文件。

-s 对于各个函数，输出已经执行的代码行所占的百分比。

lprof

`-T` 忽略正在描述的可执行文件的时间标记。通常情况下，检查文件执行的时间是为了确保使用一个执行文件的相同版本建立不同的描述文件。

`-V` 在标准错误中输出 `lprof` 的版本。

`-x` 忽略执行的次数。对于已经执行的行，只显示行号；对于没有执行的行，显示行号、符号`[U]`和源文件中的行。

lpstat

`lpstat [`*`options`*`]`

输出 `lp` 打印队列的状态，使用带有一个 *list* 参数的多个选项，忽略该列表为选项产生的所有信息。可以使用逗号区分多个 *list* 项；如果使用了双引号包括列表项，则可以使用空格间隔多个列表项。

options

`-a [`*`list`*`]`
显示 *list* 列出的打印机或种类名是否正在接受请求。

`-c [`*`list`*`]`
显示在 *list* 中指定的打印机的种类信息。

`-d` 显示默认打印机的目的地。

`-D` 用于 `-p` 之后，显示打印机的一个简要描述。

`-f [`*`list`*`]`
验证 `lp` 可以识别的形式列表。

`-l` 放在 `-f` 之后用于描述可用的形式；放在 `-p` 之后用于显示打印机的配置；放在 `-S`之后用来描述适合特定字符集或字型轮的打印机。

`-o [`*`list`*`]`
显示输出请求的状态，*list* 包含打印机名字、种类名或请求 ID。

`-p [`*`list`*`]`
显示在 *list* 中指定的打印机的状态。

`-r` 显示打印调度程序是开还是关。

`-R` 显示作业在打印队列中的位置。

`-s` 汇总打印状态（显示几乎所有的信息）。

lpstat

`-S [`*list*`]`
验证 `lp` 可以识别字符集或字型轮的 *list*。

`-t` 显示所有状态信息（报告所有内容）。

`-u [`*list*`]`
显示在 *list* 中用户的请求状态，*list* 可以是如下值：

`user`	本地计算机上的用户。
`all`	所有计算机系统中的所有用户。
`host!user`	*host* 计算机上的用户。
`host!all`	*host* 计算机上的所有用户。
`all!user`	所有计算机系统中的用户。
`all!all`	所有计算机系统中的所有用户。

`-v [`*list*`]`
显示与 *list* 中指定的各个打印机相关联的设备。

ls

`ls [`*options*`] [`*names*`]`

如果没有给出*names*，则显示当前目录下的所有文件。当使用一个或多个*names*参数时，列出包含在目录*name*中的文件或匹配文件*name*的文件。*options* 可以使用不同的格式来显示各种不同的信息。最有用的选项包括 `-F`、`-R`、`-a`、`-l` 和 `-s`。有一些选项不能在一起使用，比如 `-u` 和 `-c` 选项。

注意：Solaris 中的 `/usr/bin/ls` 命令会使用 LC_COLLATE 环境变量。其默认值 `en_US`（用于美国）可以使 `ls` 按照目录顺序进行排序（会忽略大小写）。将 LC_COLLATE 设置为 C，可以恢复按照 ASCII 码顺序排序的传统的 Unix 行为，或者使用 `/usr/ucb/ls`。

options

`-a` 列出所有文件，包括一般的隐含文件 .files。

`-A` 与 `-a` 选项作用相似，但是不包括 `.` 和 `..`（当前目录及其父目录）。该选项只用于 Solaris。

`-b` 显示八进制格式的非打印字符。

`-c` 按照信息节点修改的时间列出所有文件。

ls

`-C` 列出所有的文件（在一个终端设备上进行显示的默认格式）。

`-d` 只列出目录信息，不列出目录中的内容（通常与 `-l` 和 `-i` 选项一起使用）。

`-f` 将各个 *name* 解释为目录（忽略所有的文件）。

`-F` 通过向目录添加 `/`、向资料联机检索系统（doors）添加 `>`（只用于 Solaris）、向可执行文件添加 `*`、向先进先出（fifo）队列添加|、向符号链接添加@、向套接字添加=, 对文件名进行标记。

`-g` 与 `-l` 相似，但是会忽略属主的名字（显示组）。

`-i` 列出各个文件的信息节点。

`-l` Long 格式的列表（包括权限、属主、大小、修改时间等）。

`-L` 列出符号链接引用的文件或目录而不是链接本身。

`-m` 将列表合并到使用逗号分隔的一系列名称中。

`-n` 作用与 `-l` 相似，但是使用用户 ID 和组 ID 号来代替属主和用户组的名字。

`-o` 作用与 `-l` 相似，但是会省略组的名字（显示属主）。

`-p` 通过向目录添加“ `/` ”来标记目录。

`-q` 将非打印字符显示为`?`。

`-r` 按照相反的顺序列出所有文件（按照文件名或按照时间）。

`-R` 递归列出当前目录及子目录。

`-s` 以块为单位输出文件的大小。

`-t` 按照修改时间列出所有文件（首先显示最新的文件）。

`-u` 按照文件访问时间列出所有文件。

`-x` 通过屏幕的逐行滚动来列出所有文件。

`-1` 输出中的每一行输出一个项。

示例：

列出当前目录中的所有文件及其大小。使用多个列并对特殊文件进行标注：

```
ls -asCF
```

ls

列出目录 /bin 和 /etc 的状态：

```
ls -ld /bin /etc
```

列出当前目录下的所有 C 源文件，首先显示最旧的文件：

```
ls -rt *.c
```

计算当前目录中文件的数量：

```
ls | wc -l
```

m4

/usr/ccs/bin/m4 [*options*] [*files*]

用于 RATFOR、C 和其他程序文件 *files* 的宏处理器。

options

-B*n* 设置后推和参数集合的缓冲区为 *n*（默认是 4096）。

-D*name*[=*value*]
定义 *name* 的值为 *value*；如果没有指定 *value*，则 *name* 的值为 null。

-e 交互式操作，忽略中断。

-H*n* 设置符号表散列数组的大小为 *n*（默认是 199）。

-s 使 C 预处理程序能够进行行同步输出。

-S*n* 设置调用的堆栈大小为 *n*（默认是 100 个存储槽）。

-T*n* 设置符号缓冲区的大小为 *n*（默认是 512 个字节）。

-U*name*
不定义名字。

mail

mail [*options*] [*users*]

读取邮件（如果没有列出 *users*），或向其他 *users* 发送邮件。键入 ? 可显示命令摘要。系统管理员可以使用更多的调试选项（没有列出）。参见 **mailx** 和 **vacation**。

用于发送邮件的选项

-m *type* 在信函的标题中打印带有“ Message-type:（消息类型）”的一行，后面则显示消息的类型 *type*。

mail

`-t` 在信函的标题中打印带有“To:(发往)”的一个行，显示收信人的名字。

`-w` 强行将邮件发送给远程用户而不等待远程转换程序来完成。

读取邮件的选项

`-e` 只测试邮件是否存在，而不会将邮件打印出来。如果邮件存在，则退出状态为 0；否则为 1。

`-f` *file*
从预备的邮箱 *file* 中读取邮件。

`-F` *names*
将所有接收的邮件转发给收件人 *names*。只用于 SVR4（参见附录二中的 **vacation**）。

`-h` 显示一个消息窗口而不是显示最后的消息。

`-p` 输出所有的消息而没有任何停顿

`-P` 输出消息，显示所有的标题行。

`-q` 遇到一个中断时结束。

`-r` 首先显示最旧的消息。

mailx

`mailx [`*options*`] [`*users*`]`

读取邮件，或者向其他 *users* 发送邮件。在命令模式（比如，读取邮件时）中键入`?`或在输入模式（比如，发送邮件时）中输入`~?`，可以显示命令摘要。在设置显示变量及定义别名列表时，用户主目录中的启动文件`.mailrc`非常有用。

在 Solaris 中，`/usr/ucb/mail`和`/usr/ucb/Mail`是到`mailx`的符号链接。

options

`-B` 不对标准输入或标准输出进行缓存。只用于 Solaris。

`-b` *bcc*
向 *bcc* 发送信件的副本。如果有多个收件人，则用引号将列表引起来。只用于 Solaris。

mailx

`-c` *cc*
向*cc*发送副本。如果有多个收件人，则用引号将列表引起来。只用于 Solaris。

`-d` 设置调试。

`-e` 只测试邮件是否存在而不输出它。如果邮件存在，则退出状态为 0；否则为 1。

`-f` [*file*]
从预备文件 *file* 中读取邮件（默认是 `mbox`）。

`-F` 将消息存储到第一个收件人后面指定的文件中。

`-h` *n*
在进行了 *n* 次网络连接后停止发送邮件的尝试，或者“转发（hop）”（用于避免无限的循环）。

`-H` 只输出邮件标题摘要。

`-i` 忽略中断（主要用于调制解调器）；与忽略 `mailx` 选项作用相同。

`-I` 与 `-f` 一起用来显示已经保存的新闻文章，也包括新闻组和文章 ID 标题。

`-n` 不读取系统的启动文件 `mailx.rc` 或 `Mail.rc`。

`-N` 不输出邮件标题摘要。

`-r` *address*
为发送的邮件指定一个返回地址 *address*。

`-s` *sub*
在项目的标题字段放置字符串 *sub*。如果 *sub* 中包含了空白，则必须使用引号将其包括。

`-t` 通过在输入中使用 `To:`、`Cc:` 和 `Bcc:` 来指定收件人，可以代替命令行参数。该选项只用于 Solaris。

`-T` *file*
在文件 *file* 中记录消息 ID 和文章 ID（用于新闻文章）。

`-u` *user*
读取用户 *user* 的邮件。

`-U` 将 `uucp` 类型的地址转换为 Internet 格式。

mailx

`-v` 使用 `-v` 选项来调用 `sendmail`。只用于 Solaris。

`-V` 输出 `mailx` 和 exit 的版本号。

`-~` 处理波浪号转义，即使没有从一个终端上读取信息。只用于 Solaris。

make

`/usr/ccs/bin/make [`*`options`*`] [`*`targets`*`]`

按照当前目录下的一个描述文件中有依赖关系的指令来更新一个或多个 *targets*。默认情况下，该文件的名字为 `makefile` 或 `Makefile`。有关 `make` 的更多信息，可以参见第二十章，也可以见参考文献中列出的《Managing Projects with make》。

注意：Solaris 中的 make 比现在讲述的标准 SVR4 中的 make 具有更多的扩展功能，更多的信息可参考 *make*(1)。

options

`-e` 使用环境变量来覆盖 *makefile* 的分配。

`-f` *`makefile`*
将 *makefile* 当作描述文件，文件名－代表标准输入。

`-i` 忽略命令中的错误代码（与 `.IGNORE` 作用相同）。

`-k` 当该命令失败后，放弃当前记录，但是仍继续保持不相关记录的功能。

`-n` 只输出命令而不执行命令（主要用于测试）。

`-p` 输出宏定义和目标描述。

`-q` 进行查询，如果是最新的则返回 0；否则返回一个非 0 值。

`-r` 不使用“默认”规则。

`-s` 不显示命令行（与 `.SILENT` 作用相同）。

`-t` 接触目标文件，并使这些文件得到更新。

man

`man [`*`options`*`] [[`*`section`*`] `*`subjects`*`]`

显示联机参考页中的帮助信息，各个 *subject* 通常是联机参考页中第一节的一个命令的名字，除非指定可以在1~8之间进行选择的*section* 值。如果没有指定一个 *subject*，则必须提供一个关键字（用于 `-k` 选

man

项）或一个文件（用于 -f 选项）。除了 -M 以外，其他任何选项都不能与 -k 和 -f 一起使用。man 在 MANPATH 环境变量定义的目录中查找信息（默认是 /usr/share/man）。PAGER定义输出是如何发送到屏幕上的（默认是 more -s）。注意：在 Solaris 中，*section* 前面必须添加一个 -s。

options

- \- 通过 cat 而不是 more -s 来进行管道输出。
- -a 显示匹配 *subject* 的所有页面，只用于 Solaris。
- -d 进行调试。只评估 man 命令的运行结果，但是不执行该命令。只用于 Solaris。
- -f *files* 将一个或多个 *files* 的摘要显示在一行。与 whatis 作用相同。
- -F 搜索 MANPATH 目录而不是 windex 数据库。只用于 Solaris。
- -k *keywords* 显示包含指定关键字 *keywords* 的任何标题行。与 apropos 作用相同。
- -l 与 -a 选项相似，但是只列出页面。只用于 Solaris。
- -M *path* 在目录 *path* 中而不是在默认目录中搜索联机描述。-M 会覆盖 MANPATH。
- -r 重新设定格式但是不显示参考页。与 man - -t 相同。只用于 Solaris。
- -s *section* 指定对参考页的 *section* 部分进行搜索。Solaris 中需要其他内容而不是一条命令。
- -t 使用 troff 格式化参考页。
- -T *mac* 使用宏包 *mac* 而不是 tmac.an 来显示信息（*man* 宏）。

示例

在 mv 命令中保存文档（删除退格）：

man

```
man mv | col -b > mv.txt
```

显示与链接和编译相关的命令：

```
man -k link compile | more
```

显示所有 intro 文件的一个摘要：

```
man -f intro
```

从 Section 3M（数学库）中搜索 intro 页面：

```
man 3m intro          （用于 SVR4）
man -s 3m intro       （用于 Solaris）
```

mcs

/usr/ccs/bin/mcs [*options*] *files*

操作注释部分。mcs 添加、压缩、删除或输出一个或多个 ELF 目标文件的片段。默认的片段是 .comment。如果输入文件是一个档案，则 mcs 会作用到各个组件文件上，并且删除档案符号表（除非 -p 是指定的惟一选项）。使用 ar s 命令可以重新产生符号表。使用 mcs -d 命令可以大大减小大的执行文件的大小，通常会节约大量的磁盘空间。该命令至少需要一个选项。

options

-a *string*
: 将字符串 *string* 追加到文件 *files* 的注释部分。

-c 压缩文件 *files* 的注释部分并删除重复的内容。

-d 删除注释部分（包括标题）。

-n *name*
: 直接作用于部分 *name* 而不是 .comment 上。

-p 在标准输出上输出注释部分。

-V 在标准错误上输出 mcs 命令的版本号。

示例

```
mcs -p kernel.o          输出 kernel.o 中的注释部分
```

mesg

mesg [*options*]

改变其他用户使用 talk 的能力，或者将写入的消息发送到终端上。不使用任何参数时，会显示权限状态。

options

-n 禁止写消息。

-y 允许写消息（默认情况）。

这两个选项可以不使用前导符 -，以便与 BSD 兼容。

mkdir

mkdir [*options*] *directories*

建立一个或多个目录*directories*。为了建立一个目录，你必须在父目录中具有写的权限。参见 **rmdir**。

options

-m *mode*

设置新目录的访问模式 *mode*。

-p 如果父目录不存在，则建立它。

示例

建立一个只能读 / 执行的目录 personal：

```
mkdir -m 555 personal
```

下面是未完成的序列：

```
mkdir work; cd work
mkdir junk; cd junk
mkdir questions; cd ../..
```

通过输入下面的内容可以完成该序列：

```
mkdir -p work/junk/questions
```

mkmsgs

mkmsgs [*options*] *string_file msg_file*

将 *string_file*（一个文本字符串的列表）转换为 *msg_file*（可以通过 gettxt 读取该文件格式）。exstr 和 srchtxt 命令可以使用已经建立的 *msg_file*。

mkmsgs

options

`-i` *locale*

在下面的目录中建立文件 *msg_file*：`/usr/lib/locale/locale/LC_MESSAGES`。例如，如果*string_file*包含了德语的错误消息，则可以指定场所 *locale* 为 `german`：

`-o` 覆盖存在的 *msg_file*。

more

`more [`*options*`] [`*files*`]`

在一个终端中显示指定的文件*files*，每次显示一屏。在显示了每屏的信息后，按回车键来显示下一行，或按空格键显示下一屏。输入 `h` 可以获得附加命令的帮助；输入 `q` 可以退出；输入 `/` 可以进行搜索；输入 `:n` 可以转到下一个文件。可以使用参考页对 `more` 进行调用。

options

`-c` 通过清屏而不是滚屏来按页显示文件的内容。这样显示速度快，也容易阅读。

`-d` 显示提示信息“ press space to continue,'q'to quit（按空格键继续，输入 `q` 退出）”。

`-f` 计算逻辑行而不是屏幕上显示的行。当一些比较长的行因为超过了屏幕而折返到下一行时，该选项很有用。

`-l` 忽略进纸字符（`^L`）。

`-r` 强制使用 `^x` 形式显示控制字符。

`-s` 进行挤压，遇到多个空行时只显示一个空行。

`-u` 禁止显示下划线字符和退格字符（`^H`）。

`-w` 在退出时等待一个用户按键。

`-n` 每个“ 窗口 ”只显示 *n* 行（默认是满屏）。

`+`*num*

从行号为 *num* 的行开始显示。

`+/`*pattern*

从 *pattern* 前面两行开始显示。

more

示例

使用“清屏”模式按页显示文件 *file*，并显示提示信息：

```
more -cd file
```

在屏幕上对 doc 进行格式化，删除下划线：

```
nroff doc | more -u
```

查看 grep 命令的参考页，从“BUGS”旁边开始，并压缩多余的空白：

```
man grep | more +/BUGS -s
```

msgfmt

`msgfmt [options] pofiles`

只用于 Solaris。msgfmt 可以将“可移植的目标文件”（*file*.po）转换成可装载的消息文件，一个正在运行的应用程序可以通过 *gettext*(3C)和 *dgettext*(3C)库函数来使用这些文件。

可以使用 xgettext 从原始C源代码文件建立可移植的目标文件。一个翻译器接下来会编辑 .po 文件，提供源程序中各个字符串（或“消息”）的转换。其格式在 *msgfmt*(1)参考页中进行了描述。

一旦通过 msgfmt 进行了编译，则当设置了合适的场所后，运行程序将把这种转换用于其输出。

options

-o *file*
: 将输出放到文件 *file* 中，该选项会忽略 domain 指令和重复的 msgids。

-v
: 详细信息，重复列出消息标识符，但是并没有重新定义消息字符串。

mv

`mv [options] sources target`

用于在系统中移动文件及目录，或对它们重命名。mv 的工作机制如下表所示：

mv

源	目标	结果
文件	*name*	重新将文件的名字修改为 *name*
文件	已经存在的文件	使用源文件来覆盖已经存在的文件
目录	*name*	将目录的名字修改为 *name*
目录	已经存在的目录	将目录移动为已经存在目录的一个子目录
一个或多个文件	已经存在的目录	将文件移动到目录中

options

`--` 当名字的开头为 - 时，使用该选项。为了与过去的程序兼容，一个简单的 - 也可以起作用。

`-f` 强制进行移动，即使目标 *target* 已经存在；禁止输出受限制访问模式的消息。

`-i` 进行询问。在覆盖一个已经存在的目标以前，提示输入一个 y（yes）作为响应。

native2ascii

`/usr/java/bin/native2ascii [`*`options`*`] [`*`input`* `[`*`output`*`]]`

只用于 Solaris。将使用本地字符编码的文件转换为使用 Latin-1 或 Unicode 编码的文件。默认情况下，`native2ascii` 会从标准输入中读取数据，将数据写到标准输出中。一种替代方式是提供用于 *input* 及 *output* 的文件名来读取或写入指定的文件。

该命令支持大量的编码，更详细的内容可以参见参考页。

options

`-encoding` *encoding*

使用 *encoding* 进行编码转换，默认 *encoding* 是系统属性 `file.encoding` 的值。

`-reverse`

执行相反的操作，将 Latin-1 或 Unicode 编码转换为本地编码。

nawk

nawk [*options*] [*'programm'*] [*files*] [*variable=value*]

awk的新版本，新增加了一些功能。nawk是对操作数据比较有用的一种模式匹配语言。有关nawk的更多信息，可以参见第十一章。

options

-f *file*
从文件*file*读取程序指令而不是在命令行提供*program*指令。该选项可以使用多次，多个*file*可以进行连接来组成程序的源代码。

-F *regexp*
使用正则表达式*regexp*来隔离多个字段。

-v *variable=value*
在执行*'program'*以前，将*value*赋值给变量*variable*。

variable=value
将*value*赋值给变量*variable*。当与多个文件进行混合时，赋值将出现在进行处理的那一点上。

neqn

neqn [*options*] [*files*]

与nroff等同的预处理程序，参见第十七章。

nice

nice [*options*] *command* [*arguments*]

使用较低的权限来执行一条命令*command*或参数*arguments*（也就是说，对于其他用户是“nice”）。C shell中也内置了该命令，但是使用不同的命令语法（参见第五章）。

options

-*n* 使用最合适的*n*（1~19）来运行命令*command*，默认值是10。*n*值越高，则优先权越低。特权用户可以通过指定一个负数*n*（比如—5）来提高优先权。C shell中的nice的工作方式不同（参见第五章）。+*n*会提高优先权，-*n*会降低优先权，默认值是4。

-n *n*
与-*n*作用相同，只用于Solaris。

nl

nl [*options*] [*file*]

在逻辑页面部分中对文件 *file* 的行进行编号。在各个逻辑页面的开头，行号重新设置为1。每个页面都包含一个标题、正文和一个页脚，段落可以是空的。从正文内容开始计算行号。使用下面将描述的特殊的孤立行区分段落。定界符行作为空行拷贝到输出中。

段落定界符

`\:\:\:` 标题的开头

`\:\:` 正文的开头

`\:` 页脚的开头

options

`-b`*type*

根据类型 *type* 对行进行编号，可以是下列值：

`a` 所有的行。

`n` 没有任何行。

`t` 只有文本行（默认值）。

`p"exp"` 只有与正则表示式 *exp* 匹配的行。

`-d`*xy*

使用字符 *xy* 来分隔逻辑页（默认是 `\:`）。

`-f`*type*

与 `-b` 作用相似，但是对页脚进行编号（类型 *type* 默认是 `n`）。

`-h`*type*

与 `-b` 作用相似，但是对标题进行编号（类型 *type* 默认是 `n`）。

`-i`*n* 行号之间的增量是 *n*（默认是 1）。

`-l`*n* 将 *n* 个连续的空行当作一个行。

`-n`*format*

设置行号的格式，可以是如下值：

`ln` 所有行靠左对齐，省略前面的 0。

`rn` 所有行靠右对齐，省略前面的 0（默认值）。

`rz` 所有行靠右对齐。

nl

-p 在开始一个新页时不重新设置行号。

-s*c* 使用字符 *c* 将正文与行号分离（默认是一个制表符）。

-v*n* 各个页的号码从 *n* 开始计算（默认是 1）。

-w*n* 使用 *n* 列来显示行号（默认是 6 列）。

示例

列出当前的目录，将文件编号为 1）、2）等等：

```
ls | nl -w3 -s') '
```

对 C 源代码进行编号并进行保存：

```
nl prog.c > print_prog
```

只对开头为 #include 的行进行编号:

```
nl -bp"^#include" prog.c
```

nm

/usr/ccs/bin/nm [*options*] *objfiles*

按照字母顺序输出一个或多个目标文件（通常是ELF或COFF文件）、共享或静态库、二进制可执行程序的符号表（名称列表）。输出包括各个符号的值、类型、大小、名称等等。对符号进行分类的一个编码键也会显示出来。要求至少要提供一个目标文件。

options

-A 在各个行中写出完整的路径名或库名。只用于 Solaris。

-C 输出反改编的 C++ 符号名称。只用于 Solaris。

-D 显示 SHT_DYNSYM 符号信息。只用于 Solaris。

-e 只报表输出外部和静态的符号，已经淘汰。

-f 报表输出所有信息，已经淘汰。

-g 只写出外部（全局）的符号信息。只用于 Solaris。

-h 禁止显示标题。

-l 与 -p 一起使用，通过向编码键后面添加一个星号（*）来表示 WEAK（弱）符号。

-n 按照名称对所有的外部符号进行排序。

nm

-o 以八进制形式报表显示所有的值。

-p 将各个符号的编码键放在该符号的前面（用于分析）。

-r 在各个行中报表输出目标文件的名称。

-R 输出档案文件的名称（如果存在），后面是目标文件和符号的名称。-r 会覆盖该选项。只用于 Solaris。

-s 输出段的名字而不是段的索引。只用于 Solaris。

-t *format*
使用特定的格式 *format* 写出数字值：d 表示十进制；o 表示八进制；x 表示十六进制。只用于 Solaris。

-T 在显示时将符号名称截短。该选项已经淘汰。

-u 只报表输出未定义的符号。

-v 按照值的大小对外部符号进行排序。

-V 在标准错误中输出 nm 的版本号。

-x 以十六进制形式报表输出所有的值。

编码键

A 绝对符号。

B BSS（未初始化的数据空间）。

C 共用符号，只用于 SRV4。

D 数据对象符号。

F 文件符号。

N 无类型符号。

S 节符号。

T 文本符号。

U 未定义符号。

nohup

nohup *command* [*arguments*] &

在注销登录后仍继续执行指定的命令 *command* 和可选的命令参数 *arguments*（即不挂起命令）。在 C shell 中，nohup 是内置命令；在 Bourne shell 中，nohup 允许输出重定向，默认情况下输出会转到

nohup

nohup.out；在Korn shell中，nohup是一个别名，这个别名允许它运行的命令也使用别名（参见第四章和第五章）。

nroff

nroff [*options*] [*files*]

将文档格式化，输出到行式打印机或屏幕。参见第十二章。

od

od [*options*] [*file*] [[+] *offset*[. | b]]

八进制的内存映像，为指定的文件*file*产生一个内存映像（通常是八进制的）。文件*file*从头开始显示，除非指定了一个偏移量*offset*（通常使用八进制字节）。在下面的选项中，一个“字”是一个16位单位。

options

-A *base*
指示如何写出偏移量，前面使用d表示十进制，使用o表示八进制，使用x表示十六进制，n表示没有偏移。只用于Solaris。

-b 以八进制形式显示字节。

-c 以ASCII形式显示字节。

-C 将字节解释为基于LC_CTYPE 设置的字符。只用于Solaris。

-d 按照无符号十进制显示字。

-D 按照无符号十进制显示32位字。

-f 按照浮点数显示32位字。

-F 按照双精度显示64位字。

-j *skip*
从输入开头位置跳过*skip*个字节。在*skip*的前面可以使用一个0或0x来表示八进制或十六进制值；也可以在其后面添加一个b、k或m来表示512、1024或1048576字节的倍数。只用于Solaris。

-N *count*
最多处理*count*个输入字节。只用于Solaris。

-o 按照无符号八进制（默认值）显示字。

-O 按照无符号八进制显示32位字。

od

`-s` 按照有符号十进制显示字。

`-S` 按照有符号十进制显示 32 位字。

`-t` *`type_string`*
指定一个或多个输出类型，参见“类型字符串”部分。只用于 Solaris。

`-v` 显示详细信息，即显示所有数据。重复的行输出为 *。

`-x` 按照十六进制显示字。

`-X` 按照十六进制显示 32 位字。

`+` 如果没有指定文件 *file*，则在偏移前进行询问。

偏移量修饰符

`.` *offset* 值是十进制

`b` *offset* 值是 512 字节的块

类型字符串

类型字符串后面可以放置一个十进制的数字，用来表示有多少字节需要处理。

`a`	ASCII 命名的字符（比如 BEL 表示 `\007`）。
`c`	单个或多个字节的字符。
`d, o, u, x`	分别表示有符号的十进制、无符号的八进制、十进制和十六进制。
`f`	表示浮点数。

page

`page [`*`options`*`] [`*`files`*`]`

与 `more` 作用相同。

passwd

`passwd [`*`options`*`] [`*`user`*`]`

建立或改变与一个用户名关联的密码。只有密码的属主或特权用户可以修改该密码。属主不需要指定他们的用户名。

passwd

options

一般的用户在使用NIS或NIS+时，可以改变所谓的*gecos*信息（用户的全名、办公室等）及登录shell；否则，只有特权用户可以修改下面的内容：

`-D` *domain*
使用*domain*中的`passwd.org_dir`数据库，而不是本地域。只用于Solaris。

`-e` 修改登录shell，只用于Solaris。

`-g` 修改gecos信息，只用于Solaris。

`-r` *db*
在密码数据库*db*中修改密码，该数据库是`files`、`nis`或`nisplus`之一。只有特权用户可以使用`files`。该选项只用于Solaris。

`-s` 显示密码信息:

1. 用户名。
2. 密码状态（NP表示没有密码；PS表示有密码；LK表示进行了锁定）。
3. 密码修改的最后时间（使用*mm/dd/yy*格式）。
4. 在用户*user*重新修改密码以前必须通过的天数。
5. 密码到期以前使用的天数。
6. 在密码到期以前，警告用户*user*密码将要到期的天数。

options （只用于特权用户）

`-a` 与`-s`一起用来显示所有用户的密码信息，不应该提供user。

`-d` 删除密码，不再提示*user*。

`-f` 强制用户*user*的密码到期，*user*在下一次登录时必须修改密码。

`-h` 修改主（登录）目录，只用于Solaris。

`-l` 锁定用户*user*的密码，与`-d`选项是互斥的。

`-n` 设置用户*user*密码信息的第4项，通常与`-x`一起使用。

`-w` 设置用户*user*的第6项。

passwd

`-x` 设置用户 *user* 的第 5 项，使用 −1 来禁止密码到期，0 则像 `-f` 那样强制到期。

paste

`paste [`*`options`*`] `*`files`*

将一个或多个文件 *files* 中的相应行合并到多个垂直列中，使用制表符分隔。参见 **cut**、**join** 和 **pr**。

options

`-` 使用标准输入替换一个文件名。

`-d'`*`char`*`'`
使用 *char* 而不是制表符来隔离多个列，*char* 可以是任何常规字符或下面的转义序列：

`\n` 换行符

`\t` 制表符

`\` 反斜线

`\0` 空字符串

注意：可以通过提供多于一个的 *char* 来使用不同的字符分隔多个列。

`-s` 合并来自同一个文件中后面的行。

示例

使用文件 *x*、*y*、*z* 建立一个三列的文件 *file*：

```
paste x y z > file
```

使用两个列显示多个用户：

```
who | paste - -
```

将成对的行合并成一个行:

```
paste -s -d"\t\n" list
```

patch

`patch [`*`options`*`] [`*`sourcefile`*` [`*`patchfile`*`]]`

只用于 Solaris。patch 会读取包含通常的、`ed` 脚本或上下文格式的 `diff` 输出的一个“补丁”，并将其中包含的修改应用到源文件

patch

*sourcefile*原来的版本中。patch可以修补多个文件，但是必须能够从补丁内容确定原始文件的名字。当改变相对于整个发布版本来说很小的时候，发布补丁是对发布的源文件进行更新的一种简单的方法。

注意：该命令属于Solaris版本，它多少有点像Larry Wall的原始patch程序的旧版本。自由软件基金会现在负责维护patch。他们提供并推荐使用最新的、功能更强的版本，参见*http://www.gnu.org*。

options

`-b` 在*file*`.orig`中建立各个文件的备份，会覆盖已经存在的*file*`.orig`文件。

`-c` 补丁文件*patchfile*是一个上下文diff（来自`diff -c`或`diff -C`）。

`-d` *dir*
在使用补丁以前，将目录修改为*dir*。

`-D` *identifier*
使用C预处理程序`#ifdef`将改变括起来。

```
#ifdef identifier
 ...
#endif
```

`-e` 补丁文件*patchfile*是一个ed脚本（来自`diff -e`）。

`-i` *file*
从文件*file*而不是从标准输入中读取补丁。

`-l` 松散的补丁。补丁中任何空白字符序列可以匹配源文件*sourcefile*中的任何空白序列。其他字符必须精确匹配。

`-n` 补丁文件*patchfile*是一个一般的diff（来自`diff`而没有特殊选项）。

`-N` 忽略已经应用的补丁。通常情况下，会拒绝这种补丁。

`-o` *newfile*
代替在适当位置对各个源文件的更新，将修改文件的所有内容写入到新文件*newfile*中。如果一个文件已经更新多次，则*newfile*将包含各个中间版本的一个副本。

patch

`-p`*N* 从补丁使用的文件名中删除以N开头的路径部分。一个完整路径前面的反斜线“`/`”算作一部分。如果没有该选项，则只会使用文件名中的最后一部分。

`-r` *rejfile*
使用*rejfile*来包含不能应用的补丁，代替*file*`.rej`。被拒绝的补丁总是使用上下文diff格式。

`-R` 颠倒补丁的意义。换句话说，即假定使用`diff` *new old*产生了该补丁，而不是使用`diff` *old new*产生该补丁。

示例

更新一个软件的发布：

```
$  cd whizprog-1.1
$ patch -p1 < whizprog-1.1-1.2.diff
```

这里的许多信息实现与patch相同的功能：

```
$   find . -name '*.orig' -print | xargs rm
$  cd ..
$   mv whizprog-1.1 whizprog-1.2
```

pathchk

`pathchk [-p]` *pathnames*

只用于Solaris，用于检查路径名。该命令会校验*pathnames*指定的文件而不会违反底层文件系统的任何限制(比如一个名字可能太长)，也会校验文件可以访问(比如，如果一个中间目录缺乏搜索权限，则它就是一个问题)。`-p`选项会对*pathnames*提供附加的可移植性检查。

pax

`pax [`*options*`] [`*patterns*`]`

只用于Solaris。可移植的档案交换程序。在POSIX 1003.2工作组成员不能对`tar`或`cpio`进行标准化时，他们发明了该类程序(注1)(参见 **cpio** 和 **tar**)。

`pax`可以在四种模式中操作，这主要依赖于`-r`和`-w`的结合：

列表模式（listmode）
不使用`-r`和`-w`。列出`pax`档案的内容，可以选择将输出限制到文件名及（或）匹配一个给定模式的目录。

注1： 通常将Unix历史上的这一时期称为“tar战争”。

pax

提取模式（extract mode）

只使用 -r 选项。从一个 pax 档案中提取文件。根据需要可以建立中间目录。

存档模式（archive mode）

只使用 -w 选项。将文件归档到一个新的或已经存在的 pax 档案中。该档案会写入到标准输出中。如果需要进行备份，则可以将其重新定向到一个合适的磁带设备上。

通过模式（pass-through mode）

使用 -r 和 -w。模拟 cpio -p 将一个目录树从一个位置拷贝到另一个位置。

options

下面是四个模式可用的选项：

```
None:   c d f     n     s     v
-r:     c d f i k n o p s   u v
-w: a b   d f i     o   s t u v x X
-rw:      d i k l n   p s t u v   X
```

-a 将文件添加到档案中，该选项在一些磁带设备中可能无法使用。

-b *size*

将块大小设置为 *size*，使用字节表示，这些块将会写入到档案中。

-c 一种补充，匹配所有与模式不匹配的文件或档案成员。

-d 对于作为目录的文件或档案成员，只对目录本身进行提取或存档，而不是该目录包含的目录树。

-f *archive*

使用 *archive* 来代替标准输入或标准输出。

-i 交互地重新命名文件。对于各个文件，pax 会将一个提示写入到 /dev/tty 中，并且从 /dev/tty 中读取一行的响应。响应如下：

Return	跳过文件。
A period	取走文件。

pax

new name 除了文件使用的新名字以外，取走任何内容。

EOF 使用一个非零的退出状态立即退出。

`-k` 不会覆盖已经存在的文件。

`-l` 建立硬链接。当拷贝一个目录树（`-rw`）时，只要可能，就在源和目标层之间建立硬链接。

`-n` 选择匹配各个模式的第一个档案成员。匹配各个模式的档案成员不能超过一个。

`-o` *options*

为特定于格式的选项保留（显然没有在 Solaris 中使用）。

`-p` *privs*

为提取的文件指定一个或多个特权。*privs* 指定权限或其他将要保留或忽略的特点。

`a` 不保留文件的访问时间。

`e` 保持用户及组 ID、权限（模式）、访问及修改时间。

`m` 不保留文件的修改时间。

`o` 保持用户及组 ID。

`p` 保持权限（模式）。

`-r` 读取档案及释放文件。

`-s` *replacement*

使用 *replacement* 来修改文件或档案成员的名字，这是一个形式为 `-s`*/old/new/*`[gp]` 的字符串，类似于 ed、ex 和 sed 中的替换命令。*old* 是一个正则表达式，*new* 可以包含表示匹配的文本的 `&` 和用于子模式的 `\`*n*。后缀 `g` 表示进行全局替换；后缀 `p` 会使 pax 输出新产生的文件名。可以提供多个 `-s` 选项，但只会应用起作用的第一个选项。可以使用任何分隔符，但是不能使用 `/`。最好使用引号将参数引起来以防止shell将其当作通配符。

`-t` 在使用 pax 对档案文件进行存档以前，重置存档文件的访问时间。

`-u` 忽略比先前存在的文件或档案成员更旧的文件。该行为根据当前的模式的不同而不同。

pax

提取模式

如果档案文件是已经存在的一个同名文件更新，则提取该文件。

存档模式

如果一个已经存在的与档案成员同名的文件比档案成员更新，则使用该文件代替档案成员。

通过模式

如果源层次（或到它的链接）上的文件比目标层次上的文件更新，则使用源层次上的文件替换目标层次上的文件。

`-v` 在列表模式中，输出详细的内容表；否则，在标准错误中输出档案成员名称。

`-w` 使用给出的档案格式将多个文件写到标准输出中。

`-x` *format*

使用档案给出的格式*format*。*format*的值是`cpio`或`ustar`。两种格式的细节内容可以在IEEE 1003.1（1990） POSIX标准中找到。这两种格式是相互不兼容的；试图将使用一种格式的档案添加到使用另一种格式的档案中的做法是错误的。

`-X` 当遍历目录树时，不跨越进入不同设备上的目录（`stat`结构中的`st_dev`字段，参见*stat*(2)，类似`find`命令中的`-mount`选项）。

示例

将当前目录拷贝到磁带上：

```
pax -x ustar -w -f /dev/rmt/0m .
```

将一个主目录拷贝到另一个不同的目录中（假设在一个较大的磁盘上）：

```
#  cd /home
#   pax -r -w arnold /newhome
```

perl

`perl [`*`options`*`] [`*`programfile`*`] [`*`files`*`]`

`perl`是Perl编程语言的解释器（Unix编程工具的“瑞士军刀”）。可使用一个以上的`-e`选项提供Perl程序，如果没有使用`-e`选项，则在命令行指定的第一个文件将用于该程序。

perl

要了解Perl语言的更多知识，可以学习参考文献中列出的《Learning Perl》（本书中文版《Perl 语言入门》已由中国电力出版社出版 —— 编注）、《Programming Perl》（本书中文版《Perl 语言编程》已由中国电力出版社出版 —— 编注）及《Advanced Perl Programming》（本书中文版《高级 Perl 编程》已由中国电力出版社出版 —— 编注）等书籍。

注意：`perl`没有随SVR4或Solaris一起发布，`perl`被广泛用于开发Web、CGI、系统管理任务及其他方面。要开始学习使用Perl编程，可以访问如下站点：*http://www.perl.com*。

options

这里列出的选项用于`perl`版本5.005，补丁程序级别为2。更详细的信息参见 *perlrun*(1)。

`-a` 当与 `-n` 或 `-p` 一起使用时，会打开自动拆分模式，拆分成 `@F`。

`-c` 只检查语法，但是不执行命令。

`-d` 在调试器中运行脚本。使用 `-de 0` 命令来启动没有脚本的调试器。

`-d:`*`module`*
在作为 `Devel:`*`module`* 安装的模型控制下运行脚本。

`-D`*`flags`*
设置调试标志。*flags* 可以是由多个字母组成的字符串，也可以是这些字母对应数值的总和。参见“ 调试标志 ”。要使这些标志起作用，`perl`必须使用 `-DDEBUGGING` 进行编译。

`-e '`*`commandline`*`'`
可以用于输入单行的脚本。多个 `-e`命令可以建立一个多行的脚本。

`-F` *`regexp`*
如果 `-a`起作用，则指定一个正则表达式进行拆分。

`-h` 输出多个选项的汇总。

perl

-i[*ext*]
适当地对使用<>结构处理的文件进行编辑。旧的副本被重新命名。处理过的副本会写入到原始的文件中。可选项 *ext* 表示重命名副本使用的一个扩展名。可以使用不同的规则进行相同的操作，参见 *perlrun*(1)。

-I*dir*
与 -P选项一起使用，通知C预处理程序到哪里去查找包括文件。目录也会添加到 @INC 之前。

-l[*octnum*]
启用自动终止行的处理，比如 -1013。

-m[-]*module*
与使用 *module*( )作用相同。在 -m后面使用一个 - 时，等同于 no *module*( )。

-m[-]'*module*=*arg*[，*arg*]'， -M[<en>]'*module*=*arg*[，*arg*]'，
-M'module qw(arg ...)' 的简化形式，这可以避免使用引号将内部的参数引起来。

-M[-]'*module* [...]'
等同于use *module*...;。在 -M后面使用一个 - 时，等同于使用 no module...;。“ ... ”表示希望提供的附加代码，例如：

```
-M'mymodule qw(whizfunc wimpfunc)'
```

-n 假设你的脚本中使用了循环输入。不输出输入行（与 sed -n或 awk 相似）。

-p 假设你的脚本中使用了循环输入。不输出输入行（与sed相似）。

-P 在使用 perl 编译以前，在脚本中运行 C 预处理程序。

-s 将命令行中的 -*xxx* 解释为一个开关，并在脚本中设置相应的变量 $xxx。

-S 使用 PATH 环境变量来搜索脚本。

-T 强制进行污点检查。

-u 在编译脚本后转储核心，主要与 *undump*(1)程序一起使用。与 perl 一起提供的 Perl-to-C 编译器可以在很大程度上取代它。

-U 允许 perl 执行不安全的操作。

perl

`-v` 输出 `perl` 执行程序的版本和补丁级别。

`-V` 输出配置信息和 `@INC` 的值。

`-V:`*var*

在标准输出中输出配置变量 *var* 的值。

`-w` 输出脚本中可能出现的拼写错误和其他错误结构等方面的警告。

`-x [`*dir*`]`

从输入流中提取 Perl 程序。如果指定了 *dir*，则在运行该程序以前，`perl` 会切换到该目录下。

`-0`*val*

（即数字 0。）指定记录分隔符 `$/` 的初始值。参见 `-l`。

调试标志

值	字母	调试
1	p	符号化和分析
2	s	堆栈快照
4	l	上下文（循环）堆栈处理
8	t	跟踪执行
16	o	方法和过载分解
32	c	字符串 / 数字的转换
64	P	输出用于 `-P` 的预处理器命令
128	m	内存分配
256	f	格式化处理
512	r	正则表达式分析和执行
1024	x	语法树转储
2048	u	进行污点检查
4096	L	内存泄漏（当编译 `perl` 时需要 `-DLEAKTEST`）
8192	H	散列转储，侵占 `values()`
16384	X	暂存分配
32768	D	清除
65536	S	线程同步

pic

`pic [`*options*`] [`*files*`]`

`nroff/troff` 线图的预处理程序，参见第十七章。

pr

pr [*options*] [*files*]

按照*options*向标准输出中格式化一个或多个文件*files*。每一页包括一个标题，该标题由页码、文件名、日期和时间组成。当直接指定文件时，日期和时间是指文件的修改时间；否则，会使用当前的日期和时间。

options

-a 多列格式，列出各个行包含的项。

-d 双空格格式。

-e[*cn*]
每隔*n*个位置（默认是8）设置一个输入制表符，并使用*c*作为字段的定界符（默认是制表符）。

-f 使用进纸字符（^L）而不是一系列空行来分隔各个页。

-F 折叠输入的行（避免被-a和-m截短）。

-h *str*
使用字符串*str*替换默认的标题。

-i*cn*
在输出中，每隔*n*个位置（默认是8）使用字段定界符*c*（默认是制表符）替换空白。

-l*n* 设置页面的长度为*n*行（默认是66）。

-m 合并多个文件，在每列中输出一个文件（不能与-n和-a一起使用），并对文本进行剪切以便符合要求。参见**paste**。

-n[*cn*]
使用长度为*n*的数字（默认是5）对行进行编号，后面紧跟一个字段定界符*c*（默认是制表符）。参见**nl**。

-o*n* 各行的偏移量为*n*个空格（默认是0）。

-p 在每页前暂停。

-r 禁止输出文件中不能发现的消息。

-s*c* 使用字符*c*（默认是制表符）分隔各个列。

-t 省略页标题和页尾的空行。

pr

`-w`*n* 设置行的宽度为 *n*（默认是 72）。

`+`*num*
从页号为 *num*（默认是 1）的页开始输出。

`-n` 产生具有 *n* 个列（默认是 1）的输出；制表符扩展为与 `-i` 一样。

示例

显示一个并列的列表，省略标题和多余的行。

```
pr -m -t list.1 list.2 list.3
```

按字母顺序排列一个状态列表，每行使用 5 个列：

```
sort states_50 | pr -n -5
```

printenv

`/usr/ucb/printenv [`*variable*`]`

输出所有环境变量或者选择性地输出特定变量 *variable* 的值。SVR4 中可选择的`env`使你不仅可以看到一个变量，而且可以重新定义这些变量。

printf

`printf` *formats* `[`*strings*`]`

使用特定的格式 *formats* 输出字符串 *strings*。*formats* 可以是一般的文本字符、C 语言转义字符、*printf*(3S)格式的转换说明符，或者下面列出的一系列转换参数：

参数

`%b` 处理使用反斜线转义的一个字符串参数（不在*printf(3S)*中）。参见 **echo** 中允许使用的转义的描述。

`%s` 输出下面的字符串。

`%`*n*`$s`
输出第 *n* 个字符串。

`%[-]`*m*`[.`*n*`]s`
输出下一个字符串，使用的字段有 *m* 个字符宽。可选择限制该字段只输出字符串中最前面的 *n* 个字符。字符串一般是靠右对齐，除非指定了靠左对齐的标志 `-`。

printf

示例

```
$  printf '%s %s\n' "My files are in" $HOME
My files are in /home/arnold
$  printf '%-25.15s %s\n' "My files are in" $HOME
My files are in            /home/arnold
```

prof

/usr/ccs/bin/prof [*options*] [*object_file*]

显示一个目标文件的描述数据。该文件的符号表会与描述文件 mon.out （由通过 cc -p 编译的程序产生）进行比较。可以从 -a、-c、-n 或 -t 等排序选项中选择一个。参见 **gprof** 和 **lprof**。

options

-a 按照符号地址列出输出。

-c 按照调用的递减号码列出输出。

-C 反改编的 C++ 符号的名字。只用于 Solaris。

-g 包括非全局（静态）的函数符号（与 -l 一起使用时无效）。

-h 禁止输出报表的标题。

-l 不包括非全局的函数符号（默认值）。与 -g 一起使用时无效。

-mpf 使用 *pf* 而不是 mon.out 作为输入的描述文件。

-n 按照符号名列表输出。

-o 以八进制形式显示地址（与 -x 一起使用时无效）。

-s 在标准错误中输出一个汇总。

-t 按照总时间的百分比的递减顺序（默认值）进行列表显示。

-V 在标准错误中输出版本信息。

-x 以十六进制形式输出地址（与 -o 一起使用时无效）。

-z 包括零次使用的调用。

prs

/usr/ccs/bin/prs [*options*] *files*

一条 SCCS 命令。参见第十八章。

prt

/usr/ccs/bin/prt [*options*] *files*

只用于 Solaris。一条 SCCS 命令。参见第十八章。

ps

ps [*options*]

报告活动进程。选项 *options* 中列出的参数要么使用逗号间隔，要么使用双引号包括起来。在对产生的输出数量进行比较时，应注意 `-e > -d > -a` 和 `-l > -f`。在 BSD 版本中，*options* 的作用大不相同；可以只显示一个进程的数据。

options

`-a` 除了组引导进程和没有与一个终端建立联系的进程外，会列出所有其他进程。

`-A` 与 `-e` 作用相同，只用于 Solaris。

`-c` 列出由 `priocntl`（一条管理命令）设置的调度程序数据。

`-d` 列出除了会话引导进程以外的所有进程。

`-e` 列出所有进程。

`-f` 产生一个完全列表。

`-g`*list*
只列出组引导进程 ID 号的指定 *list*（也就是说，具有相同 ID 和组 ID 的进程）的数据。

`-G` *list*
显示其真正的组 ID 可以在 *list* 中找到的进程信息，只用于 Solaris。

`-j` 输出进程组 ID 和会话 ID。

`-l` 产生一个长的列表。

`-n`*file*
使用选择的文件 *file* 作为正在运行的内核（默认是 `/unix`）中函数名称的列表。在 SVR4 中已经淘汰。

`-o` *format*
按照格式 *format* 定制信息。很少使用。只用于 Solaris。

`-p`*list*
只列出在 *list* 中的进程 ID 的数据。

`-s`*list*
只列出在 *list* 中的会话引导进程 ID 的数据。

ps

`-t` *list*
只列出在 *list* 中的终端（比如 `tty1`）的数据。

`-u` *list*
只列出在 *list* 中的用户名的数据。

`-U` *uidlist*
显示实际用户 ID 在 *list* 中的进程的信息。只用于 Solaris。

`-y` 与 `-l` 一起使用，省略 F 和 ADDR 列及使用千字节代替 RSS 和 SZ 列的页面。只用于 Solaris。

pwd

`pwd`

输出当前目录完整的路径名（命令名称代表“输出工作目录”）。注意：内置版本 `pwd`（Bourne 和 Korn shell）和 `dirs`（C shell）速度更快，所以你可能希望定义下面的 C shell 别名：

```
alias pwd dirs -l
```

rcp

`rcp [`*options*`]` *sources target*

在计算机之间拷贝文件，源 *sources* 和目标 *target* 都是使用 *host:pathname* 形式指定的文件名，如果一个文件是在本地计算机上，则可以省略 *host:*。如果 *target* 没有包括路径，则将源文件存放到主目录中。如果你在远程主机上有一个不同的用户名，则使用 *username@hostname:file* 的形式。参见 **ssh**。

options

`-p` 将文件的修改时间、访问时间及源文件的模式保存在文件的副本中。

`-r` 如果 *target* 和 *sources* 都是目录，则同时拷贝 *source* 下的每个子树。一定要记住，符号和硬链接会当作实际文件进行拷贝；不会保存原始树的链接结构。

示例

将本地文件 `junk` 和 `test` 拷贝到计算机 `hermes` 上你的主目录中：

```
rcp junk test hermes:
```

rcp

将本地的bin目录及所有子目录拷贝到计算机diana的/usr/tools目录下：

```
rcp -r /bin diana:/usr/tools
```

将你主目录下的所有文件拷贝到计算机hera上，并将它们保存在本地目录/home/daniel中，保存时不修改时间及模式：

```
rcp -p "hera:*" /home/daniel
```

应使用引号将第一个参数包括起来，以避免文件名扩展出现在本地计算机上。

regcmp

/usr/ccs/bin/regcmp [-] *files*

表示“正则表达式编译”。编译*files*指定的一个或多个文件中的正则表达式并将输出保存到*file*.i（如果指定了-，则保存到*file*.c）中。输出是C源代码，但是*files*中的输入项使用如下形式：

C variable "*regular expression*"

该程序的目的是预编译正则表达式，以便与*regex*(3C)库例程一起使用，以避免*regcmp*(3C)函数的额外开销。

refer

refer [*options*] *files*

troff的参考文献引用预处理程序。参见第十七章。

reset

/usr/ucb/reset [*options*] [*type*]

清除终端的设置，reset会禁止CBREAK模式、RAW模式、输出延迟及奇偶校验。reset也会恢复未定义的特殊字符并启用换行符、制表符和回显的处理。当一个程序通过使终端混乱的方法而进行中断（即键盘的输入没有回显在屏幕上）时，使用该命令特别有用。要在键盘上输入reset，可能需要使用换行符（^J）来代替回车符。reset使用和tset相同的命令行参数。

rksh

rksh [*options*] [*arguments*]

ksh的受限制版本（Korn shell），用于安全的环境中。rksh可以防止你在目录外进行修改或重定向输出。参见第四章。

rlogin

`rlogin [`*`options`*`] `*`host`*

将当前使用的本地主机系统(即登录的系统)上的终端连接到一个远程的主机系统。主目录下的 `.rhosts` 文件（在远程主机上）列出了允许你不使用密码进行连接的主机名（也可选择那些主机上的用户名)。参见 **ssh**。

options

`-8` 允许传递 8 位数据，而不是 7 位数据。

`-e` *`c`*
使用转义字符 *c*（默认是 `~`)。你可以键入 `~.` 来断开到远程主机的连接，尽管通过注销会使退出更“干净”。

`-E` 没有任何转义字符。只用于 Solaris。

`-l` *`user`*
使用用户名 *user* 登录到远程主机，而不使用本地主机上的名字。

`-L` 允许在 LITOUT 模式中运行 `rlogin`（8 位数据可能只传递到输出中)。

rm

`rm [`*`options`*`] `*`files`*

删除 *files* 指定的一个或多个文件。要删除一个文件，你必须在包含该文件的目录中有“写”的权限，但是你不需要有该文件本身的权限。如果你在该文件上没有“写”的权限，则会提示你（`y` 或 `n`）进行覆盖。

options

`-f` 强制执行，删除有写保护的文件而不进行提示。

`-i` 提示输入 `y`（删除文件）或 `n`（不删除文件)。覆盖 `-f` 选项。

`-r` 如果 *file* 是一个目录，则会删除整个的目录和该目录下所有的内容，包括子目录。注意，使用该选项是比较危险的。

`-R` 与 `-r` 选项作用相同，只用于 Solaris。

`--` 标志选项的结束(`rm` 仍接受旧形式的 `-`)。当提供的文件名使用 `-` 开始时，使用该选项。

rmdel

`/usr/ccs/bin/rmdel -`*`rsid files`*

一条 SCCS 命令。参见第十八章。

rmdir

`rmdir [`*`options`*`]` *`directories`*

删除指定的目录*directories*(目录本身而不是目录的内容)。*directories*从其父目录中删除，并且必须为空（如果目录不为空，可以使用`rm -r`代替该命令)。参见 **mkdir**。

options

`-p` 删除目录 *directories* 和所有因为删除这些目录而变空的相关父目录。主要用于删除子目录树。

`-s` 禁止显示 `-p` 导致的标准错误信息。

rmic

`/usr/java/bin/rmic [`*`options`*`]` *`classes`*

只用于 Solaris。用于 Java 的远程方法调用编译器。`rmic` 使用完全限定包类名并产生框架和存根类文件来提供远程方法调用。这个类必须使用 `java` 成功地进行了编译。

对于`whiz`类中的方法`WhizImpl`, `rmic`会建立两个文件, `WhizImpl_Skel.class`和`WhizImpl_Stub.class`。该“描述”文件实现RMI的服务器端，而存根文件实现客户端。

options

`-classpath` *`path`*
使用*path*作为类文件的搜索路径，覆盖$CLASSPATH。*path*是一个使用冒号间隔的目录列表。

`-d` *`dir`*
将产生的文件放在 *dir* 目录中。

`-depend`
重新编译由其他类文件而不只是源代码引用的丢失或过期的类文件。

`-g` 产生带有行号的调试表。使用 `-O`时，也会产生有关本地变量的信息。

rmic	`-keepgenerated`
	为描述和存根保留生成的 `.java` 源文件。
	`-nowarn`
	禁止所有的警告。
	`-O` 执行优化操作，有可能产生运行快速但是比较大的类文件。它也可能降低编译的速度，使用该选项时应该进行判断。
	`-show`
	使用 RMI 编译器的 GUI 来输入类名。
	`-verbose`
	当编译及装载文件时显示信息。

rmiregistry	`/usr/java/bin/rmiregistry [`*`port`*`]` 只用于Solaris。在指定的端口*port*建立并启动一个远程的对象注册。*port*默认是1099。注册为RMI（远程方法调用）服务器和客户提供指定服务。
roffbib	`roffbib [`*`options`*`] [`*`files`*`]` 程序 `refer` 套件的一部分。参见第十七章。
rsh	`/usr/lib/rsh` `sh`（Bourne shell）的限制版本，主要用于对安全要求比较高的情况。`rsh`可以防止你在目录以外进行修改或重定向输出。参见第四章。
rsh	`rsh [`*`options`*`] `*`host`*` [`*`command`*`]` 调用一个远程shell的来自BSD的命令。该命令可以在`/usr/ucb`目录中找到，并且不要与受限制shell `rsh`混淆。在Solaris中，该命令保存在`/usr/bin`目录中。`rsh`会连接到主机并运行命令*command*。如果没有指定*command*参数，则`rsh`允许你使用`rlogin`命令登录到主机*host*。如果shell元字符需要在远程计算机上进行解释，则将它们包括在引号中。该命令有时称为`remsh`。参见**ssh**。

rsh	options `-l` *user* 使用 *user* 的登录用户名连接到主机 *host*。 `-n` 将输入转移到 `/dev/null` 中。当将 `rsh` 通过管道输出到读取标准输入但是在使用 `rsh` 以前终止的一条命令时，该选项比较有用。
sact	`/usr/ccs/bin/sact` *files* 一条 SCCS 命令。参见第十八章。
sccs	`/usr/ccs/bin/sccs [`*options*`] `*command*` [`*SCCS_options*`] [`*files*`]` 对于 SCCS 的一个用户友好界面。参见第十八章。
sccsdiff	`/usr/ccs/bin/sccsdiff` *-rsid1 -rsid2* `[`*options*`]` *files* 一条 SCCS 命令，参见第十八章。
script	`script [`*option*`] [`*file*`]` 为你的登录会话建立一条记录，将显示在屏幕上的所有内容都保存在文件 *file* 中。默认文件名为 `typescript`。`script` 会将非打印字符记录为控制字符和包括提示符。该命令对于新手比较有用，或者用于保存来自一条消耗时间的命令的输出。 option `-a` 将 `script`（脚本）记录添加到文件 *file* 中。
sdiff	`sdiff [`*options*`]` *file1 file2* 对文件 *file1* 和 *file2* 中的对应行进行比较。输出如下： *text text* 相等的行。 *text* `<` 只在文件 *file1* 中存在的行。

sdiff

`> text`
只在文件 *file2* 中存在的行。

text `|` *text*
两个文件中不同的行。

options

`-l` 只列出文件 *file1* 中相等的行。

`-o` *outfile*
将文件 *file1* 和 *file2* 中相等的行发送到 *outfile* 中。输出行的区别，并编辑 *outfile*，当进行提示时，输入下面的命令：

`e` 编辑一个空文件。

`e` *b* 编辑左边和右边的列。

`e` *l* 编辑左边的列。

`e` *r* 编辑右边的列。

`l` 将左边的列添加到 *outfile* 中。

`q` 退出编辑器。

`r` 将右边的列添加到 *outfile* 中。

`s` 安静模式，不输出相等的行。

`v` 关闭“安静模式”。

`-s` 不输出相等的行。

`-wn` 设置行的长度为 *n*（默认是 130）。

示例

用 80 列显示区别，并忽略相等的行：

```
sdiff -s -w80 list.1 list.2
```

sed

sed [*options*] [*files*]

流编辑器。编辑*files*指定的一个或多个文件而不与用户进行交互。更多的信息可以参见第十章。`-e`和`-f`选项可以使用多次，并且两者可以结合使用。

sed

options

`-e 'instruction'`
将编辑指令 *instruction* 应用到文件 files 中。

`-f script`
应用来自编辑脚本 *script* 的指令设置。

`-n` 禁止默认输出。

serialver

`/usr/java/bin/serialver [-show | classname]`

只用于 Solaris。使用适合拷贝到一个衍生类中的形式输出用于 *classname* 的 `serialVersionUID`。`-show` 选项可以启动一个简单的 GUI，在该 GUI 中你可以输入完整的类名。

sh

`sh [options] [arguments]`

执行来自一个终端或一个文件的命令的标准命令解释器（或 Bourne shell）。有关 Bourne shell 及命令行选项的更多信息，可以参见第四章。

size

`/usr/ccs/bin/size [options] [objfile ...]`

输出 *objfile* 中各节的字节数量（使用十进制表示）。在很多系统中，如果没有指定 *objfile*，则使用 `a.out`。Solaris 需要目标文件 *objfile* 的名字。

options

`-f` 输出可分配节的大小、名称及总的大小。

`-F` 输出装载节的大小、权限标志及总的大小。

`-n` 输出不可分配节或不可装载的节的大小。

`-o` 以八进制形式进行输出。

`-V` 报告程序版本号的大小。

`-x` 以十六进制的形式显示输出。

sleep

sleep *seconds*

在执行另一个命令以前等待 *seconds* 指定的秒数。通常用于 shell 脚本中，sleep 内置于 ksh93 中。

soelim

soelim [*files*]

读取nroff/troff输入文件*files*的一个预处理程序, 解析后删除 .so 请求。输入行如下：

```
.so header
```

它会被文件 header 的内容替换。通常情况下，使用 nroff 或 troff 对 .so 请求进行解析。当需要预处理输入（即通过 tbl 或 sed 发送它）或者在格式化以前需要完整的文本时，使用 soelim 最合适。

示例

在进行格式化以前在（所有）输入上运行一个 sed 脚本：

```
soelim file | sed -e 's/--/\\(em/g' | nroff -mm - | lp
```

sort

sort [*options*] [*files*]

对指定文件 *files* 中的行进行排序，通常按照字母顺序进行排列。参见 **uniq**、**comm** 和 **join**。

options

-b 忽略前导空格和制表符。

-c 检查 *files* 是否已经进行了排序，如果已经排序，则不会产生输出。

-d 按照目录顺序进行排序（忽略标点符号）。

-f 进行“折叠”，忽略大小写的区别。

-i 忽略非打印字符（即 ASCII 值在 040~176 以外）。

-k *fieldspec*
指定排序的输入字段的重要性，参见下面的完整描述。只用于 Solaris。

sort

`-m` 合并已经排序的输入文件。

`-M` 按照月份的前三个字母进行比较。

`-n` 按照算术（数字）顺序进行排序。

`-o` *file*
将输出保存到文件 *file* 中。

`-r` 颠倒排列顺序。

`-t`*c* 使用 *c*（默认是空白区域）分隔所有字段。

`-T` *dir*
将临时文件保存到 *dir* 中，只用于 Solaris。

`-u` 输入文件中相等的行在输出中只显示一次。

`-y[`*kmem*`]`
调整 `sort` 命令使用的内存数量（按照千字节）。如果没有指定 *kmem* 参数，则分配最大的内存。

`-z`*recsz*
为文件中的任意一行提供最大数量的字节数。在特定情况下，该选项可防止 `sort` 命令的异常终止。Solaris 中的 `sort` 命令接受该选项，其他 Unix 系统忽略该选项。

`+`*n* `[-`*m*`]`
在进行排序前跳过 *n* 个字段，并且最多排序到位置为 *m* 的字段。如果没有提供 *m*，则排序到行的结尾。位置使用的形式 *a.b* 表示字段 *a* 中的字符 *b*。如果没有提供 *.b*，则从字段的第一个字符开始排序。从 0 开始计数。Solaris 允许字段中使用可选的结尾修饰符，如 `-k` 选项中所示。

`-k` 中字段的规范

fieldspec 的形式如下：*fieldstart*[*type*][,*fieldend*[*type*]]。

fieldstart
一个字段的编号及形式为 *fnum*[*.schar*]的可选的开始字符。*fnum* 是字段的编号，从 1 开始计算。如果提供了 *schar*，则它表示该字段的开始字符，也从 1 开始计算。

sort

fieldend

一个字段的编号及形式为 *fnum*[*.schar*]的可选的结束字符。*fnum* 是字段的编号，从1开始计算。如果提供了 *echar*，则它表示该字段最后一个有意义的字符，也从 1 开始计算。

type

一个修饰符，可以是 `b`、`d`、`f`、`i`、`M`、`n` 或 `r` 中的一个字符，效果与相应的选项相同，除了`b`修饰符只应用于字段中，而不是整个的行。

示例

按照行号的递减顺序列出文件：

```
wc -l * | sort -rn
```

按照字母顺序排列一个单词列表，删除重复的单词并输出各个单词出现的次数：

```
sort -fd wordlist | uniq -c
```

按照第三个字段（用户 ID）中的数字顺序对密码文件进行排序：

```
sort +2n -t: /etc/passwd
```

查找一个系统中占用磁盘最多的前 20 个文件：

```
cd /home; du -sk * | sort -nr | head -20
```

sortbib

`sortbib [`*`option`*`] `*`files`*

程序 `refer` 套件的一部分，参见第十七章。

sotruss

`sotruss [`*`options`*`] `*`program`*` [`*`args`*` ...]`

只用于 Solaris。`truss` 的共享目标库版本。`sotruss` 执行 *program*，向其传递参数 *args*（如果存在），然后跟踪动态装载的共享目标库向内或（及）向外的调用。

options

`-f` 跟踪 *fork*(2)产生的子项并输出各个子项的输出。各个输出行包含了该进程的进程 ID。

sotruss

`-F` *fromlist*
: 只跟踪来自 *fromlist* 中指定库的调用，*fromlist* 是使用冒号间隔的一个库列表。默认情况是只跟踪来自主执行程序的调用。

`-o` *file*
: 将输出发送到文件 *file* 中。如果与 `-f` 一起使用，则正在执行程序的进程 ID 会添加到文件名中。

`-T` *tolist*
: 只跟踪对 *tolist* 中指定库中的例程的调用，*tolist* 是使用冒号间隔的一个库列表。默认情况是跟踪所有的调用。

spell

`spell [`*options*`] [`*files*`]`

将 *files* 指定的一个或多个文件中的单词与系统字典进行比较，然后报告所有拼写错误的单词。用于 spell 的系统文件保存在 `/usr/lib/spell` 目录中。

options

`-b` 按照英国的拼写习惯进行检查。

`-i` 忽略 `nroff` 或 `troff .so` 请求包括的文件。如果 `deroff` 不可用，则没有任何影响。

`-l` 遵循所有的包括文件（在 `.so` 或 `.nx` 请求中指定的文件）；默认情况是忽略使用 `/usr/lib` 开头的文件。

`-v` 包括来自字典列表但不是字面上的项的单词。

`-x` 显示所有可能的词干（在标准错误上）。

`+`*wordlist*
: 使用一个已经排序的单词列表文件 *wordlist* 作为一个本地字典来添加到系统字典中。*wordlist* 中的单词不会当作拼写错误。

示例

初次运行 `spell`：

```
spell file1 file2 > jargon
```

spell

在编辑jargon文件后，使用它作为特殊术语的一个列表。spell的第二次运行会产生真正的拼写错误：

```
spell +jargon file[12] > typos
```

split

split [*options*] [*infile*] [*outfile*]

将 *infile* 拆分成几个长度相等的文件，而 *infile* 仍保持不变，将拆分的文件写入到*outfile*aa、*outfile*ab 等文件中（默认是xaa、xab等）。如果*infile*是 -（或者没有使用该选项），则会从标准输入中读取数据。参见 **csplit**。

options

-n 将*infile*拆分成多个文件，每个文件的长度都是*n*行（默认是1000行）。

Solaris 选项

下面的选项只能用于 Solaris:

-a *slen*
使用 *slen* 字符作为文件名的后缀，默认是 2。

-b *n*[*m*]
将文件分成大小为 *n* 个字节的多个片。可以提供一个可选的乘数 *m*:k 表示千字节，m 表示兆字节。该选项与 -l 是互斥的。

-l*n* 与 —*n* 作用相同，与 -b 是互斥的。

示例

将 *bigfile* 分成多个长度为 1000 行的段：

```
split bigfile
```

先连接 4 个文件，然后将连接文件分成长度为 10 行的多个文件，这些文件的名字依次为 new.aa、new.ab 等。注意，如果没有 -，则 new. 将被当作不存在的输入文件：

```
cat list[1-4] | split -10 - new.
```

srchtxt

srchtxt [*options*] [*regexp*]

一个与 grep 相似的实用程序，用来查找消息文件中与正则表达式 *regexp* 匹配的文本字符串。srchtxt 是一个与 gettxt 和 mkmsgs 相似的消息处理命令。如果没有使用 *regexp* 选项，则 srchtxt 会显示来自指定文件的所有消息字符串。

options

-l *locale*
搜索保存在 /usr/lib/locale/*locale*/LC_MESSAGES 目录下的文件，*locale* 是编写消息字符串的语言。默认的 *locale* 通过环境变量 LC_MESSAGES 或 LANG 设置。如果两个环境变量都没有设置，srchtxt 会在 /usr/lib/locale/C/LC_MESSAGES 目录中进行搜索。

-m *msgfiles*
在一个或多个使用逗号分隔的文件 *msgfiles* 中搜索字符串。为 *msgfiles* 指定会覆盖 -l 选项的路径。

-s 不显示字符串中的消息数量。

ssh

ssh2 [-l *user*] *host* [*commands*]
ssh2 [*options*] [*user*@]*host*

安全的 shell。该命令是 rsh、rlogin 和 rcp 程序的安全替代者。ssh 使用加强的公钥加密技术来提供端到端的数据加密。在一些国家使用该软件时，可能有证书或专利方面的限制。

注意：在 SVR4 或 Solaris 的发布版中没有提供 ssh2。用于 Unix 的非商业版本的源代码可以从以下站点下载：*ftp://ftp.cs.hut.fi/pub/ssh*。更多的信息可以访问如下站点：*http://www.ssh.fi* 和 *http://www.ipsec.com*。

strings

strings [*options*] *files*

搜索目标或二进制文件 *files* 中使用换行符或 null 结束的有四个或四个以上可打印字符的序列。参见 **od**。

strings

options

`-a` 搜索整个*file*，而不只是目标文件的初始化数据部分。也可以将该选项指定为 `-`。

`-o` 在字符串前面显示字符串的偏移位置。

`-n` *n*

最小字符串的长度是*n*（默认是4）。也可以将该选项指定为 `-`*n*。

`-t` *format*

指定如何输出字符串的偏移量。*format*是`d`、`o`、`x`三者之一，分别表示十进制、八进制和十六进制。只用于Solaris。

strip

`/usr/ccs/bin/strip [`*options*`]` *files*

从ELF目标文件*files*或档案文件*files*中删除信息，因此会减小文件的大小并释放磁盘空间。可以删除下面的项：

1. 符号表
2. 调试信息
3. 行号信息
4. 静态符号信息
5. 外部符号信息
6. 块分隔符
7. 重新分配的位

`strip`的ELF版本提供了一些工具用于只删除最前面的三个项。

options

下面的选项涉及前面的列表：

`-b` 只删除第1、2和3项，这是默认选项。

`-l` 只删除第3项（行号信息）。

`-r` 删除第1、2、3、6项（Solaris中该选项与默认动作相同，即只删除第1、2、3等项）。

`-V` 在标准错误中输出`strip`的版本号。

`-x` 只删除第2和第3项。

stty

stty [*options*] [*modes*]

为当前设备设置终端I/O选项。没有选项时，stty会报告终端设置，此时^表示控制键，^`表示一个空值。大多数模式可以使用可选的前导符-（显示在括号中）进行切换，相应的描述也会显示在括号中。特权用户可以使用下面语法设置或读取另一个设备的设置：

stty [*options*] [*modes*] < *device*

stty是最复杂的Unix命令之一，之所以复杂是因为该命令需要处理大量的冲突、不兼容，以及非标准终端设备 —— 从打印电传打字机到CRT到窗口系统伪终端。在日常使用中，实际上只需要极少的选项。stty sane是特别值得牢记的一个。

options

-a 报告所有的选项设置。

-g 报告当前的设置。

控制模式

0 挂断连接（设置波特率为0）。

n 设置终端的波特率为*n*（比如19200）。

[-]clocal
禁止或启用调制解调器控制。

[-]cread
启用或禁止接收器。

[-]crtscts
启用或禁止使用RTS/CTS的输出硬件流控制。

[-]crtsxoff
启用或禁止使用RTS的输入硬件流控制。

cs*n* 按位选择字符的大小 （$5 \leq n \leq 8$）。

[-]cstopb
每个字符中有两个或一个停止位。

defeucw
设置每个字符按照字节表示的宽度及每个字符使用的屏幕显示列数，用于EUC（扩展的Unix代码）字符。只用于Solaris。

stty

`[-]hup`
在最后关闭时挂断或不挂断连接。

`[-]hupcl`
与`[-]hup`作用相同。

`ispeed` *n*
设置终端的输入波特率为 *n*。

`[-]loblk`
阻塞或不阻塞层输出。与`shl`一起使用，已经淘汰。

`ospeed` *n*
设置终端的输出波特率为 *n*。

`[-]parenb`
启用或禁止奇偶校验的生成及检测。

`[-]parext`
启用或禁止用于标记及空格奇偶校验的扩展奇偶校验的生成及检测。

`[-]parodd`
使用奇数或偶数奇偶校验。

输入模式

`[-]brkint`
在中断时发出或不发出信号 INTR。

`[-]icrnl`
在输入中映射或不映射回车符（`^M`）为换行符（`^J`）。

`[-]ignbrk`
在输入中忽略或不忽略中断。

`[-]igncr`
在输入中忽略或不忽略回车符。

`[-]ignpar`
忽略或不忽略奇偶校验错误。

`[-]imaxbel`
当输入行太长时，回显或不回显 BEL。

stty

`[-]inlcr`
在输入中将换行符映射为回车符或不进行映射。

`[-]inpck`
启用或禁止输入的奇偶校验检查。

`[-]istrip`
将输出字符删除到只剩 7 位或不进行删除。

`[-]iuclc`
将输入中的大写字母映射为小写字母或不进行映射。

`[-]ixany`
允许任何字符或只用 XON 来重新启动输出。

`[-]ixoff`
当队列接近为空 / 满时，发送或不发送 START/STOP 字符。

`[-]ixon`
启用或禁止 START/STOP 输出控制。

`[-]parmrk`
对奇偶错误进行标记或不进行标记。

输出模式

`bs`*n* 选择退格符的延迟类型（$n = 0$ 或 1）。

`cr`*n* 选择回车符的延迟类型（$0 \leq n \leq 3$）。

`ff`*n* 选择进纸的延迟类型（$n = 0$ 或 1）。

`nl`*n* 选择换行的延迟类型 （$n = 0$ 或 1）。

`[-]ocrnl`
在输出中将回车符映射为换行符或者不进行映射。

`[-]ofdel`
设置填充字符为 DEL 或 NULL。

`[-]ofill`
使用时间或填充字符延迟输出。

`[-]olcuc`
在输出中将小写字母映射为大写字母或不进行映射。

stty

`[-]onlcr`
在输出中将换行符映射为回车符或者不进行映射。

`[-]onlret`
在换行后执行回车换行或不执行回车换行。

`[-]onocr`
在第零列输出回车符或不输出。

`[-]opost`
进行或不进行后处理输出；忽略所有其他的输出模式。

`tab`*n*
选择水平制表符的延迟类型（$0 \leq n \leq 3$）。

`vt`*n* 选择垂直制表符的延迟类型（$n = 0$ 或 1）。

本地模式

`[-]echo`
回显键入的所有字符或者不回显。

`[-]echoctl`
将控制字符回显为`^`*char*，DEL 键回显为`^?`，或者不这样回显。

`[-]echoe`
将 ERASE 字符回显为 BS-space-BS 字符串，或者不回显。

`[-]echok`
在 KILL 字符后回显换行符，或者不回显换行符。

`[-]echoke`
在行删除中，BS-SP-BS 会擦除整个行，或者不擦除。

`[-]echonl`
回显换行符（`^J`）或者不回显。

`[-]echoprt`
回显已经擦除的字符，或者不回显。

`[-]flusho`
对输出进行刷新或不进行刷新。

`[-]icanon`
启用或禁止规范的输入（ERASE 和 KILL 处理）。

stty

`[-]iexten`
为输入数据启用或禁止扩展功能。

`[-]isig`
对 INTR、QUIT 和 SWITCH 等相关的字符进行检查，或者禁止检查。

`[-]lfkc`
与 `[-]echok` 作用相同，已经淘汰。

`[-]noflsh`
在使用 INTR、QUIT 或 SWITCH 后禁止刷新，或者进行刷新。

`[-]pendin`
在下一次读取或输入字符中重新键入挂起的输入，或者不进行键入。

`[-]stappl`
在一个同步行中的应用程序模式，或者行模式。

`[-]stflush`
启用或禁止同步行的刷新。

`[-]stwrap`
禁止或者允许截断同步行。

`[-]tostop`
当后台进程写入到终端时，发送或者不发送 `SIGTTOU`。

`[-]xcase`
改变或者不改变本地输出的大小写。

控制分配

`ctrl-char c`
设置控制字符为 *c*。*ctrl-char* 可以是如下值：`ctab`、`discard`、`dsusp`, `eof`、`eol`、`eol2`、`erase`、`intr`、`kill`、`lnext`、`quit`、`reprint`、`start`、`stop`、`susp`、`swtch`、`werase`。

`min` *`n`*
与 `-icanon`选项一起使用时，*n* 是直到由 `time`设置的超时到期前满足 `read` 系统调用的字符的最小数量。

stty

`time` *n*
与 `-icanon` 选项一起使用时，*n* 是在一个 `read` 系统调用超时以前等待的十秒时间段的数量。如果使用 `min` 设置的最小数量的字符已经被读取，则 `read` 在超时到期以前返回。

`line` *i*
设置行的规范为 *i*（$1 \leq i \leq 126$）。

合并模式

`async`
设置正常的异步通信。

`cooked`
与 `-raw` 选项一样。

`[-]evenp`
与 `[-]parenb` 和 `cs7[8]` 选项一样。

`ek` 将 ERASE 和 KILL 字符重置为 # 和 @。

`[-]lcase`
设置或不设置 `xcase`、`iuclc` 和 `olcuc`。

`[-]LCASE`
与 `[-]lcase` 选项作用相同。

`[-]markp`
启用或禁止 `parenb`、`parodd` 和 `parext` 并设置 `cs7[8]`。

`[-]nl`
设置或不设置 `icrnl` 和 `onlcr`。`-nl` 也不设置 `inlcr`、`igncr`、`ocrnl` 和 `onlret`。

`[-]oddp`
与 `[-]parenb`、`[-]parodd` 和 `cs7[8]` 选项作用相同。

`[-]parity`
与 `[-]parenb` 和 `cs7[8]` 选项作用相同。

`[-]raw`
启用或禁止原始的输入及输出（没有 ERASE、KILL、INTR、QUIT、EOT、SWITCH 或输出后处理）。

stty

`sane`
将所有的模式重置为合理的值。

`[-]spacep`
启用或禁止 `parenb` 及 `parext`，并设置 `cs7[8]`。

`[-]tabs`
扩展为空格或保持输出的制表符。

`term`
设置适合于终端类型*term*（`tty33`、`tty37`、`vt05`、`tn300`、`ti700` 或 `tek`）的所有模式（这些预定义的名字已经淘汰，所以是无用的）。

硬件流控制模式

`[-]cdxon`
在输出中启用或禁止 CD。

`[-]ctsxon`
在输出中启用或禁止 CTS。

`[-]dtrxoff`
在输入中启用或禁止 DTR。

`[-]isxoff`
在输入中启用或禁止同步的硬件流控制。

`[-]rtsxoff`
在输入中启用或禁止 RTS。

时钟模式

可能不是所有的硬件都支持这些选项：

`[x|r]cibrg`
从内部波特率产生器中获得传输或接收时钟。

`[x|r]ctset`
从发送者引导定时、CCITT V.24 circuit 114、EIA-232-D pin 15 中获得发送或接收的时钟。

stty

`[x|r]crset`
从接收者引导定时、CCITT V.24 circuit 115、EIA-232-D pin 17 中获得发送或接收的时钟。

对于使用 `t` 开头的模式，*pin* 是发送者引导定时、V.24 circuit 113、EIA-232-D pin 24；对于使用 `r` 开头的模式，*pin* 是接收者引导定时、V.24 circuit 128，没有 EIA-232-D pin。

`[t|r]setcoff`
没有发送者或接收者的定时时钟。

`[t|r]setcrbrg`
将接收波特率生成器发送到 *pin*。

`[t|r]setctbrg`
将发送波特率生成器发送到 *pin*。

`[t|r]setctset`
将发送者定时发送给 *pin*。

`[t|r]setcrset`
将接收者定时发送给 *pin*。

窗口大小

`columns` *n*
设置窗口大小为 *n* 列，也可以使用 `cols` 给出。

`rows` *n*
设置窗口大小为 *n* 行。

`xpixels` *n*
设置窗口大小为 *n* 个像素宽。

`ypixels` *n*
设置窗口大小为 *n* 个像素长。

su

`su [`*option*`] [`*user*`] [`*shell_args*`]`

用另一个用户 *user* 的有效用户 ID 建立一个 shell（即作为 *user* 进行登录）。如果没有提供 *user* 参数，则会为一个特权用户建立一个 shell（即变为一个超级用户）。输入 *EOF* 可以终止。可以通过将特定选项

su

当作 *shell_args* 传递来运行该 shell（比如，如果该 shell 运行了 sh，则可以指定 -c *command* 通过 sh 运行 *command*，或使用 -r 来建立一个受限制的 shell）。

su 会继承你的环境设置。希望切换到一个用户设置（可能帮助他们解决问题）的管理员可能会考虑使用下面的序列：

```
me$ su              切换到超级用户
Password:           输入超级用户的密码
# su - user         切换到其他用户
user$
```

option

- 经过整个的登录序列（即改变到用户 *user* 的环境）。

tail

tail [*options*] [*file*]

输出指定文件 *file* 的最后十行。-f 或 -r 二个选项一次只能使用一个。

options

-f 在文件的结尾不退出；随着文件的增长“跟随”文件。使用一个 INTR 表示文件结束（通常是 ^C）。

-r 按照相反的顺序复制行。

-*n*[*k*]
从文件结尾前的第 *n* 项开始输出。*k* 指定要计数的项，其可取值为：l（行，默认值）、b（程序块）或 c（字符）。

-k 与前面选项相同，但是使用默认数量 10。

+*n*[*k*]
与 -*n* 相似，但是该选项是从文件开头的第 *n* 项开始。

+k 与 -*k* 相似，但是从文件的开头开始计数。

示例

显示包含 .Ah 实例的最后 20 行：

```
grep '\.Ah' file | tail -20
```

继续跟踪最后的 uucp 活动：

tail

```
tail -f /var/spool/uucp/LOGFILE
```

显示变量 name 的最后 10 个字符：

```
echo "$name" | tail -c
```

反转 list 中所有的行:

```
tail -r list
```

talk

`talk user [@`*`hostname`*`] [`*`tty`*`]`

与本地计算机或计算机*hostname*上的另一个用户*user*交换键入的通信信息。当通过调制解调器进行登录并急需一些内容而打电话或发送邮件又不方便时，使用`talk`就非常有用。`talk`会将屏幕划分为两个窗口，当建立了连接后，你输入的内容会显示在上面的窗口中，而对方用户输入的内容会显示在下面的窗口中。键入`^L`可以刷新屏幕，键入`^C`（或中断）可以退出。如果*user*登录了不只一次，则可以使用*tty*来指定终端行。要求*user*已经使用过`mesg y`命令。

注意

- 多个 Unix 系统中有使用不同协议的多个版本的`talk`存在。不同的 Unix 系统之间只能进行很有限的交互。
- 如果你正在呼叫的远程用户使用了一个窗口环境，`talk`并不是很有用，因为你无法知道使用哪一个*tty*来引起对方的注意。连接请求可以很容易地显示在一个图标化的窗口中，即使你知道了远程的*tty*，被叫用户也必须输入一条`mesg y`命令来接受请求。

tar

`tar [`*`options`*`] [`*`files`*`]`

将*files*拷贝到磁带（tar,**t**ape **ar**chive 之简称）中，或者从磁带中恢复*files*。如果*files*是目录，则`tar`将会对整个子树进行操作（参见**cpio**和**pax**）。

选项作为一个组提供，后面可以按照相应的顺序放置任何参数。最初，`tar`的选项前面甚至不能使用`-`，尽管 Solaris 版本的`tar`允许有一个选项使用`-`，但是不使用也可以。在许多其他的 Unix 系统中，也

tar

可以使用常规的选项符号，各个选项前面可以加一个短划线，并使用空白字符与其他选项进行区分。一些系统实际上要求使用单独的选项。具体情况可以参考本机的文档。

注意

由于下面的一些原因，tar 最好用于通过网络交换文件或源代码档案。一般建议系统管理员使用厂商提供的备份程序（通常称为 dump 或 backup，参见本机的文档）而不是 tar 来进行系统的备份。（许多方面也同样适用于 cpio 和 pax。）

- 大多数 Unix 版本的 tar 保护前导符“/”，使其不受档案中绝对文件名的影响，这使得很难或不可能在不同的系统中对文件进行释放。
- 当 Unix 文件或目录的名字是短格式（最多 14 个字符）时，设计了 tar 档案格式。当今的 Unix 系统允许单个的文件名最多使用 255 个字符，但是 tar 档案标题限制整个路径名最多使用 100 个字符。这使得在实践中不可能或比较难于归档一个典型的 Unix 文件系统。
- 通常情况下，Unix 版本的 tar 不能修复数据错误，这与使用磁带时的情况相同。一个早期的磁带错误可以导致整个 tar 磁带无法使用。
- 因为 tar 会校检各个归档文件中描述的标题信息，而不校检实际的数据块，所以如果一个数据块在磁带上被破坏，tar 将永远无法注意到。

GNU 版本的 tar 有一些扩充功能来摆脱这些问题，其代价是牺牲到非 GNU 版本的档案格式的可移植性。可以从自由软件基金会获得该 tar 版本的源代码（*http://www.gnu.org*）。

控制选项（Solaris）

`-C` *`dir files`*

在向档案添加文件以前，将目录修改为 *dir*。应使用相对路径名。该选项可以使档案文件不使用共同的祖先目录。

tar

`-I` *file*

从 *file* 中读取将要进行归档的文件名列表，一个文件名占一行。在命令行中有太多的文件需要指定时应使用该选项。

`-X` 排除一些文件。为不应进行归档的文件的相对路径名的一个列表读取相应的文件参数，其中的每个文件参数占一行。该选项可以使用多次，并使用多个文件。这里显示的文件名将被排除，即使使用 `-I` 选项的一个文件中提供了相同的文件名。

功能选项（只能使用一个选项）

`c` 建立一个新的档案。

`r` 将 *files* 添加到档案中。

`t` 内容表。如果 *files* 保存在档案中，则输出这些文件的名字（如果没有指定 *files*，则输出所有文件的名字）。

`u` 进行更新。将不在档案中或已经修改的文件添加到档案中。

`x` 从档案中释放文件 *files*（如果没有指定 *files*，则释放所有的文件）。

options

`b`*n* 使用分块因数 *n*（默认是 1，最大是 20）。一些 Unix 系统经常允许使用较大的分块因数。

`B` 持续读取数据，直到逻辑块中充满数据。与 `rsh` 一起使用，用来跨越以太网连接。当读取标准输入时使用默认设置。只用于 Solaris，但是有时也能用于其他 Unix 系统中。

`e` 遇到意外错误时立即退出。只用于 Solaris。

`E` 使用允许长文件名、较大的文件及其他扩展名的一个扩展标题。不能移植，只用于 Solaris。

`f` *arch*

将文件保存到档案 *arch* 中，或者从档案 *arch* 中释放文件，*arch* 通常是一个设备的名称（默认情况下根据使用的系统不同，该设备的名称也不同）。如果 *arch* 是 `-`，则会适当使用标准输入或输出（比如，将一个 `tar` 档案通过管道传输到一台远程主机中）。

tar

`F`，`FF`

选择`F`时，不会对`SCCS`和`RCS`目录进行归档；选择`FF`时，则归档时排除`a.out`、`core`、`errs`文件和所有的`.o`文件。只用于Solaris。

`i` 忽略目录校验错误。只用于Solaris。

`k` *size*

指定档案的大小，使用千字节表示。大于*size*的档案被划分成多个卷。主要用于固定大小的媒体，比如软盘。只用于Solaris。

`l` 输出不能发现的链接错误信息。

`L` 紧跟在符号链接的后面。只用于SVR4。

`m` 不恢复文件的修改时间，而将其更新为文件释放的时间。

`n` 档案不是一个磁带设备，从而允许`tar`进行查找，而不是进行顺序读取，这样执行速度比较快。只用于Solaris。

`o` 将释放文件的属主修改为用户正在运行的程序的属主。这是非特权用户的默认选项。

`p` 保持释放文件的权限。如果将Solaris ACL记录在档案中，或与`c`一起使用时添加到档案中，则可以对其进行恢复。

`P` 不在档案的目录名中添加一个后缀`/`。只用于Solaris。

`v` 输出功能字母（`x`用于文件释放，`a`用于文件归档）和文件名。使用`t`时，会输出一个与`ls -l`执行结果相似的列表。

`w` 等待用户进行确认（`y`）。

n[*c*] 选择磁带驱动器*n*，使用的速度为*c*。*n*是0~7（默认是0）；*c*是l（低速）、h（高速）或n（中速，默认值）。可用于修改*arch*。（它们在很大程度上特定于系统，并且是不可移植的。这比只显式地指定*arch*更好一些。）

示例

建立一个`/bin`和`/usr/bin`（`c`）的档案，显示命令的操作（v），写入磁带的`/dev/rmt/0`位置：

```
tar cvf /dev/rmt/0 /bin /usr/bin
```

使用与`ls -l`相似的格式列出档案内容：

tar

```
tar tvf /dev/rmt/0
```

释放 /bin 目录：

```
tar xvf /dev/rmt/0 /bin
```

建立当前目录的一个档案，并将其保存到系统的/tmp/backup.tar文件中。（将一个目录备份到一个文件中，因为该目录中几乎不起作用。）

```
tar cvf /tmp/backup.tar .
```

与前一个命令相似，但是会压缩档案文件：

```
tar cvf - . | compress > /tmp/backup.tar.Z
```

（- 可以使 tar 命令知道应该将目录保存到标准输出中，然后通过管道进行重定向。）

将一个目录树从一个位置拷贝到另一个位置：

```
# cd olddir; tar cf - . | (cd newdir; tar xvpf -)
```

tbl

tbl [*options*] [*files*]

nroff/troff 表的预处理程序，参见第十七章。

tee

tee [*options*] [*files*]

复制标准输入，将其中的一个副本发送到标准输出中，将另一个副本发送到 *files* 中。

options

-a 将输出添加到 *files* 中。

-i 忽略所有的中断。

示例

在屏幕上显示一个 who 的列表，并将其保存到两个文件中：

```
who | tee userlist ttylist
```

显示拼错的单词，然后将它们添加到已经存在的 typos 中：

```
spell ch02 | tee -a typos
```

telnet

telnet [*options*] [*host* [*port*]]

使用 Telnet 协议与另一个主机 *host* 进行通信。*host* 可以是一个主机的名字，也可以是一个数字的 Internet 地址（小点格式）。telnet 有一个命令模式（使用 telnet> 提示符表示）和一个输入模式（通常是 *host* 系统中的一个登录会话）。如果没有给出 *host*，则 telnet 默认使用命令模式。也可以通过在输入模式中输入转义字符 ^]进入命令模式。在命令模式中，键入?或 help 可以列出可用的命令。

Solaris 选项

Solaris 的 telnet 提供下面的选项：

-8 使用一个 8 位数据路径。这会协商 BINARY 选项来确定输入及输出。

-c 在启动时不读取 $HOME/.telnetrc。

-d 设置 debug 选项为 true。

-e *c*
使用 *c* 作为转义字符。默认是 ^]。一个空值会禁止转义字符机制。

-E 没有转义字符。

-l *user*
使用 ENVIRON 选项来传递 USER 环境变量的值。

-L 在输出中使用一个 8 位数据路径，这会协商 BINARY 选项以便只用于输出。

-n *file*
记录 *file* 中的跟踪信息。

-r 提供一个 rlogin 风格的接口，此时转义字符是 ~，只能将其放在一个回车符后才被认可。规则的 telnet 转义字符必须在一个 telnet 命令前面使用。“ ~. 回车 ”和“ ~^Z ”会分别终止及停止一个会话。该功能在 Solaris 的未来版本中可能会改变。

test

test *expression*
或
[*expression*]

计算一个表达式 *expression* 的值，如果该值是 true，则返回一个 0 退出状态；否则，返回一个非零的退出状态。在 shell 脚本中，可以使用另一种形式[*expression*]，括号也要输入，以便与 *expression* 区分开。通常情况下，该命令与条件结构一起用于 shell 程序中。更多的信息可以参见第四章中讲述的 test。

time

time [*option*] *command* [*arguments*]

执行带有可选参数 *arguments* 的一条命令 *command*，显示总的耗费时间、执行时间、进程执行时间和进程的系统时间（都按秒计算），并在标准错误中输出时间。

option
该选项只用于 Solaris:

-p 显示实际时间、用户时间和系统时间，并使用单个空格代替制表符分隔标题和值。

timex

timex [*options*] *command* [*arguments*]

执行带有可选参数 *arguments* 的一条命令 *command*，并输出 time 命令指定的信息。使用不同选项报告进程数据。

options
-o 显示使用的总块数和字符数量。

-p *suboptions*
显示使用可能存在的 *suboptions* 选项的进程核算数据。

-s 显示总的系统活动。

用于 -p 的子选项
-f 包括 fork/exec 标志和系统退出状态。

timex

`-h` 显示“占用（hog）”因子（CPU 使用的时间片）（译注 2），而不是平均内存的大小。

`-k` 显示千核心分钟（kcore-minute）（译注 3）而不是内存大小。

`-m` 显示平均核心内存大小（这是默认的行为）。

`-r` 显示 CPU 的利用率（用户时间 /（系统时间 + 用户时间））。

`-t` 显示用户和系统的 CPU 时间。

touch

`touch [`*`options`*`] [`*`date`*`] `*`files`*

对于 *files* 指定的一个或多个文件，更新其访问时间和修改时间信息为当前的时间和日期，或更新为可选的 *date*。*date* 是使用 *mmddhhmm*[*yy*]格式的一个日期和时间。`touch` 主要用于强制其他命令按照特定的方式处理文件，比如，`make` 进行的操作及有时使用 `find` 命令，都依赖于一个文件的访问和修改时间。

options

`-a` 只更新访问时间。

`-c` 不建立不存在的文件。

`-m` 只更新修改时间。

`-r` *`file`*
使用 *file* 的访问及（或）修改时间而不是当前的时间。只用于 Solaris。

`-t` *`time`*
使用 *time* 提供的时间，其形式为[[*cc*]*yy*]*mmddhhmm*[.*ss*]。只用于 Solaris。

tput

`tput [`*`options`*`] `*`capname`*` [`*`arguments`*`]`

通过 `terminfo` 数据库输出终端权能（capability）*capname*（及与其相关的数字和字符串参数*arguments*）的值。*capname* 是一种 `terminfo`

译注 2： 计算公式为总 CPU 时间 / 耗费时间。

译注 3： 千核心分钟是进程的执行时间按秒规格化后的平均驻留大小（单位为 KB）。

tput

权能，比如clear或col。（参见参考文献中的*termcap & terminfo*。）最后的5个选项是互斥的，当指定了*capname*时，不能使用这些选项。

退出状态如下：

0 当一个 Boolean 类型的 *capname* 设置为 true 或者定义了一个字符串类型的 *capname* 时。

1 当 Boolean 值是 false 或者没有定义一个字符串时。

2 用于使用错误。

3 用于未知的终端类型 *type*。

4 用于未知的 *capname*。

options

`-T`*type*
显示终端类型 *type* 的权能（默认值是使用的终端）。

`-S` 从标准输入中读取 *capname*（这允许 `tput` 计算多于一个的 *capname*）。

`clear`
输出清屏序列。只用于 Solaris。

`init`
输出初始化字符串和扩展的制表符。

`reset`
如果字符串存在，则重新设置字符串；如果不存在，则实现与 `init` 相似的功能。

`longname`
输出终端的长名字。

示例

显示 `xterm`终端类型中列的数量：

```
tput -Txterm cols
```

定义 shell 变量 `restart` 来重新开始重置终端特性：

```
restart=`tput reset`
```

tr

tr [*options*] [*string1* [*string2*]]

将标准输入拷贝到标准输出中，执行*string1*到*string2*的字符替换或删除*string1*中的字符。System V 中要求*string1*和*string2*使用方括号括起来。BSD 版本则没有该要求。

options

`-c` 使用当前字符集中的字符补充*string1*中的字符，补充的字符是所有不在*string1*中的字符。

`-d` 从输出中删除*string1*中的字符。

`-s` 删除*string2*中重复的输出字符。

示例

将*file*中的大写字母修改为小写字母：

```
tr '[A-Z]' '[a-z]' < file
```

Solaris 允许使用的字符类别：

```
tr '[:upper:]' '[:lower:]' < file
```

将空格转换为换行符（ASCII 码是 012）：

```
tr ' ' '\012' < file
```

从`file`中删除空行并保存到一个`new.file`中（或者使用`\011`将多个连续的制表符替换为一个制表符）：

```
tr -s "" "\012" < file > new.file
```

从`file`中删除冒号，并将结果保存到`new.file`中:

```
tr -d : < file > new.file
```

使长的搜索路径更具有可读性:

```
echo $PATH | tr ':' '\n'
```

troff

troff [*options*] [*files*]

用于激光打印机或排字机的文档格式化程序。参见第十二章。

true

`true`

该命令执行空操作，只返回一个成功的退出状态（0）。通常用于 Bourne shell 脚本中。参见 **false**。

truss

`truss [`*`options`*`] `*`arguments`*

跟踪执行*arguments*时系统的调用、信号及计算机出现的错误。*arguments*可以是将要运行的Unix命令；如果指定了`-p`选项，则*arguments*是一系列进程ID，表示所跟踪的正在运行的进程。`-m`、`-r`、`-s`、`-t`、`-v`、`-w`和`-x`选项接受使用逗号间隔的参数列表。`!`会颠倒列表的意义，并且在跟踪时通知`truss`忽略列表元素（在C shell中，在`!`前面使用一个反斜线）。关键字all可以包括或排除列表中所有可能的元素。可选项`!`和相应的描述在括号中显示。

Solaris中的`truss`，也提供对动态装载的共享库中的用户级函数调用的跟踪。

当一个第三方的程序无法运行时，使用该命令可以非常方便地发现丢失的文件。通过观察系统调用`access`和`open`，你可以发现哪里的及哪些文件是应用程序希望找到而没有找到的。

许多系统具有类似的程序`trace`或`strace`，掌握这些程序的使用方法是很重要的。

options

`-a` 显示通过每个 *exec*(2)调用传递的参数。

`-c` 计算跟踪项目并显示一个摘要，而不是在它们发生时列出它们。

`-d` 在输出中输出时间信息，以*seconds.fraction*形式出现，表明相对于跟踪开始的时间。时间是指系统调用完成的时间，而不是开始的时间。只用于 Solaris。

`-D` 在输出中输出一个 delta 时间信息，使用 *seconds.fraction* 的形式，表明两个事件之间的间隔时间（也就是说，不是进行系统调用时的时间）。只用于 Solaris。

`-e` 显示各个 *exec*(2)调用传递的环境变量的值。

truss

`-f` 跟随子进程。主要用于跟踪 shell 脚本。

`-i` 只对休眠系统调用进行一次列表输出，直到完成。

`-m[!]`*`faults`*
跟踪（或从跟踪中排除）计算机 *faults* 的列表，*faults* 代表名字或号码，列在了 `<sys/fault.h>` 中（默认是 `-mall -m!flt-page`）。

`-M[!]`*`faults`*
当被跟踪进程接收到一个指定的faults时，`truss`将会使该进程处于停止状态，并从中分离出来（默认是 `-M!all`）。该进程可以随后与一个调试器连接在一起，或与使用不同选项的 `truss` 的另一个调用连接在一起。只用于 Solaris。

`-l` 显示一个多线程的进程中的轻权进程。只用于 Solaris。

`-o` *`outfile`*
将跟踪输出发送到 *outfile* 而不是标准错误中。

`-p` 跟踪一个或多个正在运行的进程，而不是一条命令。

`-r[!]`*`file_descriptors`*
显示（或不显示）用于 *file_descriptors* 的 `read` 系统调用的全部 I/O 缓冲区（默认是 `-r!all`）。

`-s[!]`*`signals`*
跟踪（或从跟踪中排除）signals的列表。signals是名字或号码，在 `<sys/signal.h>` 中列出（默认是 `-sall`）。

`-S[!]`*`signals`*
当被跟踪进程接收到一个指定的信号时，`truss` 会使进程处于停止状态，并从中分离出来（参见 `-M`）（默认是 `-S!all`）。只用于 Solaris。

`-t[!]`*`system_calls`*
跟踪（从跟踪中排除）*system_calls* 的列表。*system_calls* 是名字或号码，在《UNIX Programmer'S Reference Manual》的第二部分“ System Calls ”中对此进行了说明（参见参考文献）。默认是 `-tall`。

truss

`-T[!]`*`system_calls`*

当被跟踪进程执行一个指定的系统调用时，`truss`会使该进程处于停止状态，并从中分离出来（参见 `-M`）（默认是`-T!all`）。只用于 Solaris。

`-u[!]`*`lib`*`, ...:[:][!]`*`func`*`,...`

跟踪用户级函数调用，而不只是系统调用。*lib* 是使用逗号间隔的多个动态库名字的列表，没有 `.so.`*n* 后缀。*func* 是一个使用逗号间隔的名字列表。可以使用 shell 的通配符语法指定许多名字（这种使用应该使用引号引起来，以防止被 shell 扩展）。前面使用 `!` 表示排除库及（或）函数。使用`:`只会跟踪从外部对库进行的调用；使用`::`则会跟踪所有的调用。只用于 Solaris。

`-U[!]`*`lib`*`, ...:[:][!]`*`func`*`,...`

当被跟踪的进程执行一个指定的用户级函数时，`truss`会使该进程处于停止状态，并且从中分离出来（参见 `-M`）。只用于 Solaris。

`-v[!]`*`system_calls`*

详细模式。与 `-t` 作用相同，但是也会列出传递到 *system_calls* 的任何结构的内容（默认是 `-v!all`）。

`-w[!]`*`file_descriptors`*

显示（或不显示）用于 *file_descriptors* 的`write`系统调用的全部 I/O 缓冲区（默认是 `-w!all`）。

`-x[!]`*`system_calls`*

与 `-t`选项作用相同，但是按照原始代码（十六进制）显示系统调用的参数（默认是 `-x!all`）。

示例

跟踪用于 `lp` 命令的系统调用 `access`、`open` 及 `close` 等：

```
truss -t access,open,close lp files  > truss.out
```

跟踪 `make` 命令，包括其子进程，并将输出保存到 `make.trace` 中：

```
truss -f -o make.trace make target
```

tset

/usr/ucb/tset [*options*] [*type*]

设置终端模式。没有参数时，按照TERM环境变量对终端重新进行初始化。tset主要用于启动脚本（.profile或.login）。*type*是终端的类型，如果前面使用了一个问号，则tset会在需要时提示用户输入一个不同的类型。按回车键则使用默认的值*type*。参见**reset**。

options

- 在标准输出中输出终端的名字，主要用于将该值传递到TERM中。

-e*c* 设置删除字符是*c*，默认是^H（退格键）。

-i*c* 设置中断字符是*c*（默认是^C）。

-I 不输出终端的初始化设置。

-k *c*
设置删除行的字符为*c*（默认是^U）。

-m[*port*[*baudrate*]:*type*]
声明终端规范。*port*是端口的类型（通常是dialup或plugboard）。*tty*是终端类型，可以将?放在其前面。*baudrate*则用于检查端口的速度，前面可以使用下面的任何字符：

> 端口必须大于*baudrate*。

< 端口必须小于*baudrate*。

@ 端口必须按照*baudrate*规定的速度传输。

! 对随后的>、<或@等字符取反。

? 提示终端类型。没有响应时，使用给出的*type*。

-n 初始化新的tty驱动程序模式。由于SVR4中使用合并了BSD新的tty驱动程序功能的默认的stty设置会带来冗余，所以一般不使用该选项。

-Q 不输出“Erase set to”和“Kill set to”消息。

-r 报告终端的类型。

-s 返回TERM分配给shell环境变量的值。这一般通过eval \`tset -s\`（在C shell中，应该使用命令set noglob和unset noglob将其包围）实现。

tset

示例

设置 TERM 为 wy50:

```
eval `tset -s wy50`
```

提示用户输入终端类型（默认是 vt100）：

```
eval `tset -Qs -m '?vt100'`
```

与上面的示例相似，但是波特率必须超过 1200：

```
eval `tset -Qs -m '>1200:?xterm'`
```

通过调制解调器设置终端。如果不在拨入线路上，则?$TERM会导致 tset 提示使用 $TERM 的值作为默认的终端类型：

```
eval `tset -s -m dialup:'?vt100' "?$TERM"`
```

tty

tty [*options*]

输出你的终端的设备名称。主要用于 shell 脚本，并且经常用于需要设备信息的命令。

options

-l 如果在一个活动的同步线路上，则会显示同步的线路行号。

-s 只返回代码: 0 （一个终端）、1（不是一个终端）、2（使用了无效的选项）。

type

type *program* ...

输出对 *program* 的描述，也就是说，不管它是 shell 内置的、是一个函数还是外部命令。type 是 Bourne 和 Korn shell 的内置命令。参见第四章和 **which** 命令。

示例

描述 cd 和 ls 命令：

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

umask

umask [value]

输出文件建立模式掩码的当前值；或设置其值为 *value*，该值是一个使用三个数字表示的八进制代码，用于指定将要关闭的读、写和执行权限。该命令与 chmod 作用相反，通常用于 .login 或 .profile 文件中。在 Bourne、Korn 和 C shell 中，umask 是一个内置命令（参见第四章和第五章）。

umask 编号	文件权限	目录权限
0	rw-	rwx
1	rw-	rw-
2	r--	r-x
3	r--	r--
4	-w-	-wx
5	-w-	-w-
6	---	--x
7	---	---

示例

关闭其他文件的写权限：

```
umask 002        产生文件权限 -rw-rw-r--
```

关闭组和其他文件的所有权限：

```
umask 077        产生文件权限 -rw-------
```

注意，可以删除前面的 0。

uname

uname [*options*]

输出当前 Unix 系统的名称。

options

-a 报告所有其他选项提供的信息。

-i 显示硬件平台的名称（例如，i86pc；可以与来自 -p 选项的 i386 比较）。只用于 Solaris 。

uname

-m 显示硬件的名称。

-n 显示节点的名称。

-p 显示主机处理器的类型。

-r 显示操作系统的发布版本。

-s 显示系统的名字。如果没有提供任何选项，则该选项是默认值。

-v 显示操作系统的版本。

-S *name*
将节点的名字修改为*name*，只用于特权用户。且只用于Solaris。

-X 正如 SCO Unix 系统希望实现的那样，输出扩充信息。只用于 Solaris。

uncompress

uncompress [*option*] [*files*]

对使用compress压缩的原始文件进行恢复。隐含使用 .Z扩展名。当指定 *files* 参数时，可以省略该扩展名。

也可以使用 compress 命令的 -f 和 -v 选项。更多的信息可以参见 **compress** 命令。

option

-c 与 zcat 命令相同（只写入到标准输出中，而不会修改 *files*）。

unexpand

unexpand [*options*] [*files*]

将空格转换成适当数量的制表符。unexpand 会读取 *files* 指定的文件；如果没有提供 *files* 参数，则读取标准输入。参见 **expand**。

options

-a 只要可能，就将所有位置的空格替换为制表符，而不只是前面的空格和制表符。

-t *tablist*
按照 *tablist* 解释制表符，*tablist* 是使用空格和逗号间隔的多个数字的列表，该列表按照升序排列，并描述了输入数据的“制表位”。

unget

`/usr/ccs/bin/unget [`*options*`] `*files*

一条 SCCS 命令，参见第十八章。

uniq

`uniq [`*options*`] [`*file1* `[`*file2*`]]`

从排序文件 *file1* 的重复的相邻行中删除多余的行，并将各个重复行的一个副本拷贝到 *file2*（或者标准输出）中。该命令通常作为一个过滤器使用，并只能指定 `-c`、`-d` 或 `-u` 中的一个参数。参见 **comm** 及 **sort**。

options

`-c` 各个行只输出一次，但计算各行的所有实例。

`-d` 重复的行只输出一次，但是不输出惟一的行。

`-f` *n*
忽略一行中最前面的 *n* 个字段，并使用空格或制表符区分各个字段。只用于 Solaris。

`-s` *n*
忽略一个字段中最前面的 *n* 个字符。只用于 Solaris。

`-u` 只输出惟一的行（不输出有重复的行）。

`-n` 忽略一行中最前面的 *n* 个字段，使用空格或制表符区分各个字段。

`+n` 忽略一个字段中最前面的 *n* 个字符。

示例

将来自 `list` 中每行的一个副本发送到输出文件 `list.new` 中（`list` 必须是排过序的）：

```
uniq list list.new
```

显示哪些名字的出现多于一次：

```
sort names | uniq -d
```

显示哪些行正好出现 3 次：

```
sort names | uniq -c | awk '$1 == 3'
```

units

units

交互地提供一个公式将一个单位的数字转换为另一个单位的数字。/usr/lib/units（Solaris: /usr/share/lib/unittab）给出了units的完整列表。可以使用*EOF*退出。

unix2dos

unix2dos [*options*] *unixfile dosfile*

只用于Solaris。将使用ISO标准字符的文件转换为对应的DOS文件。如果*unixfile*和*dosfile*是同一个文件，则在转换完成后覆盖该文件。参见**dos2unix**。

options

-ascii
: 添加额外的回车符以便能用于DOS下。

-iso
: 与默认的动作一样。

-7
: 将8位的Solaris字符转换为7位的DOS字符。

unzip

unzip [*options*[*modifiers*]] *zipfile* ... [*extraction options*]
unzip -Z [*zipinfo options*] *zipfile* ...

只用于Solaris（许多现代的Unix系统都具有该命令）。unzip会输出ZIP格式的档案信息，或者从ZIP格式的档案中对文件进行解压缩。*zipfile*是一个ZIP档案文件，其文件扩展名为.zip。在命令行中可以省略.zip, unzip命令会添加该扩展名。*zipfile*也可以是一个shell风格的通配符模式（应该使用引号包括）。该命令对ZIP档案中的所有匹配文件起作用。*options*的行为将受到各种不同修饰符的影响。

在第二种形式中，*options*将使用zipinfo的选项，此时的unzip与zipinfo实现相似的功能。更多的信息可参见**zipinfo**。

要设置一种默认行为，可以在UNZIP环境变量中包括多个选项。通过在命令行的选项前面增加一个额外的减号(-)，这些选项可以覆盖$UNZIP中的设置。参见后面的示例。

unzip

当对文件进行解压缩时，如果一个文件已经存在，则unzip会提示下一步的操作，你可以选择覆盖或是跳过这个已经存在的文件、覆盖或跳过所有文件、对当前的文件重新命名等操作。

注意

- unzip及对应命令zip（Solaris中不包括该命令）是InfoZIP工程的一部分，InfoZIP是一种开放的协作压缩档案格式，Unix、Amiga、Atari、DEC VAX和Alpha VMS和OpenVMS、MS-DOS、Macintosh、Minix、OS/2、Windows NT及其他操作系统中都有相应的实现。只有与*only*相似的格式可以毫无困难地移植到这些系统中。

  Web主页是：*http://www.cdrom.com/pub/infozip*。
- 与许多Unix系统中的tar实现不同，zip在建立一个ZIP档案时会删除前面的斜线，所以在另一个站点中对该文件进行分售是没有问题的。
- Java档案格式（.jar）是基于ZIP的，zip和unzip完全可以处理.jar文件而不会出现问题。

extraction options（解压选项）

-d *dir*
对*dir*中的文件而不是当前目录中的文件进行解压缩。该选项不需要放在命令行的结尾。

-x *files*
排除。不会对与*files*匹配的档案成员进行解压缩。

options

-A 输出共享库编程接口（API）的帮助。

-c 在标准输出（CRT）中输出文件。与-p相似，但是为每个文件输出一个标题行，它允许使用-a，并自动进行ASCII到EBCDIC的转换。在unzip的使用说明中没有对该选项进行说明。

-f 刷新已经存在的文件。只对档案中那些比磁盘中更新的文件进行解压缩。unzip在进行文件覆盖以前会进行询问，除非使用了-o选项。

unzip

`-l` 使用短格式（名称、完整大小、修改时间及总和等）列出归档的文件。

`-p` 将文件解压缩后输出到标准输出中（通过管道）。只打印文件数据。不进行任何转换。

`-t` 测试归档的文件，各个文件在内存中进行解压缩，并把被解压文件的CRC与存储的CRC进行比较。

`-T` 设置档案本身的时间信息为档案中最新文件的时间信息。

`-u` 与`-f`相同，但是也会对磁盘上不存在的文件进行解压缩。

`-v` 表示详细的信息或输出诊断信息。`-v`既是一个选项，也是一个修饰符，依赖于其他的选项。它可以独立地输出`unzip ftp`站点信息、有关如何对其进行编译及设置什么样的环境变量将会起作用的信息。带有一个*zipfile*选项时，它会将压缩信息添加到`-l`提供的文件中。

`-z` 只显示档案注释。

`-Z` 当作`zipinfo`命令执行。剩余的选项是`zipinfo`选项，更多的信息参见**zipinfo**。

modifiers（修饰符）

`-a[a]`

转换文本文件。通常情况下，会将文件当作二进制文件进行提取。该选项会导致文本文件转换为本地格式的文件（比如，添加或删除在LF字符前面的CR字符）。根据需要也可以进行EBCDIC到ASCII的转换。使用`-aa`可以强制将所有文件当作文本进行解压缩。

`-b` 将所有的文件当作二进制文件。

`-B` 在文件*file*~中为各个已经覆盖的文件保存一个副本。只有使用已经定义的`UNIXBACKUP`进行编译时，该选项才可用。

`-C` 在匹配文件名时忽略大小写。主要用于对字母的大小写不敏感的非Unix文件系统中。

`-j` “删除”路径。在当前解压缩目录中解压缩所有的文件，而不是重新产生保存在档案中的目录树结构。

unzip

`-L` 将在只使用大写字母的系统中建立的档案中的文件名转换为小写字母。默认情况下，不改变解压缩文件名的大小写。

`-M` 通过内部的页面调度程序进行管道输出，类似于`more`。在出现`--More--`提示符时按回车键或空格键可以显示下一屏。

`-n` 永远不会覆盖已经存在的文件。如果一个文件已经存在，则不会对其进行解压缩，只会继续操作而不进行提示。通常情况下，`unzip`会对下一步的操作进行提示。

`-o` 覆盖已经存在的文件而不进行提示。通常与`-f`一起使用，使用该选项时应小心。

示例

列出一个 ZIP 档案的内容：

```
unzip -lv whizprog.zip
```

在主目录中解压缩 C 源文件，而不是在子目录中：

```
unzip whizprog.zip '*.[ch]' -x '*/*'
```

uptime

`uptime`

输出当前的时间、系统已经运行的时间总数、登录用户的数量及系统负载平均数等。`w`命令的第一行也会产生该输出。

users

`/usr/ucb/users [`*`file`*`]`

在一个空格间隔的列表中显示当前登录的用户。与`who -q`作用相同。从一个系统文件 *file*（默认是`/var/adm/utmp`）中读取信息。

uudecode

uudecode [*-p*] [*file*]

读取一个未编码的文件或使用相同的模式和名称来重新建立原始的文件（参见 **uuencode**）。

Solaris 提供了`-p`选项，它会将文件 *file* 解码到标准输出中。这允许在一个管线中使用 uudecode。

uuencode

uuencode [*file*] *name* | mail *remoteuser*

将一个二进制文件 *file* 转换成可以通过邮件发送到远程用户 *remote-user* 的一种形式。编码只使用打印的 ASCII 字符并包括该文件的模式和名称 *name*。当 *file* 经过 uudecode 重新转换到一个远程系统上时，输出会发送到 *name* 中。（因此，当将编码邮件消息发送到远程系统上的一个文件中时，不要将其保存到名字为 *name* 的文件中，否则将覆盖它。）注意，uuencode 可以使用标准输入，这样在对文件进行解码时，给该文件设置的名字会是惟一的参数。

Solaris 版本的 uuencode 会将本地字符集转换为编码字符集。

注意：uuencode 格式不会提供任何类型的求和检查及其他的数据完整性检查。建议首先将文件打包到一个不提供数据求和检查的档案（比如一个 .zip 文件）中，然后用 uuencode 对该档案进行处理以便可以使用电子邮件进行发送。

vacation

vacation

vacation [*options*] [*user*]

自动将邮件消息返回给一个发送人，声明你在休假。Solaris 版本的该命令应与 sendmail 一起使用。（附录二中描述了 SVR4 版本。）

使用不带有任何选项的 vacation 命令，可以初始化休假机制。该过程执行以下几个步骤：

1. 在你的主目录中建立一个 .forward 文件，该 .forward 文件包含如下内容：

   /user, "|/usr/bin/vacation user"

   user 是你登录使用的名字，该文件执行的动作实际上是将邮件传递给 *user*（即你本人），然后使用 vacation 运行输入的邮件。
2. 建立 .vacation.pag 和 .vacation.dir 文件，这些文件会保留哪些人向你发送了消息，从而他们每星期会接收到一条你的“我在休假”的消息。
3. 启动一个编辑器来编辑 .vacation.msg 的内容。该文件的内容会发送给所有向你发送邮件的人。在其正文中，收到的消息的 Subject 行的内容会替换 $SUBJECT。

vacation

删除或重命名 `.forward` 文件会禁止 vacation 的处理。

options

`-a`、`-j` 和 `-t` 这些选项用于 `.forward` 文件中，参见后面的示例。

`-a` *alias*
: 将邮件发送到 *alias*，实际上是为用户发送邮件，并应该产生一个自动应答。

`-I`
: 重新初始化 `.vacation.pag` 和 `.vacation.dir` 文件。在即将开始下一次休假以前，使用该选项最合适。

`-j`
: 不会对显示在 `To:` 或 `Cc:` 标题中的用户进行校验。

`-t` *interval*
: 默认情况下，每星期发送给所有发件人的消息不超过一条。该选项会改变默认的时间间隔。*interval* 是尾部添加了 `s`、`m`、`h`、`d` 或 `w` 的一个数字，分别表示秒、分钟、小时、天或星期。

示例

每隔三个星期向所有给定的寄件人发送不超过一条的应答：

```
$ cd
$   vacation -I
$   cat .forward
/jp, "|/usr/bin/vacation -t3w jp"
$   cat .vacation.msg
From: jp@wizard-corp.com （J. Programmer, via the
vacation program）
Subject: I’m out of the office ...
Hi. I'm off on a well-deserved vacation after finishing
up whizprog 1.0. I will read and reply to your mail
regarding "$SUBJECT" when I return.

Have a nice day.
```

val

`/usr/ccs/bin/val [`*options*`] `*file* `...`

一条 SCCS 命令，参见第十八章。

vedit

vedit [*options*] [*files*]

与运行vi的作用相同，但是使用showmode（显示模式）和novice（新手）标志设置，report（报告）标志设置为1，关闭magic（没有特殊意义的元字符）。该命令主要用于初学者。

vgrind

vgrind [*options*] *files*

只用于Solaris（来自BSD的命令）。产生非常详细的格式化源代码列表以便与troff一起使用。vgrind会格式化程序源代码，以便使用troff排版时，这些源代码排列更美观。注释使用斜体、关键字使用粗体、各个函数的名字打印在定义该函数的页边空白位置。各种语言的定义保存在/usr/lib/vgrindefs中。vgrind可以格式化一定量的语言，参见下面的-l选项。

vgrind有两种操作模式：

过滤模式

与eqn、pic和tbl相似，在传递行时不对其进行修改，除了那些使用.vS和.vE包括的内容。在该模式中，vgrind可以与其他预处理器一起用于一个管线中。

规则模式

vgrind会处理命令行中指定的所有文件，然后调用troff来输出它们。使用-作为一个文件名表示标准输入，否则vgrind将不会从标准输入中读取数据。

options

在选项字符和选项参数之间的空格是特定的，选项的使用方法如下：

-2 产生两列的输出，隐含用于$TROFF的-s8和-L（横向）选项（该选项在UCB中特定于向troff）。

-d *definitions*
使用*definitions*作为有语言定义的文件，而不是默认的文件。

-f 运行于过滤模式。

-h *header*
将标题*header*放置在每个输出页顶部的中间。

vgrind

`-l`*lang*

支持如下语言：

Bourne shell	`-lsh`
C	`-lc`
C++	`-lc++`
C shell	`-lcsh`
emacs MLisp	`-lml`
FORTRAN	`-lf`
Icon	`-lI`
ISP	`-li`
LDL	`-lLDL`
Model	`-lm`
Modula-2	`-lm2`
Pascal	`-lp`
RATFOR	`-lr`
Russel	`-lrussell`
YACC	`-lyacc`

默认是 `-lc`（用于 C 语言）。

`-n` 关键字不使用粗体。

`-s`*size*

使用磅值大小（与 `troff` 的 `.ps` 请求相同）。

`-w` 使用跨 4 个列的制表符，而不是 8 个列的制表符。

`-x` 输出索引。如果名字为 `index` 的文件保存在当前目录中，`vgrind` 会将索引写入到该文件中。该文件可以使用 `vgrind -x index` 分别进行格式化和打印。

打字机选项

下面的选项会传递到 $TROFF 指定的程序中，如果没有设置环境变量，则会传递给 `troff`：

vgrind

`-o`*list*
只输出在 *list* 中的页面，与 `troff` 中的 `-o` 选项作用相同。

`-P`*printer*
将输出发送到打印机 *printer*。

`-t` 与 `troff` 中的 `-t` 选项相同，向标准输出中发送格式化的文本。

`-T`*device*
为设备 *device* 进行格式化输出。

`-W` 使用宽行的 Versatec 打印机而不是窄行的 Varian 打印机。（这些打印机曾经在美国伯克利的加州大学使用，该选项在 Solaris 中可能不起作用。）

vi

`vi [`*options*`] [`*files*`]`

基于 `ex` 的一个面向屏幕的文本编辑器。更多的信息可以参见第八章和第九章的 `vi` 和 `ex`。`-c`、`-C`、`-L`、`-r`、`-R` 和 `-t` 等选项与 `ex` 中的这些选项作用相同。

options

`-c` *command*
进入 `vi` 并运行给定的 `vi` 命令 *command*。

`-l` 运行于 LISP 模式中以编辑 LISP 程序。

`-L` 列出由于一个编辑器或系统崩溃而保存的文件名。

`-r` *file*
在一个编辑器或系统崩溃后恢复并编辑文件 *file*。

`-R` 只读模式，不能修改文件。

`-S` 与 `-t` 一起使用，用来表示标签文件可能没有进行排序及使用一个线性搜索。只用于 Solaris。

`-t` *tag*
编辑包含标签 *tag* 的文件，并在其定义处放置编辑器（更多的信息参见 **ctags**）。

`-w`*n* 设置默认窗口大小为 *n*，当通过一个速度较慢的拨号线路进行编辑时比较有用。

vi

`-x` 提供一个密钥来对使用了 `crypt` 的文件 *file* 进行加密或解密。（注意，使用 `ps` 命令提供的密钥对其他用户是可见的。）

`-C` 与 `-x` 相似，但是假设文件 *file* 以加密的形式开始。

`+` 在文件的最后一行启动 `vi`。

`+n` 在文件的第 *n* 行启动 `vi`。

`+/`*pat*
在包含模式 *pat* 的行启动 `vi`。如果在你的 `.exrc` 文件中设置了 `nowrapscan`，则该选项不起作用。

view

`view [`*options*`] [`*files*`]`

与 `vi -R` 作用相同。

volcheck

`volcheck [`*options*`] [`*pathnames*`]`

只用于 Solaris。检查 *pathnames* 指定的一个或多个设备，以便查看可移动的媒体是否已经插好。默认情况下会检查正被卷管理程序管理的所有设备。大多数情况下用于软盘；当 CD-ROM 已经插好后，卷管理程序通常会注意到。

注意：不建议将 `-i` 和 `-t` 选项用于软驱，特别是间隔比较短时。

options

`-i` *nsec*
每隔 *nsec* 秒检查设备，默认值是间隔两秒。

`-t` *nsecs*
持续检查到下一个 *nsecs* 秒。*nsecs* 的最大值是 28000 秒（8 个小时）。

`-v` 详细信息。

w

`w [`*options*`] [`*user*`]`

输出系统的使用情况、当前登录的用户和用户进行的操作等方面的信息汇总。从本质上讲，`w` 结合了 `uptime`、`who` 和 `ps -a` 命令的功能。可以通过指定 *user* 参数来指定显示一个用户的输出。

w

options

`-h` 禁止输出标题和 `uptime` 信息。

`-l` 使用长格式进行显示（默认情况）。

`-s` 使用短格式进行显示。

`-u` 只输出标题行，等同于 `uptime`。只用于 Solaris。

`-w` 与 `-l` 作用相同。只用于 Solaris。

wait

`wait [`*`n`*`]`

等待所有后台进程的完成并报告它们的终止状态。主要用于 shell 脚本中。如果指定了参数 *n*，则只等待进程 ID 为 *n* 的进程。`wait` 是 Bourne、Korn 和 C shells 中的内部命令，更多的信息可参见第四章和第五章。

wc

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

计算单词数。显示 *files* 中的字符、单词及行的数量。如果是多个文件，则输出汇总信息。如果没有给出 *files* 参数，则从标准输入中读取数据。其他示例可参见 **ls** 和 **sort**。

options

`-c` 只输出字节的数量。

`-C` 只输出字符的数量。在一个多字节的环境中与 `-c` 是不同的。只用于 Solaris。

`-l` 只输出行的数量。

`-m` 与 `-C` 作用相同。只用于 Solaris。

`-w` 只输出单词的数量。

示例

计算登录用户的数量：

```
who | wc -l
```

计算在三个 essay 文件中单词的数量：

```
wc -w essay.[123]
```

wc

计算 `$file` 指定的文件中的行数（不显示文件名）：

```
wc -l < $file
```

what

`/usr/ccs/bin/what [`*option*`] `*files*

一条 SCCS 命令。参见第十八章。

whatis

`whatis `*commands*

在联机帮助中查找一个或多个命令 *commands*，并显示简要的描述。与 `man -f` 作用相同。MANPATH 环境变量可以影响使用该命令获得的结果。参见 **apropos**。

which

`which [`*commands*`]`

如果指定的参数 *commands* 当作一条命令运行，则列出正在运行的文件。`which` 会读取用户的 `.cshrc` 文件（使用 `source` 内置命令），检查别名并搜索 `path` 变量。Bourne 或 Korn shells 用户可以使用内部的 `type` 命令来替代它（**type** 命令可以参考第四章和第五章）。

示例

```
$   which file ls
/usr/bin/file
ls: aliased to ls -sFC
```

who

`who [`*options*`] [`*file*`]`

显示系统的当前状态信息。不使用选项时，只列出当前登录到系统中的用户的名字。可以提供一个可选的系统文件 *file*（默认是 `/var/adm/utmp`）来给出一些附加信息。通常调用 `who` 时不使用选项。但是比较有用的选项是 `am i` 和 `-u`。更多的例子，可参见 **cut**、**line**、**paste**、**tee** 和 **wc**。

options

`-a` 使用 `-b`、`-d`、`-l`、`-p`、`-r`、`-t`、`-T` 和 `-u` 选项。

`-b` 报告最后一次重新启动的信息。

`-d` 报告到期的进程。

who

`-H` 输出标题。

`-l` 报告停止的终端行。

`-m` 只报告当前的终端。只能用于 Solaris。

`-n` *x*
每行显示 *x* 个用户（只与 `-q` 一起使用）。

`-p` 报告以前产生的进程。

`-q` 即 “ quick ”，只显示用户名。

`-r` 报告运行等级。

`-s` 列出名称、行和时间字段（默认行为）。

`-t` 报告系统时钟上次进行的修改（通过 `date`）。

`-T` 报告终端是可写的（`+`）、不可写的（`-`）或未知的。

`-u` 报告终端的使用情况（空闲时间）。一个小点（`.`）表示空闲时间少于 1 分钟，old 表示多于 24 小时的空闲。

`am i`
输出正在调用用户的用户名（与 `id` 的执行结果相似）。

示例

该示例的输出是在 4 月 17 日上午 8 点产生的：

```
$ who -uH
NAME    LINE   TIME         IDLE  PID  COMMENTS
martha  ttyp3  Apr 16 08:14 16:25 2240
george  ttyp0  Apr 17 07:33   .  15182
```

由于从昨天下午（16 点）开始 `martha` 已经空闲，所以看来 `martha` 还没有开始运行。她只是简单地保留自己的登录。George 的终端当前正在使用（他喜欢让机器不停地运转）。

whoami

`/usr/ucb/whoami`

输出基于有效用户 ID 的用户名。该命令通常与标准 SVR4 中的命令 `logname` 作用相同。但是，当你作为另一个用户运行一个 `su` 会话时，`whoami` 会显示该用户的名字，而 `logname` 仍会显示你的名字。

xargs

xargs [*options*] [*command*]

执行命令*command*（可使用任何初始参数），但是从标准输入中读取剩余的参数，而不是直接指定这些参数。xargs会将参数打成几个包后传递给*command*，并允许*command*处理比它立刻打包的参数更多的参数。这些参数通常是多个文件名组成的很长的列表（例如，通常由ls或find产生），该列表经过一个管道传递给xargs。

如果没有*command*参数，则xargs实现的功能与echo相似，只是简单地将输入行打包进输出行中，并在标准输出中输出它们。

options

-e[*string*]
当遇到参数*string*（默认是下划线）时，停止传递参数。省略*string*会禁止逻辑的EOF功能。

-E *string*
使用*string*而不是下划线作为默认的逻辑EOF字符串。只用于Solaris。

-i[*string*]
向*command*中传递参数，使用当前输入的行来替换命令行中的{}实例。在Solaris中，选项*string*可用来代替{}。

-I *string*
与-i作用相同，但是会使用*string*代替{}。

-l[*n*]
使用*n*行的参数来执行命令。对于Solaris，如果提供了-l，则默认的*n*是1。

-L *n*
与-l*n*作用相同。只用于Solaris。

-n*n* 使用最多*n*个参数来执行命令*command*。

-p 提示输入一个y来确认命令*command*的每次执行。隐含-t选项。

-s*n* 各个参数列表最多可包含*n*个字符。（以前的系统中*n*最大为470。默认值与使用的系统有关。）

-t 在执行前回显各个*command*。

xargs

`-x` 如果参数列表超过了*n*个字符（来自`-s`），则退出。`-x`与`-i`和`-l`一起使用时，会自动起作用。

示例

在系统的所有文件上使用`grep`的*pattern*模式：

```
find / -print | xargs grep pattern  > out &
```

在文件对（比如，`f1.a`和`f1.b`、`f2.a`和`f2.b` ...）上运行`diff`：

```
echo $* | xargs -n2 diff
```

前面的行可以作为一个shell脚本调用，并指定文件名作为参数。

显示*file*，每行一个单词（与`deroff -w`相似）：

```
cat file  |  xargs -n1
```

将`olddir`中的文件拷贝到`newdir`中，并显示各条命令：

```
ls olddir | xargs -i -t mv olddir/{} newdir/{}
```

xgettext

`xgettext [`*options*`] `*files*
`xgettext -h`

只用于Solaris。从C和C++源文件中提取信息（特别标记的字符串），并将它们放置在一个“可移植的目标”文件（*file*`.po`）中，以便可以通过`msgfmt`进行转换及编译。默认情况下，`xgettext`只提取对*gettext*(3C)和*dgettext*(3C)函数的调用内部的字符串。源文件在命令行指定。名字为 - 的文件表示标准输入。

options

`-a` 提取所有的字符串，而不只是对`gettext`或`dgettext`函数的调用内部的字符串。

`-c` *tag*
将使用*tag*标记的源文件注释拷贝到`.po`文件中作为以#分界的注释。

`-d` *domain*
使用*domain*`.po`而不是`messages.po`作为输出文件。

xgettext

`-h` 在标准输出中输出一条帮助信息。

`-j` 将提取的信息与当前 `.po` 文件中的信息进行连接（合并）。已经存在的 `.po` 文件中的域指令会被忽略。

`-m` *prefix*
使用前缀 *prefix* 填充各个 `msgstr`。主要用于调试。

`-M` *suffix*
使用后缀 *suffix* 填充各个 `msgstr`。主要用于调试。

`-n` 将注释添加到 `.po` 文件中，用以指示使用了各个字符串的源文件名和行号。

`-p` *path*
将输出文件放到目录 *path* 中。

`-s` 按照 `msgid`（最初的字符串）对输出进行排序，并删除所有重复的内容。

`-x` *exfile*
exfile 是一个包含不被提取（即排除）的 `msgid` 的 `.po` 文件。

yacc

`/usr/ccs/bin/yacc [`*options*`]` *file*

给出包含了对上下文自由的LALR(1)语法的文件 *file*，将其转换为多个表格，以便以后进行解析并将输出发送到 `y.tab.c` 中。该命令名也代表另一个编译器到编译器。参见参考文献中列出的 **lex** 和 *lex & yacc*。

options

`-b` *prefix*
对产生的文件名使用前缀 *prefix* 代替 `y`。只用于 Solaris。

`-d` 生成 `y.tab.h`，并产生将 `yacc` 的标记代码与用户声明的标记名称相关联的 `#define` 语句。

`-l` 排除 `y.tab.c` 产生的代码中的 `#line` 结构（在调试完成后使用）。

`-p` *prefix*
使用 *prefix* 代替 `yy` 用于生成的解析器中的所有外部名称。只用于 Solaris。

yacc

-P *parser*
使用 *parser* 代替 /usr/ccs/bin/yaccpar。只用于 Solaris。

-Q*c* 将 yacc 的版本信息放到 y.tab.c 中（如果 *c* = y），或者禁止输出信息（如果 *c* = n，默认设置）。

-t 默认情况下编译运行时调试代码。

-v 生成文件y.output, 这是一个包含了诊断信息和解析表注意事项的文件。

-V 在标准错误中输出 yacc 的版本。

zcat

zcat [*files*]

将一个或多个 .Z 文件解压缩到标准输出中，并且不修改 *files*。参见 **compress**。

zip

zip [*options*] *zipfile* [*files*]

使用InfoZIP格式对文件进行归档。这些文件可以使用unzip进行恢复。在将这些文件添加到档案中时，对它们进行压缩。一般文本文件的压缩比为 2:1 或 3:1。zip 也会替换档案中已经存在的文件。不使用参数时，会显示帮助信息。参见 **zipinfo** 和 **unzip**。

默认选项可以放置在ZIPOPT环境变量中，此时不能使用 -i 和 -x 选项。ZIPOPT 中可以包括多个选项。

SVR4 或 Solaris 的发布版中没有 zip 命令，但是可以很容易地从下面的站点中获得其源代码：*http://www.cdrom.com/pub/infozip*。

在**unzip**的说明内容中有多个重要的注意事项，要了解更多的信息可以访问该站点。

options

-b *path*
当更新一个已经存在的ZIP档案时，使用*path*作为保存临时ZIP档案的目录。操作完毕后，将临时档案拷贝到新的档案中。如果在包含原始档案的文件系统中没有足够的磁盘空间用于存放新的 ZIP 档案，则应使用该选项。

zip

`-c` 在每个文件中添加一行注释。`zip` 首先执行任何文件操作，然后提示你为各个文件添加一行注释信息。

`-d` 从一个 ZIP 档案中删除一些项。如果档案是在一个 MS-DOS 系统中使用 PKZIP 建立的，则要删除的文件名必须使用大写字母表示。

`-D` 不会在档案中为各个目录建立对应项。通常情况下会建立与目录对应的项，以便目录的属性在解压缩时能够得到恢复。

`-e` 对档案进行加密。`zip` 会在终端上提示用户输入一个密码，必须连续输入两次，以防止输入错误。如果标准错误不是一个终端，则 `zip` 会退出并产生一个错误。

`-f` 如果一个文件的修改时间比档案中该文件的修改时间新，则会刷新（或替换）ZIP 档案中已经存在的项。但该选项不会向档案中添加新的文件，可以使用 `-u` 来添加新的文件。请在产生 ZIP 档案的目录中运行该命令，因为档案中只存储相对路径名。

`-F, -FF`
修正 ZIP 档案。使用该选项应小心，应首先对原档案进行备份。`-FF` 选项不相信档案中压缩的大小，它会使用特殊“ 签名 ”对档案进行扫描，以便确定不同档案成员的边界。更多的信息可以查看参考页。

`-g` 使档案增大（向档案中添加新的文件）。

`-h` 显示 `zip` 命令的帮助信息。

`-i` *`files`*
只包括 *files* 指定的文件，通常使用引号包括的 shell 通配符风格的模式来指定文件。

`-j` “ 清理 ”路径，即只存储保存的文件的名字，而不是存储任何路径名。默认情况是存储完整的路径，尽管通常使用相对路径。

`-J` 从档案中删除从前面添加的数据（比如，一个 SFX 存根，用于自解压的可执行文件）。

`-k` 建立（试图）遵循 MS-DOS 下的惯例的一个档案。这使 PKUNZIP 可以更容易地对档案进行解压缩。

zip

`-l` 只用于文本文件，将Unix换行符转换成一个CR-LF对。主要用于在MS-DOS下对档案进行解压缩。

`-ll` 只用于文本文件，将MS-DOS CR-LF转换成Unix换行符。

`-L` 显示`zip`许可证。

`-m` 将文件“移动”到ZIP档案中。实际上在成功地建立了档案后，会删除原来的文件及（或）目录。这样做多少有些危险，该选项应与`-T`一起使用。

`-n` *`suffixlist`*
不压缩具有*suffixlist*列出的后缀的文件。主要用于带有自己特定压缩方法的声音或图像文件。

`-o` 将ZIP档案的修改时间设置为该档案中最近刚修改的文件的时间。

`-q` 安静模式。不输出报告的消息和注释提示。主要用于shell脚本。

`-r` 递归地对指定的*files*中的所有文件和子目录进行归档。`-i`选项与该选项一起使用时比较有用。

`-t` *`mmddyy`*
忽略修改时间比*mmddyy*给定的日期更早的文件。

`-T` 测试新的ZIP档案的完整性。如果测试失败，则不会修改一个已经存在的ZIP档案，并且与`-m`一起使用时不会删除任何文件。

`-u` 如果指定*files*的修改日期比档案中这些文件的修改日期更新，则会更新ZIP档案中已经存在的文件。与`-f`选项相似，但是如果一些文件没有在档案中，则该选项会将这些文件添加到档案中。

`-v` 作为惟一参数，输出帮助和版本信息，提供一个指向主目录和分布式Internet站点的指针，并显示如何编译`zip`的信息。当与其他选项一起使用时，该选项会显示进度信息并提供其他的诊断信息。

`-x` *`files`*
排除指定的文件*files*，通常被指定成一个引用的shell通配符风格的模式。

zip

`-X` 不保存额外的文件属性（OS/2 上的扩展属性、用户 ID/组 ID 和 Unix 上的文件时间）。

`-y` 在 ZIP 档案中保存动态链接，而不是归档链接指向的文件。

`-z` 提示输入一个（可能是多行）描述整个 ZIP 档案的注释，并使用只包含了一个句点或 *EOF* 的行来结束注释。

`-`*n* 指定压缩速度：*n* 是介于 0~9 之间的一个数字，0 表示不进行压缩，1 表示进行快速但是最小程度的压缩，9 表示压缩速度最慢而压缩程度最高。默认值是 `-6`。

`-@` 从标准输入中读取文件的名字进行归档。文件名中包含的空格必须使用单引号包括。

示例

将当前的目录归档到 `source.zip` 文件中，只包括 C 源文件：

```
zip source -i '*.[ch]'
```

将当前的目录归档到 `source.zip` 文件中，排除目标文件：

```
zip source -x '*.o'
```

将当前目录中的文件归档到`source.zip`文件中，但是不压缩 `.tiff` 和 `.snd` 文件：

```
zip source -z '.tiff:.snd' *
```

递归将整个目录树压缩到一个档案中：

```
zip -r /tmp/dist.zip .
```

zipinfo

`zipinfo [`*`options`*`] `*`zipfile`*` ... [`*`exclusion option`*`]`

只用于 Solaris。`zipinfo` 会输出有关 ZIP 格式档案的信息。*zipfile* 是一个文件名以 `.zip` 结束的 ZIP 档案。如果档案来自命令行，则可以省略 `.zip`，`zipinfo` 会提供该扩展名。*zipfile* 也可以是一个 shell 风格的通配符模式（应该使用引号将其包括，以便与shell命令区分开），它将对 ZIP 档案中的所有匹配文件起作用。参见 **zip** 和 **unzip**。

zipinfo

exclusion option（排除的选项）

`-x` *`files`*

排除选项。不会对匹配 *files* 的档案成员进行解压缩。

options

`-1` 只列出文件名，每个文件名占一行。不输出其他内容。主要用于 shell 脚本。

`-2` 与 `-1` 作用相似，但是也允许显示文件标题、文件结尾及 ZIP 档案注释（`-h`、`-t`、`-z`）。

`-h` 输出一个标题行，包括档案名称、以字节表示的大小及文件的总数。

`-l` 使用“长”格式。与 `-m` 相似，但是还会输出以字节表示的压缩文件的大小，而不是压缩比。

`-m` 使用“中等”格式，与 `-s` 相似，但是也包括压缩因子（按照百分比计算）。

`-M` 通过内部的页面调度程序进行管道输出，与 `more` 相似。在出现 `--More--` 提示符时按回车键或空格键可以显示下一屏。

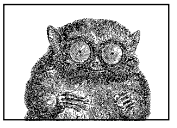
`-s` 使用“短”格式，与 `ls -l` 相似。这是默认选项。

`-t` 输出所有文件的总计信息（文件的数量、压缩及解压缩的大小、整体压缩因子）。

`-T` 按照可以排序的十进制格式（*yymmdd.hhmmss*）输出时间和日期。

`-v` 使用详细的、多页格式。

`-z` 输出档案注释。



第三章

Unix shell 概述

本章为初学者介绍了与 shell（命令解释器）有关的基本概念，同时为高级用户阐述了 Bourne shell、Korn shell 以及 C shell 的共同点及不同之处。在第四章和第五章中介绍了这三种 shell。

以下为本章关于 Unix shell 的内容

- 简述 shell
- shell 的用途
- shell 的特色
- 几种 shell 的共同功能
- 几种 shell 的区别之处

简述 shell

为了便于理解，我们可以把 Unix 操作系统比做一辆车。在驾驶的过程中，我们可能会向车发出各种各样的“ 命令 ”，让车按我们的意图来工作，比如转方向盘让车改变方向，踩油门使车加速，踩制动刹车等。但是车到底是怎样将驾驶员的命令转换为动作呢？车的驱动机制担负了该项任务，它是车与驾驶员之间联系的纽带。车可能会配备有前轮驱动、后轮驱动或者是四轮驱动，有时还可能是三种驱动的组合。

shell 是到 Unix 的用户接口。通过相同的标记，Unix 中可以有多个 shell。大多数系

统提供了一个以上的 shell 供选择。每种 shell 都有各自不同的功能，但每种 shell 都担负着命令解释以及提供工具来创建用户 Unix 环境的任务。

shell 是一个程序，通过它系统会明白用户所发出的命令（这也正是 shell 经常被称为命令解释器的原因）。对于大多数用户来说，shell 是在幕后工作的。人们并不在乎计算机是怎样工作的，只是关心计算机是否按着我们的意思去做了。就像上面所打的关于车的比方，对于大多数人来说，在踩制动刹车的时候，并不关心所驾驶的车采用的是盘式刹车器还是鼓式刹车器，只要车停住就行了。

shell 的用途

shell 有三种用途：

- 交互式使用
- 定制 Unix 会话
- 编程

交互式使用

shell 的交互式使用，就是系统等待用户在 Unix 提示符下键入一个命令。当然命令中可以包括一些特殊的符号，允许简化文件名或对输入、输出进行重定向。

定制 Unix 会话

Unix shell 定义某些变量来控制 Unix 会话的行为。例如，设定一些变量来告诉系统用哪一个目录作为用户的主目录或在哪个文件中存储邮件。一些变量是由系统预置的，其他的可以在登录时读取的启动文件中定义。启动文件中可以有 Unix 命令或特定的 shell 命令。在用户登录时，这些命令就会执行。

编程

shell 提供了一系列特定（或内置）的命令，用户可以用这些命令来编程，这类程序称为 shell 脚本（shell script）。实际上，很多内置命令可以像 Unix 命令那样交互使用，在 shell 脚本中也经常用到 Unix 命令。在执行一连串的单个命令时，脚本是非常有用的。它类似于 MS-DOS 中的批处理文件。脚本可以像在很多高级编程语言中一样重复地执行命令（在一个循环中）或者有条件地执行命令（在 `if-else` 中）。

shell 的特色

用户可能用到各种各样的 Unix shell，在这里我们阐述三种最常用的 shell。

- Bourne（或者说标准）shell 这是一种最简练也是最简单的 shell。
- Korn shell 这是 Bourne shell 的一个超集，允许用户编辑命令行。事实上，它有两个常用的版本，通常根据其发布的年份来区别。本书中所提到的分别是 ksh88 和 ksh93。
- C shell 其语法类似于 C 语言，对于交互式用户来说，它比 Bourne shell 更方便。

大多数系统不只有一个 shell，人们经常用 Bourne shell 来编写 shell 脚本，而另一个 shell 用于交互式使用。

在用户的交互式 Unix 会话期间，由 etc/passwd 文件来决定哪个 shell 生效。用户登录时，系统会检查 etc/passwd 文件中的条目。每个条目的最后一个字段指定了一个运行的程序，并作为默认的 shell（注 1）。例如：

程序名	用户的 shell
/bin/sh	Bourne shell
/bin/rsh	受限的 Bourne shell
/bin/jsh	Bourne shell, 包括作业控制
/bin/ksh	Korn shell
/usr/dt/bin/dtksh	Desktop Korn shell, 即 ksh93 的一个版本（只用于 Solaris）
/bin/rksh	受限的 Korn shell
/bin/csh	C shell

可以通过在命令行键入那个程序名来转换到另一个 shell 中。例如，为了从 Bourne shell 转换到 Korn shell，可以键入：

```
$ exec ksh
```

注意：在大多数系统中，rsh 是“远程（remote）shell”，用于通过网络在一个远程

注 1： 在 Solaris 或其他的网络 Unix 系统中，此类信息可能会从 NIS 或 NIS+ 获得。一般来说，系统管理员会为你处理这些事情，这个时候你所登录的程序名可能并没有出现在 */etc/passwd* 中。

系统上执行命令。在某些系统中，rsh 是指“受限的（restricted）shell”，remsh 是“远程 shell”。可以从本地文档中获得该类信息。

选择哪个 shell

如果你是 Unix 系统的新手，挑选哪个 shell 可能会成问题。在 ksh 普及以前，建议将 csh 用于交互式使用（因为它支持作业控制，还具备其他一些功能，这使它比 Bourne shell 更适合作为交互式 shell），而用 Bourne shell 来编写脚本（因为它是一个更有力的编程语言，使用更普遍）。

现在，ksh 应用更广泛，作为一个编程语言它向上和 Bourne shell 兼容，又具有 csh 所有的交互能力，并且功能更强大。如果可以获得，它是最好的选择。

几种 shell 的共同功能

下表列出了 Bourn shell、Korn shell 和 C shell 共同的功能。应该注意的是：Korn shell 是 Bourne shell 的增强版，所以 Korn shell 不仅包括了 Bourne shell 的所有功能，而且还具备更多的特征，像命令 bg、fg、jobs、stop 和 suspend，都只用于支持作业控制的系统中（这是现代 Unix 系统的本质特征）。

符号 / 命令	意义 / 行为
>	输出重定向
>>	向文件中追加
<	输入重定向
<<	“Here”文档（输入重定向）
|	管道输出
|&	开始协同进程，只用于 Korn shell 中
&	在后台运行进程
;	在同一行中分割命令
*	匹配文件名中的任何字符
?	匹配文件名中的单个字符
[]	匹配括号中的任何字符
()	在子 shell 中执行命令
` `	用引号内命令的输出替换命令本身
" "	部分引用（允许变量和命令扩展）

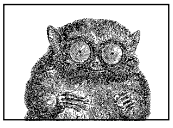
符号 / 命令	意义 / 行为
`' '`	全部引用（不进行扩展）
`\`	引用后面的字符
`$`*`var`*	使用变量值
`$$`	进程 ID
`$0`	命令名称
`$`*`n`*	第 *n* 个参数($0 \leq n \leq 9$)
`$*`	作为单个的词返回所有的参数
`#`	开始注释
`bg`	后台执行
`break`	中断循环语句
`cd`	改变目录
`continue`	继续执行循环程序
`echo`	显示输出
`eval`	计算参数
`exec`	执行一个新的 shell
`fg`	前台执行
`jobs`	显示当前活动作业
`kill`	中断运行作业
`shift`	移动位置参数
`stop`	挂起一个后台作业
`suspend`	挂起一个前台作业（如一个由 su 创建的 shell）
`time`	对命令进行计时
`umask`	为新文件设置默认的安全许可
`unset`	删除变量或函数的定义
`wait`	等待后台作业执行完毕

几种 shell 的区别之处

下表列出了三种 shell 的不同之处。

sh	ksh	csh	意义 / 行为
$	$	%	提示符
	>|	>!	强制重定向
		>>!	强制添加
>*file*2>&1	>*file*2>&1	>&*file*	将标准输出与标准错误结合
		{}	扩展列表内的元素
``	``	``	替代引起来的命令的输出
	$()		替代引起来的命令的输出（首选形式）
$HOME	$HOME	$home	主目录
	~	~	主目录符号
var=value	*var=value*	set *var=value*	变量赋值
export *var*	export *var=val*	setenv *var val*	设置环境变量
	${*nn*}		可以引用超过 9 个以上的参数
"$@"	"$@"		所有的参数都作为单独的单词
$#	$#	$#argv	参数的个数
$?	$?	$status	退出状态
$!	$!		最后的后台进程 ID
$-	$-		当前的选项
. *file*	. *file*	source *file*	读文件 *file* 中的命令
	alias *x=y*	alias *x y*	用 *x* 代替 *y*
case	case	switch/case	选择其中的一个
	cd ~-	popd/pushd	切换目录
done	done	end	结束一个循环语句
esac	esac	endsw	结束 case 或 switch 语句
exit [*n*]	exit [*n*]	exit [(*expr*)]	退出时返回一个状态
for/do	for/do	foreach	通过变量来循环
	print -r	glob	忽略 echo 转义
hash	alias -t	hashstat	显示散列的命令（跟踪别名）

sh	ksh	csh	意义 / 行为
`hash cmds`	`alias -t cmds`	`rehash`	记住命令的位置
`hash -r`	`PATH=$PATH`	`unhash`	忘记命令的位置
	`history`	`history`	列出执行过的命令
	`r`	`!!`	重新执行已执行过的命令
	`r` *`str`*	`!`*`str`*	重新执行以 *str* 开始的命令
	`r` *`x=y[cmd]`*	`!`*`cmd:s/x/y/`*	编辑命令然后执行
`if [ $i -eq 5]`	`if ((i==5))`	`if ($i==5)`	`if` 语句示例
`fi`	`fi`	`endif`	结束 `if` 语句
`ulimit`	`ulimit`	`limit`	设置资源限制
`pwd`	`pwd`	`dirs`	输出工作目录
`read`	`read`	`$<`	读取标准输入
`trap2`	`trap2`	`onintr`	忽略中断
	`unalias`	`unalias`	删除别名
`until/do`	`until/do`		开始 `until` 循环
`while/do`	`while/do`	`while`	开始 `while` 循环



第四章

Bourne shell 和 Korn shell

本章将介绍以下内容：

- 功能概述
- 语法
- 变量
- 运算表达式（只在 Korn shell 中）
- 命令历史（只在 Korn shell 中）
- 作业控制
- 调用 shell
- 受限的 shell
- 内置命令

站点 *http://www.kornshell.com* 提供了很多有关 Korn shell 的信息。如果出于非商业或仅仅是教育的目的，还可从该链接下载ksh93的代码。也可以看一下参考文献中的《Learning the Korn shell》一书。

功能概述

Bourne shell 是标准的 shell，它具备以下功能：

- 输入 / 输出重定向
- 文件名缩写用的通配符（元字符 ）
- 定制环境的 shell 变量
- 写 shell 程序用的内置命令集
- 作业控制（开始于 SVR4）

Korn shell 是向后兼容的，它是 Bourne shell 的扩充。以下功能只在 Korn shell 中有效：

- 命令行编辑（用 vi 或 emacs 编辑器的命令语法）
- 访问以前的命令（命令历史）
- 整数运算
- 更多的方式去匹配模式和替换变量
- 数组和运算表达式
- 命令名缩写（别名）
- 更多的内置命令

`ksh93` 又增加了以下内容：

- 向上兼容 POSIX
- 国际化工具
- `for` 循环运算
- 浮点运算和内置运算函数
- 结构变量名和间接变量引用
- 关联数组
- 更多的办法去匹配模式和代替变量
- 更多的内置命令

语法

本部分介绍了特定于 Bourne shell 和 Korn shell 的符号。安排了以下标题：

- 特殊的文件

- 文件名元字符
- 引用
- 命令形式
- 重定向形式
- 协同进程（只用于 Korn shell）

特殊的文件

`/etc/profile`
在登录时首先自动执行。

`$HOME/.profile`
在登录时第二个自动执行。

`$ENV`
在创建一个新的 Korn shell 时指定要读的一个文件的名称（`ksh88:`所有 shell。`ksh93:`只用于交互式 shell）。该值是被替换的变量（`ksh93:`命令和运算），用于确定实际的文件名。在处理完 `/etc/profile` 和 `$HOME/.profile` 之后登录 shell 会读取 $ENV。

`/etc/passwd`
~*name*缩写的主目录的来源（在网络系统中，此信息可能来源于NIS或NIS+，而不是用户工作站的密码文件）。

文件名元字符

`*` 匹配有零或零个以上字符的字符串。

`?` 匹配任何单个字符。

`[abc...]` 匹配括号内任何一个字符，也可用连字符指定一个范围（例如，a-z, A-Z, 0-9）。

`[!abc...]` 匹配任何不包括在括号内的字符。

在 Korn shell 中：

`?(`*pattern*`)` 匹配模式 *pattern* 的零个或一个实例。

`*(`*pattern*`)` 匹配指定模式 *pattern* 的零个或多个实例。

+(*pattern*) 匹配指定模式 *pattern* 的一个或多个实例。

@(*pattern*) 只匹配指定模式 *pattern* 的一个实例。

!(*pattern*) 匹配任何不匹配 *pattern* 的字符串。

n 匹配与（...）中的第 *n* 个子模式匹配的文本。只用于 ksh93 中。

~ 当前用户的主目录。

~name 用户 *name* 的主目录。

~+ 当前工作目录（$PWD）。

~- 先前工作目录（$OLDPWD）。

这个模式 *pattern* 可能是由分隔符“ | ”分隔的模式的序列，这意味着该匹配可应用于任何的模式。如果分隔符是 & 而不是 |，那么所有的模式必须匹配。比较而言，分隔符 & 比 | 有更高的优先级。这种扩展语法类似 egrep 和 awk 中的语法。

在 ksh93 中支持 POSIX [[=*c*=]]符号，它用于匹配有同样权重的字符，并且[[.c.]]用于指定排序序列。相应地，形式为[[:*clss*:]]的字符类，允许匹配以下字符类。

类	匹配的字符
alnum	字母数字字符
alpha	字母字符
blank	空格或制表符
cntrl	控制字符
digit	带小数的数字
graph	非空格字符
lower	小写字符
print	可打印字符
space	空白字符
upper	大写字符
xdigit	十六进制数

示例

```
$ ls new*                    列出如 new 和 new.1 的文件
$ cat ch?                    匹配如 ch9 但不匹配 ch10
$ vi [D-R]*                  匹配以大写 D 到 R 为开头的文件
$  pr !(*.o|core) | lp       只用于 Korn shell；输出不是目标文件或核心转储的文件
```

引用

引用会使字符失去其特殊的意义，按照其本身的字面意义来使用。下表列出了在 Bourne shell 和 Korn shell 中有特殊意义的字符。

字符	意义
`;`	命令分隔符
`&`	后台执行
`()`	命令分组
`|`	管道
`<> &`	重定向符号
`* ? [ ] ~ + - @ !`	文件名元字符
`"'\`	用于引用其他字符
`` ` ``	命令的替换
`$`	变量的替换（或是命令或运算的替换）
`space tab newline`	单词的分隔符

下列的字符用于引用：

`" "` 介于 " 和 " 之间的任何字符都被逐字采用，下面包含了特殊意义的字符除外。

- `$` 会发生变量（或 Korn shell 命令和运算）替换。
- `` ` `` 会发生命令的替换
- `"` 该标记表示双引号的结束

`' '` 除了另一个 ' 符号之外的介于 ' 和 ' 之间的任何字符，都被逐字采用。在这个被引用的字符串内不允许嵌入另一个 '。

`\` 在 \ 后的字符会按其原来意义逐字采用。用于 `""` 内对符号 "、`$` 和 `` ` `` 进行转义，并经常用于转义本身、空格或换行符。

`$" "`
只用于 `ksh93`，除了进行场所转换外，其用法类似 " "。

`$' '`
用于 `ksh93`，其用法类似于 `''`，不过被括起来的文本会为以下转义序列做出处理。

序列	值	序列	值
\a	警告	*nnn*	八进制值 *nnn*
\b	退格	\x*nn*	十六进制值 *nn*
\f	换页	\'	单引号
\n	换行	\"	双引号
\r	回车	\\	反斜线符号
\	制表符	\E	Escape 符号
\v	垂直制表符		

示例

```
$   echo 'Single quotes "protect" double quotes'
Single quotes "protect" double quotes
$   echo "Well,isn't that \"special\"?"
Well, isn't that "special"?
$   echo "You have `ls | wc -l` files in `pwd`"
You have 43 files in /home/bob
$   echo "The value of \$x is $x"
The value of $x is 100
```

命令形式

cmd &	在后台执行 *cmd*。
cmd1;*cmd2*	命令序列；在同一行中执行多个命令。
{*cmd1*;*cmd2*;}	在当前 shell 中，将命令做为一组来执行。
(*cmd1*;*cmd2*)	在子 shell 中，将命令做为一组来执行。
cmd1|*cmd2*	管道；将 *cmd1* 的输出作为 *cmd2* 的输入。
cmd1`cmd2`	命令替换；用 *cmd2* 的输出作为 *cmd1* 的参数。
cmd1$(*cmd2*)	Korn shell 命令替换；允许嵌套。
cmd$((*expression*))	Korn shell运算替换。用表达式*expression*的结果作为*cmd*的参数。
cmd1&&*cmd2*	逻辑与；执行 *cmd1*，如果 *cmd1* 成功执行，再执行 *cmd2*。这是一种“短路”操作；如果 *cmd1* 没有成功执行，那么不会执行 *cmd2*。
cmd1||*cmd2*	逻辑或；执行 *cmd1* 或（如果 *cmd1* 没有成功执行）*cmd2*。

这是一种“短路”操作；如果 *cmd1* 成功执行，那么不会执行 *cmd2*。

示例

```
$  nroff file > file.txt &                  在后台执行格式化
$  cd; ls                                   顺序地执行命令
$  (date; who; pwd)> logfile                重定向所有输出
$  sort file | pr -3 | lp                   将 file 排序，分页输出，然后打印
$  vi `grep -l ifdef *.c `                  编辑由 grep 找到的文件
$  egrep '(yes|no)'`cat list`               指定要搜索的文件列表
$  egrep '(yes|no)' $(cat list)             早期的 Korn shell 版本
$  egrep '(yes|no)' $(<list)                意义相同，但运行速度更快
$  grep XX file && lp file                  如果文件包含了该模式，则打印它
$  grep XX file || echo "XX not found"      如果文件没有包含该模式，则显示一个错误信息
```

重定向形式

文件描述符	名称	通常的缩写词	默认的设备
0	标准输入	stdin	键盘
1	标准输出	stdout	终端
2	标准错误	stderr	终端

通常，输入源和输出目标可以改变，正如我们在下面看到的。

简单的定向

`cmd > file`

将 *cmd* 的结果输出到 *file*（覆盖）。

`cmd >> file`

将 *cmd* 的结果输出到 *file*（添加）。

`cmd < file`

从 *file* 中获取 *cmd* 的输入。

`Cmd << text`

将 shell 脚本的内容（直到遇到一个和 *text* 相同的行为止）作为 *cmd* 的输入（*text* 也可能存储在 shell 变量中）。有时称这种命令格式为 Here document（Here 文档）。输入的内容通常用键盘键入或在 shell 程序中输入。使用这种语法的典型命令有 `cat`、`ex` 和 `sed`。（如果使用 `<<-`，当将输入的内容和文本结束标记 *text* 进行比较的时候，开头的制表符将不予考虑）。如果将 *text* 引起来，则输入的内容会逐字

通过；否则，其内容会作为变量和命令替换进行处理。在Korn shell中，也会对运算替换的输入给予处理。

Cmd<>*file*

只用于Korn shell中。在标准输入上打开*file*以便读写，而不会破坏它的内容（注1）。

用文件描述符重定向

cmd >&*n*	将*cmd*输出发送到文件描述符*n*。
cmd m>&*n*	和前面的命令相同，只是本来正常输出到文件描述符*m*的内容，转而发送到文件描述符*n*中。
cmd >&-	关闭标准输出。
cmd <&*n*	从文件描述符*n*中获取*cmd*的输入。
cmd m<&*n*	与上面的相同，只是来自文件描述符*m*的输入被来自文件描述符*n*的输入取代。
cmd <&-	关闭标准输入。
cmd <&*n*-	移动输入文件描述符*n*而不是复制它。只用于ksh93中。
cmd >&*n*-	移动输出到文件描述符*n*而不是复制它。只用于ksh93中。

多重重定向

cmd 2>*file*	将标准错误发送到*file*中，标准输出保持不变（例如，屏幕）。
cmd > *file* 2>&1	将标准错误和标准输出发送到文件*file*中。
cmd > *f1* 2>*f2*	将标准输出发送到*f1*，将标准错误发送到*f2*。
cmd | tee *files*	将*cmd*的输出发送到标准输出（一般是终端）和文件*files*中（可以参照第二章中**tee**下的示例）。
cmd 2>&1 | tee *files*	将标准输出和*cmd*的错误输出发送到标准输出（通常是终端）和文件*files*中。

在文件描述符和一个重定向符号间不允许有空格；其他情况下空格是可选的。

注1： 使用<时，打开的文件是只读的，这时写文件描述符会出错。如果使用<>，文件打开后可读写，非常适合应用于应用程序中。

示例

```
$  cat part1 > book
$  cat part2 part3 >> book
$  mail tim < report
$ sed 's/^/XX /g' << END_ARCHIVE
>  This is often how a shell archive is "wrapped",
>  bundling text for distribution. You would normally
>  run sed from a shell program, not from the command line.
> END_ARCHIVE
XX This is often how a shell archive is "wrapped",
XX bundling text for distribution. You would normally
XX run sed from a shell program, not from the command line.
```

为了将标准输出重定向到标准错误中，可执行如下命令：

```
$  echo "Usage error: see administrator" 1>&2
```

以下命令会将输出（找到的文件）发送到`filelist`，并且将错误消息（不可访问的文件）发送到文件`no_access`中：

```
$  find / -print > filelist 2>no_access
```


协同进程

协同进程是 Korn shell 才具备的功能。

cmd1`|` *cmd2*`|&`	协同进程；在后台执行管线功能，shell 会建立两路管道，以允许标准输入、标准输出的重定向。
`read -p` *var*	读取协同进程到变量 *var* 中。
`print -p` *string*	将字符串 *string* 写到协同进程中。
cmd `<&p`	*cmd* 从协同进程中接收输入。
`cmd >&p`	将 *cmd* 的输出发送到协同进程中。
`exec` *n*`<&p`	将来自协同进程的输入转移到文件描述符 *n*。
`exec` *n*`>&p`	将到协同进程的输出转移到文件描述符 *n*。

将协同进程的输入输出文件描述符转移到标准的文件描述符，就可以打开多个协同进程。

示例

```
$  ed - memo |&                开始协同进程
```

```
$   print -p /word/          发送 ed 命令给协同进程
$   read -p search           将 ed 命令的输出发送给变量 search
$   print "$search"          在标准输出上显示行
A word to the wise.
```

变量

本部分将介绍以下内容：

- 变量替换
- shell 内置变量
- 其他 shell 变量
- 数组（只用于 Korn shell）
- 规范函数（只用于 ksh93）

变量替换

ksh93 提供了结构变量，比如 pos.x 和 pos.y。要想创建任意一个变量，pos 必须已经存在，并且必须用大括号来获取它们的值。以 .sh 开头的名称只保留给 ksh 使用。

下列表达式中不允许使用空格。冒号（:）是可选的；如果包括了冒号，那么变量 *var* 必须是非空的并设置了初始值。

var=value ...	将变量 *var* 设置为 *value*。
${*var*}	使用变量 *var* 的值；如果变量 *var* 和其后面的文本是分开的，那么大括号是可选的。在 ksh93 中，如果变量名包含了句点，则它是必需的。
${*var*:-*value*}	如果变量 *var* 已设置则使用它，否则使用值 *value*。
${*var*:=*value*}	如果变量 *var* 已设置则使用它，否则使用值 *value* 并且将 *value* 赋给变量 *var*。
${*var*:?*value*}	如果变量 *var* 已设置则使用它，否则输出 *value* 并且退出（如果是非交互式的）；如果没有提供 *value*，则输出短语“parameter null or not set（参数为空或未设置）”。
${*var*:+*value*}	如果变量 *var* 已设置则使用 *value*，否则什么也不使用。

在 Korn shell 中：

${#*var*}	使用变量 *var* 的长度。
${#*}	使用位置参数的个数。
${#@}	使用位置参数的个数。
${*var*#*pattern*}	从变量的左端移走匹配模式之后返回*var*的值，移走的是最短的匹配块。
${*var*##*pattern*}	与 *#pattern* 一样，只不过移走的是最长的匹配块。
${var%pattern}	从右端移走匹配模式后返回 *var* 的值。移走的是最短的匹配块。
${*var*%%*pattern*}	与 *%pattern* 一样，只不过移走的是最长的匹配块。

在 ksh93 中：

${!*prefix**} ${!*prefix*@}	变量名称以 *prefix* 开头的变量列表。
${*var*:*pos*}	在变量 *var* 中，从位置 *pos*（以 0 为基准）提取 *len* 长的字符。
${*var*:*pos*:*len*}	如果没有 *len*，则提取剩余的字符串，其中 *pos* 和 *len* 都可能是运算表达式。
${*var*/*pat*/*repl*}	使用变量 *var* 的值，其中和 *pat* 匹配的第一部分用 *repl* 代替。
${*var*/*pat*}	使用变量 *var* 的值，其中和 *pat* 匹配的第一部分被删除。
${*var*//*pat*/*repl*}	使用变量*var*的值，其中和*pat*匹配的每一部分都用*repl*代替。
${*var*/#*pat*/*repl*}	使用变量 *var* 的值，其中和 *pat* 匹配的部分用 *repl* 代替。该匹配操作必须在值的开头执行。
${*var*/%*pat*/*repl*}	使用变量 *var* 的值，其中和 *pat* 匹配的部分用 *repl* 代替。该匹配的操作必须在值的结尾执行。

在 ksh93 中，间接变量可以允许你给变量取一个“别名”，从而可以影响到另一个的值。这是通过 typeset -n 来完成的：

```
$   greet="hello, world"                    创建初始变量
$   typeset -n friendly_message=greet       建立别名
$   echo $friendly_message                  用新名称访问以前的值
hello, world
$   friendly_message="don't panic"          改变它的值
```

```
$   echo $greet                                  以前的值已改变
don't panic
```

示例

```
$   u=up d=down blank=          将值赋予三个变量 u、d 和 blank（最后一个值是 null）
$   echo ${u}root               在这里大括号是必需的
uproot
$   echo ${u-$d}                显示变量 u 或 d 的值，因为变量 u 的值已设置，故将输出其值
up
$   echo ${tmp-`date`}          如果没有设置 tmp 的值，则会执行命令 date
Thu Feb 4 15:03:46 EST 1993
$   echo ${blank="no data"}     设置 blank 变量，因此将输出它的值（一个空行）
$   echo ${blank:="no data"}    设置 blank 变量但值为 null，因此将输出字符串
no data
$   echo $blank                 现在变量 blank 已有了一个新的值
no data
```

Korn shell 中的示例

```
tail=`${PWD##*/}`           取得当前的目录名并且从中移走以 / 为结尾的最长的字符串，
                            即移走前面的路径名只剩下结尾部分
```

shell 内置变量

内置变量由 shell 自动设置，并且通常用于 shell 脚本中。正如前面所介绍的，内置变量可能会利用变量替换模式。注意虽然总是使用 `$`，但实际上它并不是变量名的一部分。

`$#` 命令行参数的个数。

`$-` 当前有效的选项（提供给 `sh` 或 `set` 的参数）。

`$?` 上一条命令执行后返回的值。

`$$` 当前进程的进程号。

`$!` 上一个子进程的进程号。

`$0` 第一个单词，也就是命令名，如果是通过 PATH 搜索到的，这将是一个完整的路径名。

`$`*n* 命令行上的单个参数（位置参数）。在 Bourne shell 中，只允许直接引用 9 个参数（*n* = 1~9）；而在 Korn shell 中，如果指定了 `${n}`，则允许 *n* 的值比 9 更大。

`$*, $@` 命令行上所有的参数（`$1 $2 ...`）。

`"$*"` 将命令行上所有的参数作为一个字符串（`"$1 $2..."`）。

`"$@"` 命令行上所有的参数，但是每个参数都被引号引起来（`"$1" "$2"...`）。

Korn shell 会自动设置以下的变量：

`$_` 临时变量，被初始化到正在执行的脚本或程序。随后，存储上一个命令的最后的参数值。在邮件校验期间也存储匹配MAIL文件的名称。

`LINENO` 脚本或函数内正在执行的命令的行号。

`OLDPWD` 前一个工作目录（由 `cd` 设置）。

`OPTARG` 由 `getopts` 处理的最后选项的名称 。

`OPTIND` `OPTARG` 的数字索引。

`PPID` 当前 shell 的父进程的进程号。

`PWD` 当前工作目录（由 `cd` 设置）。

`RANDOM[=`*n*`]` 每次引用时产生一个随机数，如果给定 *n* 则以整数 *n* 开始。

`REPLY` 默认的答复，由 `select` 和 `read` 使用。

`SECONDS[=`*n*`]` 从shell开始启动后的秒数，或者是，如果给定*n*的值，则为shell开始启动后的秒数 +*n*。

`ksh93`，自动设置了以下的附加变量。如果变量名包括“ . ”，则引用时必须用大括号括起来。例如：`${.sh.edchar}`。

`HISTCMD` 当前命令的历史序号。

`.sh.edchar` 在处理 `KEYBD` 陷阱时键入的字符。改变它来代替导致陷阱的字符。

`.sh.edcol` 在最近的 `KEYBD` 陷阱中光标的位置。

`.sh.edmode` 如果在`vi`模式下，在`KEYBD`陷阱中，它等同于ESCAPE，否则为空。

`.sh.edtext` 在 `KEYBD` 陷阱期间在输入缓冲区中的字符。

`.sh.name` 正在运行的一个规范函数的变量的名称。

`.sh.subscript` 正在运行的一个规范函数的数组变量的下标。

`.sh.value` 在规范函数 `set` 和 `get` 里的变量的值。

`.sh.version` `ksh93` 的版本。

其他 shell 变量

shell 不会自动设置下列变量。这些变量通常用在 `.profile` 文件中，可以定义它们以满足需要。可以通过下列特定的形式给变量赋值：

```
variable=value
```

下面的列表包括了在定义变量时会用到的值类型。那些特定于 Korn shell 的，通常标记其为（K）；而特定于 `ksh93` 的，则标记为（K93）。

`CDPATH=`*`dirs`*
执行 `cd` 命令时搜索的目录。从而可以更快捷地改变目录。默认情况下不设置。

`COLUMNS=`*`n`*
（K）屏幕的列宽，用于行编辑模式和 `select` 列表。

`EDITOR=`*`file`*
（K）打开行编辑模式的路径名（路径名可以以 `emacs` 或 `vi` 结束），用于没有设置 VISUAL 的情况下。

`ENV=`*`file`*
（K）在 shell 启动时执行的脚本的名称，主要用于存储别名和函数定义，例如，`ENV=$HOME/.kshrc`（与C shell 中的 `.cshrc` 一样）。

`FCEDIT=`*`file`*
（K）命令 `fc` 所使用的文本编辑器（默认值是 `/bin/ed`）。在 `ksh93` 中由 HISTEDIT 代替。

`FIGNORE=`*`pattern`*
（K93）用于描述模式匹配期间忽略的一组文件名的模式。

`FPATH=`*`dirs`*
（K）搜索函数定义的目录，没有定义的函数通过 `typeset -fu` 设置。当第一次用到这些函数时，会搜索 FPATH（在 `ksh93` 中也会搜索 PATH）。

`HISTEDIT=`*`file`*
（K93）由 `hist` 命令使用的编辑器。覆盖 FCEDIT 的设置。

`HISTFILE=`*`file`*
（K）存储命令历史的文件（必须在 `ksh` 启动前设置），默认值是 `$HOME/.sh_history`。

`HISTSIZE=`*`n`*
（K）可利用的历史命令的数目（必须在 `ksh` 启动前设置），默认值是 128。

`HOME=`*`dir`*
主目录，由 `login` 定义（来自 `/etc/passwd` 文件）。

`IFS='`*`chars`*`'`
输入字段分隔符，默认值是空格、制表符和换行符。

`LANG=`*`dir`*
某些语言相关的程序使用的目录。

LC_ALL=*locale*　（K93）当前场所，覆盖 LANG 和其他的 LC_* 变量。

LC_COLLATE=*locale*　（K93）使用字符整理（排序）的场所。

LC_CTYPE=*locale*　（K93）使用字符类函数的场所（参照上面提到的“ 文件名元字符 ”）。

LC_NUMERIC=*locale*　（K93）使用小数点字符的场所。

LINES=*n*　（K）屏幕高度，用于 select 列表。

MAIL=*file*　用于接收邮件的默认文件，由 login 设置。

MAILCHECK=*n*　两次检查邮件之间的秒数，默认值为 600（10 分钟）。

MAILPATH=*files*　用于接收邮件并用冒号分隔的一个或多个文件，伴随着每个文件，当文件尺寸增加时，你可以提供 shell 输出的可选消息，消息和文件名由分隔符隔开，在 Korn shell 中，分隔符是?，并且默认消息是 You have mail in $_，用文件的名称代替 $_。在 Bourne shell 中，分隔符是 %，并且默认消息是 You have mail。例如，在 ksh 中，你可以输入：

```
MAILPATH="$MAIL?Ring! Candygram!:/etc/motd?New Login Message"
```

PATH=*dirlist*　执行命令时需搜索的一个或多个路径名，中间由冒号隔开。在 SVR4 中，默认值是 /bin:/usr/bin；在 Solaris，默认值是 /usr/bin:。而在标准的启动脚本中将它改为：

```
/usr/bin:/usr/ucb:/etc:
```

在 ksh93 中，对未定义函数的函数定义也会搜索 PATH。

PS1=*string*　主提示字符串，默认为 $。

PS2=*string*　二级提示符（用于多行命令），默认值是 >。

PS3=string　（K）用于 select 循环中的提示字符串，默认值是 #?。

PS4=*string*　（K）用于执行跟踪命令时的提示字符串（ksh-x 或 set -x），默认值是 +。

SHACCT=*file*　即“ shell 账户 ”。执行 shell 脚本的日志文件。在 Korn shell 中不使用。

SHELL=*file*　默认的 shell 名称（例如，/bin/sh）。

TERM=*string*	终端类型。
TMOUT=*n*	（K）如果在 *n* 秒后仍没有键入命令，则退出 shell。
VISUAL=*path*	（K）同 EDITOR，但会首先检查 VISUAL。

数组

Korn shell 支持一维数组，最多可以支持 1024 个元素。第一个元素为 0。可以用下面的方式对数组 *name* 进行初始化：

```
set -A name value0 value1 ...
```

那些指定的值就成为 *name* 的元素。然而声明数组并不是必需的。对具有下标变量的任何有效引用都会创建一个数组。

在访问数组时用 ${...} 语法。当访问在 (()) (执行自动引用的 let 命令的方式）内的数组时，它并不是必需的。注意 [和] 必须按字面键入（即它们不代表可选的语法）。

${*name*[i]}	使用数组 *name* 的元素 *i*。*i* 可能是在 let 下描述的任何运算表达式。表达式必须返回 0 和 1023 之间的数。
${*name*}	使用数组 *name* 的元素 0。
${*name*[*]} ${*name*[@]}	使用数组 *name* 的所有元素。
${#*name*[*]} ${#*name*[@]}	使用数组 *name* 中的元素个数。

Ksh93 提供了关联数组，这时候，下标是字符串而不是数字。在这种情况下，[和] 就起到了双引号的作用。可用 typeset -A 来创建关联数组。下面这个特殊的语法允许对数组的多个元素赋值：

```
data=（[joe]=30 [mary]=25）
```

可以用 ${data[joe]} 和 ${data[mary]} 的方式提取变量的值。

规范函数（只用于 ksh93）

和结构变量一起，k93 也引入了规范函数（discipline function）。无论何时访问或改变一个变量的值，都会调用这些特定的函数。对于一个名称为 x 的 shell 变量来说，可以定义下列函数：

`x.get` 当提取 x 的值时（`$x`）调用。

`x.set` 当 x 的值改变时（`x=2`）调用。

`x.unset` 当撤消 x 时（`unset x`）调用。

在规范函数中，一些特殊的变量提供那些进行改变的变量的信息：

`.sh.name` 进行改变的变量的名称。

`.sh.subscript` 进行改变的数组元素的下标。

`.sh.value` 进行赋值或返回的变量的值。在规范函数内改变该值，会改变实际赋予或返回的值。

运算表达式

Korn shell 的 `let` 命令执行运算操作。`ksh88` 限制在只允许进行整数运算的范围内。`ksh93` 也能进行浮点运算。Korn shell 提供了一个取代运算值的方式（用作命令参数或用于变量中），也可以进行基数的转换：

`$(( expr ))` 使用由括号包括的运算表达式的值。

B#n 用数字基数 *B* 来解释整数 *n*。例如，`8#100` 指定十进制数 64 的八进制等价值。

运算符

Korn shell 使用了 C 编程语言中的运算符。下表中的运算符以优先级递减的顺序排列。

运算符	描述
`++ --`	自动递增和自动递减，既可作前缀也可作后缀。只用于 `ksh93` 中
`+`	一元加法，只用于 `ksh93` 中
`-`	一元减法
`! ~`	逻辑非，二进制求反（补码）
`* / %`	乘法；除法；求余（余数）
`+ -`	加法；减法
`<< >>`	位左移；位右移
`<= >=`	小于或等于；大于或等于
`< >`	小于；大于

运算符	描述
== !=	相等；不等（左边和右边比较）
&	按位与
^	按位异或
|	按位或
&&	逻辑与（短路操作）
||	逻辑或（短路操作）
?:	内联条件计算。只用于 ksh93 中
*= /= %= = += -= <<= >>= &=^=|=	赋值
,	连续表达式计算。只用于 ksh93 中

内置运算函数（只用于 ksh93 中）

ksh93 提供了对标准运算函数集的访问。用 C 函数的语法去调用它们。

函数名称	意义	函数名称	意义
abs	绝对值	log	自然对数
acos	反余弦	sin	正弦
asin	反正弦	sinh	双曲正弦
cos	余弦	sqrt	平方根
cosh	双曲余弦	tan	正切
exp	指数函数	tanh	双曲正切
int	浮点数的整数部分		

示例

查阅 let 命令，可以获得更多的信息和示例。

```
let "count=0" "i = i + 1"                    对 i 和 count 赋值
let "num % 2"                                检测偶数
(( percent >= 0 && percent <= 100 ))         检测一个值的范围
```

命令历史

在 Korn shell 中，可以显示和修改用过的命令。可以用下面的办法来修改历史列表中的命令。

- 行编辑模式
- fc 和 hist 命令

行编辑模式

行编辑模式模拟vi和emascs编辑器的许多功能，以文件的方式来处理历史列表。调用编辑器的时候，可以键入编辑键转移到想要执行的命令行。在执行命令之前也可以改变那些命令。当准备好执行一条命令时，按回车键。

可以按几种方式来启动行编辑模式。例如，下面这几种方式是等价的：

```
$ VISUAL=vi
$ EDITOR=vi
$  set -o vi                    覆盖 VISUAL 或 EDITOR 的值
```

应注意的是，vi 是以输入模式启动的，要键入一个 vi 命令，首先要按 Escape 键。

常见编辑按键

vi 编辑器	emacs 编辑器	结果
k	CTRL-p	得到上一条命令
j	CTRL-n	得到下一条命令
/*string*	CTRL-r *string*	得到上一条包含 *string* 的命令
h	CTRL-b	往回移动一个字符
l	CTRL-f	向前移动一个字符
b	ESC-b	向后移动一个单词
w	ESC-f	向前移动一个单词
X	DEL	删除上一个字符
x	CTRL-d	删除光标当前所在的字符
dw	ESC-d	删除上一个单词
db	ESC-h	删除后一个单词
xp	CTRL-t	调换两个字符

fc 和 hist 命令

用 fc -l可以列出历史命令，并且可以用 fc -e去编辑它们。查阅“内置命令”可以得到更多的信息。

在 ksh93 中，fc 命令改名为 hist，并且预先定义了别名 fc=hist。

示例

```
$  history              列出最后用过的 16 条命令
$  fc -l 20 30          列出 20 到 30 条命令
$  fc -l -5             列出最后用过的 5 条命令
$  fc -l cat            列出最后用过的以 cat 开头的命令之后的所有命令
$  fc -ln 5 > doit      将第 5 条命令存储到文件 doit 中
$  fc -e vi 5 20        用 vi 编辑第 5 条到第 20 条的命令
$  fc -e emacs          用 emacs 编辑上一条命令
$ r                     重新执行上一条命令
$  r cat                重新执行最后的 cat 命令
$  r doc=Doc            取代后重新执行最后的命令
$  r chap=doc c         重新执行以 c 开头的最后的命令，但要将字符串 chap 改为 doc
```

作业控制

通过作业控制，可以将前台作业转到后台，也可以将后台作业转到前台执行，或挂起（暂时停止）正在运行的作业。用下列任何命令都可以启用作业控制：

```
jsh -i          用于 Bourne shell 中

ksh -m -i       用于 Korn shell 中（和下面两条命令等同）
set -m
set -o monitor
```

许多作业控制命令使用 *jobID* 作为参数。可以用下列方式指定该参数：

`%`*n*　作业号 *n*。

`%`*s*　其命令行以字符串 *s* 开头的作业。

`%?`*s*　其命令行包含字符串 *s* 的作业。

`%%`　当前作业。

`%+`　当前作业（同上）。

`%-`　前一作业。

Bourne shell和Korn shell提供了以下的作业控制命令。要想获得有关这些命令的更详细的信息，可以查阅后面章节中的“内置命令”部分。

`bg` 将一个作业置于后台运行。

`fg` 将一个作业置于前台运行。

`jobs`
列出当前活动的作业。

`kill`
终止一个作业。

`stop`
挂起一个后台作业。

`stty tostop`
如果后台作业试图发送输出到终端，则停止它。（注意`stty`不是一个内置命令。）

`suspend`
挂起一个作业控制 shell（例如由 `su` 命令创建的）。

`wait`
等待后台作业完成。

`CTRL-Z`
挂起一个前台作业。然后使用`bg`或`fg`命令。（用户终端可以使用除了`CTRL-Z`之外的东西作为挂起字符。）

调用 shell

Bourne shell 的命令解释器（`sh`）和 Korn shell 的命令解释器（`ksh`），可以用下列方式调用：

```
sh [options] [arguments]
ksh [options] [arguments]
```

`ksh`和`sh`可以在终端上执行命令，也可以从一个文件（当第一个*argument*是一个可执行的脚本时）或从标准输入（如果没有保留参数或者指定为`-s`）。如果标准输入是一个终端，或在命令行上给定了`-i`，`ksh`和`sh`会自动输出提示符。

arguments
指定的参数对应着位置参数`$1`、`$2`等。如果数组赋值是有效的（`-A`或`+A`），则参数被指定为数组元素。如果第一个参数是一个可执行的脚本，则会从中读命令，并且其余的参数会指定给`$1`、`$2`等。

options
`-c` *`str`*
从字符串 *str* 中读命令。

`-D` 输出程序里的所有 `$"..."` 字符串。只用于 `ksh93` 中。

`-i` 创建一个交互式的 shell（提示输入）。

`-I` *`file`*
为变量和命令的定义和引用建立一个交叉引用数据库。不可以编译。只用于 `ksh93` 中。

`-p` 作为一个特权用户启动（Bourne shell：不将有效用户和组 ID 设置为实际用户和组的 ID。Korn shell：不处理 `$HOME/.profile`）。

`-r` 创建一个受限的 shell（和 `rksh` 或 `rsh` 相同）。

`-s` 从标准输入中读命令，内置命令输出到文件描述符 1；所有其他的 shell 输出到文件描述符 2 中。

其余的有关 `sh` 和 `ksh` 的选项列入 `set` 内置命令中。

受限的 shell

受限的 shell 可以用以下方式调用：

```
rksh                    Korn shell
ksh -r
set -r

/usr/lib/rsh            Bourne shell
set -r
```

也可以通过给`/etc/passwd`的shell字段提供一个完整路径名，或把它们作为SHELL变量的值，来建立受限的 shell。

除了以下功能受限外，受限的 shell 和非受限的 shell 的其他功能是一样的：

- 改变目录（例如用 `cd` 命令）。
- 设置 PATH 变量。`rksh` 也禁止设置 ENV 和 SHELL。
- 为命令名或路径名指定 `/`。
- 重定向输出（例如用 `>` 和 `>>`）。`ksh` 也禁止用 `<>`。
- 增加新的内置命令（只用于 `ksh93` 中）。

由于受限的 shell 调用 `ksh` 或 `sh` 来运行脚本，因此 shell 脚本仍旧可以运行。这包括 `/etc/profile`、`$HOME/.profile` 和 $ENV 文件。

实际上由于正确地建立受限 shell 比较困难，所以一般很少使用受限的 shell。

内置命令（Bourne shell 和 Korn shell）

命令在键入时必须和 $ 提示符在一起，否则，系统会把它们当作 shell 脚本中的代码段来处理。为了方便起见，还包含了多行命令使用的保留字。

!	! *pipeline*
	
	只用于 ksh93。不要认为是一个管线。如果管线退出时为非零值，则返回的退出状态值为 0；否则返回值为 1。通常用于 if 和 while 语句中。
	
	示例
	如果用户 jane 没有登录到系统上，这段代码会输出一条消息：
	
	`if ! who | grep jane > /dev/null`
	`then`
	`  echo jane is not currently logged on`
	`fi`

#	# 忽略同一行中其后的所有文本。# 在 shell 脚本中只是作为一个注释字符而不是一个真正的命令使用。（应注意，当解释一段Bourne shell 脚本时，一些旧的系统会将第一个字符为 # 的文件当作 C shell 脚本来解释。）
#!*shell*	#!*shell*[*option*] 当作脚本的第一行调用指定的 shell。行的其余部分都被当作单个参数传递给 shell。这个功能通常由内核来实现，但一些旧的系统不一定支持该功能。有些系统限制 shell 的最大长度为 32 个字符。例如： `#!/bin/sh`

:	:
	
	空命令，返回一个退出状态0。有时用于一个旧的系统中，作为文件的第一个字符表示Bourne shell脚本。可以参照示例和**case**命令。像变量和命令替换一样，该命令仍旧作为一个有副作用的部分来处理。
	
	示例
	检查一个用户是否登录到系统上：
	
	`if who | grep $1 > /dev/null`
	`then :  # Do nothing if pattern is found`
	`else echo "User $1 is not logged in"`
	`fi`

.	. *file*[*arguments*] 在文件*file*中读取并且执行命令行。文件并不一定是可执行的，但必须存在于PATH搜索的目录中。Korn shell支持存储于位置参数中的*arguments*。
[[ ]]	[[*expression*]] 只用于Korn shell中，和test *expression*或[*expression*]意义相同，只是在[[]]中允许有附加的运算符。注意括号（[]）必须按字面键入，并且必须用空白区域包围它。 附加的运算符 && 测试表达式中的逻辑与（短路操作）。 || 测试表达式中的逻辑或（短路操作）。 < 第一个字符串按字面意义比第二个小。 > 第一个字符串按字面意义比第二个字符串大。
name()	*name*(){*command*;} 将*name*定义为一个函数。语法可以写在一行或多行上。因为Bourne shell不具备定义别名的能力，因此简单的函数可以充当别名的脚色。Korn shell提供了function关键字，可以得到同样的效果。

name()

应该记住下列语义区别：

- 在 Bourne shell 中，所有的函数都和父 shell 共享陷阱，并且是不能递归的。
- 在 ksh88 中，所有的函数都有它自己的陷阱和局部变量，是可以递归的。
- 在 ksh93 中，*name*()函数和父 shell 共享陷阱，是不可以递归的。
- 在 ksh93 中，function 函数有其自己的陷阱和局部变量，是可以递归的。用 . 命令和 function 函数一起，可赋予它 Bourne shell 语义。

示例

```
$  count(){
>   ls | wc -l
> }
```

当在命令行上执行该示例时，count 会显示当前目录下文件的数量。

alias

alias [*options*] [*name*[='*cmd*']]

只用于 Korn shell 中。给 *cmd* 分配一个具有同样意义的速记名称 *name*。如果省略了 ='*cmd*'，则会输出 *name* 的别名；如果也省略了 *name*，则会输出所有的别名。如果 alias 值的后面有空格，则下一个单词也会变成候选的别名扩展。可参见 **unalias** 命令。

在 ksh88 中设定了下面的别名。有一些使用了已经存在的 Bourne shell 或 C shell 命令的名称（指出了三种 shell 的相似之处）。

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

alias

在 ksh93 中定义了下列别名：

```
autoload='typeset -fu'
command='command '
fc='hist'
float='typeset -E'
functions='typeset -f'
hash='alias -t --'
history='hist -l'
integer='typeset -i'
nameref='typeset -n'
nohup='nohup '
r='hist -s'
redirect='command exec'
stop='kill -s STOP'
times='{ {time;} 2>&1;}'
type='whence -v'
```

options

`-p` 在每个别名前输出单词 `alias`。只用于 `ksh93` 中。

`-t` 为一个 Unix 命令 *name* 创建一个跟踪别名。Korn shell 会记住那个命令的完整路径，从而可以快速地找到并从任何目录执行它，如果没有提供任何名字，就会列出当前的跟踪别名。跟踪别名和 Bourne shell 中的散列命令类似。

`-x` 输出别名，可以用在 shell 脚本和其他的子 shell 中。如果没有提供 *name*，则列出当前输出的别名。

示例

```
alias dir='echo ${PWD##/*}'
```

autoload

`autoload[`*`functions`*`]`

只在第一次执行时装载（定义）*functions*。在 Korn shell 中是 `typeset -fu` 的别名。

bg

`bg[`*`jobIDs`*`]`

将当前的作业或 *jobIDs* 置于后台。可以查阅前面的“作业控制”一节。

break

```
break[n]
```

从for while、select、until循环中退出（或从*n*次循环中退出）。

builtin

```
builtin [-ds] [-f library] [name ...]
```

只用于 ksh93 中。可以使用该命令，在运行时从共享库文件中将新的内置命令装载到 shell 中。

如果没有指定参数，builtin会输出所有的内置命令名称；如果带参数，builtin会将每个*name*作为新的内置命令增加（像cd或pwd一样）；如果*name*中包含了一个斜线，只有路径搜索没有找到一个同样名字的命令时，才会使用那个新增加的内置版本（这样可以用更快的内置版本代替系统命令）。否则，总是会找到那个内置命令。

options

-d 删除内置命令*name*。

-f 从*library*中装载新的内置命令。

-s 只输出特别的内置命令（那些由 POSIX 指定为特别的命令）。

case

```
case value in
   pattern1) cmds1;;
   pattern2) cmds2;;
      .
      .
      .
esac
```

如果*value*匹配*pattern1*，则执行第一组命令（*cmds1*）；如果*value*匹配*pattern2*，则执行第二组命令；依次类推。必须保证每一组命令集的结尾都要有;;。*value*一般是一个位置参数或其他的shell变量。*cmds*通常是Unix命令、shell编程命令或变量赋值。模式可以用文件生成的元字符。在同一行上可指定多个模式（由|分隔开），此时，只要*value*匹配这些模式中的任何一个命令，那个相关的*cmds*就会执行。可以参照本命令的示例或命令eval。

case

Korn shell 注意事项

- 在 Korn shell 中，可以像(*pattern*)一样，用一个可选的左括号放在 *pattern* 之前。这样，在一个 $( ) 结构中可起到括号平衡作用。
- 在 Korn shell 中，可以允许一个 case 的结尾用 ;& 来代替 ;;。在这种情况下控制对下一组 *pattern* 失效。

示例

检查第一个命令行参数并采取合适的行动：

```
case $1 in       # 匹配第一个参数
   no|yes) response=1;;
   -[tT]) table=TRUE;;
   *)     echo "unknown option"; exit 1;;
esac
```

读取用户提供的行，直到用户退出为止：

```
while :          # Null 命令；总是真值
do
   echo "Type . to finish ==> \c"
   read line
   case "$line" in
      .) echo "Message done"
         break ;;
      *) echo "$line" >> $message ;;
   esac
done
```

cd

cd [*dir*]
cd [-LP] [*dir*]
cd [-LP] [-]
cd [-LP] [*old new*]

如果没有参数，则将目录改变成用户的主目录；否则，将工作目录改变成 *dir*。如果 *dir* 是一个相对的路径名而不在当前目录中，则搜索 CDPATH 变量。最后的三种命令方式特定于 Korn shell，- 代表上一个目录。第四个语法用于修改当前目录名称，即用 *new* 代替 *old*，然后切换到结果目录中。

cd

options

`-L` 对于 `cd ..` 和 PWD 的值使用逻辑路径（用户键入的，包括任何符号链接）。这是默认值。

`-P` 对 `cd ..` 和 PWD 的值用实际的文件系统物理路径。

示例

```
$ pwd
/var/spool/cron
$  cd cron uucp                cd 输出那个新目录
/var/spool/uucp
```

command

`command [pvV] name[arg ...]`

只用于`ksh93`中。如果没有`-v`或`-V`，则用给定的参数执行*name*。这个命令回避了为*name*定义的任何别名或函数。

options

`-p` 用一个预先定义的默认搜索路径而不是 PATH 的当前值。

`-v` 与`whence`一样。

`-V` 与`whence -v`一样 。

示例

为`rm`创建一个可以取得系统版本的别名，并用`-i`选项来运行它：

```
alias 'rm=command -p rm -i'
```

continue

`continue[n]`

在`for`、`while`、`select`或`until`循中跳过余下的命令，继续执行循环的下一个遍历（或跳过*n*次循环）。

disown

`disown[job...]`

只用于`ksh93`中。当一个登录 shell 退出时，不向给定的作业发送`SIGHUP`。如果没有列出作业，那么没有后台作业会接收`SIGHUP`。

do

do

一个`for`、`while`、`until`或`select`语句的命令序列中最前面的保留字。

done

done

结束一个 for、while、until 或 select 语句的保留字。

echo

echo [-n] [*string*]

将字符串 *string* 写到标准输出中。如果指定了 -n，输出不会被换行符中断；如果没有提供 *string*，则回显一个换行符。在 Korn shell 中，它是内置命令，并且模拟系统的 echo 命令（注 2）（可以参照第二章的 echo 命令）。echo 能够识别特殊的转义字符，但必须引起来（或用 \ 转义）以防止 shell 进行解释。

options

\a 报警（ASCII BEL）（不是在 /bin/sh 的 echo 中）。

\b 退格。

\c 禁止换行（和 -n 一样）。

\f 换页。

\n 换行。

\r 回车。

\t 制表符。

\v 垂直制表符。

\\ 反斜线。

\0*nnn*

由八进制数字 *nnn* 表示的 ASCII 字符，*nnn* 是一个由 0 开头的一位、两位或三位数字。

示例

```
$   echo "testing printer" | lp
$   echo "Warning: ringing bell \a"
```

注 2： 如果一次路径搜索就找到了 /usr/bin/echo，那么 ksh 的内置命令 echo 不会接受 -n 选项， echo 的这种情形是容易混乱的，建议用 printf 代替。

esac

esac

用于结束 case 语句的保留字。省略 esac 会产生常见的编程错误。

eval

eval *args*

通常，eval用在 shell 脚本中，并且 *args* 是一行包含了 shell 变量的代码。eval强迫首先进行变量扩展，并且运行由此产生的命令。这种“ 两次搜索 ”在 shell 变量包括了输入 / 输出重定向符号、别名或其他 shell 变量时是有用的。（例如，正常情况下，重定向在变量扩展之前发生，因此一个包含了重定向符的变量必须首先用 eval扩展，否则，那个重定向符号会无法解释。）可查阅 C shell 中的另一个 eval 的示例（见第五章）。

示例

这段 Bourne shell 脚本显示了 eval 怎样构造一个以正确顺序解释的命令：

```
for option
do
   case "$option" in    定义输出到哪里
   save) out=' > $newfile' ;;
   show) out=' | more' ;;
esac
done

eval sort $file $out
```

exec

exec [*command args* ...]
exec [-a *name*] [-c] [*command args* ...]

执行命令代替当前的进程（而不是创建一个新的进程）。exec对于打开、关闭或拷贝文件描述符也是有用的。第二种形式只用于 ksh93 中。

options

-a 用 *name* 作为 argv[0]的值。

-c 在执行程序前清除环境。

exec

示例

```
trap ,exec 2>&-, 0     当 shell 脚本退出（信号 0）时，关闭标准错误

$ exec /bin/csh          用 C shell 代替 Bourne shell
$   exec < infile        重新将标准输入分配给 infile
```

exit

exit[*n*]

退出一个 shell 脚本，退出状态为 *n*（例如，exit 1）。*n* 可能是 0（成功）或非 0 值（失败）。如果没有给出 *n*，则退出状态为最近的命令的退出状态。可以在命令行中执行 exit 以关闭一个窗口（退出）。退出状态在 0~255 的范围内。

示例

```
if [ $# -eq 0 ]
then
   echo "Usage: $0 [-c] [-d] file（s）" 1>&2
   exit 1          # 错误状态
fi
```

export

export [*variables*]
export [*name*=[*value*] ...]
export -p

传递（输出）一个或更多 shell 变量的值，对那些变量（默认是局部的）赋予全局意义。例如，如果一个 shell 脚本定义的变量的值被另一个 shell 脚本调用，那么必须输出这个变量。如果没有给出 *variables*，那么 export 会列出由当前 shell 输出的变量。第二种形式只用于 Korn shell 脚本中，和第一种情况相似，不同的是，用户在输出前可以给要输出的变量 *name* 赋值 *value*。第三种形式只用于 ksh93 中。

options

-p 在输出变量的名字和值之前输出 export。这样可以存储输出变量的列表以备以后读取。

示例

在 Bourne shell 中，用户可以键入：

export

```
TERM=vt100
export TERM
```

在 Korn shell 中，用户也可以用下面语句代替：

```
export TERM=vt100
```

false

`false`

在 `ksh88` 中是 `let 0` 的别名。在 `ksh93` 中，是一个内置命令，退出时返回 false 值。

fc

`fc[`*options*`][`*first*`[`*last*`]]`
`fc -3 -[`*old=new*`] [`*command*`]`

只用于 `ksh88` 中。显示或编辑在历史表中的命令（只能用 `-1` 或 `-e` 中的一个）。其中参数 *first* 和 *last* 是指定要显示或编辑的命令的数字或字符串。如果省略 *last*，`fc` 只应用于单条命令（由 *first* 指定）；如果参数 *first* 和 *last* 都省略，`fc` 会编辑上一条命令或最后 16 条命令的列表。`fc` 命令的第二种用法是对于一个历史命令 *command*，用新的字符串 *new* 代替旧的字符串 *old* 并执行修改后的命令。如果没有指明字符串，则会重新执行命令 *command* ；如果也没有指明命令 *comm-and*，则会重新执行上一条命令。命令 *command* 和 *first* 参数一样是一个数字或字符串。可以参照前面的“ 命令历史 ”一节。

options

`-e [`*editor*`]`
调用编辑器 *editor* 来编辑指定的历史命令。默认的 *editor* 由 shell 变量 FCEDIT 定义。如果该变量没有定义，则默认是 `/bin/ed`。

`-e -`
执行（或重新执行）一条历史命令；指上面提到的第二个语法行。

`-l` 列出指定的命令或一个范围里的命令，或者是最后 16 条命令的列表。

`-n` 禁止使用 `-l` 参数列出命令的行号。

`-r` 颠倒使用 `-l` 参数显示的列表中命令的顺序。

fc

fc

在 ksh93 中是 hist 命令的别名。

fg

fg [*jobIDS*]

将当前作业或 *jobIDS* 置于后台。参见前面的“作业控制”部分。

fi

fi

是结束 if 语句的保留字。（记住不要漏掉它！）

for

for x [in *list*]
do
    commands
done

使变量 *x*（在可选的值列表 *list* 中）执行 *command* 命令。如果省略了 in *list*，那么就假定为“ $@ ”（位置参数）。

示例

为命令行指定的文件标记页码，并保存每一个结果：

```
for file; do
     pr $file > $file.tmp
done
```

在章中搜索一组词（和 fgrep -f 相似）：

```
for item in `cat program_list`
do
    echo "Checking chapters for"
    echo "references to program $item..."
    grep -c "$item.[co]" chap*
done
```

从每个文件中提取出一个单词的标题并用作新文件名：

```
for file
do
    name=`sed -n 's/NAME: //p' $file`
    mv $file $name
done
```

for

for ((*init; cond; incr*))
do
    commands
done

只用于ksh93中。实现循环运算，和C中的相似。首先求出*init*的值。当条件*cond*为真时，执行循环体。在重新检查条件*cond*时计算*incr*的值。表达式中的任何一个参数都可省去，如果省略了*cond*，则把它作为真值对待。

示例
在每个奇数章中搜索一个短语：

```
for ((x=1; x <= 20; x += 2))
do
        grep $1 chap$x
done
```

function

function *name* {*commands*;}

只用于Korn shell中。将*name*定义成一个shell函数。参见前面*name*()的相关内容。

示例
定义一个统计文件的函数。

```
$   function fcount {
>           ls | wc -l
> }
```

functions

functions

在Korn shell中用于typeset -f的别名（注意命令名称中的“s”。function是Korn shell中的关键字）。参见后面提到的**typeset**命令。

getconf

getconf [*name*[*path*]]

只用于ksh93中。检索由于系统不同而不同的参数的值。*name*是那个要检索的参数；*path*是一个文件名，用于检测由于系统类型不同而变化的参数。

getconf

这些参数由 POSIX 1003.1 和 1003.2 标准定义。参见第二章中的 **getconf**。

示例

输出在 C `int` 中能够支持的整数的最大值。

```
$ getconf INT_MAX
2147483647
```

getopts

`getopts[-a` *`name`*`]` *`string name`*`[`*`args`*`]`

处理命令行参数（如果指明的话是*args*）并检查合法选项。`getopts` 用于 shell 脚本循环，并确保命令行选项的标准语法。标准语法指明命令行选项在开头有一个 `+` 或 `-`。选项可堆叠，即在相邻的字母后可以有一个 `-`。在命令行中通过指明 `--` 来结束对选项的处理。当运行 shell 脚本时，*string* 包含了由 `getopts` 识别的选项字母。有效的选项将依次处理，并存储在 shell 变量 *name* 中。如果一个选项后面跟着一个冒号，那么该选项后必定有一个或多个参数。（命令的多个参数必须作为一个 shell 词来对待，这可以通过对参数加引号或用逗号分隔来完成。应用程序必须按这种格式来处理多个参数。）`getopts` 使用 shell 变量 OPTARG 和 OPTIND。`getopts` 对于作为 `/usr/bin/getopts` 的非 Bounre shell 用户是可用的。

options

`-a` 使用无效选项错误信息中的 *name*。只用于 `ksh93` 中。

hash

`hash [-r] [`*`commands`*`]`

Bourne shell 版本。当 shell 沿着搜索路径（$PATH）找到命令时，会把发现命令的位置放于一个内部散列表中。当下次用户键入一个命令时，shell 会使用散列表中存储的值。

如果不带参数，`hash` 会列出当前散列的命令。该显示表明了命中数（hits，shell 调用那个命令的次数）和开销（cost，即找到该命令需用的工作程度）。在命中数一栏中，对于那些在相对路径中找到的命令都有一个星号（*）。

如果给出了 *commands*，shell 会把那些命令加入到散列表中。

hash

`-r` 参数用于从散列表中删除命令，或者是全部或者是那些由 *commands* 指定的命令。当给 PATH 赋值时，也会清空散列表。用 `PATH = $PATH` 来清散列表，不会影响搜索路径。如果在 $PATH 中列出的较早的目录中，用户安装了一个比当前命令更新的版本，这时使用该方法会非常有用。

hash

hash

在 Korn shell 中是 `alias -t`（在 `ksh93` 中是 `alias -t-`）的别名，它模拟 Bourne shell 中的 `hash`。

hist

`hist [`*`options`*`] [`*`first`*` [`*`last`*`]]`
`hist -s [`*`old=new`*`] [`*`command`*`]`

只用于 `ksh93` 中。用于显示或编辑历史列表中的命令（只能用 `-l` 或 `-s` 中的一个）。*first* 和 *last* 是一个数字或字符串，用于指定要显示或编辑的命令的范围。如果省略 *last*，`hist` 就只能应用于一个命令（由 *first* 指定）。如果 *first* 和 *last* 都省略，`hist` 会编辑前一条命令或最后 16 条命令的列表。该用法的第二种形式对于历史命令 *command*，用 *new* 字符串代替 *old* 字符串并执行修改后的命令。如果没有指明字符串，则命令 *command* 只会重新执行。如果也没有指明 *command*，则会执行上一条命令。命令 *command* 和 *first* 一样是一个数字或字符串。参见前面的“命令历史”部分。

options

`-e [`*`editor`*`]`
调用编辑器 *editor* 来编辑指定的历史命令。默认的 *editor* 是由 shell 变量 HISTEDIT 设置的。如果该变量没有设置，则使用 FCEDIT。如果两个变量都没有设置，默认值是 `/bin/ed`。

`-l` 列出指定的命令或一个范围内的命令，或是最后 16 条命令的列表。

`-n` 禁止通过参数 `-l` 列出命令的行号。

`-r` 颠倒由参数 `-l` 列出的命令的顺序。

`-s` 执行（或重新执行）一条历史命令，指上面的第二行语法。

history

history

显示最后的16条命令。在ksh88中是fc -l的别名。在ksh93中是hist -l的别名。

if

if *condition1*
then *commands1*
[elif *condition2*
  then *commands2*]
.
.
.
[else *commands3*]
fi

如果满足条件*conditon1*，则执行*command1*；如果满足*condition2*，则执行*command2*；如果两个条件都不满足，则执行*command3*。一般通过test命令或[[]]来指定条件。参见**test**和**[[ ]]**命令来查看条件的完整列表，也可参见：以及**exit**命令下的示例。

示例
在小于10的数前插入一个0：

```
if [ $counter -lt 10 ]
then number=0$counter
else number=$counter
fi
```

如果一个目录不存在，则创建它：

```
if [ ! -d $dir ]; then
   mkdir $dir
   chmod 775 $dir
fi
```

integer

integer

指定整数变量。在Korn shell中是typeset -i的别名。

jobs

jobs [*options*] [*jobIDs*]

列出所有正在运行或已停止的作业，或列出由*jobIDs*指定的作业。例如，用户可以检查一段较长的编译或文本格式化是否仍在运行。在退出系统之前运行它也是有用的。参见前面的“作业控制”部分。

options

`-l` 列出作业ID并处理组ID。

`-n` 列出从最后通知以来状态改变的作业。只用于ksh93中。

`-p` 只列出进程组ID。

`-x` *cmd*
用相关的进程ID代替*cmd*中的进程ID，然后执行*cmd*。在Korn shell中无效。

kill

kill [*options*] *IDs*

终止每一个指定的进程ID或作业ID。执行命令的用户必须拥有该进程或是一个特权用户。这个内置命令类似第二章描述的/usr/bin/kill。参见前面的“作业控制”部分。

options

`-l` 列出信号名（由其本身使用）。

`-n` *num*
发送给定信号的编号。只用于ksh93中。

`-s` *name*
发送给定信号的名称。只用于ksh93中。

`-`*signal*
信号编号（来自/usr/include/sys/signal.h）或信号名（来自kill -l）。信号9的kill是“绝对的”。

信号

信号是在/usr/include/sys/signal.h中定义的。这里列出了一些，但都省略了前缀SIG。在用户的系统中，可能会有更多的信号。

```
HUP        1          挂起
```

kill

INT	2	中断
QUIT	3	退出
ILL	4	非法指令
TRAP	5	跟踪陷阱
IOT	6	IOT 指令
EMT	7	EMT 指令
FPE	8	浮点异常
KILL	9	终止
BUS	10	总线错误
SEGV	11	段冲突
SYS	12	系统调用了错误参数
PIPE	13	写入管道，但没有进程来读
ALRM	14	时钟报警
TERM	15	软件终止（默认信号）
USR1	16	用户定义的信号 1
USR2	17	用户定义的信号 2
CLD	18	子进程终止
PWR	19	电源错误后重启

let

let *expressions*
或
((*expressions*))

只用于 Korn shell 中。执行一个或多个表达式 *expressions* 指定的运算。*expressions* 由数字、运算符和 shell 变量（前面不必有 $）组成。如果表达式中包含了空格或其他特殊字符，则必须将表达式引起来。可以用(())将表达式引起来。要想查阅更多的信息和示例，可以参见前面章节中的“算术表达式”。也可以查阅第二章中提到的 **expr**。

示例

下面的每个示例都会给变量 i 加 1：

```
i=`expr $i + 1`          用于 sh、ksh88 和 ksh93 中
let i=i+1                用于 ksh88 和 ksh93 中
let "i = i + 1"
((i = i + 1))
((i += 1))
(( i++))                 只用于 ksh93 中
```

nameref

nameref *newvar=oldvar* ...

在 ksh93 中是 typeset -n 的别名。可查阅本章前面“变量”部分中对间接变量的讨论。

newgrp

newgrp [*group*]

将用户的组ID改变为*group*，或返回用户的默认组。在现在的Unix系统中已经废弃了这个命令，因为用户可以存在于多个组中。

nohup

nohup

在退出后不终止一个命令。nohup是一个Korn shell别名：

```
nohup='nohup '
```

nohup后的空格可以使nohup在需要的时候将后面的命令解释为别名。

print

print [*options*] [*string* ...]

只用于Korn shell中。输出字符串*string*（默认是在标准输出中）。print包括echo函数，并可能用在大多数Unix系统中。

options

- `-` 忽略所有后面的选项。
- `--` 和 - 一样 。
- `-f` *format*
  和printf一样，用*format*作为格式化字符串，忽略-n、-r和-R选项。只用于ksh93中。
- `-n` 不用换行符结束输出。
- `-p` 发送字符串*string*给由|&创建的进程，而不是到标准输出。
- `-r` 忽略经常用于echo中的转义序列。
- `-R` 和-r一样并且忽略后来的选项（除了-n）。
- `-s` 发送字符串*string*到历史文件。
- `-u`[*n*]
  发送字符串*string*到文件描述符*n*（默认是1）。

printf

printf *format* [*val*...]

只用于ksh93中。像ANSI C中的printf函数一样格式化输出。

printf

附加的格式化字母

`%b` 扩展字符串中的转义序列（例如，将`\t`扩展为制表符等）。

`%d` 一个附加的句点，并且输出基数后可以跟着精度（例如，`%5.3.6d`可以以基数6输出）。

`%P` 将`egrep`扩充的正则表达式转化成`ksh`模式。

`%q` 输出一个被引起来的稍后可以重读的字符串。

pwd

`pwd`

`pwd [-LP]`

在标准输出上输出当前的工作目录。第二种形式用于Korn shell中。

options

选项控制被输出路径使用逻辑路径还是物理路径。可以查阅本部分以前列出的**cd**命令。

`-L` 使用逻辑路径（用户键入的，包括任何符号链接）和当前目录PWD的值，这是默认的。

`-P` 使用当前目录的实际的文件系统的物理路径。

r

`r`

重新执行上一条命令。在`ksh88`中是`fc -e -`的别名，在`ksh93`中是`hist -s`的别名。

read

`read` *`variable1`* `[`*`variable2`* `...]`

读标准输入上的一行并把每个单词赋予相应的变量，并把剩余的单词赋予最后一个变量。如果只指定一个变量，则整个行都将赋予该变量。可以参照本例和`case`命令下的示例。除非碰到*EOF*的情况，返回值都是0。

示例

```
$   read first last address
Sarah Caldwell 123 Main Street
```

read

```
$   echo "$last, $first\n$address"
Caldwell, Sarah
123 Main Street
```

read

read [*options*] [*variable1*[?*string*]] [*variable2*...]

只用于Korn shell中。和在Bourne shell中相似，只是Korn shell版本支持如下的选项和用于提示的? 语法。如果第一个变量后有?*string*，那么*string*会作为一个用户提示符显示出来。如果没有给出变量，则输入被存储到变量REPLY中。另外，ksh93还允许用户指定一个超时。

options

-A *array*
: 读到索引数组*array*中。只用于 ksh93 中。

-d *delim*
: 读直到碰到第一个*delim*而不是碰到换行符为止。用于 ksh93 中。

-p 从 |& 协处理的输出读入。

-r Raw 模式，忽略作为一个行延续字符的 \。

-s 将输入作为历史文件的一个命令存储。

-t *timeout*
: 当从一个终端或管道读的时候，如果在*timeout*秒后仍没有数据进入则返回1。这可以防止应用程序因为等待用户输入而永远挂起。只用于 ksh93 中。

-u[*n*]
: 从文件描述符 *n*（默认是 0）中读输入。

示例

提示你自己键入两个温度：

```
$  read n1?"High low: " n2
High low: 65 33
```

readonly

```
readonly [variable1 variable2 ...]
readonly -p
```

防止指定的shell变量被赋予新的值。可以访问变量（读）但不可以修改。在Korn shell中，可以通过语法*variable=value*赋予一个不可改变的新值。第二种形式只用于ksh93中。

options

`-p` 在输出只读变量的名称和值之前输出`readonly`。这可以存储一个只读变量的列表以备以后访问。

redirect

```
redirect i/o-redirection ...
```

命令exec的ksh93别名。

示例

改变shell的标准错误到控制台：

```
$ redirect 2>/dev/console
```

return

```
return[n]
```

用于一个函数定义里面。退出函数时返回状态*n*或返回执行前面命令后的退出状态。

select

```
select x [in list]
do
  commands
done
```

只用于Korn shell中。显示一列关于标准错误的菜单项，菜单项会按它们在*list*中指定的顺序编号。如果没有给出`in list`，那么就在命令行（通过`"$@"`）获取菜单项。菜单后面是一个提示字符串（由PS3设置）。在PS3提示下，用户通过键入其行号来选择一个菜单项，或通过按Return键重新显示该菜单（用户输入存储在shell变量REPLY中）。如果键入一个有效的行号，就会执行命令*commands*。输入EOF会终止循环。

select

示例

```
PS3="Select the item number: "
select event in Format Page View Exit
do
  case "$event" in
    Format) nroff $file | lp;;
    Page) pr $file | lp;;
    View) more $file;;
    Exit) exit 0;;
    *   ) echo "Invalid selection";;
  esac
done
```

这个脚本的输出如下：

```
1. Format
2. Page
3. View
4. Exit
Select the item number:
```

set

set [*options arg1 arg2* ...]

如果没有参数，那么 set 会输出当前 shell 已知的所有变量的值。选项可以被允许（*-option*）或被禁止（*+option*）。通过 ksh 和 sh，选项也可以在调用 shell 时设置（参见前面的“调用 shell”部分）。参数按顺序被赋予 $1、$2，等等。

options

+A *name*
: 将剩余的参数指定为数组 *name* 的元素。只用于 Korn shell 中。

-A *name*
: 和 +A 一样，但在赋值之前需清除 *name*。只用于 Korn shell 中。

-a 从现在开始，在定义或改变变量后自动为输出标记变量。

-b 和 -o notify 一样。单字母形式只用于 ksh93 中。

-C 和 -o noclobber 一样。单字母形式只用于 ksh93 中。

-e 如果一个命令产生非 0 退出状态则退出。在 Korn shell 中，在 shell 退出前执行 ERR 陷阱。

-f 忽略文件名元字符（例如，* ? []）。

set

`-h` 当定义命令时定位它们。Korn shell会创建跟踪别名，而Bourne shell 会散列命令名。参见 **hash**。

`-k` 环境变量的赋值（*var=value*）生效而不用管它们出现在命令行的位置。正常情况下，赋值必须出现在命令名的前面。

`-m` 启用作业控制，后台作业在一个分离的进程组中执行。`-m`通常自动被设置。只用于 Korn shell 中。

`-n` 只是读命令但不执行，在语法检查时是有用的。如果Korn shell是交互式的，它会忽略这个选项。

`-o` [*mode*]

列出 Korn shell 模式，或打开模式 *mode*。许多模式可以用其他的选项设置。模式如下：

`allexport`	和 `-a` 一样。
`bgnice`	按照较低的优先权运行后台作业。
`emacs`	设置命令行编辑器为 `emacs`。
`errexit`	和 `-e` 一样
`ignoreeof`	不处理 *EOF* 信号。要想退出 shell，键入 `exit`。
`keyword`	和 `-k` 一样。
`markdirs`	添加“ `/` ”到目录名。
`monitor`	和 `-m` 一样。
`noclobber`	阻止通过“ `>` ”重定向符修改文件，而用“ `>|` ”去修改。
`noexec`	和 `-n` 相同。
`noglob`	和 `-f` 相同。
`nolog`	在历史文件中删去函数定义。
`notify`	在后台作业终止时立即打印作业完成消息，而不等到下一个提示符出现后再报告。
`nounset`	和 `-u` 一样。
`privileged`	和 `-p` 一样。
`trackall`	和 `-h` 一样。

set

verbose 和 -v 一样。

vi 将命令行编辑器设置为 vi。

viraw 和 vi 一样，但需要处理每一个键入的字符。

xtrace 和 -x 一样。

-p 作为一个特权用户启动（即不用处理 $HOME/.profile）。

-s 对位置参数排序。只用于 Korn shell 中。

-t 执行一个命令后退出。

-u 在替换时，将清除变量视为错误。

-v 在读的时候显示每一个 shell 命令行。

-x 在执行命令的时候显示命令和参数，并且前面有一个 +（Korn shell：前面是 PS4 的值）。这提供了单步调试 shell 脚本的功能。

- 关闭 -v 和 -x，并且关闭选项处理。在 Korn shell 中用于和旧版本的 Bourne shell 兼容。

-- 用作最后的选项，“ -- ”关闭选项处理从而使以“ - ”开头的参数不会被错误地理解为选项（例如，你可以将 $1 设置为 -1）。如果在“ -- ”后没有给出参数，则清位置参数。

示例

```
set - "$num" -20 -30        设置 $1 为 $num，$2 为 -20，$3 为 -30
set -vx                     读每一个命令行，显示它，执行它，
                            再显示它（和参数一起）
set +x                      停止跟踪命令
set -o noclobber            阻止修改文件
set +o noclobber            再允许修改文件
```

shift

shift[*n*]

交换位置参数（例如，将 $2 变为 $1）。如果给出 *n*，则向左移动 *n* 个位置。通常用于在 while 循环中迭代命令行参数。在 Korn shell 中，*n* 可以是一个整数表达式。

sleep

sleep[*n*]

只用于 ksh93 中。睡眠 *n* 秒钟。*n* 可以有一个小数部分。

stop	stop [*jobIDs*] 挂起由 *jobIDs* 指定的后台作业，这是 CTRL-Z 或 suspend 的一个补充。在 ksh88 中无效。参见前面的“作业控制”部分。
stop	stop [*jobIDs*] kill-s STOP 的 ksh93 别名。
suspend	suspend 和 CTRL-Z 功能一样。经常用于停止一个 su 命令。在 ksh88 中无效，在 ksh93 中，它是 kill -s STOP $$ 的别名。
test	test *condition* 或 [*condition*] 判断一个条件 *condition*，如果它的值为真则返回退出状态 0，否则返回一个非 0 的退出状态。该命令的替换形式是用 [] 而不是单词 test。在 Korn shell 中允许附加形式 [[]]。*condition* 可以用下列表达式来构造。如果对它的描述为真则条件为真。属于 Korn shell 的用法标记为（K），属于 ksh93 的用法标记为（K93）。 文件条件 -a *file* *file* 存在。（K） -b *file* *file* 存在并且是一个特殊的块文件。 -c *file* *file* 存在并且是一个特殊的字符文件。 -d *file* *file* 存在并且是一个目录。 -f *file* *file* 存在并且是一个规则的文件。

test

`-g` *file*
file 存在并且它的 set-group-id 位已设置。

`-G` *file*
file 存在并且它的组是那个有效的组 ID。（K）

`-k` *file*
file 存在并且它的粘滞位已设置。

`-L` *file*
file 存在并且是一个符号链接。（K）

`-o` *c*
选项 *c* 打开。（K）

`-O` *file*
file 存在并且它的属主是那个有效的用户 ID。（K）

`-p` *file*
file 存在并且是一个命名管道（输入输出）。

`-r` *file*
file 存在并且是可读的。

`-s` *file*
file 存在并且大小大于 0。

`-S` *file*
file 存在并且是一个套接字。（K）

`-t` [*n*]
打开的文件描述符 *n* 和一个终端设备相关联，*n* 的默认值是 1。

`-u` *file*
file 存在并且它的 set-user-id 位已设置。

`-w` *file*
file 存在并且是可写的。

`-x` *file*
file 存在并且是可执行的。

f1 `-ef` *f2*
文件 *f1* 和 *f2* 被链接（指的是同一文件）。（K）

test

f1 -nt f2
文件 *f1* 比 *f2* 新。（K）

f1 -ot f2
文件 *f1* 比 *f2* 旧。（K）

字符串条件

`string`
字符串 *string* 不为空。

`-n` *s1*
字符串 *s1* 长度不为零。

`-z` *s1*
字符串 *s1* 长度为零。

s1 = s2
字符串 *s1* 和 *s2* 是相同的。在 Korn shell 中，*s2* 可以是通配符模式（参见本章前面的“文件名元字符”部分）。

s1 == s2
字符串 *s1* 和 *s2* 是相同的。*s2* 可以是通配符模式。比 = 更常用。（K93）

s1 `!=` *s2*
字符串 *s1* 和 *s2* 是不同的。在 Korn shell 中，*s2* 可以是通配符模式。

s1 < s2
s1 的 ASCII 值在 *s2* 的 ASCII 值的前面（只在`[[]]`结构中才有效）。（K）

s1 > s2
s1 的 ASCII 值在 *s2* 的 ASCII 值的后面（只在`[[]]`结构中才有效）。（K）

整数比较

n1 `-eq` *n2*
n1 和 *n2* 相等。

n1 `-ge` *n2*
n1 大于或等于 *n2*。

test

n1 -gt *n2*
n1 大于 *n2*。

n1 -le *n2*
n1 小于或等于 *n2*。

n1 -lt *n2*
n1 小于 *n2*。

n1 -ne n2
n1 不等于 *n2*。

组合形式

(*condition*)
如果 *condition* 为真则为真（用在分组中）。那些()应当用 \ 引起来。

! *condition*
如果 *condition* 为假则为真。

condition1 -a *condition2*
如果两个条件都为真则为真。

condition1 && *condition2*
如果两个条件都为真则为真（只在[[]]结构中有效）。（K）

condition1 -o *condition2*
如果两个条件之一为真则为真。

condition1 || *condition2*
如果两个条件之一为真则为真（只在[[]]结构中有效）。（K）

示例

下面的示例给出了可能使用测试条件的语句的第一行。

```
while test $# -gt 0              当有参数的时候
while [ -n "$1" ]                当有非空参数的时候
if [ $count -lt 10 ]             如果$count 小于 10
if [ -d RCS ]                    如果存在 RCS 目录
if [ "$answer" != "y" ]          如果 answer 不是 y
if [ ! -r "$1" -o ! -f "$1" ]    如果第一个参数不是一个可读文件
                                 或一个规则文件
```

time

`time` *command*
`time` [*command*]

只用于 Korn shell 中。执行命令 *command* 并输出总的消耗时间、用户时间和系统时间（以秒为单位）。和 Unix 命令 `time` 基本相同（参见第二章），只是这个内置版本的命令不但可给在管线中的所有命令计时，而且可以给其他内置命令计时。

第二种形式用于`ksh93`中，如果没有给出*command*，则输出总的shell用户、系统时间与所有子进程时间。

times

`times`

输出对用户和系统处理的累积时间。

trap

`trap [[`*commands*`] `*signals*`]`
`trap -p`

如果接收到任何的信号*signals*则执行命令*commands*。第二种形式只用于 `ksh93` 中，它会以适合以后再次读取的形式输出当前的陷阱设置。

一般的信号包括 0、1、2 和 15。多个命令应该引起来作为一个命令组，并且用分号隔开。如果 *commands* 是空字符串（即`trap" "`*signals*），shell 会忽略 *signals*。如果完全省略了命令 *commands*，则会重新设置由默认行为处理指定的信号。在 `ksh93` 中，如果 *commands* 是“ - ”，则重新设置 *signals* 为最初的默认值。

如果命令 *commands* 和信号 *signals* 都省略，则列出当前分配的陷阱。参见下面的示例和 **exec** 命令下的示例。

信号

信号与触发它们的事件一起列出：

0 从 shell 退出（通常是当 shell 脚本执行完时）。

1 挂起（通常是退出时）。

2 中断（通常是 `CTRL-C`）。

trap

3 退出。

4 非法指令。

5 跟踪陷阱。

6 IOT 指令。

7 EMT 指令。

8 浮点异常。

10 总线错误。

12 系统调用了一个错误的参数。

13 向一个没有进程去读的管道中写入。

14 报警超时。

15 软件中断（通常是通过 `kill`）。

`ERR` 非零退出状态。只用于 Korn shell。

`DEBUG` 任何命令的执行。只用于 Korn shell。

`KEYBD` 在 `emacs`、`gmacs` 或 `vi` 编辑模式下，读入一个键。只用于 `ksh93`。

示例

```
trap "" 2          忽略信号 2（中断）
trap 2             恢复中断 2
```

当 shell 程序退出、用户退出、按 `CTRL-C` 或执行 `kill` 时删除 `$tmp` 文件：

```
trap "rm -f $tmp; exit" 0 1 2 15
```

当 shell 程序收到信号 1、2 或 15 时，输出一个“ clear up ”消息：

```
trap 'echo Interrupt! Cleaning up...' 1 2 15
```

true

`true`

在 `ksh88` 中是 `ksh93` 的内置命令 `:` 的别名，退出时返回一个真值。

type

type *commands*

对每个命令名显示它是Unix命令、内置命令还是经过定义的shell函数。在Korn shell中，它只是whence -v的别名。

示例

```
$  type mv read
mv is /bin/mv
read is a shell builtin
```

typeset

typeset[*options*] [*variable*[*value*...]]
typeset -p

只用于Korn shell中。为每个变量赋予一种类型（和一个可选的初始值*value*），或者，如果没有提供变量，显示所有的特定类型的变量（由选项来决定）。如果指定了变量，“ -*option* ”允许那种类型，而“ +*option*”则禁止该类型。如果没有给出变量，则 -*option*输出变量名和变量值，“ +*option* ”则只输出变量名。

第二种形式只用于ksh93中。

options

-A *arr*

*arr*是一个关联数组。只用于ksh93中。

-E *d*

变量*variable*是一个浮点数。*d*是小数的位数。输出值时采用prinf %g 格式。只用于ksh93中。

-F *d*

变量*variable*是一个浮点数。*d*是小数的位数。输出值时采用printf %f格式。只用于ksh93中。

-f[*c*]

该变量是一个函数，不允许赋值。如果没有给出变量，则列出当前的函数名。标志*c*可以是t、u或x。t用于打开跟踪（和set -x一样）。u用于标记该函数为未定义，用于函数的自动装载（即当第一次用到那个函数时，通过搜索FPATH来定位该函数，在ksh93中也搜索PATH）。用x来输出该函数。注意别名**autoload**和**functions**。

typeset

`-H` 在非 Unix 系统中，将 Unix 文件名映射为主文件名。

`-i[`*n*`]`
将变量定义成基数为*n*的整数。`integer`是`typeset -i`的别名。

`-L[`*n*`]`
将变量定义为靠左对齐的字符串。*n*是字符串长度（可以根据需要从右端截取字符或用空格从右端补齐字符串）。去掉开头的空字符；如果也指定了参数 `-Z`，则开头的 0 也被去掉。如果没有指定 *n*，字段的宽度即为赋予变量的第一个值的宽度。

`-l` 将大写转换成小写。

`-n` 变量 *variable* 是对另一个变量的间接引用（一个 *nameref*）。只用于 `ksh93` 中（参见本章前面的“变量”部分）。

`-p` 输出 `typeset` 命令来重新创建所有当前变量的类型。只用于 `ksh93` 中。

`-R[`*n*`]`
将变量定义为靠右对齐的字符串，*n*为字符串长度（可以根据需要从左端截取字符或用空格从左边补齐字符串）。去掉尾部的空格 。如果没有提供 *n*，则字段的宽度即为赋予变量的第一个值的宽度。

`-r` 将变量标记为只读。参见 **readonly**。

`-t` 用一个用户定义标签来标记变量。

`-u` 将小写转换成大写。

`-x` 将变量标记为自动输出。

`-Z[`*n*`]`
如果同时用了 `-L` 选项，则去掉字符串前面所有的 0。该选项单独使用时类似选项 `-R`，只是 `-Z` 选项对数字补齐数字 0 而对文本补齐空格。

示例

```
typeset                 列出所有设置的变量的名称、值和类型
typeset -x              列出输出的变量的名称和值
typeset +r PWD          结束 PWD 的只读状态
typeset -i n1 n2 n3     三个变量是整数
typeset -R5 zipcode     zipcode 是按右对齐的并且具有五个字符的宽度
```

ulimit

ulimit[*options*] [*n*]

输出一个或多个资源限制的值，或者如果指定为*n*，则设置一个资源限制为*n*。资源限制可以是硬的（-H）也可以是软的（-S）。默认情况下，unlimit设置两种限制或输出软限制。选项用来决定对哪个资源操作。

options

-H 硬限制。任何人都可以降低一个硬限制，但只有特权用户才允许提高硬限制。

-S 软限制。必定比硬限制要低。

-a 输出所有的限制。

-c 核心文件的最大大小。

-d 数据段或堆的最大 K 字节数。

-f 文件的最大大小（默认的选项）。

-m 物理内存的最大 K 字节数。只用于 Korn shell（不是在所有的 Unix 系统中有效）。

-n 最大文件描述符加 1。

-p 管道缓冲区的大小。只用于 Korn shell 中（不是在所有的 Unix 系统中有效）。

-s 栈段的最大 K 字节数。

-t 最大的 CPU 秒数。

-v 虚拟内存的最大 K 字节数。

umask

umask [*nnn*]
umask [-S] [*mask*]

显示文件创建掩码或设置文件创建掩码为八进制值*nnn*。文件创建掩码决定会关闭哪个权限位（例如，unmask 002 产生 rw-rw-r--）。参见第二章的示例。

第二种形式只用于 ksh93 中。允许保持符号掩码。

umask

options

-S 用符号表示法输出当前的掩码。只用于 ksh93 中。

unalias

unalias *names*

unalias -a

只用于 Korn shell 中。从别名列表中删除 *names*。参见 **alias**。

options

-a 删除所有的别名。只用于 ksh93 中。

unset

unset *names*

Bourne shell 版本。删除在 *names* 中列出的函数或变量的定义。

unset

unset [*options*] *names*

删除在*names*中列出的函数或变量的定义。Korn shell版本支持选项。

options

-f 清除 *names* 中的函数。

-n 清除间接变量（nameref）*name*，而不是nameref指的那个变量。只用于 ksh93 中。

-v 清除变量 *names*（默认）。只用于 ksh93 中。

until

```
until condition
do
  commands
done
```

执行命令 *commands* 直到满足条件 *condition*。通常用 test 命令指定 *condition*。

wait

wait [*ID*]

暂停执行直到后台作业完成为止（返回退出状态 0），或者暂停执行直到指定的后台进程 *ID* 或作业 *ID* 完成为止（返回 *ID* 的退出状态）。

wait

注意 shell 变量 $! 包含了最近的后台进程的进程 ID。如果作业控制无效，*ID* 只能是一个进程 ID 号。参见前面的“作业控制”部分。

示例

```
wait $!          等待最近的后台进程完成
```

whence

whence [*options*] *commands*

只用于 Korn shell 中。显示每个命令名是 Unix 命令、内部命令、定义的 shell 函数还是别名。

options

`-a` 输出 *commands* 的所有解释。只用于 `ksh93` 中。

`-f` 跳过搜索 shell 函数。只用于 `ksh93` 中。

`-p` 搜索 *commands* 的路径名。

`-v` 详细的输出，和 `type` 一样。

while

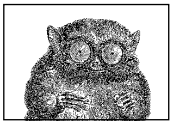
```
while condition
do
  commands
done
```

如果满足条件 *condition* 则执行 *commands*。通常用 `test` 命令来指定 *condition*。参见 **case** 和 **test** 的示例。

filename

filename

从可执行文件 *filename* 中读取并执行命令，或执行一个二进制目标文件。



第五章

C shell

本章介绍C shell，之所以如此命名，是因为它很多的编程结构与符号和C编程语言相似。其中包括以下内容：

- 功能概述
- 语法
- 变量
- 表达式
- 命令历史
- 作业控制
- 调用 shell
- 内置的C shell命令

要想得到有关C shell的更多信息，可以阅读在参考文献中列出的《Using csh & tcsh》。

功能概述

C shell具备以下功能：

- 输入输出重定向
- 用于文件名缩写的通配符（元字符）

- 定制用户环境的 shell 变量
- 整数运算
- 访问以前的命令（命令历史）
- 命令名缩写（别名）
- 用于写 shell 程序的内置命令集
- 作业控制
- 文件名完成（可选）

语法

本部分介绍了针对 C shell 的很多符号。包括以下内容：

- 特殊文件
- 文件名元字符
- 引用
- 命令方式
- 重定向方式

特殊文件

`~/.cshrc` 在 shell 调用每一个实例时执行。

`~/.login` 在 `.cshrc` 执行之后由登录 shell 执行。

`~/.logout` 在退出时由登录 shell 执行。

`~/.history` 来自以前的登录中存储的历史列表。

`/etc/passwd` `~`*name* 缩写的主目录的来源（可能来自 NIS 或 NIS+）。

文件名元字符

元字符	描述
*	匹配任何有 0 个或多个字符的字符串
?	匹配任何单个字符
[*abc*...]	匹配被括起来的字符中的任何一个；可以用连字符指定一个范围（例如，a-z、A-Z、0-9）

元字符	描述
{*abc*,*xxx*,...}	扩展括号中的每一个由逗号分隔的字符串。这些字符串不一定匹配实际的文件名
~	当前用户的主目录
~*name*	用户 *name* 的主目录

示例

```
% ls new*          匹配 new、new.1 等
% cat ch?          匹配 ch9 等，但不匹配 ch10
% vi [D-R]*        匹配文件名开头是大写 D 到大写 R 的文件
% ls {ch,app}?     先进行扩展，然后匹配 ch1、ch2、app1、app2 等
% mv info{,.old}   扩展成 mv info info.old
% cd ~tom          将目录修改为用户 tom 的主目录
```

引用

引用可以禁止字符的特殊意义，使字符按其本来意思加以使用，下表中的字符在 C shell 中有特殊的意义。

字符	意义
;	命令分隔符
&	后台执行
( )	命令分组
|	管道
* ? [ ] ~	文件名元字符
{ }	字符串扩展字符，通常不要求引用
< > & !	重定向符号
! ^	历史替代，快速替代
" ' \	用于引用其他字符
`	命令替代
$	变量替代
space tab newline	单词之间的间隔

下列的字符用于引用：

"" 位于 " 和 " 之间的所有的字符都按其字面意义加以采用，下面这些具有特殊意义的字符除外：

`$` 产生变量替代。

`` ` `` 产生命令替代。

`"` 表示双引号的结束。

`\` 转义下一个字符。

`!` 历史字符。

`newline`

换行字符。

`''` 除!(历史）和另一个'以及换行符之外，位于'和'之间的所有字符都按其字面意义加以采用。

`\` 在其后的字符会按字面意义采用。通常用于`""`中以转义`"`、`$`、`` ` ``和换行符。用于`''`内可转义换行符。经常用来转义一个历史字符（通常是!)。

示例

```
%   echo 'Single quotes "protect" double quotes'
Single quotes "protect" double quotes

%   echo "Don't double quotes protect single quotes too?"
Don't double quotes protect single quotes too?

%   echo "You have `ls|wc -l` files in `pwd`"
You have 43 files in /home/bob

%   echo The value of \$x is $x
The value of $x is 100
```

命令方式

cmd `&`	在后台执行 *cmd*。
cmd1 `;` *cmd2*	命令序列，在同一行执行多个命令。
(*cmd1* `;` *cmd2*)	子 shell，将 *cmd1* 和 *cmd2* 视为一个命令组。
cmd1 `|` *cmd2*	管道，用 *cmd1* 的输出作为 *cmd2* 的输入。
cmd1 `` `cmd2` ``	命令替代，用 *cmd2* 的输出作为 *cmd1* 的参数。
cmd1 `&&` *cmd2*	逻辑与关系，执行 *cmd1*（如果 *cmd1* 执行成功）后再执行 *cmd2*。这是一种短路操作，如果 *cmd1* 没有成功执行，*cmd2* 将永远不能执行。

cmd1 || *cmd2*　逻辑或关系，执行 *cmd1* 或（如果 *cmd1* 执行失败）*cmd2*。这是一个短路操作，如果 *cmd1* 执行成功，*cmd2* 将永远不能执行。

示例

```
%  nroff file > file.out &              后台格式化
%  cd; ls                               顺序执行
%  (date; who; pwd) > logfile           重定向所有输出
%  sort file | pr -3 | lp               先对文件排序，再分页输出，然后打印
%  vi `grep -l ifdef *.c`               编辑 grep 找到的文件
%  egrep '(yes|no)' `cat list`          指定一个搜索文件列表
%  grep XX file && lp file              如果包含了该模式，则打印文件
%  grep XX file || echo XX not found    否则，回显一个错误消息
```

重定向方式

文件描述符	名称	一般缩写	默认
0	标准输入	stdin	键盘
1	标准输出	stdout	终端
2	标准错误	stderr	终端

重定向方式可以改变一般的输入源和输出目标，参见下面的内容。

简单重定向

cmd > *file*

将 *cmd* 的输出发送到文件 *file* 中（覆盖）。

cmd >! *file*

和前一个命令意义相同，另外，还会忽略 noclobber 选项。

cmd >> *file*

将 *cmd* 的输出发送到文件 *file* 中（追加）。

cmd >>! *file*

和前一个命令意义相同，但写文件的时候忽略 noclobber 选项。

cmd < *file*

cmd 从文件 *file* 中获取输入 。

cmd << *text*

读取标准输入，直到遇到一个和 *text* 相等的行（*text* 可以存储在一个 shell 变量中）。输入通常使用终端键入或存储在shell程序中。使用这类语法的命令通常有

cat、echo、ex 和 sed。如果文本 *text* 被引起来（用任何 shell 引用机制），则该输入会按其原来的意思逐字地通过。

多重重定向

cmd >& *file*	将标准输出和标准错误发送到 *file* 中。
cmd >&! *file*	和上条命令相同，但忽略 noclobber 的设置。
cmd >>& *file*	将标准输出和标准错误添加到 *file* 的结尾。
cmd >>&! *file*	和上条命令相同，只是在追加或创建 *file* 时忽略 noclobber 选项。
cmd1 |& *cmd2*	标准错误和标准输出一起执行管道功能 。
(*cmd* > *f1*) >& *f2*	将标准输出发送到文件 *f1* 中；将标准错误发送到文件 *f2* 中。
cmd | tee *files*	将 *cmd* 的输出发送到标准输出（通常是终端）和 *files* 中（参见第二章中 **tee** 下面的示例）。

示例

```
%  cat part1 > book
%  cat part2 part3 >> book
%  mail tim < report
%  cc calc.c >& error_out
%  cc newcalc.c >&! error_out
%  grep Unix ch* |& pr
%  (find / -print > filelist) >& no_access
```

```
%  sed 's/^XX /g' << "END_ARCHIVE"
This is often how a shell archive is "wrapped"
bundling text for distribution. You would normally
run sed from a shell program, not from the command line.
"END_ARCHIVE"
XX This is often how a shell archive is "wrapped",
XX bundling text for distribution. You would normally
XX run sed from a shell program, not from the command line.
```

变量

本部分介绍以下内容：

- 变量替代
- 变量修饰符

- 预定义的 shell 变量
- `.cshrc` 文件示例
- 环境变量

变量替代

在下列替代中，大括号(`{}`)是可选的，除非必须用它来将变量名与后面的字符分开。

`${`*var*`}`	变量 *var* 的值。
`${`*var*`[i]}`	在 *var* 的 *i* 位置处选择 1 个或几个单词。*i* 可能是一个单个的数，也可能是范围 *m*-*n*、范围 -*n*（缺 *m* 暗示为 1）、范围 *m*-（缺 *n* 暗示其余的所有单词）中的数，或者是 `*`（选择所有单词）。*i* 也可以是扩展成此类值之一的一个变量。
`${#`*var*`}`	*var* 中单词的个数。
`${#`*argv*`}`	参数的个数。
`$0`	程序的名称（通常不会在交互式 shell 中设定）。
`${argv[n]}`	命令行上单个的参数（位置参数）。*n* =1~9。
`${n}`	与 `${argv[n]}`作用相同。
`${argv[*]}`	命令行上所有的参数。
`$*`	与 `$argv[*]`作用相同。
`$argv[$#argv]`	最后一个参数。
`${?`*var*`}`	如果设置了 *var*，则返回 1；否则返回 0。
`$$`	当前 shell 的进程号；在创建具有惟一名字的临时文件时，作为文件名的一部分是有用的。
`$?0`	如果知道输入文件名，则返回 1；否则返回 0。
`$<`	从标准输入中读一行。

示例

对第三到最后之间的参数（文件）进行排序，并且将输出存储到一个惟一的临时文件中：

```
sort $argv[3-]>tmp.$$
```

只有当 shell 是交互式时处理 `.cshrc` 命令（即必须设置 `prompt` 变量）：

```
if ($?prompt) then
   set commands,
   alias commands,
   etc.
endif
```

变量修饰符

除 `$?var`、`$$`、`$?0` 和 `$<` 之外，前面进行变量替代时，后面都可能跟着下面的修饰符之一。如果使用括号，修饰符在它的里面。

`:r` 返回变量的根。

`:e` 返回变量的扩展。

`:h` 返回变量的头。

`:t` 返回变量的尾。

`:gr` 返回所有的根。

`:ge` 返回所有的扩展。

`:gh` 返回所有的头。

`:gt` 返回所有的尾。

`:q` 将一个单词列表变量引起来，将其各部分分隔开。当变量中包含不应进行扩展的文件名元字符时，该选项比较有用。

`:x` 将一个模式引起来，并将它扩展成单词列表。

使用路径名修饰符的示例

下表展示了有关下面变量路径名修饰符的用法：

```
set aa=(/progs/num.c /book/chap.ps)
```

变量部分	格式	输出结果
正常变量	`echo $aa`	`/progs/num.c /book/chap.ps`
第二个根	`echo $aa[2]:r`	`/book/chap`
第二个头	`echo $aa[2]:h`	`/book`
第二个尾	`echo $aa[2]:t`	`chap.ps`

变量部分	格式	输出结果
第二个扩展	`echo $aa[2]:e`	`ps`
根	`echo $aa:r`	`/progs/num /book/chap.ps`
全部根	`echo $aa:gr`	`/progs/num /book/chap`
头	`echo $aa:h`	`/progs /book/chap.ps`
全部头	`echo $aa:gh`	`/progs /book`
尾	`echo $aa:t`	`num.c /book/chap.ps`
全部尾	`echo $aa:gt`	`num.c chap.ps`
扩展	`echo $aa:e`	`c /book/chap.ps`
全部扩展	`echo $aa:ge`	`c ps`

使用引用修饰符的示例

```
%   set a="[a-z]*" A="[A-Z]*"
%   echo "$a" "$A"
[a-z]* [A-Z]*

%   echo $a $A
at cc m4 Book Doc

%   echo $a:x $A
[a-z]* Book Doc

%   set d=($a:q $A:q)
%   echo $d
at cc m4 Book Doc

%   echo $d:q
[a-z]* [A-Z]*

%   echo $d[1] +++ $d[2]
at cc m4 +++ Book Doc

%   echo $d[1]:q
[a-z]*
```

预定义的 shell 变量

变量可以按两种方式来设置，第一种方式是分配一个值：

```
set var=value
```

第二种方式是仅仅打开它：

```
set var
```

下表展示了变量的赋值是通过一个等号（该等号后是变量接受的值的类型）来完成的，然后再对值进行描述。（注意，无论如何都不能对 argv、cwd 或 status 之类的变量进行显式赋值。）对于那些只能打开或关闭的变量来说，该表也相应介绍了在设置它们时所做的工作。C shell 会自动设置下面一些变量：argv、cwd、home、path、promt、shell、status、term 和 user。

变量	描述
argv=(*args*)	传递给当前命令的参数列表，默认是()
cdpath=(*dirs*)	当给 cd、popd 或 pushd 定位参数时，轮流搜索的目录列表
cwd=*dir*	当前目录的完整路径名
echo	在执行前重新显示每个命令行，和 csh -x 命令相同
fignore=(*chars*)	在完成文件名时要忽略的文件名后缀列表（参见 filec）
filec	如果设置了它，当按下 Escape 键的时候，在命令行上输入的部分文件名可以扩展成其完整的名称。如果匹配不止一个文件名，则键入 *EOF* 会列出可能的完全文件名
hardpaths	告诉 dirs 要显示那些是一个符号链接的目录的实际路径名
histchars=ab	一个两字符的字符串，用来设置用于历史替代和快速替代中的字符（默认是!^）
history=*n*	要存储到历史列表中的命令的个数
home=*dir*	用户的主目录，从 HOME 中初始化。~ 字符是该值的缩写形式
ignoreeof	忽略来自终端的文件结束标记（*EOF*），阻止意外退出
mail=(*n file*)	每 5 分钟或（如果指定 *n*）每 *n* 秒钟对一个或多个文件进行新邮件检查
nobeep	对不明确的文件完整性操作不振铃
noclobber	不重定向输出到一个已存在的文件，用来防止对文件的意外毁坏
noglob	关闭文件名扩展，用于 shell 脚本中
nonomatch	将文件名元字符当作字面上的字符来对待。例如，vi ch* 是创建新文件 ch* 而不是输出“ No match ”
notify	立刻通知已完成作业的用户，而不用等待下一个提示符
path=(*dirs*)	列出搜索执行命令的路径名。可用 PATH 初始化。SVR4 默认为（./usr/ucb/usr/bin）。在 Solaris 上，默认路径为（/usr/bin.）。然后标准启动脚本再将它修改为（/bin /usr/bin /usr/ucb/ 等等）

变量	描述
prompt='*str*'	提示用于交互式输入的字符串，默认是 %
savehist=*n*	退出时存储在~/.history目录中的历史命令的个数，在下一次登录时可以访问到它
shell=*file*	当前使用的 shell 程序的路径名，默认是 /bin/csh
status=*n*	最后命令的退出状态。成功时，内置命令返回 0；若失败，则返回 1
term=*ID*	终端类型的名称，和 TERM 一样
time='*n*%*c*'	如果命令执行时间超过 *n* 个 CPU 秒数，则报告用户时间、系统时间、消耗时间和 CPU 占有率。支持可选的 %*c* 标志来显示其他数据
user=*name*	用户的登录名称，在 USER 中初始化
verbose	进行历史替代之后，显示一个命令，和命令 csh -v 作用相同

.csbrc 文件示例

```
# 预定义的变量
set path=(~ ~/bin /usr/ucb /bin /usr/bin . )
set mail=(/var/mail/tom)

if ($?prompt) then              # 为交互式使用而设置
  set echo
  set filec
  set noclobber ignoreeof

  set cdpath=(/usr/lib /var/spool/uucp)
 # 现在可以键入cd宏
 # 代替cd /usr/lib/macros

  set fignore=.o                # 对filec忽略目标文件
  set history=100 savehist=25
  set prompt='tom \!% '         # 包括历史号
  set time=3

 # MY VARIABLES

  set man1="/usr/man/man1"      # 允许我执行cd $man1、ls $man1
  set a="[a-z]*"                # 允许我执行vi $a
  set A="[A-Z]*"                # 或grep string $A

 # ALIASES

  alias c "clear; dirs"         # 使用引号保护;或|
  alias h "history | more"
  alias j jobs -l
  alias ls ls -sFC              # 重新定义ls命令
```

```
  alias del 'mv \!* ~/tmp_dir'   # rm的安全替代
endif
```

环境变量

C shell维护了一系列环境变量（environment variable），它们明确地区别于shell变量，并且不是C shell的部分。shell变量只是在当前shell内部有意义，而环境变量则会自动输出，从而使环境变量成为全局变量。例如，C shell变量只在定义它们的特定脚本中才是可访问的，而环境变量可以应用到任何shell脚本、mail实用程序或用户调用的编辑器中。

可按下列方式给环境变量赋值：

```
setenv VAR value
```

按惯例，环境变量名都是大写的。你可以创建自己的环境变量，也可使用下面预定义的环境变量。

这些环境变量有一些相应的C shell变量，如下所示：

HOME
: 主目录，和home作用相同。可以彼此无关地进行修改。

PATH
: 搜索命令的路径，和path作用相同。改变任意一个路径则会修改存储在另一个中的值。

TERM
: 终端类型，和term作用相同。改变term可修改TERM，但反之则不行。

USER
: 用户名，和user作用相同。改变user可修改USER，但反之则不行。

其他环境变量包括：

EXINIT
: 类似于在.exrc启动文件中的字符串（例如，set ai），是一个ex命令的字符串。由vi和ex使用。

LOGNAME
: USER变量的另一个名字。

MAIL
: 拥有邮件的文件。由邮件程序使用。它与C shell中只用于检查新邮件的`mail`变量不一样。

PWD
: 当前目录，该值从`cwd`拷贝而来。

SHELL
: 默认情况下未定义，一旦初始化到shell，两者是一样的。

表达式

表达式用于在`@`（C shell数学运算符）、`if`和`while`语句中执行算术运算、字符串比较、文件测试等。`exit`和`set`也能指定表达式。表达式由运算符将变量和常量组合构成，就如同在C编程语言中一样。运算符优先权与在C中一样。记住下面的优先权规则是非常容易的：

- `* / %`
- `+ -`
- 将所有在`()`里的其他表达式分组，如果表达式中包括`<`、`>`、`&`或`|`，则圆括号是必需的

运算符

运算符可以是下面类型之一。

赋值运算符

运算符	描述
=	赋值
+= -=	在加/减后重新赋值
*= /= %=	在乘/除/求余后重新赋值
&= ^= |=	在对位进行AND/XOR/OR后重新赋值
++	递增
--	递减

算术运算符

运算符	描述
* / %	乘；整除；求余
+ -	加；减

位和逻辑运算符

运算符	描述
~	二进制求反（二进制反码）
!	逻辑非
<< >>	位左移；位右移
&	按位与
^	按位异或
|	按位或
&&	逻辑与（短路操作）
||	逻辑或（短路操作）
{command}	如果命令成功，则返回1；否则返回0。注意，这个情况和*command*的正常返回码是不一样的。$status变量可能更实用一些

比较运算符

运算符	描述
== !=	相等；不等
<= >=	小于或等于；大于或等于
< >	小于；大于
=~	左边字符串匹配一个包含*、?或[...]的文件名模式
!~	左边字符串不匹配一个包括*、?或[...]的文件模式

文件查询操作符

在执行检测之前对文件*file*执行命令替代和文件名扩展。

操作符	描述
`-d` *file*	该文件是一个目录
`-e` *file*	该文件存在
`-f` *file*	该文件是一个平常的文件
`-o` *file*	用户拥有该文件
`-r` *file*	用户有读权限
`-w` *file*	用户有写权限
`-x` *file*	用户有执行权限
`-z` *file*	文件长度为 0
`!`	对上面的每个操作符取反

示例

下面的示例介绍了 `@` 命令，并且假定 `n=4`。

表达式	`$x` 的值
`@ x = ($n > 10 || $n < 5)`	1
`@ x = ($n >= 0 && $n < 3)`	0
`@ x = ($n << 2)`	16
`@ x = ($n >> 2)`	1
`@ x = $n % 2`	0
`@ x = $n % 3`	1

下面的示例一般用于 `if` 和 `while` 语句的第一行。

表达式	意义
`while ($#argv != 0)`	当有参数的时候……
`if ($today[1] == "Fri")`	如果第一个词是“ Fri ”……
`if ($file !~ *.[zZ])`	如果文件不是以 `.z` 或 `.Z` 为结尾……
`if ($argv[1] =~ chap?)`	如果第一个参数是 chap 后跟着一个单字符……
`if (-f $argv[1])`	如果第一个参数是一个平常的文件……
`if (! -d $tmpdir)`	如果 `$tmpdir` 不是一个目录……

命令历史

前面执行过的命令存储在一个历史列表中。C shell 可以让你访问该列表，从而可以对命令进行校验，重复执行它们，或对命令修改后再执行。`history`内置命令可显示历史列表，预定义的变量`histchars`、`history`和`savhist`也可以影响历史机制。访问历史列表涉及三件事情：

- 进行命令替代（用 ! 和 ^）
- 进行参数替代（一个命令内的特定单词）
- 用修饰符提取或替换一个命令或单词的部分

命令替代

`!`	开始一个历史替代
`!!`	先前命令
`!`*N*	历史表中第 *N* 条命令
`!-`*N*	从当前命令开始往后第 *N* 条命令
`!`*string*	以 *string* 开始的最近的命令
`!?`*string*`?`	包含了 *string* 的最近的命令
`!?`*string*`?%`	包含了 *string* 的最近命令参数
`!$`	先前命令的最后一个参数
`!!`*string*	上一条命令，然后追加 *string*
`!`*N string*	第 *N* 条命令，然后追加 *string*
`!{`*s1*`}`*s2*	以字符串 *s1* 开始的最近的命令，然后追加字符串 *s2*
`^`*old*`^`*new*`^`	快速替代，在先前命令中将字符串 *old* 改成 *new*，执行修改后的命令

命令替代示例

事先假定下面的命令：

```
3%  vi cprogs/01.c ch002 ch03
```

事件序号	键入的命令	执行的命令
4	^00^0	vi cprogs/01.c ch02 ch03
5	nroff !*	nroff cprogs/01.c ch02 ch03
6	nroff !$	nroff ch03
7	!vi	vi cprogs/01.c ch02 ch03
8	!6	nroff ch03
9	!?01	vi cprogs/01.c ch02 ch03
10	!{nr}.new	nroff ch03.new
11	!!|lp	nroff ch03.new | lp
12	more !?pr?%	more cprogs/01.c

单词替代

单词说明符允许从先前命令行中提取单个的单词。冒号可以放在任何单词说明符的前面。在一个事件序号之后，除非是这里显示的，否则冒号是可选择的。

`:0` 命令名

`:`*n* 参数序号 *n*

`^` 第一个参数

`$` 最后一个参数

`:`*n*`-`*m* 参数 *n* 到 *m*

`-m` 单词 0 到 *m*，和：`0-`*m* 相同

`:n-` 参数 *n* 至最后一个的前一个

`:n*` 参数 *n* 至最后一个，和 *n*`-$` 相同

`*` 所有的参数，和 `^-$` 或 `1-$` 相同

`#` 当前命令行直到这点，很少用

单词替代示例

假定有下列命令：

```
13%  cat ch01 ch02 ch03 biblio back
```

事件序号	键入的命令	执行的命令
14	`ls !13^`	`ls ch01`
15	`sort !13:*`	`sort ch01 ch02 ch03 biblio back`
16	`lp !cat:3*`	`lp ch03 biblio back`
17	`!cat:0-3`	`cat ch01 ch02 ch03`
18	`vi !-5:4`	`vi biblio`

历史修饰符

可以通过一个或多个修饰符来修饰命令替代和单词替代：

输出、替代和引用

`:p` 只显示命令但不执行。

`:s/`*`old`*`/`*`new`* 用字符串 *new* 替代 *old*，只影响第一个实例。

`:gs/`*`old`*`/`*`new`* 用字符串 *new* 替代 *old*，适用于所有的实例。

`:&` 重复先前的替代（`:s` 或 `^` 命令），只影响第一个实例。

`:g&` 重复先前的替代，适用于所有实例。

`:q` 将一个单词列表引起来。

`:x` 将分隔的单词引起来。

截取

`:r` 提取第一个可用的路径名的根。

`:gr` 提取所有的路径名的根。

`:e` 提取第一个可用的路径名的扩展名。

`:ge` 提取所有的扩展名。

`:h` 提取第一个可用的路径名的头。

`:gh` 提取所有路径名的头。

`:t` 提取第一个可用的路径名的尾。

`:gt` 提取所有路径名的尾。

历史修饰符的示例

以“单词替代示例”中的第17条命令为例：

```
17%  cat ch01 ch02 ch03
```

事件#	键入的命令	执行的命令
19	!17:s/ch/CH/	cat CH01 ch02 ch03
20	!:g&	cat CH01 CH02 CH03
21	!more:p	more cprogs/01.c（只显示）
22	cd !$:h	cd cprogs
23	vi !mo:$:t	vi 01.c
24	grep stdio !$	grep stdio 01.c
25	^stdio^include stdio^:q	grep"include stdio" 01.c
26	nroff !21:t:p	nroff 01.c（是我需要的吗?）
27	!!	nroff 01.c（执行它）

作业控制

作业控制使用户可以将前台作业放到后台，将后台作业转到前台，或挂起（暂时停止）正在运行的作业。C shell对作业控制提供了下面的命令。要想获得这些命令的更多信息，参见本章后面的“内置的C shell命令”。

`bg` 将一个作业放至后台。

`fg` 将一个作业放至前台。

`jobs`
: 列出活动的作业。

`kill`
: 终止一个作业。

`notify`
: 当一个后台作业完成时通知。

`stop`
: 挂起一个后台作业。

`CTRL-Z`
: 挂起一个前台作业。

很多作业控制命令采用 *jobID* 作为参数。可以按以下方式指定参数：

`%`*n* 作业号 *n*

`%`*s* 作业的命令行是以字符串 *s* 开始的

`%?`*s* 作业的命令行中包含字符串 *s*

`%%` 当前作业

`%` 当前作业（和上面相同）

`%+` 当前作业（和上面相同）

`%-` 先前的作业

调用 shell

可以按下面的方式来调用 C shell 命令解释器：

```
csh [options] [arguments]
```

`csh` 从一个终端或文件中执行命令。在调试脚本的时候，选项 `-n`、`-v` 和 `-x` 是有用的。

下面对这些选项进行了详细的解释：

`-b` 允许余下的命令行选项作为一个指定命令的选项来解释，而不是作为csh本身的选项。

`-c` 将第一个参数作为一个要执行的命令的字符串。余下的参数通过argv数组可以使用。

`-e` 如果一个命令产生错误，则退出。

`-f` 快速启动，不用执行 `.cshrc` 或 `.login` 就可以启动 `csh`。

`-i` 调用交互式 shell（可提示输入）。

`-n` 解析命令但不执行。

`-s` 从标准输入读命令。

`-t` 执行一个命令后退出。

`-v` 执行命令前显示命令，扩展历史替代但不扩展其他的替代（如文件名、变量和命令）。与设置 `verbose` 作用相同。

`-V` 与设置 `-v` 作用相同，但也显示 `.cshrc`。

`-x` 执行命令之前先显示命令，但扩展所有的替代。与设置 `echo` 作用相同。经常将 `-x` 和 `-v` 结合使用。

`-X` 与 `-x` 作用相同，但也显示 `.cshrc`。

内置的 C shell 命令

#	# 忽略在同一行中跟在它后面的所有文本。shell 脚本把它作为注释符而不是一个真正的命令来使用。另外，一些旧的系统有时会将第一个字符为 # 的文件作为一个 C shell 脚本来解释。
#！	#!*shell* [*option*] 用于一个脚本的第一行来调用指定 shell。该行的其余部分作为单个的参数传递给 shell。该功能一般由内核实现，但一些旧系统可能不支持该功能。一些系统对 shell的最大长度有32个字符的限制。例如： `#!/bin/csh -f`
:	: 空命令（什么也不做）。返回一个退出状态 0。
alias	alias [*name*[*command*]] 指定 *name* 为命令 *command* 的缩写名称或别名。如果省略了参数 *command*，则输出 *name* 别名，如果也省略了 *name*，则输出所有的别名。别名可以在命令行上定义，但更常存储在 `.cshrc` 中，以便在登录之后生效（参见本章前面的“ `.cshrc` 文件示例 ”）。别名定义可以引用命令行参数，与历史列表相似。用 `/!*` 指所有的命令行参数，用 /!^ 指第一个参数，用 `/!$` 指最后一个参数等等。一个别名 *name* 可以是任何有效的 Unix 命令，然而，如果不键入 *name*，则会丧失其最初的意思。参见 **unalias**。 示例 在 X window 系统中设置 `xterm` 窗口的尺寸：

alias

```
alias R 'set noglob; eval `resize`; unset noglob'
```

显示包含了字符串 *ls* 的别名：

```
alias | grep ls
```

在所有的命令行参数上运行 nroff：

```
alias ms 'nroff -ms \!*'
```

拷贝被指定为第一个参数的文件：

```
alias back 'cp \!^ \!^.old'
```

使用常规的 ls，而不是它的别名：

```
% \ls
```

bg

bg [*jobIDs*]

将当前的作业或 *jobIDs* 放至后台。参见前面的“作业控制”部分。

示例

为了将一个消耗时间的进程放在后台，用户可以使用下面命令：

```
4 % nroff -ms report | col > report.txt
CTRL-Z
```

然后使用下面的任何一种方式：

```
5% bg
5% bg %          当前作业
5% bg %1         作业号 1
5% bg %nr        匹配词首字符串 nroff
5% %&
```

break

break

继续执行距离 while 或 foreach 最近的、在 end 命令后的命令。

breaksw

breaksw

终止一个 switch 语句，然后继续执行 endsw 之后的命令。

case	case *pattern* 在 switch 语句中标识模式 pattern。
cd	cd [*dir*] 将工作目录修改为*dir*，默认为用户的主目录。如果*dir*是一个相对的路径名而不是当前目录，则搜索变量 cdpath。参见本章前面的“ .cshrc 文件示例 ”部分。
chdir	chdir [*dir*] 与 cd 命令作用相同。如果希望重新将 cd 定义为别名，则可以使用该命令。
continue	continue 继续执行 while 和 foreach 的下一次迭代循环。
default	default： 规定 switch 语句中默认的情况（通常在最后）。
dirs	dirs[-l] 输出目录堆栈，首先显示当前目录，用 -l 将主目录符号（~）扩展为实际的目录名。参见 **popd** 和 **pushd**。
echo	echo [-n] *string* 将字符串 *string* 写到标准输出中，如果指定了 -n，输出不会被换行符中断。和 Unix 版本及 Bourne shell 版本不一样（/bin/echo），C shell 中的 echo 不支持转义字符。也可以参见第二章和第四章中的 **echo** 命令。
end	end 用于结束 foreach 和 while 语句的保留字。

endif

endif

用于结束 if 语句的保留字。

endsw

endsw

用于结束 switch 语句的保留字。

eval

eval *args*

一般地，eval用于 shell 脚本中，并且 *args* 是一行包含了 shell 变量的代码。eval首先强制变量扩展，然后运行生成的命令。当shell 变量包含了输入/输出重定向符号、别名或其他shell变量的时候，这种两次搜索是有用的。(例如，正常情况下重定向发生在变量扩展之前，因此一个包含了重定向符号的变量必须首先用 eval扩展，否则会无法解释那个重定向符号。) 可以参见第四章中eval下面的示例。下面也列出了 eval的其他用法。

示例

下面的代码用于 .login 文件中来确定终端属性。

```
set noglob
eval 'tset -s xterm'
unset noglob
```

下面的命令显示了 eval 的用法：

```
%   set b='$a'
%   set a=hello

% echo $b          读命令行一次
$a
% eval echo $b    读命令行两次
hello
```

exec

exec *command*

执行 *command* 来代替当前 shell，这将终止当前的 shell，而不是在它下面创建一个新的进程。

exit

`exit [(expr)]`

退出一个shell脚本，退出时带有*expr*指定的状态。零状态意味着成功，非零状态意味着失败。如果没有指定*expr*，则退出值就是`status`变量的值。可以在命令行执行`exit`以关闭一个窗口（退出）。

fg

`fg [`*`jobIDs`*`]`

将当前的作业或*jobIDs*放到前台。可以参见前面的“作业控制”部分。

示例

如果用户挂起了一个`vi`编辑会话（通过按`CTRL-Z`），那么用户就可以用下面命令中的任何一种继续`vi`的执行：

```
8% %
8%  fg
8%  fg %
8%  fg %vi              匹配词首的字符串
```

foreach

`foreach` *`name`* `(`*`wordlist`*`)`
`commands`
`end`

将变量*name*赋给*wordlist*中的每个值，并且执行在`foreach`和`end`之间的命令。用户可以将`foreach`作为在C shell 提示符下的多行命令来使用（第一个示例），也可以将它用在一个shell脚本中（第二个示例）。

示例

对所有以大写字母开头的文件重命名：

```
%   foreach i ([A-Z]*)
?   mv $i $i.new
?  end
```

检查每个命令行参数是否是一个选项：

```
foreach arg ($argv)
   # does it begin with - ?
   if ("$arg" =~ -*) then
```

foreach

```
            echo "Argument is an option"
          else
          echo "Argument is a filename"
         endif
      end
```

glob

glob *wordlist*

对 *wordlist* 作文件名替代、变量替代和历史替代。这种扩展更像 echo，只是不能识别\转义，并且单词间是用空字符作为定界符的。glob 通常用于在 shell 脚本中对一个值进行“硬编码”，以使其在脚本的其余部分保持一致。

goto

goto *string*

跳到第一个非空字符是 *string* 且其后跟着一个 :的一行，然后继续执行该行下的代码。在 goto 行上，*string* 可能是一个变量或文件名模式，但要分支去的标签必须是一个字面上的、被扩展的值，且不能出现在一个 foreach 或 while 语句的内部。

hashstat

hashstat

显示一个统计，该统计表明了通过 path 变量来定位命令的散列表的成功级别。

history

history [*option*]

显示历史事件的列表（在前面的“命令历史”中曾讨论了历史语法）。

注意：多行复合命令（例如 foreach...end）不存储在历史列表中。

options

-h 不带事件序号输出历史列表。

-r 按倒置顺序输出，最后显示最早的命令。

n 只显示最后的 *n* 条历史命令，而不是由 shell 变量 history 设置的数字。

示例

保存并执行最后 5 条命令：

history

```
history -h 5 > do_it
source do_it
```

if

`if`

开始一个条件语句，简单的格式是：

```
if (expr) cmd
```

下面并排显示了三个可能的其他格式：

```
if (expr) then      if (expr) then      if (expr) then
    cmds                cmds1               cmds1
endif               else                else if (expr)then
                        cmds2               cmds2
                    endif               else
                                            cmds3
                                        endif
```

在最简单的形式中，如果 *expr* 为真，则执行 *cmd*；否则什么也不做（重定向仍旧发生，这是一个 bug）。在其他形式中，执行一个或多个命令。如果 *expr* 为真，则继续执行 `then` 后的命令；如果 *expr* 为假，则分支到 `else`（或 `else if` 之后并且继续检查）后面的命令。更多的示例可以参见前面的“ 表达式 ”，或 **shift** 和 **while**。

示例

如果没有指定命令行参数，则执行一个默认的行为：

```
if ($#argv == 0) then
   echo "No filename given. Sending to Report."
   set outfile = Report
else
   set outfile = $argv[1]
endif
```

jobs

`jobs [-l]`

列出所有正在运行或停止的作业，`-l` 包括进程 ID。例如，用户可以检查长编译或文本格式化是否仍在运行。在退出前也是有用的。

kill

`kill [`*`options`*`] `*`ID`*

终止每个指定的进程 *ID* 或作业 *ID*。用户必须拥有该进程，或者是特权用户。这个内置命令与第二章中的 `/usr/bin/kill` 类似，但也允

kill

许符号作业名称。难处理的进程可以用信号9去终止。参见前面的“ 作业控制 ”。

options

`-l` 列出信号名（用于本身）。

`-signal`

信号编号（来自`/usr/include/sys/signal.h`）或信号名（来自`kill -l`）。采用了信号 9 的 kill 是 “ 绝对的 ”。

信号

在`/usr/include/sys/signal.h`中定义了信号, 在这里不带SIG前缀列出了这些信号。在用户的系统中可能有比这更多的信号。

HUP	1	挂起
INT	2	中断
QUIT	3	退出
ILL	4	非法指令
TRAP	5	跟踪陷阱
IOT	6	IOT 指令
EMT	7	EMT 指令
FPE	8	浮点异常
KILL	9	终止
BUS	10	总线错误
SEGV	11	段冲突
SYS	12	系统调用了错误参数
PIPE	13	向管道中写, 但没有进程来读
ALRM	14	时钟报警
TERM	15	软件终止(默认信号)
USR1	16	用户定义信号 1
USR2	17	用户定义信号 2
CLD	18	子进程终止
PWR	19	电源失败后重启

示例

如果你使用了如下命令：

```
44%   nroff -ms report > report.txt &
[1] 19536          csh 输出作业和进程 ID
```

你可以使用下面的任何一种方式终止它:

```
45%  kill 19536          进程 ID
```

kill

```
45%  kill %          当前作业
45%  kill %1         作业号 1
45%  kill %nr        词首的字符串
45%  kill %?report   匹配字符串
```

limit

`limit [-h] [`*resource* `[`*limit*`]]`

显示限制，或对当前进程和它创建的进程使用的资源进行限制。如果不给出 *limit*，则只输出对 *resource* 的当前限制。如果也省略了 *resource*，则输出所有的限制。默认情况下会显示或设置当前的限制。使用 `-h` 时，则使用硬限制。一个硬限制强加一个不能超越的绝对限制。只有特权用户才可以提高它。参见 **unlimit**。

resource（资源）

`cputime`
CPU 能花费的最大秒数，可以缩写为 cpu。

`filesize`
任何一个文件的最大长度。

`datasize`
数据（包括栈）的最大长度。

`stacksize`
栈的最大长度。

`coredumpsize`
核心转储文件的最大长度。

limit（限制）

一个数字后跟着一个可选的字符（单位说明符）。

对于 cputime：*n*h（*n* 小时）
*n*m（*n* 分钟）
mm:ss（分钟和秒）

对于其他的：*n*k（*n* 千字节，默认）
*n*m（*n* 兆字节）

login	`login [`*user*`|-p]` 用 `/bin/login` 代替用户 *user* 的登录 shell。`-p` 保持了环境变量。
logout	`logout` 终止登录的 shell 。
nice	`nice [±`*n*`] `*command* 改变 *command* 的执行优先权；如果什么也没给出，则改变当前 shell 的优先权（参见第二章中的 **nice**），该优先权的范围是 −20 至 20，默认为 4。该范围的优先顺序和我们所想像的相反：−20 给出最高的优先权（最快地执行），20 给出最低的优先权。 `+`*n*　对优先权值加 *n*（降低作业优先权）。 `-`*n*　对优先权值减 *n*（提高作业优先权）。只能由特权用户使用。
nohup	`nohup [`*command*`]` “ 没有挂起的信号 ”。在关闭终端行后（即，当挂断一个电话或退出时）不终止 *command*。在 shell 脚本中使用没有 *command* 的 nohup 命令，以使脚本不被终止（参见第二章中的 **nohup**）。
notify	`notify [`*jobID*`]` 当一个后台作业完成后立即报告（而不是等待用户退出一个长编辑会话）。如果省略了 *jobID*，则假定是当前的后台作业。

onintr	`onintr `*label*
	`onintr -`
	`onintr`
	“ 正在中断 ”。用于在 shell 脚本中处理中断信号（与 Bourne shell 中的 `trap 2` 和 `trap ""2` 命令相似）。第一种形式类似 `goto` *label*，此时如果捕捉到中断信号（例如，`CTRL-C`），则脚本分支至 *label*；第二种形式使脚本忽略中断。在一个其运行不能被干扰的脚本的开头或代码段之前，第二种形式是有用的；第三种形式用于存储由先前的命令 `onintr-` 禁止的中断处理。

onintr

示例

```
onintr cleanup          中断发生时转至“ cleanup ”
 .
 .                      shell 脚本命令
 .
cleanup:                中断标号
  onintr -              忽略相应的中断
  rm -f $tmpfiles       删除任何创建的文件
  exit 2                返回一个错误状态并退出
```

popd

popd [+*n*]

删除目录栈的当前条目，或从栈中删除第 *n* 条。当前条目有序号 0 并显示在左边。可参见 **dirs** 和 **pushed**。

pushd

pushd *name*
pushd +*n*
pushd

第一种形式将工作目录修改为 *name*，并且将其添加到目录栈中；第二种形式将目录的第 *n* 项移动到开始位置，使之成为工作目录（条目序号从 0 开始）。如果不带参数，pushd 会交换目录的前两项，并变成新的当前目录。可以参见 **dirs** 和 **popd**。

示例

```
5%  dirs
/home/bob /usr
6% pushd /etc                   将 /etc 添加到目录栈
/etc /home/bob /usr
7% pushd +2                     将目录第三项转变成第一项
/usr /etc /home/bob
8% pushd                        交换目录的前两项
/etc /usr /home/bob
9% popd                         丢弃当前项，并移动到下一项
/usr /home/bob
```

rehash

rehash

重新计算 path 变量的散列表。在当前会话期间无论何时创建一个新命令，都应用 rehash。这可以允许 shell 定位并执行该命令。（如果新命令没有列在 path 目录中，在执行 rehash 前，应将目录添加到 path 中。）可以参见 **unhash** 命令。

repeat

repeat *n command*

执行 *command* 的 *n* 个实例。

示例

通过在一个文件中存储/usr/dict/words的25次拷贝来产生一个测试文件。

```
%  repeat 25 cat /usr/dict/words > test_file
```

从终端读 10 行并存储在 item_list 中：

```
%  repeat 10 line > item_list
```

向 report 中追加 50 个样本文件：

```
%  repeat 50 cat template >> report
```

set

set *variable=value*
set *variable*[*n*]=*value*
set

将变量 *variable* 设置为 *value*；如果指定了多个值，则用值的列表来设置变量。如果给定了下标 *n*，则将变量中的第 *n* 个单词设置为 *value*（变量包含的单词的数量不能少于单词的序号）。如果没有给定参数，则显示所有设置变量的名称和值。参见本章前面的“预定义的 shell 变量”部分。

示例

```
%  set list=(yes no maybe)          对一个词的列表赋值
%  set list[3]=maybe                对存在的词列表的一项赋值
%  set quote="Make my day"          对一个变量赋值
%  set x=5 y=10 history=100         对几个变量赋值
%  set blank                        对 blank 赋空值
```

setenv

setenv [*name* [*value*]]

对一个环境变量 *name* 赋值 *value*。按惯例，*name* 应用大写。*value* 可能是单个的词或一个引起来的字符串。如果没有给出 *value*，则赋空值。没有给出参数时，显示所有环境变量的名称和值。对 USER、TERM 和 PATH 变量来说，setenv 不是必需的，因为它们会自动地从 user、term 和 path 中输出。可以参见前面的“环境变量”。

shift

shift [*variable*]

假如给出*variable*，则转换一个词列表变量中的词。例如，*name*[2]会变成*name*[1]。如果没有给出参数，则转换位置参数（命令行参数），例如，$2变成$1。在while循环中经常用到shift。可参见**while**下的相应示例。

示例

```
while ($#argv)                  当有参数时
   if (-f $argv[1])
      wc -l $argv[1]
   else
      echo "$argv[1] is not a regular file"
   endif
   shift                        下一个参数
end
```

source

source [-h] *script*

从C shell脚本中读取并且执行命令。如果有-h，则命令会添加到历史列表中，但不会执行。

示例

```
source ~/.cshrc
```

stop

stop [*jobIDs*]

挂起当前的后台作业或由*jobIDs*指定的后台作业，该命令是CTRL-Z或suspend的补充。

suspend

suspend

挂起当前的前台作业，和CTRL-Z相似。经常用于终止su命令。

switch

switch

依靠一个变量的值来处理命令。当用户希望处理三个以上选择的时候，switch是if-then-else语句的最佳替代。如果*string*变量匹配*pattern1*，则执行第一组命令；如果匹配*pattern2*，则执行第二组命令；其他依次类推。如果没有模式匹配，则执行默认情况下的命令。*string*可以由命令替代、变量替代或文件名扩展来指定。模式可以用

switch

模式匹配符号 *、?和[ ]来指定。当执行完 *commands* 之后，可使用 breaksw 来退出 switch 语句。如果省略了 breaksw（很少这样做），则 switch 继续执行另一组命令，直到碰到 breaksw 或 endsw。下面是 switch 的一般语法，同时并列列出了一个示例，该示例用于处理第一个命令行参数。

```
switch (string)      |    switch ($argv[1])
 case pattern1:      |      case -[nN]:
     commands        |          nroff $file | lp
     breaksw         |          breaksw
 case pattern2:      |      case -[Pp]:
     commands        |          pr $file | lp
     breaksw         |          breaksw
 case pattern3:      |      case -[Mm]:
     commands        |          more $file
     breaksw         |          breaksw
     .               |      case -[Ss]:
     .               |          sort $file
     .               |          breaksw
 default:            |      default:
     commands        |          echo "Error-no such option"
                     |          exit 1
     breaksw         |          breaksw
endsw                |      endsw
```

time

time [*command*]

执行 *command* 并且显示它花费了多少时间。如果没有给出参数，则 time 用于 shell 脚本中，并对它进行计时。

umask

umask [*nnn*]

显示文件创建掩码，或将文件创建掩码设为八进制 *nnn*。文件创建掩码决定关闭哪一个权限位。参见第二章该项的示例。

unalias

unalias *name*

从别名列表中删除 *name*。要想获得更多信息，可参见 **alias**。

unhash

unhash

删除内部的散列表。C shell 停止使用被散列的值，并且通过搜索 path 中的目录来定位一个命令。可参见 **rehash**。

unlimit

unlimit [*resource*]

消除对资源*resource*的限制。如果没有给出*resource*，则取消对所有资源的限制。要想获得更多的信息，可以参见 **limit**。

unset

unset *variables*

删除一个或多个变量。变量名可以用文件名元字符来指定为一个模式。参见 **set**。

unsetenv

unsetenv *variable*

删除一个环境变量。文件名匹配无效。参见 **setenv**。

wait

wait

暂停执行，直到所有后台作业完成或接收到一个中断信号。

while

while (*expression*)
    commands
end

只要表达式为真（计算为非零），则执行 while 和 end 之间的 *commands*。break 和 continue 可以中断或继续该循环。可以参见 **shift** 下的示例。

示例

```
set user = (alice bob carol ted)
while ($argv[1] != $user[1])
  检查匹配以循环通过每一个用户
   shift user
  如果在循环通过时没有匹配…
   if ($#user == 0) then
     echo "$argv[1] is not on the list of users"
     exit 1
   endif
end
```

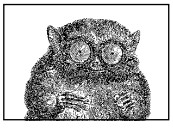
@	@ *variable=expression* @ *variable*[*n*]=*expression* @ 将运算表达式 *expression* 的值赋给 *variable*；如果指定了下标 *n*，则对变量的第 *n* 个元素赋值；如果既没有指定 *variable* 也没有给出 *expression*，则输出所有 shell 变量的值（和 `set` 相同）。表达式运算符和示例都在前面的“ 表达式 ”一节列出。下面两个特殊的形式也是有效的： @ *variable*++ 变量加一。 @ *variable*-- 变量减一。

第二部分

文本编辑和处理

第二部分概述了在Unix中用于文本编辑的命令集和相关的工具。第六章将回顾文本编辑中的一个重要方面，即模式匹配。本部分主要包括如下内容：

- 第六章，模式匹配
- 第七章，emacs编辑器
- 第八章，vi编辑器
- 第九章，ex编辑器
- 第十章，sed编辑器
- 第十一章，awk编程语言



第六章

模式匹配

大量的Unix文本处理工具让用户去搜索（有时还可以改变）文本模式而不是固定的字符串。这些工具包括ed、ex、vi和sed、awk编程语言，还有grep和egrep命令。文本模式（通常称为正则表达式）既包含有正常字符，又混合有特殊字符（称为元字符）。

本章将介绍以下主题：

- 文件名与模式
- Unix 程序列出的元字符
- 元字符
- 示例

要想获得关于正则表达式的更多信息，参见在参考文献中列出的《Mastering Regular Expressions》。

文件名与模式

用于模式匹配中的元字符区别于用于文件名扩展中的元字符（参见第四章和第五章）。当用户在命令行上提供了一个命令的时候，首先由shell识别特殊的字符，其次才是程序，因此，没有引用的元字符通常被shell解释为文件名扩展。例如：

```
$  grep [A-Z]* chap [12]
```

该命令由 shell 转变成：

```
$ grep Array.c Bug.c Comp.c chap1 chap2
```

然后在 Bug.c、comp.c、hap1 和 chap2 中试图找到模式 Array.c。为了绕过 shell 并且将特殊字符传递给 grep，使用引用：

```
$ grep "[A-Z]*" chap[12]
```

在大多数情况下，使用双引号就足够了，但只有单引号才是最安全的。

注意：在模式匹配过程中，？是用来匹配正则表达式中的零个或一个实例的，而在文件名扩展中，它是用来匹配一个字符的。

Unix 程序列出的元字符

一些元字符在一个程序中是有效的，但在另一个程序中则未必。在表 6-1 中用着重号（•）标记元字符对 Unix 程序有效。而标记为 P 的项则是由 POSIX 指定的，使用时应仔细检查你的系统版本。（在 Solaris 中，在 /usr/xpg4/bin 中的版本可接受这些项目。）在表后面提供了完整介绍。

表 6-1：Unix 元字符

符号	ed	ex	vi	sed	awk	grep	egrep	意义
.	•	•	•	•	•	•	•	匹配任何字符
*	•	•	•	•	•	•	•	匹配零个或多个前面的字符
^	•	•	•	•	•	•	•	匹配行/字符的开头
$	•	•	•	•	•	•	•	匹配行/字符的结尾
\	•	•	•	•	•	•	•	转义后面的字符
[]	•	•	•	•	•	•	•	匹配集合中的一个项
\(\)	•	•	•	•		•		存储以后会再利用的模式[a]
n	•	•	•	•		•		重新利用匹配的子模式
{}					•P		•P	匹配一个实例的范围

表 6-1：Unix 元字符（续）

符号	ed	ex	vi	sed	awk	grep	egrep	意义
\{ \}	•			•		•		匹配一个实例的范围
\< \>	•	•	•					匹配单词的开头或结尾
+				•		•		匹配一个或多个前面的字符
?				•		•		匹配零个或一个前面的字符
|				•		•		分隔开要匹配的选项
()				•		•		将要匹配的表达式分组

a： 在\(\)中，被存储的子模式可能在匹配期间再利用。见表 6-2。

注意：在 ed、ex、vi 和 sed 中，用户既要指定一个搜索模式（在左边），又要指定一个代替模式（在右边）。表 6-1 中的元字符只对搜索模式是有意义的。

在 ed、ex、vi 和 sed 中，表 6-2 中列出的元字符只对代替模式有效。

表 6-2：代替模式元字符

符号	ex	vi	sed	ed	意义
\	•	•	•	•	转义后面的字符
\n	•	•	•	•	存储在\(\)中的文本匹配模式
&	•	•	•	•	文本匹配搜索模式
~	•	•			再使用前面的替换模式
%				•	再使用前面的替换模式
\u \U	•	•			改变字符成大写
\l \L	•	•			改变字符成小写
\E	•	•			关闭先前的\U 或\L
\e	•	•			关闭先前的\u 或\l

元字符

搜索模式

下面表中的字符，只在搜索模式中有特殊的意义。

字符	模式
.	匹配除了换行符之外的任何单个字符。在 awk 中也可能匹配换行符
*	匹配任意数量（或无）的在它前面的单个字符。前面的字符也可能是一个正则表达式，例如，. 意味着任意字符，.* 则意味着“ 匹配任意数量的任何字符 ”
^	匹配在行或字符串的开头处跟着的正则表达式
$	匹配在行或字符串的结尾前面的正则表达式
[]	匹配括号包围起来的字符中的任何一个 连字符（-）指出连续字符的范围。而符号 ^ 在括号中的第一个字符处，则正好使上述意义相反 :它匹配在列表中没有列出的任意一个字符。在第一个字符处的一个连字号或右括号 (]), 会作为列表中的一个成员来对待。所有的其他元字符都作为列表中的成员来对待（照字面意思）
{*n*, *m*}	匹配直接在它前面的单个字符出现的范围。前面的字符也可能是一个元字符。{*n*}匹配精确的 *n* 次出现, {*n*, }匹配至少 *n* 次出现, {*n*, *m*}匹配介于 *n* 和 *m* 之间的任意次数的出现。*n* 和 *m* 必须介于 0 和 255 之间，且包含 0 和 255
\{*n*, *m*\}	和上面的{*n*, *m*}意义相同，但在括号前面有反斜杠
\	取消后面的字符的特殊意义
\(\)	将在 \ (和\) 之间括起来的模式存储在一个特殊的地方。在一个单行上可以存储最多 9 个模式。由子模式匹配的文本可以按转义序列 \1 到 \9 替代而再利用
n	将围在 \（和 \）中的第 *n* 个子模式再利用为该点处的模式。*n* 是一个从 1 到 9 的数字，1 开始在左边。也可以参见后面的示例
\< \>	匹配在词的开头 (\<) 或结尾 (\>) 的字符
+	匹配一个或多个正则表达式前的实例
?	匹配 0 个或一个正则表达式前的实例
|	匹配由前面或后面指定的正则表达式
()	对括起来的正则表达式的组匹配

很多 Unix 系统允许使用 POSIX 的“字符类”，即由方括号括起来的一组字符。这些类被包括在`[:`和`:]`中。例如，`[[:alnum:]]`匹配单个的字母数字字符。

类	匹配的字符
`alnum`	字母数字字符
`alpha`	字母字符
`blank`	空格或制表符
`cntrl`	控制字符
`digit`	小数位
`graph`	非空格字符
`lower`	小写字符
`print`	可打印字符
`space`	空白字符
`upper`	大写字符
`xdigit`	十六进制数

代替模式

下表中的字符只在代替模式中有特殊的意义。

字符	模式
`\`	取消跟着的字符的特殊意义
`\`*n*	恢复以前由`\(`和`\)`保存的第 *n* 个模式匹配的文本。*n* 是从 1 到 9 的数字，1 开始于最左边
`&`	将由搜索模式匹配的文本作为代替模式的部分重新使用
`~`	在当前的代替模式中重新使用先前的代替模式，必须是在代替模式中的惟一的字符(`ex`和`vi`)
`%`	在当前代替模式中重新使用先前的代替模式。必须是代替模式中惟一的字符(`ed`)
`\u`	将代替模式中的第一个字符转化为大写
`\U`	将全部的代替模式转化成大写
`\l`	将代替模式的第一个字符转化成小写
`\L`	将全部的代替模式转化成小写
`\e,\E`	取消先前的`\u`、`\U`、`\l`和`\L`

示例

当使用 grep 或 egrep 时，要用引号将正则表达式括起来。（假如该模式包含 $，必须用单引号，例如，'*pattern*'。）当使用 ed、ex、sed 和 awk 时，尽管（除 awk 外）任意的定界符都起作用，但通常用 \ 将正则表达式围起来。下表列出了一些示例。

模式	匹配的内容
bag	字符串 *bag*
^bag	该行的开头是 *bag*
bag$	*bag* 在该行的结尾
^bag$	*bag* 是该行惟一的单词
[Bb]ag	*Bag* 或 *bag*
b[aeiou]g	第二个字母是一个元音字母
b[^aeiou]g	第二个字母是一个辅音字母（或大写字母或符号）
b.g	第二个字母是任意的字符
^...$	只包括三个字符的任意行
^\.	以一个点开始的任意行
^\.[a-z][a-z]	和上面意义相同，跟着两个小写字母（例如，troff 请求）
^\.[a-z]\{2\}	和上面意义相同，只用于 ed、grep 和 sed 中
^[^.]	任何一个不以一个点开始的行
bugs*	*bug*、*bugs*、*bugss* 等等
"word"	在引号中的一个单词
"*word"*	一个单词，带或不带引号
[A-Z][A-Z]*	一个或多个大写字母
[A-Z]+	和上面意义相同，只用于 egrep 或 awk
[[:upper:]]+	和上面意义相同，为 POSIX egrep 或 awk
[A-Z].*	一个大写字母，后面跟着 0 个或多个大写字母
[A-Z]*	0 个或多个大写字母
[a-zA-Z]	任意字母
[^0-9A-Za-z]	任何符号或空白（而不是字母或数字）
[^[:alnum:]]	和上面相同，使用 POSIX 字符类

egrep 或 awk 模式	匹配的内容
`[567]`	*5*、*6* 或 *7* 中任意一个
`five|six|seven`	*five*、*six* 或 *seven* 中的任何一个
`80[2-4]?86`	*8086*、*80286*、*80386* 或 *80486*
`80[2-4]?86|(Pentium(-II)?)`	*8086*、*80286*、*80386*、*80486*、*Pentium* 或 *Pentium-II*
`compan(y|ies)`	*company* 或 *companies*

ex 或 vi 模式	匹配的内容
`\<the`	如 *theater* 或 *the* 之类的单词
`the\>`	如 *breathe* 或 *the* 之类的单词
`\<the\>`	单词 *the*

Ed、sed 或 grep 模式	匹配的内容
`0\{5,\}`	在一行中 5 个或更多的零
`[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}`	U.S. 社会安全号码（*nnn-nn-nnnn*）
`\(why\).*\`	出现两个 *why* 的一行
`\([[:alpha:]_][[:alnum:]_.]*\) = \1;`	C/C++ 简单的赋值语句

搜索和代替的示例

表 6-3 中的示例列出了对 sed 或 ex 有效的元字符。注意 ex 命令要以一个冒号开头。空白符用□来标记，制表符用⟹ 来标记。

表 6-3：搜索和代替

命令	结果
`s/.*/( & )/`	重新对整个的行执行动作，但要增加圆括号
`s/.*/mv & &.old/`	将一个单词列表（每行一个单词）转变成 mv 命令
`/^$/d`	删除空白行
`:g/^$/d`	和上面相同，用在 ex 编辑器中
`/^[□⟹]*$/d`	删除空白行，加上只包含空白符或制表符的行
`:g/^[□⟹]*$/d`	和上面相同，用于 ex 编辑器中
`s/□□*/□/g`	将一个或多个空白符变成一个空白符
`:%s/□□*/□/g`	和上面相同，用于 ex 编辑器
`:s/[0-9]/Item &:/`	将数字转变为项目标号（在当前行）
`:s`	重复对第一个出现的替代

表 6-3：搜索和代替（续）

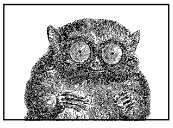
命令	结果
`:&`	和上面相同
`:sg`	和上面相似，但是对该行所有出现的替代进行重复
`:&g`	和上面相同
`:%&g`	重复全部的替代（即对所有的行）
`:.,$s/Fortran/\U&/g`	对于当前行至最后一行，将单词改变成大写
`:%s/.*/\L&/`	将整个文件的单词改变成小写
`:s/\<./\u&/g`	对当前行的每个单词的第一个字母改成大写（对标题有用）
`:%s/yes/No/g`	全部将一个单词改成 *No*
`:%s/Yes/~/g`	全部将一个不同的单词改变成 *No*（先前的代替）

最后，举一些关于调换单词位置的 sed 示例。下面是一个简单的两个单词的调换：

```
s/die or do/do or die/                    调换单词
```

真正的技术在于用拥有的缓冲区来调换变量模式的位置。例如：

```
s/\([Dd]ie\) or \([Dd]o\)/\2 or \1/     用拥有的缓冲区来调换
```



第七章

emacs 编辑器

本章讲述以下内容：

- 介绍
- 按组概述命令
- 按键概述命令
- 按名称概述命令

要想获得更多的关于 emacs 的信息，可以参见列在参考文献中的《Learning GNU Emacs》。

介绍

虽然 emacs 不是 SVR4 或 Solaris（注 1）的一部分，但该文本编辑器可用于很多 Unix 系统，因为它是对 vi 的一个受欢迎的代替。这本书为 GNU emacs（版本 20.3）提供了资料，可以从自由软件基金会（Free Software Foundation）获得（*http://www.gnu.org*）。

为了启动一个 emacs 编辑会话，键入：

```
emacs [file]
```

注 1： Sun 研究组编程环境，可独立于 Sun 而利用，和 GNU emacs 派生出来的 Xemacs 一起出现。

在一些系统中，GNU emacs 是通过键入 gmacs 而不是通过键入 emacs 来调用的。

注意事项

emacs 命令要使用 CTRL 键和 Meta 键（Meta 通常是 Esc 键）。在本章中，符号 C- 表示同时按下 CTRL 键和后面的那个字符。M- 的用法类似 C- 的用法，只不过按下的是 Meta 键。当用 Esc 键模拟 Meta 时，在敲击下一个键时没有必要按下 Meta 键。但假如用户的键盘本身有 Meta 键的时候，那么它就如同 CTRL 或 Shift 键，用户应当将它与其他的键同时按下。

在下面的命令表中，第一栏列出了按键，最后一栏是对它的描述。如果有中间一栏，则列出了命令名。该命令可以通过键入 M-x 后跟着命令名来访问。假如用户不能确定命令的名称，可以键入空白符或回车，那么 emacs 会列出用户所需的命令。

因为 emacs 是一个相当全面的编辑器，包含了成千上万的命令，为了快速地找到必须省略一些命令。用户可以通过键入 C-h（帮助）或 M-x（命令名）来浏览命令集。

必要的命令

假如你刚刚接触 emacs，可以参照在这里列出的最重要的命令：

按键	描述
C-h	进入在线帮助系统
C-x C-s	保存文件
C-x C-c	退出 emacs
C-x u	取消最后的编辑（可以重复）
C-g	放弃当前的命令操作
C-p	上一行或字符
C-n	下一行或字符
C-f	前一行或字符
C-b	后一行或字符
C-v	前一屏
M-v	后一屏
C-s	向前搜索字符
C-r	向后搜索字符
C-d	删除后面的字符
Del	删除前面的字符

常见的问题

一个很常见的问题是在终端上的Del或Backspace键并不和习惯上一样，它并不删除光标前的字符。相反，它调入一个帮助提示。这是由于不兼容的终端引起的。一个稳妥的做法是在你的home目录创建一个名称为`.emacs`的文件（或编辑一个已经存在的文件），并添加以下的行：

```
(keyboard-translate ?\C-h ?\C-?)
(keyboard-translate ?\C-\\ ?\C-h)
```

那么Del或Backspace键就会起作用了，并且可以通过按C-\来调入帮助（任意选择键的次序）。

另一个问题是当你从远程终端登入时C-s可能会导致终端挂起。这是由一个以前比较流行的握手协议造成的。可以通过按C-q重新启动终端，但并不能帮助进入包含C-s的命令。惟一的解决办法（除了用一个更先进的拔号协议外）是创建一个新的键组合来代替C-s。

按组概述命令

注意：`C-`指CTRL键，`M-`指Meta键。

文件处理命令

按键	命令名	描述
C-x C-f	find-file	查找文件并读文件
C-x C-v	find-alternate-file	读另一个文件；代替由C-x C-f读的文件
C-x i	insert-file	在光标位置插入文件
C-x C-s	save-buffer	保存文件（可能会挂起终端；用C-q重启）
C-x C-w	write-file	写缓存内容到文件
C-x C-c	save-buffers-kill-emacs	退出`emacs`
C-z	suspend-emacs	挂起`emacs`（用`exit`或`fg`重启）

光标移动命令

按键	命令名	描述
C-f	forward-char	移动到下一个字符（右）
C-b	backward-char	移动到前一个字符（左）

按键	命令名	描述
C-p	previous-line	移动到上一行（上面的）
C-n	next-line	移动到下一行（下面的）
M-f	forward-word	移动至下一个词
M-b	backward-word	移动到上一个词
C-a	Beginning-of-line	移动至行的开头
C-e	end-of-line	移动到行尾
M-a	backward-sentence	移动到下一句
M-e	forward-sentence	移动至前一句
M-{	backward-paragraph	移动至下一段
M-}	forward-paragraph	移动至上一段
C-v	scroll-up	向前滚动一屏
M-v	scroll-down	向后滚动一屏
C-x [	backward-page	向后移动一页
C-x]	forward-page	向前移动一页
M->	end-of-buffer	移动到文件尾
M-<	beginning-of-buffer	移动至文件开头
无	goto-line	到文件的第 *n* 行
无	goto-char	到文件的第 *n* 个字符
C-l	recenter	重画屏幕，当前行在屏幕中心
M-n	digit-argument	重复下一个命令 *n* 次
C-u n	universal-argument	重复下一个命令 *n* 次

删除命令

按键	命令名	描述
Del	backward-delete-char	删除前面的字符
C-d	delete-char	删除光标下的字符
M-Del	backward-kill-word	删除前面的单词
M-d	kill-word	删除光标所在的单词
C-k	kill-line	从光标处删到行尾
M-k	kill-sentence	删除光标所在的句子
C-x Del	backward-kill-sentence	删除前面的句子

按键	命令名	描述
C-y	yank	恢复用户所删除的
C-w	kill-region	删除一块已标记的区域（参见下面的部分）
无	backward-kill-paragraph	删除前面的段
无	kill-paragraph	从光标处删除至段尾

段和区域

按键	命令名	描述
C-@	set-mark-command	对一个区域的开头（或结尾）做标记
C-Space	（和上面相同）	
C-x C-p	mark-page	对页做标记
C-x C-x	exchange-point-and-mark	交换光标和标记的位置
C-x h	mark-whole-buffer	对缓冲区做标记
M-q	fill-paragraph	重新格式化段
无	fill-region	重新格式化区域内的个别段
M-h	mark-paragraph	对段做标记

停止和取消命令

按键	命令	描述
C-g	keyboard-quit	终止当前的命令
C-x u	advertised-undo	取消最后的编辑（可以重复地执行）
无	revert-buffer	重新恢复缓冲区至最后存储文件的状态（或自动存储）

调换命令

按键	命令	描述
C-t	transpose-chars	调换两个字母
M-t	transpose-words	调换两个单词
C-x C-t	transpose-lines	调换两行
无	transpose-sentences	调换两个句子
无	transpose-paragraphs	调换两段

“大写”命令

按键	命令	描述
M-c	capitalize-word	对单词的第一个字母大写
M-u	Upcase-word	对单词大写
M-l	downcase-word	对单词小写
M- 一; M-c	negative-argument; capitalize-word	对前面的单词的第一个字母大写
M- 一 M-u	negative-argument; upcase-word	对前面的单词大写
M- 一 M-l	negative-argument; downcase-word	对前面的单词小写
无	capitalize-region	对区域的第一个词大写
C-x C-u	upcase-region	对区域大写
C-x C-l	downcase-region	对区域小写

词缩写命令

按键	命令名	描述
无	abbrev-mode	进入（或退出）单词缩写模式
C-x a i g	inverse-add-global-abbrev	显示全局缩写，然后是定义
C-x a i l	inverse-add-local-abbrev	显示局部缩写，然后是定义
无	unexpand-abbrev	取消最后单词的缩写
无	write-abbrev-file	写单词缩写文件
无	edit-abbrevs	编辑单词缩写
无	list-abbrevs	查看单词缩写
无	kill-all-abbrevs	取消这个会话的缩写

缓冲区处理命令

按键	命令名	描述
C-x b	switch-to-buffer	移向指定的缓冲区
C-x C-b	list-buffers	显示缓冲区列表
C-x k	kill-buffer	删除指定的缓冲区

按键	命令名	描述
无	kill-some-buffers	查询删除每个缓冲区
无	rename-buffer	将缓冲区名改变成指定的名称
C-x s	save-some-buffers	询问是否存储每个修改的缓冲区

窗口命令

按键	命令名	描述
C-x 2	split-window-vertically	将当前的窗口分成两个，一个在另一个顶部
C-x 3	split-window-horizontally	将当前的窗口分成并排的两个
C-x >	scroll-right	向右滚动窗口
C-x <	scroll-left	向左滚动窗口
C-x o	other-window	移到另一个窗口
C-x 0	delete-window	删除当前的窗口
C-x 1	delete-other-windows	删除除这个之外的所有窗口
无	delete-windows-o	删除所有在一个给定的缓冲区上的窗口
C-x ^	enlarge-window	扩大窗口
无	shrink-window	缩小窗口
C-x }	enlarge-window-horizontally	使窗口变宽
C-x {	shrink-window-horizontally	使窗口变窄
M-C-v	scroll-other-window	滚动其他窗口
C-x 4 f	find-file-other-window	在另一个窗口找一个文件
C-x 4 b	switch-to-buffer-other-window	在另一个窗口选择一个缓冲区
C-x 5 f	find-file-other-frame	在一个新图文框中查找一个文件
C-x 5 b	switch-to-buffer-other-frame	在另一个图文框中选择一个缓冲区
无	compare-windows	比较两个缓冲区；显示第一个不同点

特殊的 shell 字符

按键	命令名	描述
C-c C-c	comint-interrupt-subjob	终止当前作业
C-c C-d	comint-send-eof	文件的结尾字符
C-c C-u	comint-kill-input	删除当前行
C-c C-w	backward-kill-word	删除前面的单词
C-c C-z	comint-stop-subjob	挂起当前作业

缩进命令

按键	命令名	描述
C-x .	set-fill-prefix	用该行的开头到光标列的字符作为“填充前缀”。这个前缀由前添加到该段的每一行。通过在第一列键入该命令可取消前缀
无	indented-text-mode	主模式：每个制表符定义后继行的缩进
无	text-mode	退出缩进的文本模式；返回至文本模式
M-C-\	indent-region	对一个区域缩进以匹配区域的第一行
M-m	back-to-indentation	移动光标到行的第一个字符
M-C-o	split-line	拆分光标所在的行；缩进到光标列
无	fill-individual-paragraphs	重新格式化缩进的段，保持缩进

居中命令

按键	命令名	描述
M-s	center-line	将光标所在的行居中
无	center-paragraph	将光标所在的段居中
无	center-region	将当前定义的区域居中

宏命令

按键	命令名	描述
C-x (	start-kbd-macro	开始宏定义
C-x)	end-kbd-macro	结束宏定义
C-x e	call-last-kbd-macro	执行最后定义的宏
M-n C-x e	digit-argument and call-last-kbd-macro	执行最后定义的宏 *n* 次
C-u C-x (	universal-argument and start-kbd-macro	执行最后定义的宏，然后增加按键
无	name-last-kbd-macro	对最后创建的宏命名（存储之前）
无	insert-keyboard-macro	将指定的宏插入文件
无	load-file	装载存储的宏文件

按键	命令名	描述
无	macroname	执行一个存储的键盘宏
C-x q	kbd-macro-query	在一个宏定义中插入一个查询
C-u C-x q	无	在一个宏定义中插入一个递归的编辑
M-C-c	exit-recursive-edit	退出一个递归的编辑

基本的缩进命令

按键	命令名	描述
M-C-\	indent-region	缩进一个区域以匹配区域中的第一行
M-m	back-to-indentation	移动至行上的第一个非空白符
M-^	delete-indentation	连接这一行至前面的行

详细的信息帮助命令

按键	命令名	描述
C-h a	command-apropos	涉及这个概念的命令
无	apropos	涉及这个概念的函数和变量
C-h c	describe-key-briefly	这个按键顺序执行的命令
C-h b	describe-bindings	这个缓冲区绑定的所有组合键
C-h k	describe-key	这个按键顺序执行的命令，并且它完成了什么工作
C-h l	view-lossage	最后键入的 100 个字符是什么
C-h w	where-is	这个命令的绑定键是什么
C-h f	describe-function	这个函数的作用是什么
C-h v	describe-variable	这个变量指的是什么，它的值是多少
C-h m	describe-mode	告诉我当前缓冲区所在的模式
C-h s	describe-syntax	这个缓冲区的语法表

帮助命令

按键	命令名	描述
C-h t	help-with-tutorial	运行 emacs 指南
C-h i	info	启动该信息文档阅读器
C-h n	view-emacs-news	查看修改 emacs 信息

按键	命令名	描述
C-h C-c	describe-copying	查看 emacs 一般公共许可
C-h C-d	describe-distribution	从 FSF 中查看一个排序的 emacs 信息
C-h C-w	describe-no-warranty	查看 emacs（非）授权

按键概述命令

以下两个字母表展示了 emacs 命令。提醒：C- 指 CTRL 键；M- 指 Meta 键。

CTRL 键序列

按键	命令名	描述
C-@	set-mark-command	标记一个区域的开头（或结尾）
C-Space	（和上面的相同）	
C-]	无	退出递归编辑并且退出查询替代
C-a	beginning-of-line	移动至行的开头
C-b	backward-char	向后移动一个字符（左）
C-c C-c	comint-interrupt-subjob	终止当前作业
C-c C-d	comint-send-eof	文件结束字符
C-c C-u	comint-kill-input	删除当前行
C-c C-w	backward-kill-word	删除前面的单词
C-c C-z	comint-stop-subjob	挂起当前作业
C-d	delete-char	删除光标下的字符
C-e	end-of-line	移动至行尾
C-f	forward-char	向前移动一个字符（右）
C-g	keyboard-quit	终止当前命令
C-h	help-command	进入在线帮助系统
C-h a	command-apropos	涉及这个概念的命令
C-h b	describe-bindings	所有的这个缓冲区的绑定键
C-h C-c	describe-copying	查看 emacs 一般公共许可
C-h C-d	describe-distribution	从 FSF 中查看排序的 emacs 信息

按键	命令名	描述
C-h C-w	describe-no-warranty	查看 emacs（非）授权
C-h c	describe-key-briefly	这个按键顺序执行的命令
C-h f	describe-function	这个函数的功能
C-h i	info	启动信息文档阅读器
C-h k	describe-key	这个按键顺序执行命令，它的功能
C-h l	view-lossage	键入的最后 100 个字符
C-h m	describe-mode	告诉我当前缓冲区的模式
C-h n	view-emacs-news	查看修改 emacs 的信息
C-h s	describe-syntax	这个缓冲区的语法表
C-h t	help-with-tutorial	运行 emacs 指南
C-h v	describe-variable	这个变量指的什么，并且它的值是多少
C-h w	where-is	这个命令的绑定键是什么
C-k	kill-line	从光标处删到行尾
C-l	recenter	行重画屏幕，当前行在屏幕中心
C-n	next-line	移动到下一行（向下）
C-p	previous-line	移动到前面的行（向上）
C-r Meta	无	开始向后非递增搜索
C-r	无	重复向后非递增搜索
C-r	无	进入递归编辑（在查询替代期间）
C-r	isearch-backward	开始向后递增的搜索
C-s Meta	无	开始向前非递增搜索
C-s	无	重复向前非递增搜索
C-s	isearch-forward	开始向前递增搜索
C-t	transpose-chars	调换两个字母
C-u n	universal-argument	重复下一个命令 *n* 次
C-u C-x (	universal-argument and start-kbd-macro	执行最后定义的宏，然后增加按键
C-u C-x q	无	在宏定义中插入递归编辑
C-v	scroll-up	向前移动一屏

按键	命令名	描述
C-w	kill-region	删除一个标记的区域
C-x (	start-kbd-macro	开始宏定义
C-x)	end-kbd-macro	结束宏定义
C-x [	backward-page	向后翻一页
C-x]	forward-page	向前翻一页
C-x ^	enlarge-window	扩大窗口
C-x {	shrink-window-horizontally	使窗口变窄
C-x }	enlarge-window-horizontally	使窗口变宽
C-x <	scroll-left	向左滚动窗口
C-x >	scroll-right	向右滚动窗口
C-x .	set-fill-prefix	用从行的开头到光标列的字符作为“ 填充的前缀 ”。这个前缀由前添加到段的每一行。通过在第 1 列键入这个命令取消前缀
C-x 0	delete-window	删除当前的窗口
C-x 1	delete-other-windows	删除除了这一个之外的所有窗口
C-x 2	split-window-vertically	将当前的窗口分成两个, 一个在另一个的顶部
C-x 3	split-window-horizontally	将当前的窗口分成并排的两个
C-x 4 b	switch-to-buffer-other-window	选择在其他窗口的一个缓冲区
C-x 4 f	find-file-other-window	在其他的窗口寻找一个文件
C-x 5 b	switch-to-buffer-other-frame	选择在另一个图文框中的一个缓冲区
C-x 5 f	find-file-other-frame	在一个新的图文框中寻找一个文件
C-x C-b	list-buffers	显示缓冲区列表
C-x C-c	save-buffers-kill-emacs	退出 emacs
C-x C-f	find-file	找到文件并读文件
C-x C-l	downcase-region	对区域小写
C-x C-p	mark-page	标记页
C-x C-q	无	选定缓冲区的只读状态

按键	命令名	描述
C-x C-s	save-buffer	存储文件（可能会挂起终端；用 C-q 可重启）
C-x C-t	transpose-lines	调换两行
C-x C-u	upcase-region	对区域大写
C-x C-v	find-alternate-file	读一个替换的文件，代替用 C-x C-f 读的文件
C-x C-w	write-file	写缓冲区内容到文件
C-x C-x	exchange-point-and-mark	调换光标和标记的位置
C-x DEL	backward-kill-sentence	删除前面的句子
C-x a i g	inverse-add-global-abbrev	显示全部缩写，然后是定义
C-x a i l	inverse-add-local-abbrev	显示局部缩写，然后是定义
C-x b	switch-to-buffer	移动到指定的缓冲区
C-x e	call-last-kbd-macro	执行最后定义的宏
C-x h	mark-whole-buffer	标记缓冲区
C-x i	insert-file	在光标处插入文件
C-x k	kill-buffer	删除指定的缓冲区
C-x o	other-window	移动到另一个窗口
C-x q	kbd-macro-query	在宏定义处插入一个查询
C-x s	save-some-buffers	询问是否存储每个修改的缓冲区
C-x u	advertised-undo	取消最后的编辑（可以重复执行）
C-y	yank	恢复你所删除的
C-z	suspend-emacs	挂起 `emacs`（用 `exit` 或 `fg` 重启）

Meta 键序列

按键	命令名	描述
Meta	无	退出一个查询替换或成功的搜索
M- — M-c	negative-argument; capitalize-word	对前面的单词的第一个字母大写

按键	命令名	描述
M- — M-l	negative-argument; downcase-word	对前面的单词小写
M- — M-u	negative-argument; upcase-word	对前面的单词大写
M-$	spell-word	对光标后的单词进行拼写检查
M-<	beginning-of-buffer	移到文件的开头
M->	end-of-buffer	移到文件的结尾
M-{	backward-paragraph	向后移动一段
M-}	forward-paragraph	向前移动一段
M-^	delete-indentation	连接这行到前面的行
M-*n*	digit-argument	重复下一个命令 *n* 次
M-*n* C-x e	digit-argument and call-last-kbd-macro	执行最后定义的宏 *n* 次
M-a	backward-sentence	向后移动一句
M-b	backward-word	向后移动一个单词
M-C-\	indent-region	缩进一个区域以匹配区域的第一行
M-C-c	exit-recursive-edit	退出一个递归编辑
M-C-o	split-line	分割光标所在的行；缩进到光标列
M-C-v	scroll-other-window	滚动另外的窗口
M-c	capitalize-word	对单词的第一个字母大写
M-d	kill-word	删除光标所在的单词
M-DEL	backward-kill-word	删除前面的单词
M-e	forward-sentence	向前移动一个句子
M-f	forward-word	向前移动一个单词
无	fill-region	重新格式化一个区域中的个别段
M-h	mark-paragraph	标记段
M-k	kill-sentence	删除光标所在的句子
M-l	downcase-word	对单词小写
M-m	back-to-indentation	移动光标到行上的第一个非空字符

按键	命令名	描述
M-q	fill-paragraph	重新格式化段
M-s	center-line	将光标所在的行置于中心
M-t	transpose-words	调换两个单词
M-u	upcase-word	对单词大写
M-v	scroll-down	向下移动一屏
M-x	无	通过命令名访问命令

按名称概述命令

下面列出的 emacs 命令是按命令名依字母次序排列的。用 M-x 可以访问命令名。提醒：C- 指 CTRL 键；M- 指 Meta 键。

命令名	按键	描述
macroname	无	执行存储过的键盘宏
abbrev-mode	无	进入（或退出）单词缩写模式
advertised-undo	C-x u	取消最后的编辑（可以重复执行）
apropos	无	涉及这个命令的函数和变量
back-to-indentation	M-m	移动光标到行的第一个非空字符
backward-char	C-b	向后移动一个字符（左）
backward-delete-char	Del	删除前面的字符
backward-kill-paragraph	无	删除前面的段
backward-kill-sentence	C-x Del	删除前面的句子
backward-kill-word	C-c C-w	删除前面的单词
backward-kill-word	M-Del	删除前面的单词
backward-page	C-x [	向后翻一页
backward-paragraph	M-{	向后移动一段
backward-sentence	M-a	向后移动一句
backward-word	M-b	向后移动一个单词
beginning-of-buffer	M-<	移到文件的开头

命令名	按键	描述
beginning-of-line	C-a	移到行的开头
call-last-kbd-macro	C-x e	执行最后定义的宏
capitalize-region	无	对区域的第一个字母大写
capitalize-word	M-c	对单词的第一个字母大写
center-line	M-s	将光标所在的行置于中心
center-paragraph	无	将光标所在的段置于中心
center-region	无	将当前定义的区域置于中心
comint-interrupt-subjob	C-c C-c	终止当前作业
comint-kill-input	C-c C-u	删除当前行
comint-send-eof	C-c C-d	文件结束字符
comint-stop-subjob	C-c C-z	挂起当前作业
command-apropos	C-h a	涉及这个概念的命令
compare-windows	无	比较两个缓冲区；显示第一个不同点
delete-char	C-d	删除光标下的字符
delete-indentation	M-^	连接这一行到前面的一行
delete-other-windows	C-x 1	删除除了这一个之外的所有窗口
delete-window	C-x 0	删除当前窗口
delete-windows-on	无	删除一个给出缓冲区中的所有窗口
describe-bindings	C-h b	在这个缓冲区里所有的绑定键
describe-copying	C-h C-c	查看 emacs 一般公共许可
describe-distribution	C-h C-d	从 FSF 中查看关于排序的 emacs 的信息
describe-function	C-h f	函数的功能
describe-key	C-h k	这个按键序列运行的命令，其功能是什么
describe-key-briefly	C-h c	这个按键序列运行的命令
describe-mode	C-h m	告诉我当前缓冲区所处的模式
describe-no-warranty	C-h C-w	查看 emacs（非）授权
describe-syntax	C-h s	这个缓冲区的语法表
describe-variable	C-h v	这个变量指的是什么，并且它的值是多少

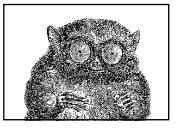
命令名	按键	描述
digit-argument and call- last-kbd-macro	M-*n* C-x e	执行最后定义的宏 *n* 次
digit-argument	M-n	重复下一个命令 *n* 次
downcase-region	C-x C-l	对区域小写
downcase-word	M-l	对单词小写
edit-abbrevs	无	编辑单词缩写
end-kbd-macro	C-x)	结束宏定义
end-of-buffer	M->	移到文件结尾
end-of-line	C-e	移到行结尾
enlarge-window	C-x ^	扩大窗口
enlarge-window-horizontally	C-x	使窗口变宽
exchange-point-and-mark	C-x C-x	交换光标和标记的位置
exit-recursive-edit	M-C-c	退出一个递归的编辑
fill-individual-paragraphs	无	重新格式化缩进的段，保持缩进
fill-paragraph	M-q	重新格式化段
fill-region	无	重新格式化在区域内的单个的段
find-alternate-file	C-x C-v	读一个替换的文件，替换由用C-x C-f读的文件
find-file	C-x C-f	找到文件并读它
find-file-other-frame	C-x 5 f	在一个新图文框中找文件
find-file-other-window	C-x 4 f	在另外的窗口中找文件
forward-char	C-f	向前移动一个字符（右）
forward-page	C-x]	向前翻一页
forward-paragraph	M-}	向前移动一段
forward-sentence	M-e	向前移动一句
forward-word	M-f	向前移动一个单词
goto-char	无	到文件的第 *n* 个字符
goto-line	无	到文件的第 *n* 行

命令名	按键	描述
help-command	C-h	进入在线帮助系统
help-with-tutorial	C-h t	运行 emacs 指南
indent-region	M-C-\	缩进一个区域以匹配区域中的第一行
indented-text-mode	无	主模式 ;每个制表符为后续行定义了一个新的缩进
info	C-h i	启动信息文档阅读器
insert-file	C-x i	在光标处插入文件
insert-keyboard-macro	无	向一个文件中插入指定的宏
inverse-add-global-abbrev	C-x a i g	显示全部缩写，然后是定义
inverse-add-local-abbrev	C-x a i l	显示局部缩写，然后是定义
isearch-backward	C-r	开始向后递增的搜索
isearch-backward-regexp	C-r	和上面相同，但搜索的是正则表达式
isearch-forward	C-s	开始向前递增的搜索
isearch-forward-regexp	C-r	相同，但搜索的是正则表达式
kbd-macro-query	C-x q	往宏定义中插入一个查询
keyboard-quit	C-g	终止当前的命令
kill-all-abbrevs	无	取消这个会话的缩写
kill-buffer	C-x k	删除指定的缓冲区
kill-line	C-k	从光标处删除到行尾
kill-paragraph	无	从光标处删除到段尾
kill-region	C-w	删除一个标记的段
kill-sentence	M-k	删除光标所在的句子
kill-some-buffers	无	询问删除每个缓冲区
kill-word	M-d	删除光标处的单词
list-abbrevs	无	查看单词缩写
list-buffers	C-x C-b	显示缓冲区列表
load-file	无	装载所存储的宏文件
mark-page	C-x C-p	标记页

命令名	按键	描述
mark-paragraph	M-h	标记段
mark-whole-buffer	C-x h	标记缓冲区
name-last-kbd-macro	无	命名创建的最后的宏（存储之前）
negative-argument; capitalize-word	M- — M-c	对前面的单词的第一个字母大写
negative-argument; downcase-word	M- — M-l	对前面的单词小写
negative-argument; upcase-word	M- — M-u	对前面的单词大写
next-line	C-n	移到下一行（向下）
other-window	C-x o	移到其他的窗口
previous-line	C-p	移到前面的行（向上）
query-replace-regexp	C-% Meta	查询－代替一个正则表达式
recenter	C-l	重画屏幕，当前行在屏幕的中心
rename-buffer	无	改变缓冲区名称为指定的名称
replace-regexp	无	无条件地代替一个正则表达式
re-search-backward	无	仅向后搜索正则表达式
re-search-forward	无	仅向前搜索正则表达式
revert-buffer	无	恢复缓冲区成文件最后被存储的状态（或自动存储）
save-buffer	C-x C-s	存储文件（可能会挂起终端；用 C-q 重启）
save-buffers-kill-emacs	C-x C-c	退出 emacs
save-some-buffers	C-x s	询问是否存储每个修改的缓冲区
scroll-down	M-v	向后滚动一屏
scroll-left	C-x<	向左滚动窗口
scroll-other-window	M-C-v	滚动其他的窗口
scroll-right	C-x>	向右滚动窗口
scroll-up	C-v	向前翻一屏

命令名	按键	描述
set-fill-prefix	C-x.	用行的开头到光标列之间的字符作为“ 填充前缀 ”。这个前缀由前添加到该段的每一行。通过在第 1 列键入这个命令可取消该前缀
set-mark-command	C-@ or C-Space	标记一个区域的开头（或结尾）
shrink-window	无	缩小窗口
shrink-window-horizontally	C-x {	使窗口变窄
spell-buffer	无	对当前缓冲区进行拼写检查
spell-region	无	对当前区域进行拼写检查
spell-string	无	对小型缓冲区中键入的字符串进行拼写检查
spell-word	M-$	对光标之后的词进行拼写检查
split-line	M-C-o	在光标处分割行；缩进到光标列
split-window-vertically	C-x 2	将当前窗口分成两个，一个在另一个的顶端
split-window-horizontally	C-x 3	将当前窗口分成两个并排的窗口
start-kbd-macro	C-x (	开始宏定义
suspend-emacs	C-z	挂起 `emacs`（用 `exit` 或 `fg` 重启）
switch-to-buffer	C-x b	移动到指定的缓冲区
switch-to-buffer-other-frame	C-x 5 b	在另一图文框中选择一个缓冲区
switch-to-buffer-other-window	C-x 4 b	在其他窗口中选择一个缓冲区
text-mode	无	退出缩进文本模式；返回到文本模式
transpose-chars	C-t	调换两个字母
transpose-lines	C-x C-t	调换两行
transpose-paragraphs	无	调换两段
transpose-sentences	无	调换两个句子
transpose-words	M-t	调换两个单词
unexpand-abbrev	无	取消最后的单词缩写
universal-argument	C-u *n*	重复下一个命令 *n* 次

命令名	按键	描述
universal-argument and start-kbd-macro	C-u C-x (	执行最后定义的宏，然后增加按键
upcase-region	C-x C-u	对区域大写
upcase-word	M-u	对单词大写
view-emacs-news	C-h n	查看 emacs 更新消息
view-lossage	C-h l	用户键入的最后 100 个字符
where-is	C-h w	这个命令绑定的键
write-abbrev-file	无	写单词缩写文件
write-file	C-x C-w	将缓冲区内容写到文件
yank	C-y	恢复所删除的



第八章

vi 编辑器

本章包括以下内容：

- 复习 vi 操作
- 移动命令
- 编辑命令
- 保存和退出
- 访问多文件
- 和 Unix 交互
- 宏
- 杂项命令
- 键的字母顺序列表
- 设置 vi

vi 发音为 “ vee eye ”。

除了最初的Unix vi之外，还有很多自由使用的克隆的vi。参考文献中的《Learning the vi Editor》介绍了最初的 vi 和其他克隆的 vi 编辑器。

复习 vi 操作

本部分提供以下复习内容：

- 命令行语法
- vi 模式
- vi 命令的语法
- 状态行命令

命令行语法

开始一个 vi 会话有三个最一般的方式：

```
vi file
vi +n file
vi +/pattern file
```

你可以打开*file*来编辑，可以选择*n*行来编辑*file*或者在匹配*pattern*的第一个行来编辑*file*。假如没有指定*file*，vi会打开一个空的缓冲区。参见第二章可以获得有关vi命令行选项的更多信息。

注意：vi和ex实际上是同样的程序，因此，要熟悉ex命令集也可以参见第九章中的材料。

命令模式

一旦打开文件，你就处于命令模式中了。在命令模式下，可以：

- 调用插入模式
- 发出编辑命令
- 移动光标到文件中不同的位置
- 调用 ex 命令
- 调用一个 Unix shell
- 保存或退出文件的当前版本

插入模式

在插入模式中，可以在文件中输入新文本。按Esc键可以退出插入模式并回到命令模式。下面的命令可以调用插入模式：

a 在光标后追加

A 在行尾追加

c 开始更改操作

C 更改到行尾

i 在光标前插入

I 在行的开头处插入

o 打开当前行下面的一行

O 打开当前行上面的一行

R 开始覆盖文本

s 替代一个字符

S 替代整个行

vi 命令的语法

在 vi 中，命令通常有以下的形式：

[*n*] *operator* [*m*] *object*

基本的编辑操作符 *operator* 是：

c 开始一个更改

d 开始一个删除

y 开始移出（或拷贝）

假如操作的对象是当前行的话，用操作符：cc，dd，yy 也是一样的。否则编辑操作符的对象要由光标移动命令或模式匹配命令来指定。*n* 和 *m* 是操作符执行的次数，或操作符执行的对象的数目。假如既指定了 *n* 又指定了 *m*，结果就是 $n \times m$。

一个对象可以表现为以下文本块：

单词（word）	包括空白字符（空格或制表符）或标点符号在内的任何字符。一个大写的对象是一种只识别空白字符的变体形式。
句子（sentence）	结尾是 .，! 或 ? 后跟着两个空白符。
段（paragraph）	直到下一个空行或由 para= 选项定义的段宏。
节（section）	直到下一个由 sect= 选项定义的节头。

示例

`2cw` 更改下两个单词。

`d}` 删除直到下一段。

`d^` 往回删除到行的开头。

`5yy` 拷贝下面的 5 行。

`y]]` 拷贝直到下一节。

状态行命令

用户输入的大多数命令并不回显到屏幕上。而屏幕底部的状态行用于回显这些输入的命令：

`/` 向前搜索模式。

`?` 向后搜索模式。

`:` 调用一个 ex 命令。

`!` 调用一个 Unix 命令并把缓冲区中的一个对象作为它的输入，并且用命令的输出来代替它。

在状态行上输入的命令必须通过按回车键来开始。另外，错误消息和来自 `CTRL-G` 命令的输出会显示在状态行上。

移动命令

在一个命令前的数字会用来重复该移动。移动命令也是用于更改、删除或移出（yank）的对象。

字符

`h`, `j`, `k`, `l` 左，下，上，右（←，↓，↑，→）。

`Spacebar` 右。

文本

`w`, `W`, `b`, `B` 向前，向后一个单词。

e, E 单词的结尾。

), (下一个的开头，当前句子。

}, { 下一个的开头，当前段。

]], [[下一个的开头，当前节。

行

0, $ 当前行的第一个，最后一个位置。

^ 当前行的第一个非空字符。

+, - 下一行，前一行的第一个字符。

Return 下一行的第一个字符。

n| 当前行的第 *n* 列。

H 屏幕的顶行。

M 屏幕当中的行。

L 屏幕最后的行。

*n*H 顶行后的 *n* 行。

*n*L 最后行前的 *n* 行。

屏幕

CTRL-F
CTRL-B 向前、向后滚动一屏。

CTRL-D
CTRL-U 向下、向上滚动二分之一屏。

CTRL-E
CTRL-Y 在窗口底部、顶部显示更多的行。

z Return 将光标所在的行重新定位到屏幕顶部。

z. 将光标所在的行重新定位到屏幕中部。

z- 将光标所在的行重新定位到屏幕底部。

CTRL-L
CTRL-R 重画屏幕（不滚动）。

搜索

/*text* 向前搜索 *text*。

n 重复先前的搜索。

N 按相反的方向重复搜索。

/ 重复向前搜索。

? 向后重复先前的搜索。

?*text* 向后搜索 *text*。

/*text*/+*n* 到 *text* 后的第 *n* 行。

?*text*?-*n* 到 *text* 前的第 *n* 行。

% 寻找当前圆括号、花括号、方括号内的匹配。

f*x* 向前搜索到当前行的 *x*。

F*x* 向后搜索到当前行的 *x*。

t*x* 向前搜索到当前行中 *x* 前的字符。

T*x* 向后搜索到当前行中 *x* 后的字符。

, 改变最后执行的 f、F、t 或 T 的方向。

; 重复最后的字符搜索（f、F、t 或 T）。

行号

CTRL-G 显示当前行号。

*n*G 移动到行号 *n*。

G 移动到文件的最后一行。

:*n* 移动到行号 *n*。

标记位置

m*x* 用字符 *x* 标记当前的位置。

`*x* 移动光标到标记 *x*。

'*x* 移动到包含 *x* 的行的开头。

`` 返回到前面的标记（或到一个搜索前面的位置）。

`''` 和上相似，但返回到行的开头。

编辑命令

c、d 和 y 是最基本的编辑操作符。

插入新的文本

a	在光标后追加。
A	追加到行尾。
i	插入到光标前。
I	在行的开头插入。
o	打开光标下一行。
O	打开光标上一行。
Esc	终止插入模式。
CTRL-J	向下移动一行。
Return	向下移动一行。
CTRL-I	插入一个制表符。
CTRL-T	移动到下一个设置的制表符处。
Backspace	往回移动一个字符。
CTRL-H	往回移动一个字符。
CTRL-U	删除当前行。
CTRL-V	将下一个字符引起来。
CTRL-W	往回移动一个单词。

改变和删除文本

cw	更改单词。
cc	更改行。
C	从当前的位置到行尾更改文本。
dd	删除当前行。

*n*dd　　删除 *n* 行。

D　　删除行余下的部分。

dw　　删除一个单词。

d}　　删除一直到下一段。

d^　　往回删除到行的开头。

d/*pat*　　删除直到第一个模式的出现。

d*n*　　删除直到下一个模式的出现。

df*a*　　在当前行上删除直到 *a* 并且包括 *a*。

dt*a*　　在当前行上删除直到 *a*（不包括 *a*）。

dL　　在屏幕上删除直到最后一行。

dG　　删除到文件结束。

p　　在光标后插入最后删除的文本。

P　　在光标前插入最后删除的文本。

r*x*　　用 *x* 代替字符。

Rtext　　用新的 text（覆盖）代替，开始于光标处。

s　　替换字符。

4s　　替换 4 个字符。

S　　替换整个行。

u　　取消最后的更改。

U　　恢复当前行。

x　　删除当前光标位置。

X　　往回删除一个字符。

5X　　删除前面的 5 个字符。

.　　重复最后的更改。

~　　改变大小写。

拷贝和移动

Y　　拷贝当前行到新的缓冲区。

`yy` 拷贝当前行。

`"`*x*`yy` 将当前行移出到缓冲区 *x*。

`"`*x*`d` 删除到缓冲区 *x* 中。

`"`*X*`d` 删除并追加到缓冲区 *x* 中。

`"`*x*`p` 放置缓冲区 *x* 的内容。

`y]]` 拷贝直至下一个节的头。

`ye` 拷贝至单词尾。

缓冲区名称是字母 `a`～`z`。大写名称追加文本至指定的缓冲区。

保存和退出

向一个文件写意味着保存该编辑并且修改文件的更改时间。

`ZZ` 退出 `vi`，只有做出改变时才写文件。

`:x` 和 `ZZ` 一样。

`:wq` 写并退出文件。

`:w` 写文件。

`:w` *file* 保存拷贝到文件 *file*。

`:`*n*`,`*m*`w` *file* 将 *n* 至 *m* 行写到新文件 *file*。

`:`*n*`,`*m*`w >>` *file* 将 *n* 至 *m* 行追加到现有文件 *file*。

`:w!` 写文件（覆盖保护）。

`:w!` *file* 用当前的缓冲区覆盖文件 *file*。

`:w %.`*new* 写当前命名为 *file* 的缓冲区，并以 *file.new* 命名。

`:q` 退出 `vi`。

`:q!` 退出 `vi`（放弃编辑）。

`Q` 退出 `vi` 并调用 `ex`。

`:vi` `Q` 命令后返回到 `vi`。

`:e` *file2* 不离开 `vi` 编辑 *file2*。

`:`*n* 编辑下一个文件。

`:e!` 在最后的写时返回当前文件的版本。

`:e #`	编辑备用文件。
`%`	当前文件名。
`#`	备用文件名。

访问多文件

`:e` *`file`*	编辑另一个文件 *file*；当前文件变成备用文件。
`:e!`	在最后的写时返回当前文件的版本。
`:e+`*`file`*	在 *file* 尾开始编辑。
`:e+`*`n file`*	在第 *n* 行打开 *file*。
`:e #`	在备用文件中打开到先前的位置。
`:ta` *`tag`*	在 *tag* 处编辑文件。
`:n`	编辑下一个文件。
`:n!`	强制下一个文件。
`:n` *`files`*	指定 *files* 的新列表。
`CTRL-G`	显示当前文件和行号。
`:args`	显示要编辑的多个文件。
`:rew`	重新倒回多个文件的列表至顶部。

和 Unix 交互

`:r` *`file`*	将光标后文件 *file* 的内容读入。
`:r !`*`command`*	将当前行后命令 *command* 的输出读入。
`:`*`m`*`r !`*`command`*	和上面意义相同，但位置在第 *n* 行后（0 为文件顶部）。
`:!`*`command`*	运行 *command* 然后返回。
`!`*`object command`*	发送缓冲区 *object* 到 Unix 命令 *command*；用输出替换。
`:`*`n`*`,`*`m`*`! command`	发送 *n* 至 *m* 行到 *command*；用输出替换。
`n``!!`*`command`*	发送 *n* 行到 Unix 命令 *command*；用输出替换。
`!!`	重复最后的系统命令。
`:sh`	创建子 shell；用 *EOF* 返回文件。

CTRL-Z	挂起编辑器，用 fg 重新开始。
:so *file*	从 *file* 中读并执行 ex 命令。

宏

:ab *in out*	用 *in* 作为 *out* 的缩写。
:unab *in*	删除 *in* 的缩写。
:ab	列出缩写。
:map *c sequence*	以字符 *c* 映射命令的序列 *sequence*。
:unmap *c*	删除字符 *c* 的映射。
:map	列出映射的字符。
:map! *c sequence*	映射字符 *c* 到输入模式 *sequence*。
:unmap! *c*	删除输入模式映射（可能需要用CTRL-V将那个字符引起来）。
:map!	列出映射为输入模式的字符。

下面的字符在命令模式中没有使用，可以映射为用户定义的命令：

字母

```
g K q V v
```

CTRL 组合键

```
^A ^K ^O ^W ^X
```

符号

```
_ * \ =
```

（注意：“ = ”在设置 Lisp 模式时会被 vi 编辑器用到。vi 的不同版本会用到这些字符中的一部分，因此在使用它们之前最好先检查 vi 版本号。）

杂项命令

J	连接两行。
:j!	连接两行，并保留空白符。
<<	对这行向左移动一个移位宽度（默认是 8 个空格位）。
>>	对这行向右移动一个移位宽度（默认是 8 个空格位）。

>}　向右移动至段尾。

<%　向左移动直到匹配的圆括号、花括号或方括号（光标必须在那个匹配的符号上）。

键的字母顺序列表

为了简捷起见，用 ^ 来标记控制字符。

^]　在光标下的文本上执行一个标签查找。

a　将文本追加到光标后。

A　追加文本到行尾。

^A　未使用。

b　向后直到当前行中的单词的开头。

B　向后直到单词的开头处，忽略标点符号。

^B　向后滚动一个窗口。

c　更改操作符。

C　更改到当前行的结尾。

^C　在命令模式中未使用；结束插入模式（stty 中断字符）。

d　删除操作符。

D　删除到当前行的尾部。

^D　向下滚动半个窗口（命令模式）。

^D　向后移动一个制表位（插入模式）。

e　移动到单词尾。

E　移动至单词尾，忽略标点符号。

^E　在窗口底部显示更多的行。

f　在当前行向前查找键入的下一个字符。

F　在当前行向后查找键入的下一个字符。

^F　向前滚动一个窗口。

g 未使用。

G 到指定的行或文件结尾。

^G 在状态行上输出关于文件的信息。

h 左箭头光标键。

H 移动光标到 Home 位置处。

^H 左箭头光标键；在插入模式中为退格键（Backspace）。

i 在光标前插入文本。

I 在行上第一个非空格字符前插入文本。

^I 在命令模式中未使用；在插入模式中和制表符（Tab 键）一样。

j 下箭头光标键。

J 连接两行。

^J 下箭头光标键；在插入模式中为向下移动一行。

k 上箭头光标键。

K 未使用。

^K 未使用。

l 右箭头光标键。

L 移动光标至窗口中最后的位置。

^L 重画屏幕。

m 在寄存器中（a～z）标记当前光标位置。

M 移动光标到窗口的中部位置。

^M 回车。

n 重复最后的搜索命令。

N 从相反的方向重复最后的搜索命令。

^N 下箭头光标键。

o 打开当前行下面的行。

O 打开当前行上面的行。

^O 未使用。

p 将移出或删除的文本放置在光标后或光标下。

P 将移出或删除的文本放置在光标前或光标上。

^P 上箭头光标键。

q 未使用。

Q 退出 vi 并调用 ex。

^Q 未使用（在一些终端上，为重新开始数据流）。

r 用你键入的下一个字符代替在光标处的字符。

R 代替字符。

^R 重画屏幕。

s 更改光标下的字符为键入的字符。

S 更改整个行。

^S 未使用（在一些终端上，为停止数据流）。

t 向前移动光标到键入的下一个字符前。

T 向后移动光标到键入的下一个字符后。

^T 返回到标签栈中先前的位置（Solaris vi 命令模式）。
假如设置了 autoindent，则缩进另一个制表位（插入模式）。

u 取消最后做的更改。

U 恢复当前行，放弃更改。

^U 向上滚动半个窗口。

v 未使用。

V 未使用。

^V 在命令模式中未使用；在插入模式中，将下一个字符引起来。

w 移动到下一个单词的开头。

W 移动到下一个单词的开头，不考虑标点符号。

^W 在命令模式中未使用；在插入模式中退到单词的开头。

x 删除光标下的字符。

X 删除光标前的字符。

^X 未使用。

y 移出或拷贝操作符。

Y 对当前行执行拷贝。

^Y 在窗口顶部显示更多的行。

z 重新定位包含光标的行。z 后必须跟着: Return（重新定位行到屏幕的顶部）、. （重新定位行到屏幕的中部）、或 -（重新定位行到屏幕的底部）。

ZZ 退出编辑器，保存更改。

^Z 挂起 vi（只在有作业控制的系统上起作用）。

设置 vi

本部分介绍以下内容：

- :set 命令
- 和:set 一起的有效选项
- .exrc 文件示例

:set 命令

该命令允许用户指定选项, 从而改变用户编辑环境的功能。选项可以放置到~/.exrc 文件中或在 vi 会话期设置。

假如放置到 .exrc 文件中，则不应该带冒号：

:set *x* 使选项 *x* 有效。

:set no*x* 禁止选项 *x*。

:set *x*=val 给选项 *x* 赋值。

:set 显示更改的选项。

:set all 显示所有的选项。

:set *x*? 显示选项 *x* 的值。

:set 使用的选项

表 8-1 包括了重要的 set 命令选项的详细解释。第一栏中选项是以字母顺序列出的，如果选项可以缩写的话，其缩写包括在圆括号中。如果没有提供明确的 set 命令（或者手动或在 .exrc 中），那么第二栏显示 vi 使用的默认设置。最后一栏解释了当选项有效时的意义。

表 8-1 :set 选项

选项	默认	描述
autoindent(ai)	noai	在插入模式，缩进每一行与上面或下面的行保持平齐。和 shiftwidth 选项一起使用
autoprint(ap)	ap	在每个编辑器命令后显示改变（对于全局代替只显示最后的代替）
autowrite(aw)	noaw	在用 :n 打开另一个文件或用 :! 给出 Unix 命令之前，假如有更改的话则自动写（保存）该文件
beautify(bf)	nobf	在输入期间忽略所有的控制字符（除了制表符、换行符或进纸符）
directory(dir)	/tmp	对 ex/vi 存储缓冲区文件的目录进行命名。（目录必须可写）
edcompatible	noedcompatible	记住最近替代命令（全局确认）使用的标记，并且下一个替代命令会使用它们。尽管有该名称，实际上没有 ed 版本会支持这种方式
errobells(eb)	errorbells	当错误发生时响铃
exrc(ex)	noexrc	允许执行驻留在用户的主目录外的 .exrc 文件
hardtabs(ht)	8	定义终端硬件制表符的边界
ignorecase(ic)	noic	在搜索期间忽略大小写
lisp	nolisp	按合适的 Lisp 格式插入缩进。( )、{}、[[和]] 被修改成 Lisp 意义
list	nolist	将制表符输出为 ^I；用 $ 标记行尾（使用 list 来告诉结束字符是制表符还是空格）
magic	magic	通配符 .（点）、*（星号）和 [] （方括号）在模式中都有特殊的意义

表 8-1 :set 选项（续）

选项	默认	描述
mesg	mesg	当在 vi 中编辑时允许在终端上显示系统消息
novice	nonovice	要求使用长 ex 命令名，例如 copy 或 read
number(nu)	nonu	在编辑会话期间，在屏幕的左边显示行数
open	open	允许从 ex 进入 *open* 或 *visual* 模式。虽然并没在 Solaris vi 中，但这个选项通常用在 vi 编辑器中，或许就在你的 Unix vi 版本中
optimize(opt)	noopt	当输出多行时消除行尾的回车，当输出开头是空白符（空格或制表符）的行时加速输出到哑终端上
paragraphs(para)	IPLPPPQP LIpplpipbp	用 { 或 } 定义移动的段定界符。值中的那对字符就是开始段的 troff 宏的名称
prompt	prompt	当给出 vi 的 Q 命令时，显示 ex 提示符（:）
readonly(ro)	noro	如果在写后不用 ! 的话，任何的写（保存）文件都会失败（和 w、ZZ 或 autowrite 一起生效）
redraw(re)		无论何时执行编辑（换句话说，插入模式会覆盖现有的字符，并且删除的行会迅速关闭）都会重画屏幕。默认依靠线速和终端类型 noredraw在哑终端上在慢速时是有用的 :删除的行会显示 @，并且插入的文本会覆盖现有的文本直到按 Escape 为止
remap	remap	允许嵌套的映射序列
report	5	无论什么时候你做了影响到一定行数的编辑时，在状态行上显示消息。例如，6dd会报告消息“ 6 lines deleted（删除了 6 行）”
scroll	[1/2window]	使用 ^D 和 ^U 命令滚动的行数
sections(sect)	SHNHH HU	定义 [[和]] 移动的节定界符。值中的那对字符是开始一个节的 troff 宏的名称
shell(sh)	/bin/sh	用于 shell 转义（:!）和 shell 命令（:sh）的 shell 路径名。默认值来自于 shell 环境，依据不同的系统而变化

表 8-1 :set 选项（续）

选项	默认	描述
shiftwidth(sw)	8	当使用 autoindent 选项时定义向后（^D）制表符的空格数，并且用于 << 和 >> 命令
showmatch(sm)	nosm	在 vi 中，当键入) 或 } 时，光标会移动来匹配 (或 {（假如没有匹配，会响铃报错）。在编程时有用
showmode	noshowmode	在插入模式中，在提示行上显示一个消息指出你正在做的插入类型。例如“ OPEN MODE ”或“ APPEND MODE ”
slowopen(slow)		在插入期间拖延显示。默认依靠线速和终端类型
tabstop(ts)	8	在编辑会话期间定义一个制表符的缩进的空格数（打印机仍旧用为 8 的系统制表符）
taglength(tl)	0	定义对标签有重要意义的字符的数目。默认（零）意味着所有的字符都是重要的
tags	*tags/usr/lib/tags*	定义包含标签的文件的路径名（参见 Unix ctags 命令）（默认情况下，vi 会在当前目录和 /usr/lib/tags 目录中搜索文件 tags）
tagstack	tagstack	在一个栈上允许标签堆叠
term		设置终端类型
terse	noterse	显示较短的错误消息
timeout(to)	timeout	在 1 秒钟后键盘映射超时[a]
ttytype		设置终端类型，它只是 term 的另一个名称
warn	warn	显示警告消息,“ No write since last change（上次改变后没有写）”
window(w)		在屏幕上显示文件的一定数目的行。默认依靠线速和终端类型
wrapscan(ws)	ws	搜索绕回至文件的任意一端

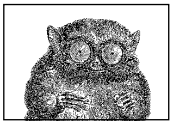
表 8-1 :set 选项（续）

选项	默认	描述
wrapmargin(wm)	0	定义右边页空白。假如大于 0，自动插入回车来分断行
writeany(wa)	nowa	允许保存到任何文件

a： 当你有几个键的映射时（例如，:map zzz 3dw），可能会用到 timeout。否则你必须在 1 秒钟内键入 zzz。当你有一个光标键插入模式映射时（例如，:map! ^[OB ^[ja），应当用 timeout，否则直到你键入另一个键时 vi 才会对 Escape 响应。

.exrc 文件示例

```
set nowrapscan wrapmargin=7
set sections=SeAhBhChDh nomesg
map q :w^M:n^M
map v dwElp
ab ORA O'Reilly & Associates, Inc.
```



第九章

ex 编辑器

ex 行编辑器是屏幕编辑器 vi 的基础。在 ex 中的命令只在当前行或在一个文件中一个行的范围内起作用。更常见的是，你会在 vi 内使用 ex。在 vi 中，ex 命令前有一个冒号，并且通过按回车键输入。

你也可以从它自身调用 ex（从命令行），就和调用 vi 一样（可以用这种方式执行一个 ex 脚本）。也可以用 vi 命令 Q 来退出 vi 编辑器并进入 ex。

本章提供了以下内容：

- ex 命令语法
- 按字母顺序概述 ex 命令

要想获得更多的信息，可以参见参考文献中列出的《Learning the vi Editor》。

ex 命令语法

从 vi 中输入一个 ex 命令，键入：

```
:[address] command [options]
```

起始处的：指的是一个 ex 命令。当键入该命令时，它会回显到状态行上。通过按回车键执行命令。*address* 是行号或 *command* 对象的行的范围。下面会提到 *options* 和 *address*。ex 命令也会在“ 按字母顺序概述 ex 命令 ”部分加以解释。

退出 ex 有以下几种方式：

`:x` 退出（保存更改并退出）。

`:q!` 不保存更改就退出。

`:vi` 在当前文件上转换到 vi 编辑器。

address（地址）

假如没有给出 *address*，则当前行就是命令的对象。假如地址指定了一个行的范围，该格式为：

x,*y*

x 和 *y* 是第一个和最后的地址行（在缓冲区中 *x* 必须在 *y* 之前）。*x* 和 *y* 每一个都可以是一个行号或一个符号。在解释 *y* 以前用“ ; ”而不是“ , ”来设置当前行到 *x*。符号 1,$ 代表了文件中所有的行，和 % 一样。

地址符号

`1,$`	文件中所有的行。
`x,y`	从第 *x* 行到第 *y* 行。
`x;y`	从第 *x* 行到第 *y* 行，并且当前行重新设置到 *x*。
`0`	文件的顶部。
`.`	当前行。
`n`	绝对行号 *n*。
`$`	最后一行。
`%`	所有的行；和 `1,$` 一样。
x-*n*	*x* 以前的 *n* 行。
x+*n*	*x* 以后的 *n* 行。
`-[`*n*`]`	前面的一行或 *n* 行
`+[`*n*`]`	后面的一行或 *n* 行。
`'x`	用 *x* 标记的行。
`''`	先前的标记。
/pattern/	向前到匹配模式 *pattern* 的行。

?*pattern*? 向后到匹配模式 *pattern* 的行。

如果需了解模式的话可以参见第六章。

options

! 指出一个命令的变体形式，覆盖正常行为。

count

重复执行命令的次数。和在 vi 中的命令不一样，count 不能在命令之前，因为 ex 命令前的一个数会当作一个行地址来对待。例如，d3 指的是从当前行开始删除 3 行，而 3d 则指的是删除第 3 行。

file

命令影响到的文件的名称。% 代表当前文件；# 代表先前的文件。

按字母顺序概述 ex 命令

ex 命令可以按指定的惟一缩写形式输入。在本列表中，命令全名显示在页边空白处，而其缩写形式用在语法行中。示例是假定在 vi 中键入的，所以它们包括“：”提示符。

abbrev

ab [*string text*]

定义 *string*，它将被转化为 *text*。假如没有指定 *string* 和 *text* 的话，会列出所有的当前缩写。

示例

注意：当键入 ^V 后跟着回车时，会出现 ^M。

```
:ab ora O'Reilly & Associates, Inc.
:ab id Name:^MRank:^MPhone:
```

append

[*address*] a[!]
text
.

在指定的 *address* 追加 *text*，或者，如果没有指定 *address*，则在当前的 *address* 追加 *text*。在输入期间，加一个“！”来约束 autoindent 设置。即，假如允许 autoindent 的话，则“！”会禁止它。

args

`ar`

输出参数列表中的成员（在命令行上命名的文件），而当前的参数输出在括号（[]）中。

change

[*address*] c[!]
text
.

用 *text* 代替指定的行。加一个“！”在输入 *text* 期间改变 autoindent 设置。

copy

[*address*] co *destination*

拷贝包括在 *address* 中的行到指定的 *destination* 地址中。命令 t（“ to ”缩写）是 copy 的同义词。

示例

```
:1,10 co 50
```

delete

[*address*] d [*buffer*]

删除包括在 *address* 中的行。如果指定了 *buffer*，则保存或追加文本到指定的缓冲区。缓冲区的名称是小写字母 a～z。大写字母名称会追加文本到缓冲区。

示例

```
:/Part I/,/Part II/-1d      删除到“ Part II ”上的行
:/main/+d                   删除“ main ”下面的行
:.,$d                       从该行删除到最后一行
```

edit

e[!] [+*n*] [*filename*]

在 *filename* 上开始编辑。如果没有给出 *filename*，则从当前文件的拷贝开始。加一个“！”用来编辑新文件，即使自从最后一次更改以来没有保存当前文件。如果带 +*n* 参数，则开始在第 *n* 行上编辑。或 *n* 可能是一个形式为 */pattern* 的模式。

edit

示例

```
:e file
:e#
:e!
```

file

`f [`*filename*`]`

在认为没有处于编辑状态时，将当前文件的名称改变为*filename*。假如没有指定*filename*，则输出当前文件的状态。

示例

```
:f %.new
```

global

[*address*] g[!]/*pattern*/[*commands*]

在包含*pattern*的所有行上执行*commands*，或者如果指定了*address*，则在该范围内的所有行上执行*commands*。假如没有指定*commands*，则输出所有这样的行。加一个“！”用于指定在所有不包括*pattern*的行上执行*commands*。也可参见 **v**。

示例

```
:g/Unix/p
:g/Name:/s/tom/Tom/
```

insert

[*address*] i[!]
text
.

在指定的*address*前的行上插入*text*，或者在没有指定*address*的情况下，在当前位置上插入*text*。加一个“！”用于在输入文本期间改变 autoindent 设置。

join

[*address*] j[!] [*count*]

将指定范围的文本放到一行上，并且带有白空间调整，即在句点“.”之后提供两个空格的位置，而在“)”之后没有空格，否则有一个空格字符。加“！”用于阻止白空间调整。

示例

```
:1,5j!          连接第一个5行，保留白空间
```

k

[*address*] k *char*

用 *char* 来标记给出的 *address*，*char* 是一个单个的小写字母。以后可返回到带有 'x 的行。k 和 mark 等价。

list

[*address*] l [*count*]

输出指定的行，制表符显示为 ^I，并且行的结尾显示为 $。l 只是 :set list 的一个临时版本。

map

map[!] [*char commands*]

定义一个名称为 *char* 的键盘宏作为指定命令 *commands* 的序列。*char* 通常是一个单个的字符，或序列 #*n*，在键盘上表现为一个功能键。用一个“！”去创建一个输入模式宏。如果没有参数则列出当前定义的宏。

示例

```
:map K dwwP           调换两个词
:map q :w^M:n^M       写当前的文件；转到下一个
:map! + ^[bi(^[ea)    将先前的词括在括号中
```

mark

[*address*] ma *char*

用 *char* 标记指定的行，*char* 是一个单个的小写字母。以后可以返回带有 'x 的行。和 k 意义相同。

move

[*address*] m *destination*

将由 *address* 指定的行移动到 *destination* 地址处。

示例

```
:.,/Note/m /END/     移动文本块到包括“ END ”的行后
```

next

n[!] [[+*n*] *filelist*]

编辑来自命令行参数列表中的下一个文件。用 args 可以列出这些文件。假如提供了 *filelist*，就用 *filelist* 代替当前的参数列表并且开始在第一个文件上编辑。参数 +*n* 表示开始在第 *n* 行编辑。或 *n* 可以是形式为 */pattern* 的一个模式。

next

示例

```
:n chap*          开始在所有的“ chapter ”文件上编辑
```

number

[*address*] nu [*count*]

输出由*address*指定的每一个行，前面是它的缓冲区行号。用#作为number的一个替换的缩写。*count*指定了要显示的行数，从*address*开始。

open

[*address*] o [/*pattern*/]

在*address*指定的行进入开放模式(vi)，或在匹配*pattern*的行进入开放模式。用Q可退出开放模式。开放模式可以让你使用常规的vi命令，但一次只有一行。在慢的拨号线上（或在远距离的Internet telnet连接上）可能会有用。

preserve

pre

保存当前的编辑器缓冲区好像系统将要崩溃一样。

print

[*address*] p [*count*]

输出由*address*指定的行。*count*指定要输出的行数，从*address*开始。P是另一个缩写形式。

示例

```
:100;+5p          显示100行以及下面的5行
```

put

[*address*] pu [*char*]

恢复先前删除或从*char*指定的缓冲区中移出的行到*address*指定的行上。假如没有指定*char*，则恢复最后删除或移出的文本。

quit

q[!]

终止当前的编辑会话。用!放弃自从最后一次保存以来做出的更改。假如编辑会话包括从未访问过的参数列表中的相应文件，则通过键入q!或键入q两次退出。

read	[*address*] r *filename* 拷贝 *filename* 的文本到由 *address* 指定的行后。假如没有指定 *filename*，则使用当前的文件名。 示例 `:0r $HOME/data` 读文件到当前文件的顶部
read	[*address*] r !*command* 读 Unix 命令 *command* 的输出到由 *address* 指定的行后的文本中。 示例 `:$r !cal` 将日历放到文件尾
recover	rec [*file*] 从系统保存区域恢复文件 *file*。
rewind	rew[!] 倒出参数列表并在列表中的第一个文件开始编辑。加一个“！”用于指定即使自从最后的更改后当前文件没有保存的情况下也要倒出参数列表。

set	se *parameter1 parameter2...*
	
	用每个*parameter*设置一个选项的值, 或者, 假如没有提供*parameter*, 则输出由默认所更改的选项。为了绑定选项，每个 *parameter* 都可以用短语*option*或*nooption*表示 ;其他的选项可以用语法*option=value*来赋值。如果给出 `all` 的话则输出当前的设置。形式 `set` *option*?显示 *opton* 的值。可以参见第八章中 `set` 选项的列表。
	
	示例
	`:set nows wm=10`
	`:set all`

shell	sh 创建一个新的 shell。当 shell 终止时重新开始编辑。

source

so *file*

从 *file* 中读并执行 ex 命令。

示例

```
:so $HOME/.exrc
```

substitute

[*address*] s [/*pattern*/*replacement*/] [*options*][*count*]

用 *replacement* 取代指定行上 *pattern* 的每个实例。假如省略 *pattern* 和 *replacement*，则重复最后的替代。*count* 指定了要替代的行数，从 *address* 开始。可以参见第六章相应的示例（要讲清楚的是命令名在 Solaris vi 中不起作用）。

options

- c 在每个更改之前提示确定。
- g 替代每个行上所有的 *pattern* 的实例（全局）。
- p 输出做出替代的最后一行。

示例

```
:1,10s/yes/no/g          在第一个 10 行上替代
:%s/[Hh]ello/Hi/gc       确定全局替代
:s/Fortran/\U&/ 3        大写在下面三行上的 "Fortran"
```

t

[*address*] t *destination*

拷贝包括在 *address* 中的行到指定的 *destination* 处。t 和 copy 意义相同。

示例

```
:%t$                拷贝文件并将其增加到尾部
```

tag

[*address*] ta *tag*

将编辑的中心转到 *tag*。

示例

运行 ctags，然后转到包含 *myfunction* 的文件。

tag

```
:!ctags *.c
:tag myfunction
```

unabbreviate

una *word*

从缩写列表中删除 *word*。

undo

u

取消由最后的编辑命令做出的更改。

unmap

unm[!] char

从键盘宏列表中删除 *char*。用 ! 来删除输入模式宏。

v

[*address*] v/*pattern*/[*commands*]

在所有的不包含 *pattern* 的行上执行 *commands*。假如没有指定 *commands*，则输出所有这样的行。v 和 g! 意义相同。

示例

```
:v/#include/d        删除所有除“ #include ”之外的行
```

version

ve

输出编辑器当前的版本号和最后更改的日期。

visual

[*address*] vi [*type*] [*count*]

在由 *address* 指定的行上进入可视模式（vi）。用 Q 可以退出。*type* 可以是一个 -、^ 或 .（参见 z 命令）。*count* 指定了一个最初的窗口尺寸。

visual

vi [+*n*] *file*

开始在可视模式（vi）中编辑 *file*，可以选择从第 *n* 行开始。

write

[*address*] w[!] [[>>] *file*]

将由 *address* 指定的行写到 *file*，或者假如没有给定 *address*，则写缓冲区中的所有内容。假如也省略了 *file*，则存储缓冲区的内容到当前

write

的文件名。假如用到 >>*file* 的话，则写内容到指定的 *file* 的结尾。加一个“！”用于迫使编辑器写并且覆盖 *file* 的任何当前内容。

示例

```
:1,10w name_list        拷贝第一个 10 行到 name_list
:50w >> name_list       现追加第 50 行
```

write

[*address*] w !*command*

将由 *address* 指定的行写到 *command*。

示例

```
:1,66w !pr -h myfile | lp     输出文件的第一页
```

wq

wq[!]

写并且退出在一个移动中的文件。该文件总是被写。标记迫使编辑器写并覆盖文件中的任何当前内容。

xit

x

自从最后的一次写文件后如被更改则写文件；然后退出。

yank

[*address*] ya [*char*][*count*]

将由 *address* 指定的行放置在指定的缓冲区 *char* 中。假如没有给出 *char*，则将行放置至一般的缓冲区中。*count* 指定要移动的行数，从 *address* 开始。

示例

```
:101,200 ya a
```

z

[*address*] z [*type*][*count*]

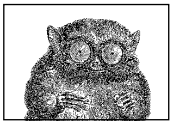
在顶部输出由 *address* 指定的行的文本的窗口。*count* 指定要显示的行数。

type（类型）

+ 将指定的行放置到窗口的顶部（默认）。

z	- 将指定的行放置至窗口的底部。 . 将指定的行放置到窗口的中心。 ^ 输出先前的窗口。 = 将指定的行放置到窗口的中心并离开当前行。
!	[*address*] !*command* 在一个 shell 中执行 Unix 命令 *command*。假如指定了 *address*，则将包含在 *addresss* 的行作为 *command* 的标准输入。并且用输出和错误输出代替那些行（这称为通过命令过滤文本）。 示例 `:!ls` 列出当前目录中的文件 `:11,20!sort -f` 对当前文件的 11~20 行排序
=	[*address*]= 输出由 *address* 指定的行的行号。默认为最后行的行号。
<>	[*address*]<[*count*] 或 [*address*]>[*count*] 向左(<)或向右(>)移动由 *address* 指定的行。移动行的过程中只有开头的空格和制表符会增加或删除。*count* 指出要移动的行数，从 *address* 开始。`shiftwidth` 选项控制移动的列数。重复“ < ”或“ > ”会增加移动数。例如，:>>> 移动是 :> 的三倍。
address	*address* 输出由 *address* 指定的行。
RETURN	输出文件中的下一行。
&	[*address*]&[*options*][*count*] 重复先前的替代（s）命令。*count* 指出要替代的行数，从 *address* 开始。*options* 和替代命令是一样的。

&	示例 `:s/Overdue/Paid/` 在当前行上替代一次 `:g/Status/&` 在所有的“ Status ”行上重新执行替代
~	[*address*]~[*count*] 用来自最近的s（替代）命令中的代替模式以代替最后用过的正则表达式（尽管来自于一次搜索而不是来自于s命令）。这的确是难以理解的。要想了解到详细的信息可以参见《Learning the vi Editor》中的第六章。



第十章

sed 编辑器

本章提供了以下内容：

- 对 sed 概念的综述
- 命令行语法
- sed 命令的语法
- 按组概述 sed 命令
- 按字母顺序概述 sed 命令

要想看到更多的内容，可以参见参考文献中的《sed & awk》。

对 sed 概念的综述

sed是一个非交互式的, 或者说面向流的编辑器。它对脚本进行解释并按脚本执行动作。之所以说它是面向流的，是因为和很多 Unix 程序一样，sed 是通过程序输入流并定向到标准输出的。例如，sort 是面向流的；vi 则不是。sed的输入通常来自于文件或管道, 但可以从键盘引导。输出默认到屏幕但可以在一个文件中捕获或发送到一个管道。

sed 的一般用法

- 自动编辑一个或多个文件。

- 简化反复编辑多文件。
- 写转化程序。

sed 有如下操作

- 拷贝输入的每一行到一个 “ 模式空间 ”，这是一个执行编辑操作的内部缓冲区。
- 在一个 sed 脚本中的所有编辑命令按顺序应用到输入的每一行。
- 编辑命令应用到所有的行（全局），除非有行地址限制。
- 假如一个命令改变了输入，后续的命令和地址检测会应用到模式空间中的当前行，而不是最初的输入行。
- 最初的输入文件不改变, 因为编辑命令修改的是最初输入行的一个拷贝。该拷贝被发送到标准输出（但也可重定向到一个文件）。
- sed 也支持 “ 保持空间 ”，这是一个分离的缓冲区，可以用来保存数据以备以后提取。

命令行语法

调用 sed 的语法有两种形式：

```
sed [-n] [-e] 'command' file(s)
sed [-n] -f scriptfile file(s)
```

第一种形式允许你在命令行上指定一个编辑命令, 该命令用单引号引起来。第二种形式允许你指定一个脚本文件 *scriptfile*，即一个包含 sed 命令的文件。两种形式可以一起使用，并且可以使用多次。假如没有指定 *file(s)*，则 sed 从标准输入上读取。

一般有下列的选项：

`-n` 禁止默认输出；sed 只显示由 p 命令或 s 命令中的 p 标志指定的行。

`-e` *`cmd`*
下一个参数是一个编辑命令。在指定多脚本或命令时是有用的。

`-f` *`file`*
下一个参数是一个包含编辑命令的文件。

假如脚本的第一行是 #n，sed 的执行就如同已指定了 -n 一样。

sed 命令的语法

sed 命令有下面的一般形式：

```
[address[,address]][!]command [arguments]
```

sed拷贝输入的每行到模式空间中。sed指令由地址*addresses*和编辑命令*commands*组成。假如命令的地址匹配模式空间中的行，则命令就应用到该行。假如一个命令没有地址，则该命令将应用到每个输入行。假如一个命令改变了模式空间的内容，则后续的命令和地址就应用到模式空间中的当前行，而不是最初的输入行。

命令*commands*由一个单个的字母或符号构成，它们会在以后以按组和字母顺序的形式加以介绍。参数*auguments*包括提供给 b 或 t 的标号，还有提供给 r 或 w 的文件名以及 s 的替代标记。地址 *addresses* 会在下面加以介绍。

模式地址

一个 sed 命令可以指定 0、1 或 2 个地址。一个地址可以是一个行号，符号 $（最后一行），或一个由斜线围起来的正则表达式（/*pattern*/）。正则表达式在第六章中介绍。另外，\n可匹配模式空间中的任何换行符（来自 N 命令），但不能是模式空间结尾的换行符。

假如命令指定了	那么命令应用到
没有地址	每一个输入行
一个地址	匹配地址的任何行。一些命令只接受一个地址：a、I、r、q 和 =
两个由冒号分隔的地址	第一个匹配的行和所有的后续行直到并包括匹配第二个地址的行
一个跟有!的地址	所有不匹配该地址的行

示例

s/xx/yy/g	在所有的行上替代（都发生）
/BSD/d	删除包含 BSD 的行
/^BEGIN/,/^END/p	输出 BEGIN 和 END 之间的，并包括 BEGIN 和 END
/SAVE/!d	删除不包括 SAVE 的任何行

`/BEGIN/,/END/!s/xx/yy/g` 在所有的行上替代，除了 BEGIN 和 END 之间的

在 sed 中，括号“ {} ”用于将一个地址嵌套在另一个里面或应用多命令到同一个地址。

```
[/pattern/[,/pattern/]]{
command1
command2
 }
```

左括号必须结束该行，并且右括号必须自己在一行上。并且确保在括号后没有空格。

按组概述 sed 命令

在下面的列表中，sed 命令是按功能分组的，并且进行了简单的介绍。全面的介绍（包括语法和示例）会在下面的按字母顺序概述部分找到。

基本的编辑

a\ 在一行后追加文本。

c\ 代替文本（通常一个文本块）。

i\ 在一行前插入文本。

d 删除行。

s 执行替代。

y 转换字符（和 Unix 的 tr 相似）。

行信息

= 显示一行的行号。

l 以 ASCII 形式显示控制字符。

p 显示该行。

输入 / 输出处理

n 跳过当前行并且到下面的行。

r 将另一个文件的内容读到输出流。

`w` 将输入行写到另一个文件。

`q` 退出 sed 脚本（没有进一步的输出）。

移出和放置

`h` 拷贝到保持空间，清除原有内容。

`H` 拷贝到保持空间，追加到原有内容后面。

`g` 从保持空间恢复，清除目标行。

`G` 从保持空间恢复，追加到模式空间。

`x` 交换保持空间和模式空间的内容。

分支命令

`b` 分支到标号或脚本尾。

`t` 和 `b` 相似，但只在替代后分支。

`:`*`label`* 由 `t` 或 `b` 分支到的标号。

多输入处理

`N` 读输入的另一行（创建内嵌的换行符）。

`D` 删除直到内嵌的换行符。

`P` 输出直到内嵌的换行符。

按字母顺序概述 sed 命令

`#`	`#` 在 sed 脚本中开始一个注释。只在作为第一行的第一个字符才有效。（一些版本允许在任何地方注释，但最好不要依赖它。）假如脚本的第一行是 `#n`，那么执行 sed 就如同已指定了 `-n` 一样。
`:`	`:`*`label`* 标记一行用于 `b` 或 `t` 控制转移。标号最多可以包含 7 个字符。

=	[*/pattern/*]= 将由 *pattern* 定位的每行的行号写到标准输出。

a	[*address*]a\
	text
	
	追加 *text* 到由 *address* 匹配的每行的后面。假如 *text* 超过一行，必须在换行符前用反斜杠符号 \ 对它们加以隐藏。*text* 通过第一个不按这种方式隐藏的换行符来结束。该 *text* 在模式空间中无效，并且后续的命令不可能应用到它。不管在模式空间的当前行发生了什么，当完成编辑命令的列表时，这个命令的结果会发送到标准输出。
	
	示例
	`$a\`
	`This goes after the last line in the file\`
	`(marked by $). This text is escaped at the\`
	`end of each line, except for the last one.`

b	[*address1*[,*address2*]]b[*label*]
	
	无条件地转移控制到脚本中的：*label* 处。也就是，在 *label* 后的命令是应用到当前行的下一条命令。假如没有指定 *label*，则控制直接到脚本的尾部，因此不再有命令应用到当前行。
	
	示例
	`# Ignore tbl tables; resume script after TE:`
	`/^\.TS/,/^\.TE/b`

c	[*address1*[,*address2*]]c\ *text* 用 *text* 代替（更改）由地址选择的行（参见 **a** 命令了解 *text*）。当指定了行的范围时所有的行作为一个整体被 *text* 的一个拷贝代替。实际上模式空间的内容被删除，并且没有后续的编辑命令可以应用到模式空间（或 *text*）。 示例 `# Replace first 100 lines in a file:`

c

```
1,100c\
\
<First 100 names to be supplied>
```

d

[*address1*[,*address2*]]d

从模式空间中删除选定的行。因此该行不会被传递到标准输出。读一个新的输入行，并且用脚本中的第一个命令恢复编辑。

示例

```
# delete all blank lines:
/^$/d
```

D

[*address1*[,*address2*]]D

删除由N命令创建的多模式空间的第一部分（直到内嵌的换行符），并且用脚本中的第一个命令重新恢复编辑。假如这个命令清空了模式空间，则读一个新的输入行，如同执行 d 命令一样。

示例

```
# Strip multiple blank lines, leaving only one:
/^$/{
N
/^\n$/D
}
```

g

[*address1*[,*address2*]]g

将保持空间（参见**h**和**H**）中的内容粘回到模式空间中，清除模式空间原先的内容。示例显示了一个拷贝行的简单办法。

示例

这个脚本收集了所有包含单词*Item:*的行，并且将它们拷贝到文件中一个最近的位置标记处，覆盖该位置标记：

```
/Item:/H
/<Replace this line with the item list>/g
```

G

[*address1*[,*address2*]]G

和g相似, 只是一个换行符和保持空间被粘贴到模式空间的结尾而不是覆盖它。示例显示了一个剪切和粘贴行的简单办法。

示例

这个脚本收集了所有包括单词*Item:*的行, 并且将它们移到文件中一个位置标记的后面。原先的*Item:*行被删除。

```
/Item:/{
H
d
}
/Summary of items:/G
```

h

[*address1*[,*address2*]] h

拷贝模式空间到保持空间, 即一个特殊的临时缓冲区。保持空间中原先的内容被删除。可以在编辑一行前用 h 去保存它。

示例

```
# Edit a line; print the change; replay the original
/Unix/{
h
s/.* Unix \(.*\) .*/\1:/
p
x
}
```

示例输入：

```
This describes the Unix ls command.
This describes the Unix cp command.
```

示例输出：

```
ls:
This describes the Unix ls command.
cp:
This describes the Unix cp command.
```

H

[*address1*[,*address2*]]H

追加一个换行符，然后追加模式空间的内容到保持空间中。即使保持空间是空的，H仍旧追加一个换行符。H就像是一个递增的拷贝。可以参见 **g** 和 **G** 下面的示例。

i

[*address*]i\
text

在匹配 *address* 的每行前插入 *text*（要了解 *text* 可以参见 **a** 命令）。

示例

```
/Item 1/i\
The five items are listed below:
```

l

[*address1*[,*address2*]]l

列出模式空间的内容，用ASCII代码显示非打印字符。长行被折叠。

n

[*address1*[,*address2*]]n

读下一个输入行到模式空间。当前行被发送到标准输出，并且下一行变成当前行。控制传递给后面为n的命令而不是在脚本的顶端重新开始的命令。

示例

在 *ms* 宏中，一个节的头出现在一个 .NH 宏下的行上。为了输出头文本的所有行，用 sed -n 调用这个脚本。

```
/^\.NH/{
n
p
}
```

N

[*address1*[,*address2*]]N

追加下一个输入行到模式空间的内容中，这个新行和模式空间的原先内容由一个换行符分隔开（这个命令的原意是允许模式匹配越过两行）。用 \n去匹配内嵌的换行符，可越过多行去匹配模式。参见 **D** 下面的示例。

N

示例

和 **n** 下的示例相似，但要一起输出 .NH 行和头标题：

```
/^\.NH/{
N
p
}
```

连接两行（用空格代替换行符）

```
/^\.NH/{
N
s/\n/ /
p
}
```

p

[*address1*[,*address2*]]p

输出选定的行。要注意的是这可能会导致重复输出，除非用命令行选项 #n 或 -n 来禁止默认输出。通常用在改变流控制（d、n、b）的命令前并且可以阻止当前行的输出。参见 **h**、**n** 和 **N** 下的示例。

P

[*address1*[,*address2*]]P

输出由N命令创建的多行模式空间的第一部分（直到内嵌的换行符）。假如 N 不被应用到一个行的话和 p 一样。

示例

假定有两种格式的函数：

```
function(arg1, arg2)
function(arg1,
         arg2)
```

下面的脚本改变参数 2，而不用考虑它是否和函数名出现在同一行上。

```
s/function(arg1, arg2)/function(arg1, XX)/
/function(/{
N
s/arg2/XX/
P
D
}
```

q

[*address*]q

当遇到*address*的时候退出。和由先前的r或a命令追加到它的内容中的任何文本一道，选定的行首先写到输出设备上（假如默认输出没有禁止的话）。

示例

删除选定的行后面的所有内容：

```
/Garbled text follows:/q
```

只输出一个文件中的第一个50行：

```
50q
```

r

[*address*]r *file*

读*file*中的内容并将其追加到模式空间的内容之后。在r和文件名之间必须有且只允许有一个空格。

示例

```
/The list of items follows:/r item_file
```

s

[*address1*[,*address2*]]s/*pattern*/*replacement*/[*flags*]

在每一个选定的行上用*replacement*代替*pattern*。假如用到模式地址，则模式//指的是指定的最后的模式地址。可以用任何定界符。用*pattern*或*replacement*内的\可转义定界符。可以指定下列的标记：

g 在每个定位的行上代替所有的*pattern*的实例，而不只是第一个实例。

p 假如成功地替代则输出该行，假如完成了多个成功的替代，则sed会输出该行的多个拷贝。

w *file*
假如做完了一次代替则将该行写到*file*中，最大限度可以打开10个不同的文件。

n 在每个定位的行上代替*pattern*的第*n*个实例，*n*可以是1到512范围中的任何数，默认是1。

s

示例

这里有一些短的经过注释的脚本：

```
# Change third and fourth quote to ( and ):
/function/{
s/"/(/3
s/"/)/4
}

# Remove all quotes on a given line:
/Title/s/"//g

# Remove first colon and all quotes; print resulting lines:
s/://p
s/"//gp

# Change first "if" but leave "ifdef" alone:
/ifdef/!s/if/ if/
```

t

[*address1*[,*address2*]]t [*label*]

假如成功地进行了替代则在定位的行上执行检测，如果是这样的话，则分支到由:*label*（参见 **b** 和**:**）标记的行。假如没有指定 *label*，则控制分支到脚本的底部。t 命令就如同C编程语言或各种各样的编程语言中的 case 语句。检测每一种情况，当为真的时候退出该结构。

示例

假定想填充一个数据库中的空字段。你有下面的字段：

```
ID: 1 Name: greg Rate: 45
ID: 2 Name: dale
ID: 3
```

想让它成为这样：

```
ID: 1 Name: greg Rate: 45 Phone: ??
ID: 2 Name: dale Rate: ?? Phone: ??
ID: 3 Name: ???? Rate: ?? Phone: ??
```

需要检测已经存在的字段数目。这里有一个脚本（字段用制表符分隔）：

```
#n
/ID/{
s/ID: .* Name: .* Rate: .*/& Phone: ??/p
t
```

t

```
s/ID: .* Name: .*/& Rate: ?? Phone: ??/p
t
s/ID: .*/& Name: ???? Rate: ?? Phone: ??/p
}
```

w

[*address1*[,*address2*]]w *file*

追加模式空间的内容到*file*中。这个动作发生在遇到该命令时而不是模式空间被输出时。在w和*file*之间有且只能有一个空格。在一个脚本中最大限度可以有10个不同的文本被打开。假如文件不存在，这个命令会创建它；假如文件存在的话，在每次执行脚本时它的内容会被覆盖。将输出定向到同一个文件中的多个写命令会追加到文件尾。

示例

```
# Store tbl and eqn blocks in a file:
/^\.TS/,/^\.TE/w troff_stuff
/^\.EQ/,/^\.EN/w troff_stuff
```

x

[*address1*[,*address2*]]x

交换模式空间和保持空间中的内容。参见 **h** 命令中的示例。

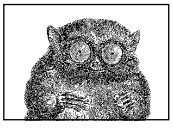
y

[*address1*[,*address2*]]y/*abc*/*xyz*/

转换字符。改变*a*的每个实例到*x*，*b*到*y*，*c*到*z*等。

示例

```
# Change item 1, 2, 3 to Item A, B, C ...
/^item [1-9]/y/i123456789/IABCDEFGHI/
```



第十一章

awk 编程语言

本章提供了以下内容：

- 概念综述
- 命令行语法
- 模式和过程
- 内置变量
- 操作符
- 变量和数组赋值
- 用户定义的函数
- 分组的 awk 函数和命令列表
- 实现限制
- 按字母顺序概述函数和命令

要想看到更详细的内容，可以参见参考文献中列出的《sed & awk》。

概念综述

awk 是一个用于处理文件的模式匹配程序，尤其当这些文件是数据库的时候。awk 的

新版本称为 nawk，提供了额外的功能（注 1）。每个现代的 Unix 系统都有一个新的 awk 的版本，并且建议使用这些新版本。

不同的系统中两个版本的名称不同。作为旧版本和新版本，有的用 oawk 和 awk，而有的分别是 awk 和 nawk。仍有其他的系统只使用新版本 awk。假如你的 awk 是旧版本，下面的示例会显示所发生的情况：

```
$  awk 1 /dev/null
awk: syntax error near line 1
awk: bailing out near line 1
```

假如是新版本的话 awk 会正常退出。

来自贝尔实验室的最新的 awk 源代码可以从 Brian Kernighan 的主页下载，它的地址是：*http://cm.bell-labs.com/~bwk*。Michael Brennan 的 mawk 也是可以使用的，可以通过匿名 FTP 从 *ftp://ftp.whidbey.net/pub/brennan/mawk1.3.3.tar.gz* 处下载。还有，自由软件基金会有一个称为 gawk 的 awk 版本也可以从 *ftp://gnudist.gnu.org/gnu/gawk/gawk-3.0.4. tar.gz* 下载。三个程序都实现“新的”awk。因此下面会提到诸如“只用于 nawk”，“应用到三个”等。gawk 有另外的功能。

应用最初的 awk，你可以：

- 将一个文本文件认为是由一个文本数据库中的记录和字段构成。
- 执行算术和字符串操作。
- 使用诸如有条件的和循环之类的编程结构体。
- 产生格式化的报表。

用 nawk，你也可以：

- 定义自己的函数。
- 从一个脚本执行 Unix 命令。
- 处理 Unix 命令的结果。
- 方便地处理命令行参数。
- 更容易地使用多输入流。

注 1： 它并不是那么新。在 1984 年添加了额外的功能，在 1987 年它第一次与 System V Release 3.1 一同发布。然而，该名字在绝大多数的系统上没有改变。

- 刷新打开的输出文件和管道（最新的贝尔实验室的 awk）。

另外，用 GNU awk（gawk），你可以：

- 用正则表达式去分离记录和字段。
- 跳到下一个文件的开头，而不只是下一个记录。
- 执行功能更强大的字符串替代。
- 检索和格式化系统时间值。

命令行语法

调用 awk 的语法有两种形式：

```
awk [options] 'script' var=value file(s)
awk [options] -f scriptfile var=value file(s)
```

可以直接从命令行上指定一个脚本，或者在脚本文件 *scriptfile* 中存储一个脚本并且用 -f 来指定它。nawk 允许多个 -f 脚本。在命令行上可以对变量赋值。该值可以是一个文本值，一个 shell 变量（$*name*），或是一个命令替代（`` `*cmd*` ``），但该值只有在执行 BEGIN 语句之后才可用。

awk 可以对一个或多个文件 *files* 进行操作。假如什么也没给出（或指定的是 " - "），awk 会从标准输入上读取。

通常使用的选项有：

`-F` *`fs`*
: 设置字段分隔符为 *fs*。这和设置系统变量 FS 一样。最初的 awk 只允许字段分隔符是一个单个的字符。nawk 则允许 *fs* 是一个正则表达式。每个输入行或记录可用空白符（空格或制表符）或由一些其他的用户定义的记录分隔符分成字段。字段可以用变量 $1，$2，...，$n 来访问，$0 指的是整个记录。

`-v` *`var=value`*
: 给变量 *var* 赋一个值 *value*。这允许在脚本开始执行前赋值（只在 nawk 中有效）。

要想在分开的行上输出每个记录的前三个（由冒号分隔的）字段：

```
awk -F: '{ print $1; print $2; print $3 }' /etc/passwd
```

更多的示例参见 " 简单的模式－过程示例 " 部分。

模式和过程

awk 脚本由模式和过程构成：

```
pattern { procedure }
```

两者都是可选的。假如省略了模式 *pattern*，则{*procedure*}就应用到所有的行；假如省略了过程{*procedure*}，则输出匹配的行。

pattern（模式）：

一个模式可以是下面的任何情况：

```
/regular expression/
relational expression
pattern-matching expression
BEGIN
END
```

- 表达式可以由引起来的字符串、数字、操作符、函数、定义了的变量，或任何预定义变量组成，预定义变量在“ 内置变量 ”部分加以了介绍。
- 正则表达式使用元字符的扩充集，并且在第六章的“ 模式匹配 ”部分加以了介绍。
- ^和$分别指一个字符串（比如字段）的开头和结尾，而不是一行的开头和结尾。特别是这些元字符不会匹配一个字符中间的内嵌换行符。
- 关系表达式使用关系操作符，在本章后面列出的“ 操作符 ”中有关系操作符。例如，$2>$1选择那些第二个字段比第一个大的行。比较可以是字符串也可以是数字。因此，依靠在$1和$2中的数据类型，awk 会做数字的或字符串的比较。这可以从一个记录改变到下一个记录。
- 模式－匹配表达式使用操作符 ~（匹配）和!~（不匹配）。可以参见本章后面的“ 操作符 ”部分。
- BEGIN 模式允许指定过程，该过程在处理第一个输入行之前发生（ 一般地，在这里用户定义全局变量）。
- END 模式让你指定在读最后的输入行之后发生的过程。
- 在 nawk 中，BEGIN 和 END 模式可以出现多次。那些合并的过程就好像一个大的过程一样。

除了 BEGIN 和 END 之外，模式也可以用布尔操作符 ||（或）、&&（与）和!（非）组合而成。一个行的范围也可以用逗号分隔的模式指定：

pattern,pattern

过程

过程由一个或多个命令、函数或变量赋值组成，用换行符或分号分隔开，并且包含在大括号内。命令分成 5 组：

- 变量或数组赋值
- 输出命令
- 内置函数
- 控制流命令
- 用户定义的函数（只用于 nawk）

简单的模式 - 过程示例：

- 输出每行的第一个字段：

```
{ print $1 }
```

- 输出包含模式 *pattern* 的所有行：

```
/pattern/
```

- 输出包含模式 *pattern* 的行的第一个字段：

```
/pattern/ { print $1 }
```

- 选择包含超过两个字段的记录：

```
NF > 2
```

- 将输入记录作为一组行来解释，直到碰到一个空行。每行都是一个单个的字段：

```
BEGIN { FS = "\n"; RS = "" }
```

- 按交换的次序输出字段 2 和字段 3，但只限于第一个字段匹配字符串“ URGENT ”的行上：

```
$1 ~ /URGENT/ { print $3, $2 }
```

- 计算并输出发现的模式 *pattern* 的数量：

```
/pattern/ { ++x }
END { print x }
```

- 加第二列的数并输出总计：

```
{ total += $2 }
END { print "column total is", total}
```

- 输出少于 20 个字符的行：

```
length($0) < 20
```

- 输出每个以 *Name:* 开始并且只包含七个字段的行：

```
NF == 7 && /^Name:/
```

- 按相反的次序输出每个输入记录的字段，每行只有一个：

```
{
      for (i = NF; i >= 1; i--)
               print $i
}
```

内置变量

版本	变量	描述
awk	FILENAME	当前文件名
	FS	字段分隔符（一个空格）
	NF	当前记录的字段数
	NR	当前记录数
	OFMT	数字（"%.6g"）和转化成字符串的输出格式
	OFS	输出字段分隔符（一个空格）
	ORS	输出记录分隔符（一个换行符）
	RS	记录分隔符（一个换行符）
	$0	整个的输入记录
	$*n*	当前记录的第 *n* 个字段；字段用 FS 分隔
nawk	ARGC	命令行上参数的数目
	ARGV	一个包含命令行参数的数组，索引从 0 到 ARGC-1
	CONVFMT	数字的字符串转化格式（"%.6g"）（POSIX）
	ENVIRON	环境变量的一个关联数组
	FNR	和 NR 相似，只是相对于当前的文件
	RLENGTH	由 match()函数匹配的字符串的长度
	RSTART	由 match()函数匹配的字符串的第一个位置
	SUBSEP	数组下标的分隔字符（"\034"）
gawk	ARGIND	当前输入文件的 ARGV 中的下标

版本	变量	描述
	ERRNO	当 getline 的重定向失败或假如 close() 失败时指出错误的一个字符串
	FIELDWIDTHS	一个由空格分隔的字段宽度的列，用于取代 FS 分隔记录
	IGNORECASE	当为真时，所有的正则表达式匹配，字符串比较和调用 index() 都忽略大小写
	RT	通过 RS 匹配的文本，在 gawk 中可以是一个正则表达式

操作符

下表列出了在 awk 中有效的操作符，是按优先递增顺序列出的。注意：作为一般扩充的 ** 和 **= 不是 POSIX 的 awk 的部分。

操作符	意义
= += -= *= /= %= ^= **=	赋值
?:	C 条件表达式（只用于 nawk）
||	逻辑或（短路操作）
&&	逻辑与（短路操作）
in	数组成员（只用于 nawk）
~ !~	匹配和不匹配正规表达
< <= > >= != ==	关系操作符
（空行）	连接
+ -	加，减
* / %	乘，除和求余（余数）
+ - !	一元加和减，逻辑非
^ **	求幂
++ --	加一和减一，既可以是前缀也可以是后缀
$	字段引用

变量和数组赋值

变量可以通过一个“ = ”来赋值。例如：

```
FS=", "
```

通过使用操作符 +、-、/ 和 % 也可以将表达式赋给变量。

使用 split()函数（见后面)可以创建数组，或可以在一个赋值语句中简单地对它们进行命名。数组元素的下标可以为数字（*array*[1],...,*array*[*n*]）或字符串。下标为字符串的数组称为关联数组（*associative array*）（注 2)。例如，为了数你拥有的 widget 的数目，可以使用如下的脚本：

```
/widget/ { count["widget"]++ }          计数
END { print count["widget"] }           输出数组
```

可以使用特殊的 for 循环来读一个关联数组的所有元素：

```
for (item in array)
          process array[item]
```

数组的下标可以作为 item 来利用，数组的元素值可以用 array[item]来访问。

通过检测数组的下标是否存在（只用于nawk）可以使用操作符in去检查一个元素是否存在：

```
if (index in array)
          ...
```

该序列检测array[index]的存在，但不能使用它去检测由array[index]引用到的元素的值。

还可以使用删除语句删除数组的单个元素（只用于 nawk 中)。

转义序列

在字符串和正则表达式常量内可以使用以下的转义序列。注意：\x转义序列是一个一般的扩展，它不是 POSIX awk 的一部分。

注 2： 实际上，在 awk 中所有的数组都是关联的，在使用数字作为数组下标之前会把它们转化成字符串。关联数组是 awk 最强大的功能之一。

序列	意义	序列	意义
\a	报警（铃）	\v	垂直制表符
\b	退格	\\	反斜线符号
\f	换页	*nnn*	八进制值 *nnn*
\n	换行	\x*nn*	十六进制值 *nn*
\r	回车	\"	双引号（在字符串内）
\t	制表符	\/	斜线（在正则表达式中）

用户定义的函数

nawk 允许你定义自己的函数。这使得将必须重复执行的步骤封闭到一个位置更容易了，并且可以在程序的任何位置再次使用该代码。注意：对于用户定义的函数，在调用函数时，函数名和左括号之间不允许有空格。

下面的函数将字符串中的每个单词的第一个字母大写。它有一个参数 input，还有 5 个写成附加参数的局部变量。

```
# capitalize each word in a string
function capitalize(input, result, words, n, i, w)
{
         result = ""
         n = split(input, words, " ")
         for (i = 1; i <= n; i++) {
                   w = words[i]
                   w = toupper(substr(w, 1, 1)) substr(w, 2)
                   if (i > 1)
                             result = result " "
                   result = result w
         }
         return result
}

# main program, for testing
{ print capitalize($0) }
```

输入数据：

```
A test line with words and numbers like 12 on it.
```

程序输出：

```
A Test Line With Words And Numbers Like 12 On It.
```

分组的 awk 函数和命令列表

下表对函数和命令进行了分类。

算术函数	字符串函数	控制流语句	I/O 处理	时间函数	编程
atan2[a]	gensub[b]	break	close[a]	strftime[b]	delete[a]
cos[a]	gsub[a]	continue	fflush[c]	systime[b]	function[a]
exp	index	do/while[a]	getline[a]		system[a]
int	length	exit	next		
log	match[a]	for	nextfile[c]		
rand[a]	split	if	print		
sina	sprintf	return[a]	printf		
sqrt	sub[a]	while			
srand[a]	substr				
	tolower[a]				
	toupper[a]				

a： 在 nawk 中有效

b： 在 gawk 中有效

c： 在贝尔实验室的 awk 和 gawk 中有效

实现限制

awk 的很多版本都有各种各样的实现限制，有关的情况例如：

- 每个记录的字段数
- 每个输入记录的字符数
- 每个输出记录的字符数
- 每个字段的字符数
- 每个 printf 字符串的字符数
- 在文本字符串中的字符数
- 字符类中的字符数
- 文件打开的数目
- 管道打开的数目

- 处理 8 位字符和都是 0（ASCII 中的 NUL）的字符的能力

在 gawk 中对这些项都没有限制，只是限于机器结构和 / 或操作系统。

按字母顺序概述函数和命令

下面按字母排序的列表包括了所有在 awk、nawk 和 gawk 中有效的关键字和函数。nawk 包括了所有旧的 awk 函数和关键字再加上一些附加的内容（标记为**{N}**）。gawk 包括了所有的 nawk 函数和关键字再加上一些附加的内容（标记为**{G}**）。用**{B}**标记的项表示在贝尔实验室的 awk 中有效。没有任何标记的项表示在所有的版本中都有效。

atan2	atan2(*y*, *x*) 返回括号内 *y*/*x* 弧度的反正切值。{N}
break	break 从 while、for 或 do 循环中退出。
close	close(*filename-expr*) close(*command-expr*) 在 awk 的大多数实现中，可以同时打开 10 个文件和拥有一个管道。因此，nawk 提供 close 函数来允许关闭一个文件或管道。它把打开该管道或文件的表达式视为一个参数。这个表达式必须与打开文件或管道的表达式完全相同；甚至空白符都是相当重要的。{N}
continue	continue 开始 while、for 或 do 循环的下一次遍历。
cos	cos(*x*) 返回一个角的弧度 *x* 的余弦值。{N}

delete	`delete` *`array`*`[`*`element`*`]` `delete` *`array`* 从数组 *array* 中删除元素 *element*。那个括号必须按字面键入。第二种形式是一个一般的扩充，一次删除数组的所有元素。{N}

do	`do`
	　`statement`
	`while (expr)`
	
	循环语句。执行语句 *statement*，然后计算 *expr* 的值，假如为真再执行语句 *statement*。一系列的语句必须放置在大括号内。{N}

exit	`exit[`*`expr`*`]` 没有读到新的输入则从脚本中退出。假如存在`END`过程的话，则将要执行它。可选的 *expr* 是 `awk` 的返回值。
exp	`exp(`*`x`*`)` 返回 *x* 的幂（e^x）。
fflush	`fflush([`*`output-expr`*`])` 刷新任何缓冲区，该缓冲区和打开的输出文件或管道 *output-expr* 相关联。{ B } `gwak`扩展这个功能。假如没有提供任何 *output-expr*，则刷新标准输出。假如 *output-expr* 是空字符串（`""`），它会刷新所有打开的文件和管道。{ G }
for	`for (`*`init-expr`*`;`*`test-expr`*`;`*`incr-expr`*`)` 　*`statement`* C 风格的循环结构。*init-expr* 对一个计数器变量赋初值。*text-expr* 是在每次执行语句 *statement* 之前都要计算的关系表达式。当 *test-expr* 为假时退出循环。*incr-expr* 在每次通过之后递增计数器变量。所有的表达式都是可选的。省略的 *test-expr* 被认为是真值。一系列的语句必须放置在大括号内。

for	`for (`*item* `in` *array*`)` 　*statement* 为读关联数组设计的特殊的循环。对于数组 *array* 中的每一个元素，都执行语句 *statement*，该元素可以通过 *array*[*item*]来引用。一系列的语句必须放置到大括号内。

function	`function` *name*`(`*parameter-list*`) {`
	　statements
	`}`
	
	创建由 awk 语句 *statements* 构成的用户定义函数 *name*，语句 *statements* 会应用到指定的参数列表。当调用函数时在 *name* 和左括号之间不允许有空格。{N}

getline	`getline [`*var*`][<`*file* 或 *command*`|getline [`*var*`]` 读下一个输入行。最初的awk并不支持此语法去打开多输入流。第一种形式从 *file* 中读输入，第二种形式读 *command* 的输出。两种形式都一次读一个记录，并且每次执行语句，它会得到下一个输入的记录。通过设置 NF、NR 和 FNR，记录赋给 $0 并被分成字段。假如指定 *var* 的话，结果会赋给 *var*，并且 $0 和 NF 不改变。因此假如结果赋给一个变量的话，当前的记录不会改变。getline 实际上是一个函数，假如成功地读了一条记录它会返回 1，到文件结尾的时候会返回 0，不成功会返回－1。{N}
gensub	`gensub(`*r,s,h*`[,`*t*`])` 一般的替代函数。用 *s* 替代字符串 *t* 中的正则表达式的匹配。假如 *h* 是一个数字，则代替第 *h* 个匹配。假如它是 "g" 或 "G" 则全局替代。假如没有提供 *t*，就用 $0。返回新的字符串值。最初的 t 不会修改（比较 **gsub** 和 **sub**）。{G}

gsub	gsub(*r*,*s*[,*t*]) 对在字符串*t*中的正则表达式*r*的每一个匹配全部执行替代*s*。假如没有提供*t*，默认为$0。返回替代数。{N}

if	if (*condition*)
	  statement
	[else
	  statement]
	
	假如条件 *condition* 为真，执行语句 *statement(s)*；否则执行可选的 else 子句中的语句 *statement*。条件可以是使用任何关系操作符 <、<=、= =、!=、>= 或 > 的表达式，也可以是数字成员操作符 in 以及模式匹配操作符 ~ 和 !~（例如，if ($1 ~ /[Aa].*/)）。一串的语句必须放到大括号内。另外 if 也可以直接跟在 else 之后以便进行一连串的检测或决定。

index	index(*str*,*substr*) 返回 *substr* 在 *str* 中的位置（开始为 1），假如 *substr* 不存在于 *str* 中则返回 0。
int	int(*x*) 通过截去 *x* 的小数部分返回其整数值。
length	length([*arg*]) 返回 *arg* 的长度，假如没有参数的话则返回 $0 的长度。
log	log(*x*) 返回 *x* 的自然对数（基为 *e*）。
match	match(*s*,*r*) 这是一个函数，用于匹配由字符串 *s* 中的正则表达式 *r* 指定的模式。或者返回在 *s* 中的位置（从匹配开始），或者假如没有发生匹配则返回0。设置RSTART和RLENGTH的值分别为匹配的开始位置和长度。{N}

next	`next` 读下一个输入行并通过模式 / 过程语句开始新的循环。
nextfile	`nextfile` 停止处理当前的输入文件并从下一个文件的第一个记录开始通过模式 / 过程语句开始新的循环。{B}{G}
print	`print[`*output-expr*`[, ...]][`*dest-expr*`]` 计算 *output-expr* 并将它定向到标准输出，后面跟着 `ORS` 的值。每一个由逗号分隔的 *output-expr* 在输出中都用 `OFS` 的值隔开。假如没有 *output-expr* 则输出 `$0`。 输出重定向 *dest-expr* 是一个可选的表达式，它将输出定向到一个文件或管道。 `>`*file* 将输出定向到一个文件，并覆盖它原来的内容。 `>>`*file* 追加输出到文件，保留它原先的内容。在这两种情况下，假如文件并不存在，则都会创建一个文件。 `|` *command* 将输出作为输入定向到一个 Unix 命令。 注意不要对同一个文件混合使用 `>` 和 `>>`。一旦用 `>` 打开一个文件，则后续的输出继续追加到该文件中直到关闭这个文件为止。 当完成一个文件或管道时记住一定要调用 `close()` 函数。假如不这样做，最终会导致违反系统对同时打开文件的数目限制。
printf	`print(`*format*`[,`*expr-list*`])[`*dest-expr*`]` 一个可选择的输出语句，它借鉴了 C 语言。它能产生格式化的输出，也能输出数据而不自动生成换行符。*format* 是一个格式规范的字符串和常量。*expr-list* 是一个相对应于格式说明符的参数列表。对于 *dest-expr*，可参见 **print** 中对 *dest-expr* 的解释。

printf

格式 *format* 遵循 C 语言 *printf(3S)* 库函数的惯例。这里列出了几种常见的格式：

`%s`	一个字符串。
`%d`	一个小数。
`%`*n*`.`*m*`f`	一个浮点数：*n*= 数字的总数。*m*= 小数点后数字的数目。
`%[-]`*nc*	*n* 指定了格式类型为 *c* 的最小的字段长度，而 - 用来调整字段值左对齐；否则值是按右对齐的。

和任何字符串相似，*format* 可以包括嵌入的转义序列：`\n`（换行）或 `\t`（制表符）是最常见的。通过引起整个的参数可以将空格和文本放置在 *format* 参数中。假如要输出多表达式，就应当指定多个格式。

示例

使用脚本：

```
{ printf("The sum on line %d is %.0f.\n", NR, $1+$2) }
```

下面是输入行：

```
 5   5
```

产生输出，后面跟着换行符：

```
The sum on line 1 is 10.
```

rand

`rand()`

产生 0 和 1 之间的一个随机数。当执行脚本时这个函数每次会返回同样的数列，除非使用 `srand()` 函数设定随机数据产生器。{N}

return

`return[`*expr*`]`

在由用户定义的函数内使用来退出函数，并返回 *expr* 的值。假如没有提供 *expr* 则函数的返回值是未定义的。

sin

`sin(`*x*`)`

返回角的弧度 *x* 的正弦。{N}

split	split(*string*, *array*[, *sep*]) 将字符串*string*分成数组的元素*array*[1],...,*array*[*n*]。这个字符串是根据分隔符*sep*来分开的。假如没有指定*sep*，则使用FS。返回已创建的数组元素的数目。
sprintf	sprintf(*format* [, *expressions*]) 返回已格式化了的一个或多个表达式*expressions*的值，格式化使用指定的*format*格式（参见 **printf**）。数据只被格式化但并不输出。
sqrt	sqrt(*arg*) 返回*arg*的平方根。
srand	srand([*expr*]) 使用可选的*expr*为随机数产生器设定一个新的种子（seed）。默认是一天的时间。返回值是以前的随机数种子。{N}
strftime	strftime([*format*[,*timestamp*]]) 根据*format*格式化时间戳*timestamp*。返回格式化了的字符串。*timestamp*是一个从1970年1月1日午夜开始算起按格林威治标准时间（UTC）的日时值，是以秒为单位的。*format*字符串与sprintf的相似（参见 **systime**的示例）。假如省略了*timestamp*，则默认为现在的时间。假如省略了*format*，则默认为产生的输出值和date相似。{G}
sub	sub(*r*,*s*[,*t*]) 用*s*替代字符串中正则表达式*r*的第一个匹配。假如没有提供*t*，则默认为$0。假如成功则返回 1，否则返回 0。
substr	substr(*string*, *beg* [, *len*] 返回开始位置为*beg*的字符串*string*的子串，并且长度最大为指定的*len*。假如没有给出长度，则使用字符串的其他部分。

system

`system(`*command*`)`

是一个函数，它执行指定的命令 *command* 并且返回它的状态。那个执行的命令的状态通常指的是成功或失败。零值意味着执行成功。非零则意味着某种失败。你所执行的命令的文档会告诉你详细的情况。

该命令的输出在 `awk` 脚本内是无法处理的。可以使用 *command*`|getline` 将命令的输出读到脚本中。{N}

systime

`systime()`

返回自 1970 年 1 月 1 日以来格林威治标准时间的日时值，以秒为单位。{ G }

示例

记录一个数据处理程序的起始和结束时间：

```
BEGIN {
          now = systime()
          mesg = strftime("Started at %m/%d/%Y %H:%M:%S", now)
          print mesg
}
process data ...
END {
          now = systime()
          mesg = strftime("Ended at %m/%d/%Y %H:%M:%S", now)
          print mesg
}
```

tolower

`tolower(`*str*`)`

将 *str* 中的所有大写字母转换成小写并返回转换后的字符串（注 3）。{N}

注 3： 早期的nawk版本并不支持`tolower()`和`toupper()`函数。然而它现在是awk的POSIX规范的一部分，并且包括在 SVR4 nawk 中。

toupper	toupper(*str*) 将 *str* 中的所有小写字母转换成大写并返回转换后的值。{N}
while	while(*condition*) *statement* 当 *condition* 条件为真时执行 *statement* 语句（参见 **if** 中关于允许条件的说明）。必须用大括号将一连串的语句括起来。

printf 格式

printf 和 sprintf 格式化说明符有下面的形式：

```
%[flag][width][.precision] letter
```

控制字母是必需的。下面所列出的是格式转化控制字母：

字符	描述
c	ASCII 字符
d	十进制整数
i	十进制整数（在 POSIX 中添加的）
e	浮点格式（[-]*d. precision* e[+-]*dd*）
E	浮点格式（[-]*d. precision* E[+-]*dd*）
f	浮点格式（[-]*ddd. precision* ）
g	对于 e 或 f 选择最短的，尾部的零全部删去
G	对于 E 或 f 选择最短的，尾部的零全部删去
o	无符号的八进制值
s	字符串
u	无符号的十进制值
x	无符号的十六进制数；使用 a-f 指 10 到 15
X	无符号的十六进制数；使用 A-F 指 10 到 15
%	字面意义的 %

下面是可选的标志：

字符	描述
-	对字段内已格式化的值靠左对齐
空格	正值使用一个空格为前缀，负值使用一个减号为前缀
+	数值总带一个符号，即使一个正值也如此
#	使用另一种形式：%o 前面带 0；%x 和 %X 前缀分别为 0x 和 0X；%e，%E 和 %f 在结果中总带有小数点；而 %g 和 %G 会删除后面的 0
0	用零而不是空格填充输出，这只发生在字段的宽度比转换后的结果更宽时

可选的宽度 *width* 指的是要输出的字符的最小数。如果结果的长度不够大则必须补满。0 标志表示用 0 来补满；否则用空格补满。

精度也是可选的。它的意思根据控制字母而变化，有关它的说明在下表中：

转换	精确度的意义
%d，%i。%o %u，%x，%X	要输出的数字的最小数目
%e，%E，%f	小数点右边数字的数目
%g，%G	有意义的数字的最大数目
%s	要输出的字符的最大数目

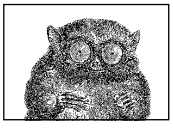
第三部分

文本格式化

第三部分将介绍用于文档格式化的Unix工具。这些工具不再是标准SVR4的一部分，但在BSD（伯克利软件发行中心）兼容包中和SVR4一起提供。它们已成为Solaris的标准部分（`pic`除外）。

许多Unix厂商提供一个增强的格式化工具集 —— 在有些情况下，需要额外付费。

- 第十二章，nroff和troff
- 第十三章，mm宏
- 第十四章，ms宏
- 第十五章，me宏
- 第十六章，man宏
- 第十七章，troff预处理程序



第十二章

nroff 和 troff

本章提供以下内容：

- 介绍
- 命令行调用
- 概念综述
- 默认的请求操作
- 按组概述请求
- 按字母顺序概述请求
- 转义序列
- 预定义寄存器
- 特殊字符

介绍

nroff和troff都是用于格式化文本文件的Unix程序。nroff用于格式化行式打印机和高质量字母打印机的输出；也可以在屏幕上显示输出。troff用于对排版打印机和激光打印机的输出格式化。同样的命令对两个程序都有效；nroff只是简单地忽略它不能执行的命令。

nroff和troff不是标准SVR4的部分，但包括在兼容包内。这里对这个版本加以了

说明。另外，我们还提到了 ditroff，或者说设备无关性 troff，这是 troff 的后续版本。绝大部分功能 ditroff 和 troff 是一样的，有区别时最初的 troff 被称为 otroff。Solaris troff 是设备无关性版本，并且是 Solaris 发布的一个标准部分。

一些 Unix 厂商还提供了特定于厂商的 nroff/troff 版本。其他的根本不包括它们。各种各样的增强软件包也是可以利用的，比如来自 SoftQuard 的 sqtroff 或者来自自由软件基金会的 groff（注 1）。这些包都包括附加的请求或转义序列。要想获得更精确的信息，应当查询和你的特定 Unix 版本一起提供的文本处理手册。

还有，假如能用 checknr 程序，你应当在你的 troff 文档中使用它。注意 troff 的设备无关性版本是不支持 8 位的。你的系统可能只支持 otroff。

命令行调用

nroff 和 troff 可以按以下的方式从命令行调用：

```
nroff [options] [files]
troff [options] [files]
```

很多选项对于两个格式化程序都是相同的。

nroff/troff 选项

-F*dir*
: 在目录 *dir* 中搜索字体表。

-i 处理文件 *files* 之后读标准输入。

-m*name*
: 向输入文件 *files* 中由前添加一个宏文件。过去，/usr/lib/tmac/tmac.*name* 或 /usr/share/lib/tmac/tmac.*name* 有一个是 *name* 宏的位置。而 Solaris 则使用 /usr/share/lib/tmac/*name*。实际的位置和文件名在不同的 Unix 系统中是不同的。

-n*N* 第一个输出页页号为 *N*。

注 1： gruff 尤其值得注意；它对标准 troff 有许多有用的扩充并且很稳定（参见 *http://www.gnu.org*）。

-o *list*

打印只包括在 *list* 中的页，这里页号是由逗号分隔的。范围可以指定为 *n*—*m*、—*m* （第一页到 *m* 页）或 *n*—（*n* 到文件尾）。

-r*aN*

设置寄存器 *a* 为 *N*。该寄存器 *a* 限制为一个字符的名字。

-s*N*每 *n* 页停止，这允许改变存纸盒。通过按回车（在 nroff 中）或在排字机上按开始按钮（在 troff 中）可重新启动。

-T*name*

准备好为打印机或排字机 *name* 指定的输出。对于设备名，可以查询特定的文档或咨询有经验的人员。

-u*N*重击位置 3 处的字体 N 次。一般用于调整粗体字的加粗系数。

-z 除了由 .tm 请求产生的信息外丢弃其他的输出（只用于 otroff）。

只用于 nroff 的选项

-e 在调整输出行时，将单词等间距地隔开（使用终端分辩率而不是完全增加空格）。

-h 通过使用制表符代替 8 个等字符宽度的空格，从而加快输出的速度。

-q 同时调用 .rd 请求的输入 / 输出。

只用于 troff 的选项

-a 格式化可打印的ASCII近似值。这样不用生成打印好的输出就可以找到页的合计数。

-f 当格式化完成后不停止排字机（只用于 otroff）。

-N 作为 nroff 而不是作为 troff 运行（只用于最新的 ditroff 版本）。

示例

通过 tbl 预处理程序来运行 chap1，然后使用 *mm* 宏来格式化结果，并且将寄存器 N 设置为 5（设置页码风格），等：

```
tbl chap1 | troff -mm -rN5 |  spooler  &
```

使用 *ms* 宏格式化 chap2；第一页为 7，但只打印 8～10 页、15 页、18 页直到文件的结尾：

```
nroff -ms -n7 -o8-10,15,18- chap2 | col > chap2.txt &
```

概念综述

本部分将详细介绍怎样为 nroff 和 troff 准备好输入。提供了以下主题：

- 请求和宏
- 一般请求
- 指定度量单位
- 引起行断开的请求
- 内嵌的格式化控制

请求和宏

通过将详细的代码（称为请求）内嵌到文本源文件中来指定格式化。这些代码在它们运行时引导 nroff 和 troff 去工作。例如，为了将文本行居中，可在文件中键入以下代码：

```
.ce
this text should be centered.
```

当格式化完毕后，则输出会居中显示：

```
                    This text should be centered.
```

有两种格式化代码的类型：

- 请求，提供最基本的指令
- 宏，请求的预定义组合

请求，通常也称为原始的请求，它允许直接控制一个页布局和格式化的几乎任何特征。宏会将请求组合起来创建一个总的效果。从某种意义上来说，请求就像原子，而宏则像分子。

所有的nroff/troff请求名称都是由两个小写的字母组成的。宏的名称一般是大写或大小写混合的。

如果想详细了解标准的宏软件包，可参见第十三章、第十四章、第十五章和第十六章。

一般请求

最常用到的请求有：

```
.ad .ds .ll .nr .sp
.br .fi .na .po .ta
.bp .ft .ne .ps .ti
.ce .in .nf .so .vs
.de .ls
```

例如，一个简单的宏可以这样写：

```
.                   \" Ps macro - show literal text display
.de Ps              \" Define a macro named "Ps"
.sp .5              \" Space down half a line
.in 1i              \" Indent one inch
.ta 10n +10n        \" Set new tabstops
.ps 8               \" Use 8-point type
.vs 10              \" Use 10-point vertical spacing
.ft CW              \" Use constant width font
.br                 \" Break line (.ne begins count on next line)
.ne 3               \" Keep 3 lines together
.nf                 \" No-fill mode (output lines as is)
..                  \" End macro definition
```

指定度量单位

在请求中，数字参数可能有一个指定度量单位的刻度指示器。下表列出了一些有效的指示器和它们的意义。应当了解的是所有的度量都会在内部转化成基本的单位（最后一栏中列出了这些变换）。一个基本单位即为在打印机设备上可能的最小尺寸。设备分辨率（比如，每英寸 600 点）决定了一个基本单位的大小。*T* 指出了当前的磅值，而 R 指定了设备分辨率。

刻度指示器	意义	等价单位	基本单位转换
c	厘米	0.394 英寸	*R* / 2.54
i	英寸	6 点活字或 72 磅	*R*
m	Em	T 磅	*R* × *T* / 72
n	En	0.5em	*R* × *T* / 144
p	磅	1/72 英寸	*R* / 72
P	点活字	1/6 英寸	*R* / 6
u	基本单位		1
v	垂直行间隔		当前的基本单位值
无	默认		

值得注意的是，在 nroff/troff 中所有的数都用整数存储在内部。这可以在命令中应用于分数值：

```
.sp .5
```

向下留出当前垂直间隔的一半的空间。

一个“ em ”即为在当前的字体和磅值下字母“ m ”的宽度。一个“ en ”即为在当前字体和磅值下字母“ n ”的宽度。应注意在 nroff 中“ em ”和“ en ”都为一个字符的宽度。

除了总是使用磅值的 .ps 外，你可以为下表中的任何一个请求指定一个刻度指示器。假如没有给出单位，则使用默认的单位（第二栏列出了在前面的表中说明的刻度指示器）。对于面对水平方向上的请求，默认的单位是 ems。而面对垂直方向上的请求，默认通常为垂直行间隔。

请求	默认的刻度指示器	请求	默认的刻度指示器
.ch	v	.pl	v
.dt	v	.po	v
.ie	u	.ps	p
.if	u	.rt	v
.in	m	.sp	v
.ll	m	.sv	v
.lt	m	.ta	m
.mc	m	.ti	m
.ne	v	.vs	p
.nr	u	.wh	v

引起行断开的请求

即使当前输出行没有完全填充的情况下，nroff/troff 写输出行的时候也可能会发生行断开。绝大多数请求可以由文本点缀而不会在输出中产生行断开。下面的请求则可能会产生行断开：

```
.bp .ce .fi .in .sp
.br .cf .fl .nf .ti
```

假如你需要阻止这些请求引起的行断开，可以在这些请求开头使用“ no break ”控制

字符（一般是′）而不是点（.）。例如，.sp 会立刻执行，而 .sp 则等到输出行完全填充才开始，只有到那时才会增加一个换行。

内嵌的格式化控制

除了请求和宏（它们都被写在它们自己的分离的行上）之外，你还可以在文本行内使用内嵌的格式化控制。它们一般提供了以下的功能：

一般的格式化

相当多的格式化控制是可以利用的，比如转换字体（\f）、改变磅值（\s）、计算宽度（\w），还有很多其他的。例如：

```
This text is in \fIitalic\fR, but this is in roman.
This text is \s-2VERY SMALL\s0 but this text is not.
```

特殊的字符

预定义的特殊的照排机字符，例如加重号 \(bu(•)、左手 \(lh(☜)和右手 \(rh(☞)。

字符串

用户定义的字符序列，和宏一样，但应用内联。例如：

```
.\" define a shorthand for UNIX
.ds UX the \s-1UNIX\s0 Operating System
...
Welcome to \*(UX.
While \*(UX may appear daunting at first,
it is immensely powerful. ...
```

数字寄存器

和编程语言中的变量一样，它存储数字值，这些数字值可以按一个格式范围（小数、罗马字体，等等）来打印。它们可以设置成自动递增或自动递减，并且在写宏包的时候尤其有用，用于管理标题的自动编号、脚注、数字等等。例如：

```
.nr Cl 0 1 \" Chapter Level
.de CH
.bp
\\n+(Cl. \\$1 \\$2 \\$3
..
```

它创建了一个使用数字寄存器 Cl 作为“章的层次”的宏。该宏的前三个参数变成章的标题。另外的斜线符号必须用在宏定义的内部。

本章后面的部分将介绍预定义的特殊字符，字符串和数字寄存器以及所有可用的转义序列。

默认的请求操作

nroff/troff对格式化环境进行初始化。例如，除非你重新设置行的长度，nroff/troff总会使用6.5英寸的尺度。大多数的请求可以改变默认的环境，并且列在了表12-1中。第二栏列出了在使用请求前的原始值或默认值。假如没有原始值，则用一个连字符（-）。对于没有使用可选参数的请求，第三栏列出了其会产生的效果。在这里，假如请求不接受参数或要求参数时使用了连字符。

表12-1：影响默认环境的请求

请求	原始值	假如没有参数	描述
.ad	调整	调整	调整页边空白
.af	小写的阿拉伯字母	—	将格式赋给寄存器
.am	—	用 .. 结束调用	向宏中追加命令
.bd	关闭	—	加粗字体
.c2	'	'	设置 no-break 控制字符
.cc	.	.	设置控制字符
.ce	关闭	将一行置中	将行置于中心位置
.ch	—	清除陷阱	改变陷阱位置
.cs	关闭	—	设定常量间距
.cu	关闭	一行	连续的下划线 / 斜体
.da	—	结束转移	转移文本并追加到宏
.de	—	使用 .. 结束宏	定义一个宏
.di	—	结束转移	将文本转移到宏
.dt	—	清除陷阱	设置一个转移陷阱
.ec	\	\	设置转义字符
.eo	打开	—	关闭转义字符
.ev	0	先前的环境	改变环境（下推）
.fc	关闭	关闭	设置字段定界符和填充字符
.fi	填充	—	填充行
.fp	1=R	—	装配字体（在位置 1～4 上）
	2=I		
	3=B		
	4=S		
.ft	罗马字体	先前的字体	设置字体

表 12-1：影响默认环境的请求（续）

请求	原始值	假如没有参数	描述
.hc	\%	\%	设置连字符
.hy	模式 1	模式 1	设置连字模式
.ig	—	以 .. 结束	禁止（忽略）输出的文本
.in	0	先前的缩进	设置缩进
.it	—	清除陷阱	设置输入行陷阱
.lc	.	无	设置前导字符
.lg	关闭(nroff) 打开(troff)	打开	设置连字模式
.ll	6.5 英寸	先前的行宽	设置行宽
.ls	单个空格	先前的模式	设置行间距
.lt	6.5 英寸	先前的标题宽	设置标题宽
.mc	—	关闭	设置页边字符
.mk	—	内部的	标记垂直位置
.na	调整	—	不调整页边空白
.ne	—	一个垂直行	假如有空间的话将行保持在同一页上
.nf	填充	—	不填充行
.nh	打开	—	关闭连字符
.nm	关闭	关闭	行编号模式
.nn	—	一行	对下面的 N 行不编号
.ns	空模式	—	允许非空模式
.nx	—	结束文件	到一个文件
.pc	%	关闭	设置页码字符
.pl	11 英寸	11 英寸	设置页高
.pn	页码 1	—	设置页码
.po	0 (nroff)； 26/27 英寸 (otroff) 1 英寸(ditroff)	先前的页边距	改变页边距
.ps	10	先前的磅值	设置磅值
.rd	—	响铃	读终端
.rt	—	内部的	返回至标记的垂直位置

表 12-1：影响默认环境的请求（续）

请求	原始值	假如没有参数	描述
.sp	—	一个垂直行	输出空白区
.ss	12/36em	忽略	设置字符间距
.sv	—	一个垂直行	保存（存储）间距
.ta	8 en (nroff)； 1/2 英寸(troff)	—	定义制表位设置
.tc	—	—	设置制表符
.ti	0	—	缩进下一行
.tm	—	换行	输出一个消息，然后继续
.tr	—	—	在输出上转换成对的字符
.uf	斜体	斜体	设置下划线字体
.ul	0	一行	对一行加下划线 / 斜体
.vs	1/6 英寸(nroff) 12 磅(troff)	先前的值	对行设置垂直间距

在nroff/troff中，注释的开头有一个\"。开头为.且包含未知请求的行会被忽略。在文本的开头一般不要放置空白符。这可能会引起中断，并且 nroff 和troff会按字面意义对待开头的空白符。

注意：对 nroff/troff 规范的参考资料是贝尔实验室计算机科技报告 #54，由 J.F.Ossanna 和 B.W.Kernighan 编写的 Troff 用户手册 。它来自 *http://cm.bell-labs.com/cm/cs/cstr/54.ps.gz* 的附录部分。这个文档介绍了转移、环境、字段、寄存器、字符串和陷阱。

按组概述请求

为了帮助为特定的任务查找合适的请求，下面按主题列出了 85 条 nroff/troff 请求。

字符输出

.cu 连续的下划线 / 斜体。

.lg 连字模式。

.tr 转换字符。

.uf 设置下划线字体。

`.ul` 下划线 / 斜体。

有条件的处理

`.el` *if-else* 的 *else* 部分。

`.ie` *if-else* 的 *if* 部分。

`.if` *If* 语句。

定制 n/troff 请求

`.c2` 设置 no-break 控制字符。

`.cc` 设置控制字符。

`.ec` 设置转义字符。

`.eo` 关闭转义字符。

`.hc` 设置连字符。

`.pc` 设置页码字符。

诊断输出

`.ab` 输出一个消息，然后中断。

`.fl` 刷新输出缓冲区。

`.ig` 禁止（忽略）输出文本。

`.lf` 设置行号和文件名。

`.mc` 设置页边字符。

`.pm` 输出宏的名称和大小。

`.tm` 输出一个消息，然后继续。

字体和字符大小

`.bd` 加粗字体。

`.cs` 设置常量间距。

`.fp` 装配字体（在 1～4 位置上）。

`.ft` 设置字体。

`.ps` 设置磅值。

`.ss` 设置字符间距。

水平位置调整

`.in` 缩进。

`.ll` 设置行宽。

`.lt` 设置标题宽。

`.po` 改变页边距。

`.ti` 缩进下一行。

`.tl` 指定三部分的标题。

连字符连接

`.hw` 设置硬编码的连字符连接。

`.hy` 设置连字模式。

`.nh` 关闭连字符。

输入 / 输出转换

`.cf` 将原始文件拷贝到输出。

`.ex` 从 `nroff/troff` 退出。

`.nx` 到一个文件。

`.pi` 经管道输出到一个 Unix 命令。

`.rd` 读终端。

`.so` 到一个文件，然后返回。

`.sy` 执行一个 Unix 命令。

行编号

`.nm` 行编号模式。

`.nn` 取消行编号。

宏和字符串处理

.am 向一个宏追加。

.as 向一个字符串追加。

.ch 改变陷阱位置。

.da 转移文本并且追加到宏。

.de 定义宏。

.di 将文本转移到宏。

.ds 定义字符串。

.dt 设置转移陷阱。

.em 设置结尾的宏。

.ev 改变环境。

.it 设置输入行计数陷阱。

.rm 清除宏、请求或字符串。

.rn 对宏、请求或字符串改名。

.wh 设置陷阱位置。

数字寄存器

.af 把格式赋给寄存器。

.nr 定义一个数字寄存器。

.rr 清除一个数字寄存器。

分页

.bp 开始一个新页。

.mk 标记垂直位置。

.ne 假如有空间则保持行在同一页上。

.pl 设置页高。

.pn 设置页号。

.rt 回到标记的垂直位置。

制表符

.fc 设置字段定界符和填充字符。

.lc 设置前导字符。

.ta 定义制表位设置。

.tc 设置制表符。

文本调整

.ad 调整页边空白

.br 断开输出行。

.ce 将行置中。

.fi 填充行。

.na 不调整页边空白。

.nf 不填充行。

垂直间距

.ls 行间距（例如，单个空格）。

.ns 允许非空模式。

.os 从 .sv 输出垂直间距。

.rs 恢复间距模式。

.sp 输出空白间距。

.sv 保存（存储）间距。

.vs 设置行的垂直间距。

按字母顺序概述请求

.ab

.ab[*text*]

中断并且输出消息*text*。假如没有给定*text*则输出User Abort消息。

.ad

.ad[*c*]

根据格式*c*调整输出行。填充模式必须打开（参见**.fi**）。如果没有参数，则相当于.ad 1。当前的调整模式存储在寄存器.j中，并跟有值:0=1，1=b，3=c，5=r（参见**.na**）。

c的值

b 对齐行。

n 对齐行。

c 将行置于中心。

l 左对齐行。

r 右对齐行。

.af

.af *r c*

将格式*c*赋给寄存器*r*。

c的值

1 0、1、2，等等。

001 000、001、002，等等

i 小写罗马数字。

I 大写罗马数字。

a 按字母顺序的小写字母。

A 按字母顺序的大写字母

示例

使用*ms*宏标出前面几页的页码：

```
af PN i                    设置页码寄存器PN为i
```

.am	`.am` *xx* [*yy*] 接受后续的请求并且追加到宏 *xx* 中。在调用到 *.yy* 时结束追加（或 `..`，即当省略 *yy* 时）。
.as	`.as` *xx string* 追加字符串 *string* 至字符串寄存器 *xx* 中。字符串 *string* 可以包含空格并用一个换行符终止字符串。词首的引号（"）忽略。
.bd	`.bd [s]` *f n* 重击字体为 *f* 的字符 *n* 次。假如指定了 `s` 的话，则当字体 *f* 有效时对为特殊字体的字符重击 *n* 次。
.bp	`.bp` [*n*] 开始一个新页。对下一页标记页码 *n*。
.br	`.br` 用换行符断开行（输出部分的行）。
.c2	`.c2` *c* 使用 *c*（而不是 ′）作为 no-break 控制字符。
.cc	`.cc` *c* 使用 *c*（而不是 .）作为控制字符引入请求和宏。
.ce	`.ce`[*n*] 将下面的 *n* 行（默认为 1）置于中心，假如 *n* 为 0 则停止置于中心。*n* 只应用于包含输出文本的行。空行不计算在内。
.cf	`.cf` *file* 拷贝文件 *file* 的内容到输出并且不做解释（只用于 `ditroff`）。

.ch	.ch *xx* [*n*] 将宏 *xx* 的陷阱位置改变到 *n*。假如 *n* 不存在，则清除陷阱。
.cs	.cs *f n m* 对字体 *f* 使用常量间距。常量间距宽度是 *n*/36em。假如给出 *m* 的话，则用 *m* 磅取代 em。 例如 `.cs CW 18` 压缩常量宽度字体的间距
.cu	.cu [*n*] 对下面的 *n* 行加连续的下划线（包括字间的空格）。假如 *n* 为零，则停止加下划线。用 .ul 只对可见的字符加下划线。在 troff 中可以用 .uf 请求取代。在 troff 中，.cu 和 .ul 都生成斜体（必须使用宏来加下划线）。
.da	.da [*xx*] 转移后面的文本并追加到宏 *xx* 中。假如没有参数，则结束转移。
.de	.de *xx* [*yy*] 定义宏 *xx*。调用到 *.yy* 时则结束定义（或当省略了 *yy* 时为 ..）。
.di	.di[*xx*] 转移后面的文本到新定义的宏 *xx* 中。假如没有参数，则结束转移。
.ds	.ds *xx string* 定义 *xx* 来包含 *string*。词首的引号（"）忽略。
.dt	.dt *n xx* 在位置 *n* 处设置转移陷阱（在转移内部），去调用宏 *xx*。

.ec	.ec[*c*] 设置转义字符 *c*。默认为 \。
.el	.el *if-else* 的 *else* 部分（参见下面的 .ie）。
.em	.em *xx* 设置结束宏为 *xx*。当所有其他的输出完毕后自动执行 *xx*。
.eo	.eo 关闭转义字符机制。所有的转义字符按字面输出。
.ev	.ev[*n*] 改变环境为 *n*。例如，在当前的环境中存储了很多的请求，这些请求会影响到水平位置、连字符连接或文本调整等。假如省略了 *n* 则恢复先前的环境。*n* 的最初值为 0，并且 $0 \leq n \leq 2$。必须使用不带参数的 .ev返回到原先的环境，否则会发生栈溢出（ditroff仅仅忽略一个无效的参数并且给出警告）。
.ex	.ex 退出格式化程序并不再执行文本处理。为了产生统一的格式通常使用 .nx。
.fc	.fc *a b* 设置字段定界符为 *a*，设置填充字符为 *b*。
.fi	.fi 打开填充模式，和 .nf 相反。默认为打开。
.fl	.fl 刷新输出缓冲区。用于交互式的调试。

.fp

`.fp` *n f*

将字体*f*指定给位置*n*。在otroff中*n*的范围是从1到4，而在ditroff中为 1 到 99。

示例

```
.fp 7 CW     \" 在位置 7 是常宽
.fp 8 CI     \" 在位置 8 是不变的斜体
.fp 9 CB     \" 在位置 9 是不变的粗体
```

.ft

`.ft` *f*

改变字体为*f*，*f*是一（或两）个字符的字体名，或者用 `.fp`指定一个字体位置。和转义序列 `\f` 相似。

.hc

`.hc[`*c*`]`

改变输入连字指示字符为*c*。默认是 `\%`。

.hw

`.hw` *words*

为*words*指定连字符点（例如，`.hw spe-ci-fy`）。对*words*列表的总数有一个限制，总共大约为 128 个字符。

.hy

`.hy`

打开（$n \geq 1$）或关闭（$n=0$）连字符连接。也可参见 **.nh**。

n 的值

1 无论何时都以连字符连接。

2 页上的最后一个单词不用连字符。

4 不分开前面的两个字符。

8 不分开最后的两个字符。

14 使用所有的三个限制。

.ie

`.ie[!]`*`condition anything`*
`.el` *`anything`*

if-else 的 *if* 部分。假如 *condition* 为真则执行 *anything*。否则执行 `.el` 请求后面的 *anything*。`.ie/.el` 对可嵌套。有关 *condition* 的语法在下面的 **.if** 中介绍。

示例

```
.ie !'\\$1'2' .MC 1.8i 0.2i
.el .MC 2.5i 0.25i
```

.if

`.if [!]`*`condition anything`*

假如条件 *condition* 为真，则执行 *anything*。使用 ! 会否定 *condition*。假如 *anything* 运行超过一行，则必须用 `\{`和 `\}`划界。

condition

`o`	假如页码为奇数则为真。
`e`	假如页码为偶数则为真。
`n`	假如处理器为 `nroff` 则为真。
`t`	假如处理器为 `troff` 则为真。
`"`*`str1`*`"`*`str2`*`"`	假如 *str1* 等同于 *str2* 则为真。经常用于检测传递给宏的参数值。
`expr`	假如表达式 *expr* 的值大于 0 则为真。

表达式

表达式通常包括数字寄存器并可能使用下列的操作符：

`+ -`	加法，减法
`/ *`	除法，乘法
`%`	求余
`< >`	小于，大于
`<= >=`	小于或等于，大于或等于
`= ==`	相等

.if

`!` 逻辑非

`&` 逻辑与

`:` 逻辑或

注意：表达式是从左到右计算的；没有操作符优先。可能会使用圆括号去迫使形成一个计算的顺序。

示例

```
.if t .nr PD 0.5v   \" 定义 ms 段落间距
 .if !"\\$2"" \{ \" 假如参数 2 是非空的则按粗体输出它
 \fB\\$2\fP\}
```

.ig

`.ig [`*yy*`]`

忽略后面的文本，直到遇上开始为 `.`*yy* 的行（默认为 `..`，和 `.de` 一样）。用于在大的文本块或宏定义外做注释。

.in

`.in[[`±`]`*n*`]`

设置缩进为 *n* 或增加缩进 ± *n*。假如没有参数，则恢复原先的缩进。当前的缩进存储在寄存器 `.i` 中，默认的刻度为 em。

.it

`.it` *n xx*

对输入行计数设置陷阱，以便在读了 *n* 行输入文本后再调用宏 *xx*。

.lc

`.lc` *c*

设置前导重复字符（`\a` 的值）为 *c* 而不是 `.`（点）。

.lf

`.lf` *n filename*

为后续的错误消息设置行号为 *n* 和文件名为 *filename*（只用于最近的 `ditroff` 版本）。修改寄存器 `.c` 和 `.F`。

.lg

`.lg` *n*

假如省略 *n* 或 *n* 为非零则打开连字模式。

.ll	`.ll [[±]`*n*`]` 设置行宽为*n*或增加行宽±*n*。假如没有给出参数则恢复先前的行宽。当前的行宽存储在寄存器 `.l` 中。默认值为 6.5 英寸。
.ls	`.ls[`*n*`]` 设置行间距为 *n*。假如没有给出参数，则恢复原先的行间距。原始值为 1。 示例 `.ls 2` 生成双空格的输出
.lt	`.lt[`*n*`]` 设置标题宽为 *n*（默认的刻度为 em）。假如没有给出参数，则恢复原先的值。
.mc	`.mc [`*c*`] [`*n*`]` 设置页边的空白字符为 *c*，并且将 *c* 置于距页边空白的右边 *n* 处。假如省略 *c*，则关闭页边空白字符。假如省略了 *n*，则使用原先的值。最初的值在 `nroff` 中为 2 英寸，而在 `troff` 中为 1 em。 这个命令通过用于在文档中生成“ 变化条 ”。参见第二章中的**diffmk**。
.mk	`.mk[`*r*`]` 将当前垂直位置标记在寄存器 *r* 中。用 `.rt` 或 `.sp|\nr` 返回标记。
.na	`.na` 不调整页边的空白。当前的调整存储在寄存器 `.j` 中。也可参见 **.ad**。
.ne	`.ne` *n* 假如在这页上容不下 *n* 行，则开始一个新页。
.nf	`.nf` 不填充或调整输出行。也可参见 **.ad** 和 **.fi**。

.nh

`.nh`

关闭连字符连接。也可参见 **.hy**。

.nm

`.nm [`*n m s i*`]`

对输出行编号（假如 $n \geq 0$），或关闭行编号（假如 $n=0$）。± *n* 设置最初的行号；*m* 设置编号间隔；*s* 设置号码和文本的分离；*i* 设置文本的缩进。用于代码段、诗等。也可参见 **.nn**。

.nn

`.nn` *n*

不对下面的*n*行编号。但仍跟踪行号，可以采用 `.nm +0`重新恢复。参见 **.nm**。

.nr

`.nr` *r n* `[`*m*`]`

对数字寄存器 *r* 赋值 *n*，*m* 是可选的，用于自动递增。

示例
设置“边框宽（box width）”寄存器为行宽减缩进：

```
.nr BW \n(.l-\n(.i
```

为 *ms* 宏设置页面布局值：

```
.nr LL 6i                          行宽
.nr PO ((8.25i-\n(LLu)/2u)         页边距
.nr VS \n(PS+2                     垂直间隔
```

在 `nroff` 中，自动递增一个脚注计数器寄存器：

```
.nr footcount 0 1                  对每页恢复为0
```

注意：在一个宏定义内，`\n` 应当是 `\\n`。

.ns

`.ns`

打开非空模式。也可参见 **.rs**。

.nx

`.nx file`

切换到文件 *file* 并且不返回到当前文件。也可参见 **.so**。

.os	.os 输出在先前的 .sv 请求中指定的保存的空间。
.pc	.pc *c* 在 nroff/troff 编码中使用 *c*（而不是 %）作为页码字符。
.pi	.pi *command* 通过一个 Unix 命令 *command* 从管道输出格式化程序，而不是将它送到标准输出上（只用于 ditroff 和 nroff）。请求必须发生在输出之前。 示例 .pi /usr/bin/col 使用 col 处理 nroff 输出
.pl	.pl[[±]*n*] 设置页高为 *n* 或增加页高 ± *n*。假如没有参数，则恢复默认值。当前的页高存储在寄存器 .p 中。默认为 11 英寸。
.pm	.pm 输出所有定义的宏名和宏大小。
.pn	.pn[[±]*n*] 设置下一个页码为 *n* 或增加页码 ± *n*。当前的页码存储在寄存器 % 中。
.po	.po[[±]*n*] 左边距为 *n* 或对当前的边距增加 ± *n*。假如没有参数，则返回先前的边距。当前的页边距存储在寄存器 .o 中。
.ps	.ps *n* 设置磅值为 *n*（只用于 troff，在 nroff 中接受但会忽略）。当前的磅值存储在寄存器 .s 中。默认为 10 磅。

.rd	.rd [*prompt*] 在输出可选的 *prompt* 后，从终端读输入。
.rm	.rm *xx* 清除请求、宏或字符串 *xx*。
.rn	.rn *xx yy* 将请求、宏或字符串 *xx* 改名为 *yy*。
.rr	.rr *r* 清除寄存器 *r*。也可参见 **.nr**。
.rs	.rs 恢复间距（禁止非空模式）。参见 **.ns**。
.rt	.rt[± *n*] 返回（只向上）到标记的垂直位置。或距离页顶或转移 ± *n* 处。参见 **.mk**。
.so	.so *file* 改变输出到 *file*，然后返回到当前文件；即读另一个 *file* 的内容到当前的文件中。参见 **.nx**。
.sp	.sp *n* 留出 *n* 个空白行。默认为 1。可以为 *n* 使用任何的垂直值（和适当的单元说明符一起）。
.ss	.ss *n* 设置空格字符大小为 *n*/36em（在 nroff 中无效）。
.sv	.sv *n* 保存 *n* 个空白行；用 .os 输出保存的空白行。

.sy

`.sy` *command* [*args*]

使用可选的参数执行 Unix 命令（只用于 `ditroff`）。

示例

搜索第一个参数；积累在一个临时文件里。

```
.sy sed -n ′s/\\$1/Note: &/p′ list >> /tmp/notesfile
```

（注意在`\\$1`中附加的反斜杠。这个示例发生在一个宏定义里。当第一次读宏时会剥掉一个反斜杠，因此第二个反斜杠对于正确地执行宏是必需的。）

.ta

`.ta n[`*t*`] [+]m[`*t*`]`

设置制表位在 *n*、*m* 等位置处。假如没有给出 *t*，制表位是左调整的。使用 + 指相对于前面的制表位移动。

t 的值

`L` 左调整

`R` 右调整

`C` 置于中心

.tc

`.tc` *c*

设置制表符为 *c*（代替空白符）。当扩展制表符时 `nroff/troff` 使用 *c*。例如，当格式化一个目录表时可以使用 `.tc .`。

.ti

`.ti [[±]`*n*`]`

暂时的缩进。用 *n* 缩进下一个输出行，或为下一个输出行对当前的缩进加 ± *n*。默认尺度为 em。

示例

```
.in 10
.ti -5
这段的第一行伸出 5em ...
.in -10
```

.tl

`.tl ' l' c' r '`

指定左边的、中间的或右边的标题。标题宽由 `.lt` 而不是 `.ll` 指定。使用 `%` 可获得页码，例如，`.tl ''- % -''`。

.tm

`.tm` *text*

终端消息。输出有关标准错误的文本 *text*。用于调试，也用于生成索引和交叉引用。

.tr

`.tr` *ab...*

转换字符 *a*（一对字符的第一个）为 *b*（一对字符的第二个）。

示例

生成大写字母并以后恢复。用于标题宏：

```
.tr aAbBcCdDeEfFgGhHiIjJkKlLmM    等等
.tr aabbccddeeffgghhiijjkkllmm    等等
```

.uf

`.uf` *f*

设置下划线字体为 *f*（通过 `.ul` 或 `.cu` 改变），默认为斜体。

.ul

`.ul` [*n*]

对后面的 *n* 个输入行加下划线（在 `troff` 中变为斜体）。不对字间的空格加下划线。`.cu` 用于连续的下划线。在 `troff` 中可以用 `.uf` 请求切换到下划线字体，但必须使用一个宏来加下划线。

.vs

`.vs [n]`

设置垂直间距为 *n*。假如没有参数，恢复原先的间距。当前的垂直间距存储在寄存器 `.v` 中。默认为 1/6 英寸。

.wh

`.wh` *n* [*xx*]

“ when ”请求。当到达位置 *n* 处时，执行宏 *xx*；负值按页底部计算。假如没有指定 *xx*，则清除该位置处的陷阱（陷阱在执行给定宏的页上）。假如使用 `.ch` 将一个宏放到另一个宏上，则两个陷阱就处在同一个位置上。使用 `.wh` 它们不能放在同一个位置处。

转义序列

转义序列	作用
\\	阻止或延迟对 \ 的解释
\e	输出转义字符的可打印版本（通常为 \）
\′	′（锐音符）；和 \(aa 相同
\`	`（沉音符）；和 \(ga 相同
\-	-（在当前字体中的减号）
\.	句点（点）
\ 空格符	不能填充的一个空格尺寸的间隔字符
\ 换行符	隐藏（忽略）换行符
\0	数字宽度的空格
\\|	1/6em 宽的间隔字符（在 nroff 中为 0）
\^	1/12em 宽的间隔字符（在 nroff 为 0）
\&	不输出，0 宽的字符
\!	透明的线指示器
\"	注释的开始
\$*n*	插入宏参数 $1 \le n \le 9$
\%	默认的可选的连字符
\(*xx*	名为 *xx* 的字符。可参见后面的 “ 特殊的字符 ” 部分
*x 或 *(*xx*	插入字符串 *x* 或 *xx*
\a	不加解释的前导字符
\b′abc...′	创建大括号的函数
\c	处理和当前行连续的下面的行
\C′*abcd*′	名为 *abcd* 的字符（只用于 ditroff）
\d	向下移动 1/2em（在 nroff 中为 1/2 行）
\D′l*x,y*′	从当前的位置 *x,y* 划一条线（只用于 ditroff）
\D′*c d*′	在当前的位置左边画一个直径为 *d* 的圆（只用于 ditroff）
\D'e *d1 d2*'	在当前位置左边画一个水平直径为 *d1* 而垂直直径为 *d2* 的椭圆（只用于 ditroff）

转义序列	作用
\D'a *x1 y1 x2 y2*'	在当前位置按反时针方向画弧，中心在 *x 1* , *y 1* ，端点在 *x1+x2,y1+y2* （只用于 ditroff）
\D'~*x1 y1 x2 y2...*'	从当前的位置通过指定的坐标画云形轨（只用于 ditroff）
\f*x*或\f(*xx*或\f*n*	改变成字体 *x* 或 *xx* 或到位置 *n*。假如 *x* 为 P，则返回先前的字体
\g*x*或\g(*xx*	格式化数字寄存器 *x* 或 *xx*，适合于和 .af 一起使用
\h'*n*'	局部的水平移动；向右移动 *n*，或者，假如 *n* 为负值则向左移
\H'*n*'	设置字符高度为 *n* 磅，不改变宽度（只用于 ditroff）
\k*x*	在寄存器 *x* 中标记水平输入位置
\l'*nc*'	画长度为 *n* 的水平线（*c* 可选）
\L'*nc*'	画长度为 *n* 的垂直线（*c* 可选）
\n*x*, \n(*xx*	向数字寄存器 *x* 或 *xx* 中插数
\n+*x*, \n+(*xx*	向数字寄存器 *x* 或 *xx* 中插数，应用自动递增
\n-*x*, \n-(*xx*	向数字寄存器 *x* 或 *xx* 中插数，应用自动递减
\N'*n*'	当前字体的字符数 *n*（只用于 ditroff）
\o'*abc...*'	加粗字符 *a*、*b*、*c*...
\p	断开并展开输出行
\r	反向垂直移动 1em（在 nroff 中为反转行）
\s*n*, \s±*n*	改变磅值为 *n* 或加 *n*，例如，\s0 返回先前磅值
\s(*nn*, \s±(*nn*	就像 \s，但允许明确的两字符磅值（仅用于最新的 ditroff）
\S'*n*'	输出向右倾斜 *n* 度。负值为向左倾斜。0 值关闭倾斜（只用于 ditroff）
\t	不解释的水平制表符
\u	反向（向上）移动 1/2em（在 nroff 中为 1/2 行）
\v'*n*'	垂直移动；向下移动 *n*，或，假如 *n* 为负则向上移动
\w'*string*'	将字符串 *string* 的宽度添写进去
\x'*n*'	附加的行间隔函数（负 *n* 在以前加空格，正 *n* 在以后加空格）
\X'*text*'	作为设备控制函数输出 *text*（只用于 ditroff）
\z*c*	采用 0 宽度输出 *c*（不留空间）
\{	开始多行有条件的输入

转义序列	作用
\}	结束多行有条件的输入
x	*x*，上面没有列出的任何字符

预定义寄存器

有两种类型的预定义寄存器：只读和读写。即使其中一些实际上返回字符串值，它们也都是通过 \n 转义序列进行访问的。

只读寄存器

.$ 在当前宏命令级可用的参数个数。

$$ troff 进程的进程 ID（只用于 ditroff）。

.A 假如使用 -a 选项，则在 troff 中设置为 1；在 nroff 中总为 1。

.F 当前输入文件的名称（只用于最新的 ditroff）。

.H 用基本单位表示的所能达到的水平分辨率。

.L 当前的行间距值（由 .ls 设置）（只用于最新的 ditroff）。

.R 未使用的数字寄存器的数目（只用于最新的 ditroff）。

.T 假如使用选项 -T 则在 nroff 中设置为 1；在 otroff 中总为 0；在 ditroff 中，字符串 *(.T 包含了 -T 的值。

.V 用基本单位表示的所能达到的垂直分辨率。

.a 最近一次使用 \x′n′ 而产生的后续行的附加行间距。

.b 加粗等级（只用于最新的 ditroff）。

.c 从当前输入文件读入的行数。

.d 在当前转移中的当前垂直位置；若没有转移，则和寄存器 nl 相等。

.f 当前的字体序号（在 otroff 中为 1 到 4；在 ditroff 为 1 到 99）。

.h 在当前页或转移中的文本基线的高水位线标记。

.i 当前的缩进。

.j 当前的调整模式。

.k 当前输出的水平位置。

`.l` 当前的行宽。

`.n` 在前一输出行上文本部分的长度。

`.o` 当前的页边距。

`.p` 当前的页长度。

`.s` 当前的磅值。

`.t` 到下一个陷阱的距离。

`.u` 在填充模式中为 1，在非填充模式中为 0。

`.v` 当前的垂直行间距。

`.w` 先前字符的宽度。

`.x` 保留的版本相关寄存器。

`.y` 保留的版本相关寄存器。

`.z` 当前转移的名称。

读写寄存器

`%` 当前的页码。

`ct` 字符类型（由 `\w` 函数设置）。

`dl` 最后完成转移的宽度（最大）。

`dn` 最后完成转移的高度（垂直大小）。

`dw` 一周的当前天数（1 到 7）。

`dy` 一月的当前天数（1 到 31）。

`hp` 在输出行上的当前水平位置。

`ln` 输出行号。

`mo` 当前的月（1 到 12）。

`nl` 最后输出的文本基线的垂直位置。

`sb` 基线下字符串的深度（由 `\w` 函数产生）。

`st` 基线上面字符串的高度（由 `\w` 函数产生）。

`yr` 自从 1900 年的年数[a]。

a： 这里有一个潜在的 Y2K 问题。在 2000 年将为 100。

特殊字符

本部分列出了以下的特殊字符：

- 驻留在标准字体上的字符
- 混杂的字符
- 创建大括号符号。
- 数学符号
- 希腊字符

下面第一个表中的字符在标准字体上可用。其余表的字符在特殊字体上可以使用。

表 12-2：在标准字体上的字符

输入	字符	字符名
'	’	闭引号
`	‘	开引号
\(em	—	长破折号（“ m ”的宽度）
\(en	-	短破折号（“ n ”的宽度）
\-	-	在当前字体中的减号
-	-	连字号
\(hy	-	连字号
\(bu	•	着重号
\(sq	□	正方形
\(ru	_	比例尺
\(14	1/4	1/4
\(12	1/2	1/2
\(34	3/4	3/4
\(fi	fi	fi 连字符
\(fl	fl	fl 连守符
\(ff	ff	ff 连字符
\(Fi	ffi	ffi 连字符
\(Fl	ffl	ffl 连字符
\(de	°	度
\(dg	†	剑号

表 12-2：在标准字体上的字符（续）

输入	字符	字符名
\(fm	′	下标
\(ct	¢	分币符号
\(rg	®	注册商标
\(co	©	版权

表 12-3：混杂的字符

输入	字符	字符名称
\(sc	§	节
\(aa	′	锐音符
\′	′	锐音符
\(ga	‘	沉音符
\`	‘	沉音符
\(ul	_	下划的破折号
\(->	→	右箭头
\(<-	←	左箭头
\(ua	↑	上箭头
\(da	↓	下箭头
\(br	|	框界尺
\(dd	‡	双剑号
\(rh	☞	向右的手
\(lh	☜	向左的手
\(ci	◯	圆

表 12-4：创建大括号符号

输入	字符	字符名
\(lt	⎧	大括号左半部分的顶部
\(lk	⎨	大括号左半部分的中部
\(lb	⎩	大括号左半部分的底部
\(rt	⎫	大括号右半部分的顶部
\(rk	⎬	大括号右半部分的中部
\(rb	⎭	大括号右半部分的底部

表 12-4：创建大括号符号（续）

输入	字符	字符名
\(lc	⎡	大方括号左半部分的顶部
\(bv	⎪	粗垂直线
\(lf	⎣	大方括号左部分的底部
\(rc	⎤	大方括号右半部分的顶部
\(rf	⎦	大方括号右半部分的底部

表 12-5：数学符号

输入	字符	字符名
\(pl	+	数学加号
\(mi	−	数学减号
\(eq	=	数学等号
\(**	*	数学星号
\(sl	/	斜线（匹配反斜线）
\(sr	√	平方根
\(rn	‾	根的短延伸部分
\(>=	≥	大于或等于
\(<=	≤	小于或等于
\(==	≡	恒等于
\(~~	≈	约等于
\(ap	∞	近似
\(!=	≠	不等于
\(mu	×	乘
\(di	÷	除
\(+-	±	加减
\(cu	∪	杯形符（并集）
\(ca	∩	帽形符（交集）
\(sb	⊂	... 的子集
\(sp	⊃	... 的父集
\(ib	⊆	包含于或等于
\(ip	⊇	包含或等于
\(if	∞	无穷大

表 12-5：数学符号（续）

输入	字符	字符名
\(pd	∂	偏导数
\(gr	∇	梯度
\(no	¬	非
\(is	∫	积分号
\(pt	∝	与 ... 成比例
\(es	∅	空集
\(mo	∈	属于
\(or	|	或

希腊字符

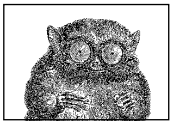
和大写英文字母等价的字符可以在标准字体上使用，在表 12-6 中的字符只能存在于特殊字体上。

表 12-6：希腊字符

输入	字符	字符名	输入	字符	字符名
\(*a	α	alpha	\(*A	A	ALPHA
\(*b	β	beta	\(*B	B	BETA
\(*g	γ	gamma	\(*G	Γ	GAMMA
\(*d	δ	delta	\(*D	δ	DELTA
\(*e	ε	epsilon	\(*E	E	EPSILON
\(*z	ζ	zeta	\(*Z	Z	ZETA
\(*y	η	eta	\(*Y	H	ETA
\(*h	θ	theta	\(*H	Θ	THETA
\(*i	ι	iota	\(*I	I	IOTA
\(*k	κ	kappa	\(*K	K	KAPPA
\(*l	λ	lambda	\(*L	Λ	LAMBDA
\(*m	μ	mu	\(*M	M	MU
\(*n	ν	nu	\(*N	N	NU
\(*c	ξ	xi	\(*C	Ξ	XI
\(*o	o	omicron	\(*O	O	OMICRON
\(*p	π	pi	\(*P	Π	PI

表 12-6：希腊字符（续）

输入	字符	字符名	输入	字符	字符名
\(*r	ρ	rho	\(*R	Ρ	RHO
\(*s	σ	sigma	\(*S	Σ	SIGMA
\(ts	ς	Terminal sigma			
\(*t	τ	tau	\(*T	Τ	TAU
\(*u	υ	upsilon	\(*U	ϒ	UPSILON
\(*f	φ	phi	\(*F	Φ	PHI
\(*x	χ	chi	\(*X	Χ	CHI
\(*q	ψ	psi	\(*Q	Ψ	PSI
\(*w	ω	omega	\(*W	Ω	OMEGA



第十三章

mm 宏

本章将介绍以下主题:

- 按字母顺序概述 *mm* 宏
- 预定义的字符串名
- *mm* 宏用到的数字寄存器
- 其他保留的宏名和字符串名
- 示例文档

按字母顺序概述 mm 宏

.1C	.1C 返回单列格式。
.2C	.2C 开始双列格式。
.AE	.AE 结束摘要（参见 **.AS**）。

.AF

.AF [*company name*]

首页替换格式。改变首页的“主题/日期/来自”格式。如给出参数，其他标题不会受影响；没有参数，则不显示公司名和标题。

.AL

.AL [*type*] [*indent*] [1]

初始化编号列表或字母顺序列表。指定列表类型 *type* 和文本的缩进 *indent*。如第三个参数为 1，则项与项之间不允许有空格。用 .LI 标记列表中的每一项，用 .LE 结束列表。默认类型是编号列表。默认文本缩进在寄存器 Li 中指定。

type

1 阿拉伯数字

A 大写字母

a 小写字母

I 大写罗马数字

i 小写罗马数字

.AS

.AS [*type*] [*n*]

按指定的类型 *type*、缩进 *n* 个空格开始摘要。只用 .TM 和 .RP。用 .AE 结束。

type

1 在封面或首页上的摘要

2 在封面上的摘要

3 在备注上的摘要，作为文件封面

.AT

.AT *title*

在正式备注中，作者的头衔 *title* 列于作者名字之后。

.AU

.AU *name* [*init*] [*loc*] [*dept*] [*ext*] [*room*]

作者姓名 *name* 和其他信息（最多有 9 个参数）放在正式备注的开头。

.AV	.AV *name* *name* 的批准签字行。关闭正式备注中的宏。
.B	.B [*barg*] [*parg*] ... 设置 *barg* 为加粗字（在 nroff 中为下划线或加粗），设置 *parg* 为先前的字体。最多可设 6 项参数。
.BE	.BE 结束页底的文字块，并在脚注（如果有）与页脚之间输出。参见 **.BS**。
.BI	.BI [*barg*] [*iarg*] 设置 *barg* 项为加粗字（在 nroff 中为下划线或加粗），设置 *iarg* 项为斜体。最多可设 6 项参数。
.BL	.BL [*indent*] [1] 初始化项目符号表。指定文本的缩进为 *indent*，默认值为 3，由寄存器 Pi 指定。如第 2 项参数为 1，则项目间不允许有空行。
.BR	.BR [*barg*] [*rarg*] 设置 *barg* 项为加粗字（在 nroff 中为下划线或加粗），设置 *rarg* 项为罗马字。最多可设 6 项参数。
.BS	.BS 在页底开始文字块的输出，并在脚注（如果有）与页脚之间输出。用 .BE 结束。
.CS	.CS [*pgs*] [*other*] [*tot*] [*figs*] [*tbls*] [*ref*] 封面信息，提供给正式备注使用。参数分别表示各个项目的个数，通常由系统自动算出，也可以指定一个值代替计算的结果。

.DE	.DE 结束由 .DS 开始的静态显示或由 .DF 开始的浮点显示。
.DF	.DF [*type*] [*mode*] [*rindent*] 开始浮点显示。 即如果文字输出所需要的空间超过当前页剩余的空间，则留作下页显示，并由后面的文字填满当前页（参见 De 和 Df 寄存器）。默认类型*type*是不缩进的，默认模式*mode*是不填充的，*rindent*是一个数量，按照这个数量缩短行的长度值以便与右边界对齐来放置文本。用 .DE 结束显示。 type L 或 0　不缩进（默认值）。 I 或 1　标准缩进。 C 或 2　每行各自居中。 CB 或 3　整块居中。 mode（模式） N 或 0　不填充模式（默认值）。 F 或 0　填充模式。
.DL	.DL [*indent*] [1] 初始化虚线列表。指定文本的缩进为 *indent*，默认值为 3，由寄存器 Pi 指定。如果第 2 项参数为 1，则项目间不允许出现空行。
.DS	.DS [*type*] [*mode*] [*rindent*] 开始静态显示。即如果页面剩余空间不足以显示全部内容，则进行分页，所剩文字在下一页顶部继续开始显示。关于*type*、*mode*和*rindent*的内容，参见 **.DF** 。用 .DE 结束显示。
.EC	.EC [*caption*] [*n*] [*flag*] 等式标题。可选用参数代替默认的数字，标志*flag*决定数字*n*的用法。参见 **.EQ**。

.EC	flag（标志） 0 *n* 是数字前缀（默认值）。 1 *n* 是后缀。 2 *n* 代替数字。
.EF	.EF ['*left*'*center*'*right*'] 输出三部分字符串作为偶数页的页脚，这三部分在偶数页的底部向左对齐、居中对齐、向右对齐。
.EH	.EH ['*left*'*center*'*right*'] 输出三部分字符串作为偶数页的页头，这三部分在偶数页的顶部向左对齐、居中对齐、向右对齐。
.EN	.EN 结束等式显示。参见 **.EQ**。
.EQ	.EQ [*text*] 开始等式显示，按照 eqn 进行等式处理，用 *text* 作标号（参见 **.EC**）。用 .EN 结束等式显示。关于 eqn 详见十七章。
.EX	.EX [*caption*] [*n*] [*flag*] 展示标题 *caption*。可选用参数代替默认的数字，标志 *flag* 决定了数字 *n* 的用法。 flag 0 *n* 是数字前缀（默认值）。 1 *n* 是后缀。 2 *n* 代替数字。
.FC	.FC [*text*] 使用 *text* 进行正式关闭。

.FD

`.FD [n] [1]`

设置默认的脚注格式为*n*，用下面的表进行说明。第二项参数为`1`，每次遇到一个第一层标题，则脚注编号从 1 开始。

值	连字符连接	对齐	文字缩进	标号调整
0	关	开	开	靠左
1	开	开	开	靠左
2	关	关	开	靠左
3	开	关	开	靠左
4	关	开	关	靠左
5	开	开	关	靠左
6	关	关	关	靠左
7	开	关	关	靠左
8	关	开	开	靠右
9	开	开	开	靠右
10	关	关	开	靠右
11	开	关	开	靠右

.FE

`.FE`

结束脚注。参见 **.FS**。

.FG

`.FG [title] [n] [flag]`

后跟插图标题*title*。可选用参数代替默认的数字，标志*flag*决定数字*n*的用法。

flag

`0` *n* 是数字前缀（默认值）。

`1` *n* 是后缀。

`2` *n* 代替数字。

.FS

`.FS [c]`

用 *c* 作为指示器开始脚注。默认值是已编号的脚注。用 `.FE` 结束。

.H

.H n [*heading*] [*suffix*]

输出第*n*层的编号标题，*n*的值为1至7。选项*suffix*添加在标题后面，可作为脚注标记，或其他不需在目录表中出现的文字。更多的信息可以参见下面的相应部分。

数字寄存器

Ej 弹出页面。

Hb 在标题后面断开。

Hc 标题居中。

Hi 标题后第一段的类型。

Hs 标题后的间距。

Hu 无编号标题。

字符串

HF 控制字体。

HP 字号（磅值）。

宏

.HM 标题标记。

.HU 无编号标题。

.HX, .HY, .HZ 用户提供的宏，在页眉输出时调用。

.HC

.HC [*c*]

用字符*c*作为连字符指示器。

.HM

.HM [*H1*] ... [*H7*]

设置7个层次的标题标记类型。每个标题可以是阿拉伯数字（1或001）、罗马字（i或I）和字母（a或A）。

.HU

.HU *heading*

无编号标题。同.H，但不输出标题标记（参见数字寄存器Hu）。

.HX	.HX *dlevel rlevel text* 用户提供的退出宏，在输出标题之前执行。 如果 .H由用户调用，导出层次*dlevel*等于真正的层次*rlevel*。如果使用了 .HU，*dlevel*等于 Hu 寄存器的值，*rlevel*的值为 0。在两种情况下，*text* 都是实际的标题内容。
.HY	.HY *dlevel rlevel text* 用户提供的退出宏，在输出标题的中间执行。有关 *dlevel*、*rlevel* 和 *text* 的内容，参见 **.HX**。
.HZ	.HZ *dlevel rlevel text* 用户提供的宏，在输出标题之后执行。有关 *dlevel*、*rlevel* 和 *text* 的内容参见 **.HX**。
.I	.I [*iarg*] [*parg*] 设置 *iarg* 为斜体（在 nroff 中为下划线），设置 *parg* 为前面用过的字体。最多使用 6 个参数。
.IB	.IB [*iarg*] [*barg*] 设置 *iarg* 为斜体（在 nroff 中为下划线），设置 *barg* 为粗体。最多使用 6 个参数。
.IR	.IR [*iarg*] [*rarg*] 设置 *iarg* 为斜体（在 nroff 中为下划线），设置 *rarg* 为罗马字。最多使用 6 个参数。
.LB	.LB *n m pad type* [*mark*] [*LI-space*] [*LB-space*] 列表开始。完全控制列表格式。每个列表项前用 .LI开始，用 .LE结束列表： *n* 文本缩进。 *m* 标记缩进。

pad

与标记关联的填充内容。

type

如果为 0，则用指定的标记 *mark*。如果不为 0，而且标记 *mark* 为 `1`、`A`、`a`、`I` 或 `i`，则列表自动编号或以字母顺序排列。在这种情况下，类型 *type* 控制了标记 *mark* 的显示方式。例如：如果 *mark* 当前值为 1，*type* 取以下结果：

类型	结果
`1`	1.
`2`	1)
`3`	(1)
`4`	[1]
`5`	<1>
`6`	{1}

mark

每个列表项的符号或文本标签。*mark* 可为空值（创建悬挂缩进）、文本字符串，或为 `1`、`A`、`a`、`I`、`i`，会自动创建以数字编号或以字母顺序编号的列表。参见 **.AL**。

LI-space

在每个 `.LI` 宏引起的列表项之间输出的空行数。默认值为 1。

LB-space

由 `.LB` 宏自己输出的空行数。默认值为 0。

.LC

`.LC [`*n*`]`

清除直到 *n* 层的列表。

.LE

`.LE [1]`

结束由 `.AL`、`.BL`、`.DL`、`.LB`、`.ML` 或 `.VL` 开始的列表项。参数 `1` 表示在列表之后留出一个空白行（.5v）。

.LI

.LI [*mark*] [1]
text

列表项。列表必须进行初始化（参见 **.AL**、**.BL**、**.DL**、**.LB**、**.ML** 和 **.VL**），并用 .LE 关闭。如果指定了 *mark*，它代替列表初始化宏设置的 *mark* 值。如果 *mark* 与第二个参数 1 一起指定，则指定的 *mark* 作为当前标记的前缀。

.ML

.ML *mark* [*indent*] [1]

用指定的 *mark* 初始化列表，*mark* 可为一个或多个字符。指定文字的缩进（默认值比 *mark* 多一个空格）。如果第三个参数为 1，则省略列表项之间的空格。

.MT

.MT [*type*] [*title*]

指定备注的类型 *type* 和标题 *title*。控制正式备注的格式，必须在其他元素（如：.TL、.AF、.AU、.AS 和 .AE）之后指定。用户提供的标题 *title* 成为页码的前缀。

type

0	无类型
1	文件的备注（默认）
2	程序员的附注
3	工程师的附注
4	发布的文件
5	外部文字
string	输出字符串 *string*

.ND

.ND *date*

新的日期。修改出现在正式备注中的日期。

.NE	.NE 注释结束。参见 **.NS**。
.nP	.nP 低两格缩进编号段落。参见 **.P**。
.NS	.NS [*type*] 注释开始。与 .MT 1 和 .AS 2/.AE（文件备注）一起使用指定封面注释。否则用于正式备注的结尾。指定注释类型 *type*。 type 0 拷贝到（默认） 1 拷贝到（要小心留意） 2 拷贝（不带 att.）到 3 att 4 atts 5 Enc 6 Encs 7 在单独的封页之下 8 写文字到 9 备注到 10 拷贝（带 atts.）到 11 拷贝（不带 atts.）到 12 摘要到 13 完全备注到 *string* 拷贝字符串 *string* 到
.OF	.OF ['*left*'*center*'*right*'] 在奇数页的底部分别向左对齐、居中对齐、向右对齐输出页脚字符串。

.OH	.OH ['*left*'*center*'*right*'] 在奇数页的顶部分别向左对齐、居中对齐、向右对齐输出页眉字符串。
.OK	.OK [*topic*] 其他关键字。指定出现在正式备注封面上的题目 *topic*。最多有 9 个参数。
.OP	.OP 强制使用一个奇数页。
.P	.P [*type*] 另起一段。可指定段类型 *type* 代替默认值。可设置不同的寄存器来控制默认格式： Pt 文档的段类型（默认值为 0）。 Pi 缩进值（默认值为 3n）。 Ps 段间距（默认值为一个空行）。 Np 此项设置为 1 会产生编号段落。 type 0 左对齐（默认）。 1 缩进 2 除显示（.DE）、列表（.LE）和标题（.H）外，还进行缩进。
.PF	.PF ['*left*'*center*'*right*'] 在每页的底部分别向左对齐、居中对齐、向右对齐输出页脚字符串。在字符串中用 \\\\nP 得到页号。参见 **.EF** 和 **.OF**。
.PH	.PH ['*left*'*center*'*right*'] 在每页的顶部分别向左对齐、居中对齐、向右对齐打印页眉字符串。在字符串中用 \\\\nP 得到页号。参见 **.EH** 和 **.OH**。

.PM	.PM [*type*] 每页的属性标记。 type P 保密。 N 公开。
.PX	.PX 用户退出的页面标题。在恢复默认环境后调用。参见 **.TP**。
.R	.R 返回罗马字体（在 nroff 中是结束下划线或加粗）。
.RB	.RB [*rarg*] [*barg*] 设置 *rarg* 为罗马字，设置 *barg* 为字体加粗。最多有 6 个参数。
.RD	.RD [*prompt*] 从终端读输入，可选用 *prompt* 作为提示。
.RF	.RF 参考文本的结尾。参见 **.RS**。
.RI	.RI [*rarg*] [*barg*] 设置 *rarg* 为罗马字，设置 *barg* 为斜体。最多有 6 个参数。
.RL	.RL [*indent*] [1] 初始化参考列表，以方括号中的数字为列表编号。设置文本缩进，默认值通过寄存器 Li 设置。如果第二个参数为 1，则省略列表项间的空格。
.RP	.RP [*counter*] [*skip*] 产生参考页面。

.RP

counter

0 重置参考计数器（默认）。

1 不重置参考计数器。

skip

0 放在一个独立的页面上（默认）。

1 不发布下面的 .SK。

2 不发布前面的 .SK。

3 不发布前面或后面的 .SK。

.RS

.RS [*strname*]

开始自动编号的参考。用 .RF 结束。如果提供了 *strname*，则将它用作troff预处理程序的字符串，该字符串保存了参考的编号（用方括号括起来）及适当行的动作。这样以后就可以从文档的文字中再次引用这些参考。

示例

```
J. Programmer\*(Rf
.RS W1
.I "Whizprog \- The Be All and End All Program,"
J. Programmer, Wizard Corp, April 1, 1999.
.RF
describes the design of
.IR whizprog .
The second chapter\*(W1
presents

an especially insightful analys is. ...
```

.S

.S [[±]*n*] [[±]*m*]

设置字号为 *n*（磅值），垂直间距为 *m*（只用于 troff）。另一方面，每个参数值可由当前值（C）、默认值（D）或以前的值（P）增加或减小来指定。默认字号为 10，默认垂直间距为 12。

.SA

.SA [*n*]

设置右页边距为*n*。对于nroff，默认值不进行调整；对于troff，则进行调整。

n 可取如下值：

0 不调整。

1 调整。

.SG

.SG [*typist*] [1]

在签名行中作者名部分增加打字员*typist*（作者名从.AU宏中获得）。如果第二项参数为1，作者的地址、部门等与第一作者的名字置于同一行，而不与最后一名作者置于同一行。

.SK

.SK *n*

跳过*n*页。类似于.bp请求。

.SM

.SM *x* [*y*] [*z*]

字符串大小减小一磅。与多项参数关联，减小值如下表所列参数：

其中的参数	动作
1	第一个字符串减小一磅
2	第一个字符串减小一磅
3	中间字符串减小一磅

.SP

.SP [*n*]

输出*n*个空行。不会累加两个相继.SP宏的空行请求。

.TB

.TB [*title*] [*n*] [*flag*]

提供表格标题*title*。可选用参数代替默认的数字，标志*flag*决定了数字*n*的用法。

.TB	flag 0 *n* 是数字的前缀（默认）。 1 *n* 是后缀。 2 *n* 代替数字。
.TC	.TC [*slevel*] [*spacing*] [*tlevel*] [*tab*] [*head1*] ... 用参数指定的格式生成目录表。寄存器C1设置的值决定了目录表保存的标题层次。 *slevel* 设置了前面有空格的标题层次。*spacing* 设置了标题前的空格数，默认值为 1；第一层标题前有一个空行。 *tlevel* 和 *tab* 决定页码的位置。如果标题层次小于或等于 *tlevel*，则与页码一起在右边空白处输出；否则，标题和页码用两个空格隔开。如果页码在右边空白处，同时 *tab* 为0，则使用点作为前导字符进行输出；否则使用空格。
.TE	.TE 结束表格。参见 **.TS**。
.TH	.TH [*N*] 结束表格标题，前面必须用 .TS H开始。用N禁止表格标题，直到下一页。
.TL	.TL [*charge* [*file*]] *text* 给出正式备注的标题。*charge* 和 *file* 是备注的“ charging case ”和“ filing case ”；在 Bell 系统之外，没有太大用处。
.TM	.TM [*n*] 给出技术性备注的数字 *n*。

.TP	.TP 页面顶部的宏。在新的一页开始处自动调用，并在有标题输出时执行它。参见 **.PH**。
.TS	.TS [*H*] 启动由 tbl 处理的表格。用 H 将表格标题放置到所有页面上。用 .TH 结束表格标题。用 .TE 结束表格。有关 tbl 的更多信息，参见第十七章。
.TX	.TX 在目录列表的标题前执行用户定义的宏。
.TY	.TY 在目录列表的页眉前，执行用户定义的宏。
.VL	.VL *n* [*m*] [1] 初始化可变项目列表。用于产生缩进的或有标号的段落。文本缩进 *n* 个空格，标记缩进 *m* 个空格。如果第三个参数为 1，则省略列表项间的空格。用 .LI 开始列表项，并指定每个项目的标号，用 .LE 结束列表。
.VM	.VM [*n*] [*m*] 上下边距。页顶留 *n* 行空白，页底留 *m* 行空白。
.WC	.WC [*x*] 将列宽或脚注宽度修改为 *x*。 x 的取值 FF 所有的脚注与第一个相同。 -FF 关闭 FF 模式。正常默认模式。 WD 加宽显示。 -WD 使用默认列宽模式。

.WC	WF 加大脚注宽度。 -WF 关闭 WF 模式。

预定义的字符串名

BU 项目符号，与 \(bu 作用相同。

Ci 目录表中列表项的分层缩进。

DT 如果没有替换，则表示当前日期，格式为月、日、年（例如：January 1, 2000）。

EM Em 破折号（在 troff 中用 em 破折号，在 nroff 中用双连字符）。

F 脚注编号生成程序。

HF 每层标题使用的字体（1= 罗马，2= 斜体，3= 粗体）。

HP 每层标题使用的字号。

Le 标题设置为 “ LIST OF EQUATIONS ”。

Lf 标题设置为 “ LIST OF FIGURES ”。

Lt 标题设置为 “ LIST OF TABLES ”。

Lx 标题设置为 “ LIST OF EXHIBITS ”。

RE SCCS 版本号和 *mm* 宏的版次。

Rf 参考编号生成程序。

Rp 标题设置为 “ REFERENCES ”。

Tm 商标符号。小字号的 “ TM ”，位于文字后面的上半行。

mm 宏用到的数字寄存器

表 13-1 列出了 *mm* 宏的数字寄存器。寄存器名的旁边有一个剑符†，表示只在命令行中设置或在格式化程序读取*mm*宏定义前设置寄存器的值。每个寄存器有单个字符的名称，可在命令行中加 -r 选项设置。

表 13-1：mm 宏数字寄存器

寄存器	描述
A†	如果置为1，则忽略技术性备注的标题，并在字母开头留出适当的空格（参见 **.AF** 宏）
Au	首页忽略作者信息（参见 **.AU** 宏）
C†	表示复制文件类型（原件、草稿等）的标志
Cl	为目录表保存的标题层数（参见 **.TC** 宏）。默认值为 2
Cp	如果设置为 1，则插图和表格的列表与目录表在同一页出现；否则，在另一页显示
D†	如果设置为 1，则使用 debug 模式（即使遇到常见的严重错误，*mm* 宏也会继续执行下去）。默认值为 0
De	如果设置为 1，则每次浮点显示后会弹出页面。默认值为 0
Df	设置浮点显示的格式（参见 **.DF** 宏）
Ds	设置静态显示前后使用的空格
E†	主题 / 日期 / 来自的字体。0（粗体，默认值）或 1（罗马字）
Ec	等式计数器，每执行一次 .EC 宏后，计数器加 1
Ej	在标题前弹出页面的标题层次。默认值为 0，不弹出
Eq	如设置为 1，则在左侧页边空白处写出等式标号。默认值为 0
Ex	展示计数器，每执行一次 .EX 宏后，计数器加 1
Fg	插图计数器，每执行一次 .FG 宏后，计数器加 1
Fs	脚注垂直间距
H1 ...H7	标题层次计数器，从 1 至 7，每执行相应层的 .H 宏使计数器递增，或如寄存器 Hu 给出了层数，执行 .HU 宏使计数器递增。寄存器 H2 至 H7 由较低层数的 .H 宏（或 .HU 宏）重置为 0
Hb	在正文输出之前，断开的标题层次。默认值为 2
Hc	居于中间的标题层次。默认值为 0
Hi	标题之后的缩进类型。值为 0（向左对齐），1（缩进，置认值），2（除遇到 .H 宏、.LC 宏、.DE 宏外，缩进）
Hs	标题之后有空格的标题层次。默认值为 2
Ht	标题的编号类型：1（单独的）或 0（连接的，默认值）

表 13-1：mm 宏数字寄存器（续）

寄存器	描述
Hu	设置无编号标题的层次。默认值为 2
Hy	如设置为 1，则允许使用连字符。默认值为 0
L†	设置页面的长度。默认值为 66v
Le	目录列表后输出等式列表的标志：0（不输出，默认值）或 1（输出）
Lf	与 Le 类似，但是输出插图列表的标志
Li	默认的列表缩进值。默认值：对于 nroff 为 6n，对于 troff 为 5n
Ls	设置嵌套列表项之间的间距。默认值为 6（所有列表层之间的间隔）
Lt	与 Le 类似，但是输出表格列表的标志
Lx	与 Le 类似，但是输出展示列表的标志
N†	设置页号类型： 0 所有页都有页眉（默认值） 1 第一页页眉当页脚输出 2 第一页没有页眉 3 分节页为页脚 4 只有调用 .PH 宏，才有页眉 5 分节页和分节图为页脚
Np	设置段落编号的风格：0（无编号，默认值）或 1（有编号）
O	页面偏移量。对于 nroff，值为无刻度的数字，只表示字符的位置，默认值为 9（.75i）；对于 troff，值为有刻度的数字，默认值为 .5i
Oc	设置目录表中页编号的风格：0（小写罗马，默认值）或 1（阿拉伯数字）
Of	设置插图标题中的编号分隔符：0（句点，默认值）或 1（连字符）
P	当前页号
Pi	段落缩进值。默认值：对于 nroff 为 5n；对于 troff 为 3n
Ps	段间距。默认值为 3v
Pt	段类型。值为 0（左对齐，默认值），1（缩进），2（除遇到 .H 宏、.LC 宏、.DE 宏外，缩进）
Pv	通过设置 0（默认）来禁止“ PRIVATE ”（私有）页眉
Rf	参考计数器，每执行一次 .RS 宏后，计数器加 1

表 13-1：mm 宏数字寄存器（续）

寄存器	描述
S†	默认 troff 字号，默认值为 10。垂直间距为 \nS+2
Si	显示的标准缩进。默认值：对于 nroff 为 5n；对于 troff 为 3n
T†	nroff 输出设备类型。为指定的设备设置寄存器
Tb	表格计数器，每执行一次 .TB 宏后，计数器加 1
U†	调用 .H 和 .HU 后的 nroff 下划线风格。如未设置，用连续的下划线；如已经设置，则标点符号和空白不需下划线。默认值为 0
W†	页宽（行和标题长度）。默认值为 6i

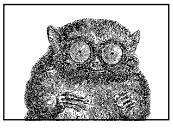
其他保留的宏名和字符串名

在 *mm* 宏中，可以安全使用的宏或字符串名如下：由单个小写字母组成；由两个字符组成，其中的第一个字符为小写字母，第二个字符为除小写字母以外的其他任何字符。这其中，只有 c2 和 np 已经使用。

示例文档

```
.ND "April 1, 1999"
.TL
Whizprog \- The Be All and End All Program
.AF "Wizard Corp."
.ds XX "012 Binary Road, Programmer's Park, NJ 98765-4321"
.AU "J. Programmer" "" XX
.AT "Coder, Extraordinaire"
.\" Abstract
.AS 1
This memorandum discusses the design and
mplementation of
.I whizprog ,
the next generation of really
.B cool
do-it-all programs.
.AE
.\" Released paper
.MT 4
.H 1 Requirements
.P
The following requirements were identified. ...
```

```
.H 1 Analysis
.P
Here is what we determined. ...
.H 1 Design
.P
After much popcorn, we arrived at the
following design. ...
.H 1 Implementation
.P
After more popcorn and lots of Jolt Cola, we
implemented
.I whizprog
using ...
.H 1 Conclusions
.P
We're ready to blow the socks off the market!
.SG
.CS
```



第十四章

ms 宏

本章将介绍以下主题：

- 按字母顺序概述 *ms* 宏
- 页面布局的数字寄存器
- 保留的宏名和字符串名
- 保留的数字寄存器名
- 示例文档

按字母顺序概述 ms 宏

.1C	.1C 在 .2C 或 .MC 宏后返回单列格式。调用 .1C 宏将产生分页。
.2C	.2C 开始双列格式。用 .1C 返回单列格式。
.AB	.AB 在封面上开始摘要。用 .AE 结束摘要。

.AE	.AE 结束由 .AB 开始的摘要。
.AI	.AI *name* *address* 输出作者所在单位的名称、地址等。在封面的排序上，通常跟在 .AU 宏之后。有多个成对的作者 / 单位名时，最多可重复 9 次。
.AU	.AU *name* 输出作者姓名。在封面的排序上，通常跟在 .TL 宏后，放在 .AL 宏前；有多名作者时，最多可重复 9 次。
.B	.B [*text*] [*text2*] 以黑体输出文本 *text*。如果提供 *text2*，将它与 *text* 连接，但用前一种字体输出。如果没有提供参数，则等同于 .ft3 或 .ftB。
.B1	.B1 为下面的文字加边框。用 .B2 结束边框。
.B2	.B2 结束加边框的文本（用 .B1 开始）。
.BD	.BD 开始块的显示。完全按原文件样式输出正文，以最长一行为基准居中对齐。与 .DS B 作用相同。用 .DE 结束。
.BX	.BX *word* 为字 *word* 加边框。由于填充的问题，通常不能一次为多个字加边框。若要为多个字加边框，则应用不可填充的空格分隔各个字（*space*）。

.CD	.CD 开始居中显示，每行显示独立居中。与 .DS C作用相同。用 .DE结束。
.DA	.DA 输出当天的日期，作为每页的中间页脚。
.DE	.DE 结束由 .DS开始显示的文本。
.DS	.DS [*type*] 开始显示文本。用 .DE结束。 type B 文字块靠左对齐，整块居中；参见**.BD**。 C 居中显示；参见**.CD**。 I 缩进显示（默认）；参见**.ID**。 L 靠左居中显示；参见**.LD**。
.EN	.EN 结束由 .EQ开始的等式显示。
.EQ	.EQ 开始显示由 eqn处理的等式。用 .EN结束。关于 eqn，详见十七章。
.FS	.FS 开始脚注。以连续行写脚注文本。用 .FE结束。
.FE	.FE 结束由 .FS开始的脚注。

.I	.I [*text*] [*text2*] 用斜体输出文本*text*。如果提供参数*text2*，则将它与*text*连接，但用前一种字体输。如果两个参数都没有提供，则等同于.ft 2或.ft I。
.ID	.ID 开始缩进显示。完全按原文件样式输出文本，但缩进8 en。与.DS I作用相同。使用.DE结束。
.IP	.IP *label n* 缩进带有悬挂标号*label*的段落*n*个空格。.RS和.RE用于嵌套缩进。
.KE	.KE 结束由.KS开始的静态排版或由.KF开始的浮点排版。
.KF	.KF 开始浮点排版。用.KE结束。也就是说，如果文本输出所需要的空间超过当前页剩余空间，则这些文本会留在下一页排版，用显示后面的文本填充当前页。
.KS	.KS 开始静态排版。用.KE结束。由.KS与.KE包围的文本显示在同一页上。如当前页面容纳不下，则会分页显示。
.LD	.LD 开始靠左对齐显示。文字块居中对齐，但块内每行文字靠左对齐。与.DS L作用相同。用.DE结束。
.LG	.LG 增大2个字号（只用于troff）。用.NL恢复正常类型。

.LP	`.LP` 开始块段落。段间间距由寄存器 PD 决定。默认值：`troff` 为 .5v，`nroff` 为 1 行。
.MC	`.MC` *cw gw* 开始多列模式，列宽为 *cw*，列间距为 *gw*。当前行长能容纳多少列，宏即可产生多少列。用 `.1C` 返回单列模式。
.ND	`.ND date` 给出日期，不用当前日期。参见 **.DA**。
.NH	`.NH [`*n*`]` *heading text* 编号的节标题；节号 *n* 自动增加。
.NL	`.NL` 恢复默认的类型尺寸（只用于 `troff`）。在 `.LG` 或 `.SM` 之后使用。
.PP	`.PP` 开始标准的缩进段落。段落缩进值保存在寄存器`PI`中（默认为5 en）。
.QE	`.QE` 结束由 `.QS` 开始的引用段落。`.QS`/`.QE` 与 `.QP` 相似。
.QP	`.QP` 开始引用的段落，两端缩进，段前段后留空行，并且（在 `troff` 中）字号减小 1 磅。
.QS	`.QS` 开始引用的段落，保持当前字号和垂直间距。

.R	.R 返回罗马字；本质上与 .ft R 作用相同。
.RE	.RE 结束用 .RS 开始的一层相对缩进。
.RP	.RP 为“ released paper ”初始化标题页。
.RS	.RS 右移。增加一层相对缩进。用 .RE 结束。通常与 .IP 一起使用。
.SG	.SG 输出一个签名行。
.SH	.SH *heading text* 无编号的节标题。参见 **.NH**。
.SM	.SM 换成较小的字号（只用于 troff）。用 .NL 恢复正常类型。
.TE	.TE 结束由 tbl 处理的表格。参见 **.TS**。
.TH	.TH 结束表格标题。必须以 .TS H 开头。
.TL	.TL *multiline title* 封面的标题行。可给出多行标题，用下一个宏（在封面的序列中通常为 .AU）结束。

.TS	.TS [*H*] 开始由tbl处理的表格。用H宏将表格标题放在所有页上（用.TH结束表格标题）。用.TE结束表格。关于tbl的内容，参见第十七章。
.UL	.UL 下面的文本加下划线，适用于troff。

页面布局的数字寄存器

名字	意义	默认值
CW	列宽	7/15 行宽
FL	脚注长度	11/12 行宽
FM	页底空白	1 英寸
GW	列间距	1/15 行宽
HM	页顶空白	1 英寸
LL	行宽	6 英寸
LT	标题宽度	6 英寸
PD	段间距	.3v
PI	段缩进值	5 en
PO	页面偏移量	1 英寸
PS	字号	10 磅
QI	引用缩进	5 en
VS	垂直行间距	12 磅

保留的宏名和字符串名

下列宏名和字符串名由*ms*宏软件包使用。为确保兼容性，在现有的宏中，避免使用这些名字。斜体的*n*表示这些名字中包含有数字（通常为数字寄存器取的值）。

```
,    .]   :    [.   [c   [o   ^    `    ~
1C   2C   AB   AE   AI   An   AT   AU   AX
B    B1   B2   BB   BG   BT   BX   C    C1
C2   CA   CC   CF   CH   CM   CT   DA   DW
DY   EE   EG   EL   EM   EN   En   EQ   EZ
FA   FE   FF   FG   FJ   FK   FL   FN   FO
```

```
FS   FV   FX   FY   HO   I    IE   IH   IM
In   IP   IZ   KD   KF   KJ   KS   LB   LG
LP   LT   MC   ME   MF   MH   MN   MO   MR
ND   NH   NL   NP   OD   OK   PP   PT   PY
QE   QF   QP   QS   R    R3   RA   RC   RE
Rn   RP   RS   RT   S0   S2   S3   SG   SH
SM   SN   SY   TA   TC   TD   TE   TH   TL
TM   TQ   TR   TS   TT   TX   UL   US   UX
WB   WH   WT   XF   XK   XP
```

保留的数字寄存器名

下列数字寄存器名称由 *ms* 宏软件包使用。斜体的 *n* 表示这些名字中包含有数字（通常为其他数字寄存器取的值）。

```
nT    AJ    AV    BC    BD    BE    BH    BQ    BW
CW    EF    FC    FL    FM    FP    GA    GW    H1
H2    H3    H4    H5    HM    HT    I0    IF    IK
IM    IP    IR    IS    IT    IX    In    Jn    KG
KI    KM    L1    LE    LL    LT    MC    MF    MG
ML    MM    MN    NA    NC    ND    NQ    NS    NX
OJ    PD    PE    PF    PI    PN    PO    PQ    PS
PX    QI    QP    RO    SJ    ST    T.    TB    TC
TD    TK    TN    TQ    TV    TY    TZ    VS    WF
XX    YE    YY    ZN
```

当编写自己的宏时，最安全的办法是用大小混写的字母作为宏名。（用大写字母会与保留的宏名冲突，用小写字母会与 `troff` 请求冲突。）

示例文档

```
.ND April 1, 1999
.\" Released paper
.RP
.TL
Whizprog \- The Be All and End All Program
.AU
J. Programmer
.AI
Wizard Corp.
012 Binary Road
Programmer's Park, NJ 98765-4321
USA
.\" Abstract
```

```
.AB
This memorandum discusses the design and
implementation of
.I whizprog ,
the next generation of really
.B cool
do-it-all programs.
.AE
.NH
Requirements
.PP
The following requirements were identified. ...
.NH
Analysis
.PP
Here is what we determined. ...
.NH
Design
.PP
After much popcorn, we arrived at the
following design. ...
.NH
Implementation
.PP
After more popcorn and lots of Jolt Cola,
we implemented
.I whizprog
using ...
.NH
Conclusions
.PP
We're ready to blow the socks off the market!
.SG
```

第十五章

me 宏

本章将介绍以下主题：

- 按字母顺序概述 *me* 宏
- 预定义字符串
- 预定义数字寄存器
- 示例文档

按字母顺序概述 me 宏

.1c	.1c 返回到单列格式。参见 **.2c**。
.2c	.2c 进入双列格式。用 .bc 强制产生一个新列。用 .1c 结束双列格式。
.ar	.ar 用阿拉伯数字设置页码。

.b	.b *w x* 将 *w* 设置为粗体，*x* 为前一种字体。
.(b	.(b *type* 开始块的排版。用 .)b 结束。 type C 文字块居中排版。 F 文字块分散对齐排版。 L 文字块左对齐排版。
.)b	.)b 结束由 .(b 开始的块排版。
.ba	.ba *n* 设置基本的缩进值为 *n*。
.bc	.bc 开始列；用在 .2c 之后。
.bi	.bi *w x* 设置 *w* 为粗斜体，*x* 为前一种字体。
.bl	.bl *n* 留出 *n* 个空行。在块内等同于 .sp *n*。
.bu	.bu 开始一个标有项目符号的段落。
.bx	.bx *w x* 在文字框内设置 *w*，*x* 紧跟在文字框外。

.+c	.+c *title* 开始有标题 *title* 的章。
.$c	.$c *title* 开始有标题 *title* 的编号章。
.$C	.$C *keyword n title* 用户可定义的宏。由 .$c 调用，提供关键字 *keyword*（如：“ Chapter ”或“ Appendix ”）、章（chapter）或附录（appendix）号（*n*）和标题 *title*。
.(c	.(c 开始居中对齐的块。用 .)c 结束。
.)c	.)c 结束由 .(c 开始、居中对齐的块。
.(d	.(d 开始延迟的文本。用 .)d 结束。
.)d	.)d 结束延迟的文本。用 .pd 输出文本。
.ef	.ef *′l′c′r′* 在所有偶数页的底端分别靠左对齐、居中对齐和靠右对齐打印页脚。
.eh	.eh *′l′c′r′* 在所有偶数页的顶端分别靠左对齐、居中对齐和靠右对齐输出页眉。
.EN	.EN 结束由 .EQ 开始显示的等式。

.ep	.ep 结束本页并输出脚注。
.EQ	.EQ *format title* 开始显示由 eqn 处理的等式，使用输出格式 *format* 并在等式右边空白处输出标题 *title*。用 .EN 结束。关于 eqn 详见第十七章。 format C 居中对齐。 I 缩进。 L 居左对齐。
.$f	.$f 调用输出页脚。
.(f	.(f 开始脚注正文。用 .)f 结束。
.)f	.)f 结束由 .(f 开始的脚注正文。
.fo	.fo *′ l′ c′ r′* 在所有页的底端分别靠左对齐、居中对齐和靠右对齐输出页脚。
.GE	.GE 结束由 gremlin 创建的图片。必须与前面的 .GS 一起使用。只在 *me* 最近的版本上才有。
.GF	.GF 结束由 gremlin 创建的图片，并“回扫”到原来纵向的位置上。必须与前面的 .GS 一起使用。只在 *me* 最近的版本上才有。

.GS	.GS [*flag*] 开始由 gremlin 创建的图片，后面必须跟有 .GE 或 .GF。只在 *me* 最近的版本上才有。（gremlin 是画图工具，与 pic 相似，由 UCB 开发。）默认动作是使图片居中。 flag 的取值 L 将图片置于左边空白的旁边。 R 将图片置于右边空白的旁边。
.$H	.$H 通常指未定义的宏，在一个页面上输出文本之前立即调用。可用于列标题等。
.$h	.$h 调用输出页眉。
.he	.he *′l′c′r′* 在所有页的顶端分别靠左对齐、居中对齐和靠右对齐输出页眉。
.hl	.hl 画一条与页宽相等的横线。
.hx	.hx 不在下页输出页眉和页脚。
.i	.i *w x* 设置 *w* 为斜体（在 nroff 中为下划线），*x* 为前一种字体。
.IE	.IE 结束由 ideal 创建的图片。必须与前面的 .IS 宏一起使用。只在 *me* 最近的版本上才有。

.IF	`.IF` 结束由`ideal`创建的图片，“回扫”到原来纵向的位置上。必须与前面的`.IS`宏一起使用。只在*me*最近的版本上才有。
.IS	`.IS` 开始由`ideal`创建的图片，后面必须用`.IE`宏或`.IF`宏。只在*me*最近的版本上才有。（`ideal`是与`pic`类似的画图工具，由贝尔实验室开发。）
.ip	`.ip` *label n* 带有悬挂标号的段落缩进*n*个空格。
.ix	`.ix` [±*n*] 缩进但不断行。等于*n*中的`'`。
.(l	`.(l` *type* 开始列表。用`.)l`结束。 type `C` 居中列表。 `F` 分散对齐列表。 `L` 左对齐列表。
.)l	`.)l` 结束由`.(l`开始的列表。
.ll	`.ll` +*n* 设置行宽为+*n*（所有环境下）。这是一个宏，不是`nroff/troff`的`.ll`请求。

.lo	`.lo` 装入局部定义的宏的集合（通常为 `/usr/lib/me/local.me`）。不在最近版本中。
.lp	`.lp` 开始块段落（居左对齐）。
.m1	`.m1` *n* 设置页面顶部和标题间的间隔为 *n* 个空格。
.m2	`.m2` *n* 设置标题和正文第一行之间的间隔为 *n* 个空格。
.m3	`.m3` *n* 设置页脚与正文的间隔为 *n* 个空格。
.m4	`.m4` *n* 设置页脚与页底的间隔为 *n* 个空格。
.n1	`.n1` 在页边空白处从 1 开始编号多行。
.n2	`.n2` *n* 在页边空白处从 *n* 开始编号多行；如果 *n* 为 0，则停止编号。
.np	`.np` 开始一个有编号的段落。当前编号通过 `\n(%p` 进行访问。
.of	`.of` *'l'c'r'* 在所有奇数页的顶端分别靠左对齐、居中对齐和靠右对齐输出页脚。

.oh	.oh *′l′c′r′* 在所有奇数页的顶端分别靠左对齐、居中对齐和靠右对齐输出页眉。
.$p	.$p *title n d* 输出带有指定标题 *title*、节号 *n*、节深度 *d* 的节标题。
.$0	.$0 *title n d* 在每次调用 .$p 后自动调用。通常未定义，但可用于将每个节标题自动输出到目录表中，或用于其他类似功能。
.$*n*	.$*n* 在输出节深度 *n*（*n* 为 1~6）前调用的一些陷阱。由 .$p 调用。
.pa	.pa [±*n*] 等同于 .bp。
.pd	.pd 输出延迟的文本，由 .(d 和 .)d 指示。
.PE	.PE 结束由 pic 创建的图片。必须与前面的 .PS 一起使用。只在 *me* 最近的版本中才有。
.PS	.PS *vert indent* 开始由 pic 创建的图片。后面必须使用 .PE。只在 *me* 最近的版本才有。 *vert* 是为图片提供的垂直空间，*indent* 用来决定距左边空白多远的位置放置图片。
.pp	.pp 开始缩进的段落。

.q	`.q` *w x* 将 *w* 用双引号引起来，*x* 直接跟在引号外面。
.(q	`.(q` 开始主要的引用。用 `.)q` 结束。
.)q	`.)q` 结束由 `.(q` 开始的主要引用。
.r	`.r` *w x* 设置 *w* 为罗马字，*x* 为前一种字体。
.rb	`.rb` *w x* 设置 *w* 为粗体，*x* 为前一种字体。
.re	`.re` 每隔 0.5 英寸（在 `troff` 中）或每隔 0.8 英寸（在 `nroff` 中）设置一个制表符。
.ro	`.ro` 用罗马数字设置页号。
.$s	`.$s` 用 1.5 英寸的横线分隔脚注。
.sh	`.sh` 开始编号的节标题。
.sk	`.sk` 保留下页为空白。类似于 `troff .bp` 请求。

.sm	`.sm` *small reg* 将 *small* 和 *reg* 相连接，用 *small* 设置小一磅字号。只在最近的 *me* 版本中才有。
.sx	`.sx` +*n* 在第 *n* 层开始一个段落。
.sz	`.sz` *n* 设置字符字号为 *n*，并成比例地设置行间隔。
.TE	`.TE` 结束表格。参见 **.TS**。
.TH	`.TH` 结束表格标题。必须与前面的 `.TS H` 一起使用。
.th	`.th` 初始化一篇论文（不在最近的版本中）。
.tp	`.tp` 初始化一个标题页。
.TS	`.TS [H]` 开始由 `tbl` 处理的表格。用 `H` 将表格标题置于所有页（用 `.TH` 结束表格标题）。用 `.TE` 结束表格。关于 `tbl` 详见第十七章。
.u	`.u` *w x* *w* 使用下划线，并设置 *x* 为前一种字体。
.uh	`.uh` *title* 开始无编号的节标题 *title*。

.(x	.(x 开始索引条目。用 .)x 结束。
.)x	.)x [*page*] [*author*] 结束由 .(x 开始的索引条目。用 .xp 输出索引。 这些参数是可选项。如果 *page* 是下划线 " _ "，则会省略索引条目的页号。否则，*page* 为将要使用的页号，而不是自动计算的页号。 第二项参数为在条目结尾靠右对齐输出 ;例如, 可能会在输出作者姓名时用到。如果指定了作者，则必须有 *page* 项：使用 \n% 得到当前页号。
.xl	.xl *n* 设置行宽为 *n*（只用于当前环境）（这实际是 nroff/troff 的 .ll 请求）。
.xp	.xp 输出索引。参见 **.(x** 和 **.)x**。
.(z	.(z 开始浮点排版。
.)z	.)z 结束浮点排版。
.++	.++ *type header* 定义输入文章的节。用 *header* 标题字符串指定类型 *type*。 type A 附录 AB 摘要 B 参考文献

.++	C 章 P 预分节（目录表等） RA 附录，页号从 1 开始 RC 章，页号从 1 开始

预定义字符串

在许多最新版本的*ms*宏中出现了有剑符（†）的条目。需要在系统中仔细查对这些条目。

*	脚注号，由 .)f 宏递增
#	延迟文本号
[	上标；上移并紧缩字体
]	撤消上标
<	下标；下移并紧缩字体
>	撤消下标
-	3/4 em 短横线
dw	代表星期几，当作一个单词
mo	月，当作一个单词
td	今天的日期，形式为 January 20, 1999
lq	左引号
rq	右引号
$n†	节名
’†	锐音符
`†	沉音符
qa†	所有的
qe†	存在的
,†	变音符号
:†	元音变音

^† 插入记号

o† 圆圈（即斯的那维亚语的 Å）。用法为 A*o

v† 将“ v ”转换为捷克语 ě 。用法为 e*v

{† 开始上标

}† 结束上标

~† 波浪线

预定义数字寄存器

最近版本的 *ms* 出现了有剑符（†）的条目。需要仔细查对系统的这些条目。

$0† 节深度

$1† 第一节

$2† 第二节

$3† 第三节

$4† 第四节

$5† 第五节

$6† 第六节

$V† 显示相对的垂直间距

$c 当前列号

$d 廷迟文本号

$f 脚注号

$i† 段落基本缩进

$l 列宽

$m 起作用的列号码

$p 编号段落号

$s 列缩进

$v† 正文中相对垂直间距

bi 显示（块）缩进

bm 底部标题边界

bs 显示（块）前 / 后间距

bt† 块排版边界

ch 当前章号

df† 显示字体

es† 等式前 / 后的间距

ff† 脚注字体

fi† 脚注缩进（只对第一行）

m 页脚边界

fp† 脚注字号

fs 脚注前的间隔

fu† 脚注右缩进（距页右边空白）

hm 页眉边界

ii 层次交错的段落缩进

pf 段落字体

pi 段落缩进

po† 模拟页面偏移量

pp 段落字号

ps 段前间隔

qi 引用缩进（也用短线）

qp 引用文字字号

qs 引用文字前 / 后间隔

sf† 节标题字体

si† 每个节深度基本缩进值

so† 附加节标题偏移量

sp† 节标题字号

s† 节前的间隔

tf 标题字体

tm 顶层标题边界

`tp` 标题字号

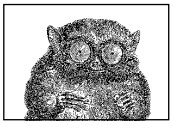
`xs` 索引条目前的间隔

`xu†` 索引右边的缩进（距页右边的空白）

`zs` 浮点排版文字前 / 后的间隔

示例文档

```
.tp
.(l C
Whizprog \- The Be All and End All Program
.sp
by
.sp
.ce 2
J. Programmer
Wizard Corp.
.)l
.+c Abstract
This memorandum discusses the design and
implementation of
.i whizprog ,
the next generation of really
.b cool
do-it-all programs.
.+c "The Whole Story"
.sh 1 Requirements
.pp
The following requirements were identified. ...
.sh 1 Analysis
.pp
Here is what we determined. ...
.sh 1 Design
.pp
After much popcorn, we arrived at the
following design. ...
.sh 1 Implementation
.pp
After more popcorn and lots of Jolt Cola, we
implemented
.i whizprog
using ...
.+c "Conclusion"
.pp
We're ready to blow the socks off the market!
```



第十六章

man 宏

本章将介绍以下主题：

- 按字母顺序概述 *man* 宏
- 预定义字符串
- 内部名称
- 示例文档

按字母顺序概述 man 宏

所有宏可有6项参数，用以改变字体或产生标题。用双引号引起多个字可得到较长的标题。

`.TS`、`.TE`、`.EQ`和`.EN`宏未由 *man* 宏定义。但由于`nroff`和`troff`忽略未知请求，所以它们仍然可用于参考页；`tbl`和`eqn`可正常工作。

.B	`.B [`*`text ...`*`]` 设置参数为粗体，各个参数之间留一个空格。如果未提供参数，则会将下一个输入行设为粗体。

.BI	.BI *barg iarg ...* 设置交替的 *barg* 为粗体，*iarg* 为斜体 , 中间不插入空格。
.BR	.BR *barg rarg ...* 设置交替的 *barg* 为粗体，*rarg* 为罗马字，中间不插入空格。
.DT	.DT 重置制表位为默认值，间隔为 1/2 英寸。
.HP	.HP [*indent*] *tag text* 开始一个有悬挂缩进的段落，左边有一个悬挂的标签。选项 *indent* 为段落的缩进量。标签文本跟在下一行。参见 **.TP** 中的示例。
.I	.I [*text ...*] 设置参数为斜体，各个参数之间留一个空格。如果没有参数，下一个输入行应设为斜体。
.IB	.IB *iarg barg ...* 设置交替的参数 *iarg* 为斜体，*barg* 为粗体，中间不插入空格。

.IP	.IP *tag* [*indent*]
	开始一个有悬挂缩进的段落，左边有一个悬挂的标签。与 .HP 和 .TP 不同，标签 *tag* 会作为参数传递给宏。选项 *indent* 是段落的缩进量。
	示例
	`.IP 1.`
	`The first point is ...`
	`.IP 2.`
	`The second point is ...`

.IR	.IR *iarg rarg ...* 设置交替的 *iarg* 为斜体，*rarg* 为罗马，中间不插入空格。

.IX	.IX *text* 索引宏。只用于 Solaris。只为 SunSoft 内部使用。
.LP	.LP 另起一段。就像 .PP。
.P	.P 另起一段。就像 .PP。

.PD	.PD [*distance*]
	
	设置段间距为*distance*。没有参数时，重置为默认值。最有用的是在一个段落中使用多个标签。
	
	示例
	显示两个选项完成同一件事情：
	
	`.PP`
	`.I Whizprog`
	`accepts the following options.`
	`.TP \w'\fB\-\^\-help\fP'u+3n`
	`.PD 0`
	`.B \-h`
	`.TP`
	`.PD`
	`.B \-\^\-help`
	`Print a helpful message and exit.`

.PP	.PP 另起一段。这个宏重置所有默认值，如字号、字体和间距。
.RB	.RB *rarg barg* ... 设置交替的*rarg*为罗马字，*barg*为粗体，中间不插入空格。
.RE	.RE 结束一个相对的缩进。每个 .RE要与前面的一个 .RS匹配。参见**.RS**的示例。

.RI	.RI *rarg iarg ...* 设置交替的 *rarg* 为罗马字，*iarg* 为斜体，中间不插入空格。

.RS	.RS [*indent*]
	
	开始一个相对的缩进。各个连续的 .RS 都增加一次缩进量。选项 *indent* 为后面文本的缩进量。各个 .RS 必须与 .RE 配合使用。
	
	示例
	`.PP`
	`There are a number of important points to remember.`
	`.RS`
	`.IP 1.`
	`The first point is ...`
	`.IP 2.`
	`The second point is ...`
	`...`
	`.RE`
	`Forget these at your own risk!`

.SB	.SB *arg ...* 设置参数为粗体，用较小的磅值，并使用空格隔开。
.SH	.SH *arg ...* 节标题。开始一个新的节，如 NAME 或 SYNOPSIS。用双引号引起多个字可得到较长的标题。
.SM	.SM *arg ...* 设置参数为罗马字，用较小的字号，用空格隔开。
.SS	.SS *arg ...* 子节标题。开始一个新的子节。用双引号引起多个字可得到较长的标题。

.TH

.TH *title section date ...*

表示标题。这是参考页用到的第一个宏，用于设置页眉和页脚行。标题 *title* 是参考页名；*section* 是带有参考页的一个节（一个数字，可能还跟有一个字母）；*date* 是参考页最后更新的日期。该宏后面的参数在不同的系统中有不同的规则。对于 Solaris，第四和第五个参数分别为左边的页脚和中间的页眉。

示例

```
.TH WHIZPROG 1L "April 1, 1999"
.SH NAME
whizprog \- do amazing things
...
```

.TP

.TP [*indent*]
tag text

开始一个有悬挂缩进的段落，左边有一个悬挂的标签。选项 *ident* 是段落的缩进量。标签跟在下一行。见 **.PD** 中的示例。

示例

```
.TP .2i
1.
The first point is ...
.TP .2i
2.
The second point is ...
```

预定义字符串

以下字符串是预定义的。其中，只有 R 和 S 记录在文档中。

字符串	在 troff 中的作用	在 nroff 中的作用
*lq	`` (“)	"
*(rq	'' (”)	"
*(PN	当前页号	当前页号
*(R	\(rg (®)	(Reg.)
*(S	恢复默认字号	恢复默认字号

内部名称

Solaris 的 *man* 宏使用了一些由 `]`、`}` 和 `)` 开头的宏、字符串和数字寄存器名。在自己的文件中应避免使用这些名字。

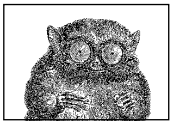
数字寄存器 `D`、`IN`、`LL`、`P`、`X`、`d`、`m` 和 `x` 由 Solaris 的 *man* 宏内部使用。在调用 `.TH` 宏前，应使用 `.nr D 1` 在奇数页和偶数页上产生不同的页脚（注 1）。

示例文档

```
.TH WHIZPROG 1 "April 1, 1999"
.SH NAME
whizprog \- do amazing things
.SH SYNOPSIS
.B whizprog
[
.I options
] [
.I files
\&... ]
.SH DESCRIPTION
.I Whizprog
is the next generation of really
.B cool
do-it-all programs. ...
.SH OPTIONS
.PP
.I Whizprog
accepts the following options.
.TP \w'\fB\-\^\-level\fP'u+3n
.PD 0
.B \-h
.TP
.PD
.B \-\^\-help
Print a helpful message and exit.
.TP
.BI \-\^\-level " level"
Set the level for the
.B \-\^\-stun
```

注 1: 经过测验实际的宏，得到了这条信息。文档中没有进行记载，所以实际内容可能会有所不同。

```
option.
.TP
.B \-\^\-stun
Stun the competition, or other beings, as needed. ...
.SH SEE ALSO
.IR "Whizprog \- The Be All and End All Program" ,
by J. Programmer.
.PP
.IR wimpprog (1)
.SH FILES
.B /dev/phaser
.br
.B /dev/telepath
.SH CAVEATS
.PP
There are a number of important points to remember.
.RS
.IP 1.
Use
.B \-\^\-help
to get help.
.IP 2.
Use
.B \-\^\-stun
with care. ...
.RE
Forget these at your own risk!
.SH BUGS
The
.B \-\^\-stun
option currently always uses
.BR "\-\^\-level 10" ,
making it rather dangerous.
.SH AUTHOR
J. Programmer,
.B jp@wizard-corp.com
```



第十七章

troff 预处理程序

本章分为以下 4 节，每节涵盖了 nroff/troff 格式系统的一个预处理程序：

- `tbl`
- `eqn`
- `pic`
- `refer`

每个预处理程序将代码翻译为`nroff/troff`请求和转义序列。它们处理的信息仅限于定义的宏，其他输入的文本不予处理。通常，会将一个或多个这样的预处理程序作为命令管道的一部分进行调用，来格式化一个文件：

```
$ pic file | tbl | eqn | troff options | spooler
```

在多用户系统中，一般总有一个通用的 shell 脚本用于格式化。可以选择不同的命令行选项来指定特殊格式命令用到的预处理程序。当然，也可以分别调用这些预处理程序。这对于确认语法的正确性或确定错误的位置是有帮助的。例如，命令：

```
$ tbl file
```

处理每个`.TS/.TE`宏对之间的输入内容，并将其转换成`tbl`代码。其他输入直接输出而未加改变。

在 SVR4 中，这些命令是 BSD 兼容软件包的一部分，可在`/usr/ucb`目录中找到。对于 Solrais，除`pic`而外，它们是系统的标准组成部分，可在`/usr/bin`目录下找到。

troff（groff, 见 *http://www.gnu.org*）的 GUN 版本与 tbl、eqn、pic 和 refer 版本一起提供。

tbl

tbl 是在 nroff/troff 中用于格式化表格的预处理程序。当在命令管线中使用时，tbl 必须在 eqn 之前。这使得对输出的处理更有效。tbl 命令语法如下：

```
tbl [options] [files]
```

tbl的规范参考文献是由L.L.Cherry 和M.E. Lesk发表的《Tbl-A Program to Format Tables》一文，编入 1990 在 Holt Rinehart &Winston 出版，由 AT&T 贝尔实验室的编辑 M.D. McIlroy 和 A.G. Hume 负责的《UNIX Programmer's Manual》第 10 版、卷 2，本文可从以下地址下载：*http://cm.bell-labs.com/cm/cs/doc/76/tbl.ps.gz*。

options

-me 在文件前预先加上 *me* 宏。

-mm 在文件前预先加上 *mm* 宏。

-ms 在文件前预先加上 *ms* 宏。

-TX 只以整行输出。当用nroff进行格式化或输出到一个不支持分行输出的设备上时有用（Solaris 的 tbl 没有该选项）。

一般的代码模式

在文本文件中，tlb代码形式如下：

```
.TS H
options;
format1
format2.
Column Titles
.TH
Item1    Item2    Item3
Item1    Item2     Item3 ...
.TE
```

tbl要成功地处理一个表，更多地取决于标题行，标题行由一个选项列表行和一个或多个格式行组成。如果表格的区域没有被文本块符号T{和T}围起来，表格的每个输入区域必须由制表符或指定的制表符号分隔开，而且表格每排要全部在一行内输入。

tbl 宏

`.TS` 开始表格。

`.TE` 结束表格。

`.TS H` 当表格连续在几页输出时有用。用 `.TH` 定义输出在每页上的标题。

`.TH` 与 `.TS H` 一起使用，结束表格标题的位置。

`.T&` 用新的格式行继续表格。

options

选项影响整个表格。选项由逗号或空格分隔，但选项行必须用分号结束。

`center` 将表格居中置于版面。

`expand` 以当前左右边界整版排版。

（*blank*） 靠左边界排版（默认）。

`box` 给表格加边框。

`doublebox` 给表格加双边框。

`allbox` 给表格内每项加边框。

`tab(`*x*`)` 指定制表符号为 *x*，代替制表符。

`linesize` *n* 设置线宽或边框为 *n* 磅。

`delim` *xy* 将 *x* 和 *y* 作为 `eqn` 的定界符。

格式

格式行对表格的每一列和每一行的布局独立发生作用。每个格式行有一个关键的字母可控制表格的一列。一列里的项必须由空格隔开，格式部分必须由句号结束。除最后一行外，每个格式行对应于表格的每一行，但在遇到下一个 `.T&` 之前，其下面的内容（如果有的话）都采用相同的格式。

关键字母

`c` 居中对齐。

`l` 靠左对齐。

`r` 靠右对齐。

n 对齐数字项。

a 按照字母顺序对齐子列。

s 水平跨越列，从前一列至本列。

^ 垂直跨越行，从上一行至本行。

关键修饰符

以下这些项必须跟随一个关键字母。

b 粗体。

i 斜体。

f*x* 字体 *x*。

p*n* 磅值 *n*。

v*n* 垂直行间距的磅值。只用于文本块。

t 从顶上的横线开始垂直跨越表格项（即，从 ^ 开始）。

e 等宽的列。

w(*n*) 最小列宽。同文本块一起用。*n* 可由任一可接受的 troff 单元给出。

n 列与列间的距离（默认值为 3，单位用 en）。

| 用单竖线分隔列。用于关键字母之间。

|| 用双竖线分隔列。用于关键字母之间。

_ 用单横线分隔行。代替关键字母。

= 用双横线分隔行。代替关键字母。

数据

数据部分包括表格的标题和正文。表格的每个项必须用制表符分隔开。在下面的描述中，➡ 代表制表符。

.xx 可能要用到 troff 请求（如 .sp *n* 和 .na 等）。

\ 每行的最后一个字符，合并当前行与下一行（隐藏换行符）。

\^ 跨越表格上下两行的项，将上面的项下移并垂直居中。

_ or = 一行中惟一的字符，将单横线或双横线扩展为表格的整个宽度。

\$_ or\$= 将单竖线或双竖线扩展为表格的列宽度。

`\_` 将单横线扩展为列内容的宽度。

`\R`*x* 输出 *x*s, 宽度与列内容相同。

`...`➠`T{` 作为表格项开始文本块。必须结束一行。当输入的文本多于一行或跨行输出多行时，有必要使用它。

`T}`➠`...` 结束文本块。必须开始一行。

一个 tbl 的示例

输入：

```
.TS
center box linesize(6) tab(@);
cb s s.
Horizontal Local Motions
_
.T&
ci | ci s
ci | ci s
ci | ci | ci
c | l s.
Function@Effect in
\^@_
\^@troff@nroff
_
\eh'n'@Move distance N
\e(space)@Unpaddable space-size space
\e0@Digit-size space
_
.T&
c | l | l.
\e|@1/6 em space@ignored
\e^@1/12 em space@ignored
.TE
```

结果：

水平的局部移动		
功能	作用于	
	troff	*nroff*
\h'n' \(space) \0	移动距离为N 没有填充的一个空格大小的间距 数字大小的间距	
| \^	1/6em间距 1/12em间距	忽略 忽略

eqn

eqn 是为便于数学公式排版而设计的预处理程序，与 nroff 一起使用 neqn。eqn 命令行语法如下：

```
eqn [options] [files]
```

eqn 的规范参考是《Typesetting Mathematics-User's Guide》，由 L.L.Cherry 和 B.W. Kernighan 撰写，发表于《UNIX Programmer's Manual》，第 10 版，第 2 卷，由 AT&T 贝尔实验室的编辑 M.D. McIlroy 和 A.G. Hume 负责，Holt Rinehart & Winston 出版，1990。该论文可从 *http://cm.belllabs.com/cm/cs/doc/74/eqn.ps.gz.* 下载。

options

`-d`*xy*
: 用 *x* 和 *y* 作为起始和终止定界符，与指定 eqn 命令 `delim` *xy* 相同。

`-f`*n* 改变为字体 *n*，与 `gfont` 命令相同。

`-p`*n* 减小上标和下标的字号 *n* 磅。如果未指定 `-p`，默认值是 3 磅。

`-s`*n* 减小字号 *n* 磅，与 `gsize` 命令相同。

`-T`*dev*
: 格式化输出到设备 *dev* 上。默认值从 TYPESETTER 环境变量中得到。与 neqn 一起使用无效（Solaris eqn 上没有该选项）。

eqn 宏

`.EQ` 开始数学排版。

`.EN` 结束数学排版。

用 checkeq 命令检查未匹配的宏对（并不是所有的系统都有该命令）。

数学符号

以下字符序列是公认的，翻译过来的意思为：

字符	翻译	字符	翻译
>=	≥	approx（约等于）	≈
<=	≤	hothing（空）	
==	≡	cdot（点）	.
!=	≠	times（乘）	×
+-	±	del	∇
->	→	grad（算子）	∇
<-	←	...	…
<<	<<	,...,	,…,
>>	>>	sum（求和）	Σ
inf（无穷）	∞	int（积分）	∫
partial（偏微分）	∂	prod（积）	Π
half（一半）	1/2	union（并）	∪
prime（撇）	′	inter（交）	∩

数学运算符号

数字、圆括号、方括号、标点符号及以下数学运算符以罗马字输出：

```
sin     cos     tan     arc
sinh    cosh    tanh
and     if      for     det
max     min     lim
log     ln      exp
Re      Im
```

希腊字符

可用大写或小写输出希腊字母。只要按希腊字母的发音拼写, 就可得到该字母。由于有些大写的希腊字母等同于罗马字，系统不支持使用大写希腊字母（如：A 代表 alpha，B 代表 beta）。

名称	字符	名称	字符
alpha	α	tau	τ
beta	β	upsilon	υ
gamma	γ	phi	π
delta	δ	chi	χ
epsilon	ε	psi	ψ
zeta	ζ	omega	ω
eta	η	GAMMA	Γ
theta	θ	DELTA	$\triangle$
iota	ι	THETA	Θ
kappa	κ	LAMBDA	Λ
lambda	λ	XI	Ξ
mu	μ	PI	Π
nu	ν	SIGMA	Σ
xi	ξ	UPSILON	Υ
omicron	o	PHI	Φ
pi	π	PSI	Ψ
rho	ρ	OMEGA	Ω
sigma	σ		

可区别的标记

有些关键字可用于在顶上标记字符。eqn会将标记置于合适的高度，会使用合适长度的上、下短横线。

字符	翻译
`x dot`	$\dot{x}$
`x dotdot`	$\ddot{x}$
`x hat`	$\hat{x}$
`x tilde`	$\tilde{x}$
`x vec`	$\vec{x}$
`x dyad`	$\overleftrightarrow{x}$
`x bar`	$\bar{x}$
`x under`	$\underline{x}$

eqn 认可的关键字

除字符名和可区别的标记外，eqn 认可以下关键字：

`above`	分隔堆间分块或矩阵的列。
`back` *n*	向后水平移动 em 的 *n* 个 1/100。
`bold`	改成粗体。
`ccol`	居中排列矩阵的一列。
`cpile`	创建一个居中的堆（与 `pile` 相同）。
`define`	为经常使用的字符串创建一个名字。
`delim` *xy*	定义两个字符来标记将要内部输出的 eqn 等式的左右边界。用 `delim off` 关闭定界符。
`down` *n*	向下移动 em 的 *n* 个 1/100。
`fat`	使当前的字体加粗。
`font` *x*	改成字体 *x*，*x* 为字体的名称或编号。
`from`	在求和、积分等相似的结构中指定下限。
`fwd` *n*	向前水平移动 em 的 *n* 个 1/100。
`gfont` *x*	将所有公式设置为全局字体 *x*。
`gsize` *n*	将所有公式设置为全局字号 *n*。
`italic`	改成斜体。
`lcol`	将矩阵的列设为左对齐。
`left`	创建大方括号、大花括号和竖线等。
`lineup`	排列不同行公式中的标记。
`lpile`	堆元素左对齐。
`mark`	标记公式的水平位置。与 `lineup` 一起用。
`matrix`	创建矩阵。
`ndefine`	创建一个定义，只在 neqn 运行时起作用。
`over`	创建分数。
`pile`	创建垂直的堆，每个元素上下居中对齐。
`rcol`	靠右对齐矩阵的列。

`right`	创建大的方括号、大花括号和竖线等。必须有一个匹配的 `left`。
`roman`	设置下面的常量为罗马字。
`rpile`	靠右对齐堆的元素。
`size` *n*	改变字号为 *n*。
`sqrt`	以下公式元素取平方根。
`sub`	开始下标。
`sup`	开始上标。
`tdefine`	创建只用于 eqn 的定义。
`to`	在求和、积分等相似的结构中指定上限。
`up` *n*	向上移动 em 的 *n* 个 1/100。
`~`	在输出时产生额外的空格。
`^`	产生空格，空格为 ~ 产生的空格的一半。
`{ }`	迫使 eqn 将一个元素视为一个单元。
`"..."`	引号内的字符串不遵循 eqn 的改变。

优先使用顺序

如果不用括号，eqn 执行的操作以下表中从左至右的顺序执行。

```
dyad    vec     under   bar
tilde   hat     do      dotdot
fwd     back    down    up
fat     roman   italic  bold
size    sub     sup     sqrt
over    from    to
```

这些操作组合到左边：

```
over   sqrt   left  right
```

所有其他操作组合到右边。

eqn定义了表达数学运算的语言。这样，数学公式内项目的组合、排序就有一定的语法规则。要了解全面的情况，见贝尔实验室备忘录。

eqn 示例

输入：

```
.EQ
delim %%
.EN
%sum from i=0 to inf c sup i~=~lim from {m -> inf}
sum from i=0 to m c sup i%
.EQ
delim off
.EN
```

结果：

$$\sum_{i=0}^{\infty} c^i = \lim_{m \to \infty} \sum_{i=0}^{m} c^i$$

输入：

```
.EQ
x ~=~ left [ { -b ~+-~ sqrt {b sup 2 - ~4ac} }
over 2a right ]
.EN
```

结果：

$$x = \left[\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \right]$$

pic

pic是图形语言程序，用于方便地画出简单的流程图和框图。pic提供了很多画图的方法，不但可用缩写词，而且还可将几何学与英语结合起来。例如，可以指定线条的方向、大小和起始点，也可以通过简单地描述“从这到那”达到同样的效果。

pic 命令行语法如下：

```
pic [files]
```

在 pic 中，可以用开始新的一行或分号结束对图元对象的完整描述。在前一行结尾用反斜杠（\），也可以在下一行继续对单个图元的描述。注释前面需要加井字符号（#）。

Solaris 没有 pic 命令。

pic 的规范参考资料是贝尔实验室计算机科技报告 #116，由 B.W. Kernighan 撰写。这篇文章可从 *http://cm.belllabs.com/cm/cs/cstr/116.ps.gz* 下载。这个文件描述的新版 pic 比这里所叙述的具有更多的功能，但是这些功能可能不通用。如果打算认真地用 pic 进行工作，应该看看这些内容。

pic 宏

`.PS [`*h* `[`*w*`]]` 开始 pic 的描述。如果指定了 *h* 和 *w*，则是需要图片的高度和宽度。可以扩大或缩小整个图片以填满空间。

`.PS <` *file* 在当前行的位置上读文件 *file* 的内容。

`.PE` 结束 pic 描述。

`.PF` 结束 pic、描述，返回到匹配 .PS 前的垂直位置。

在图片描述中内嵌的 troff 请求或宏未加改变地通过。它们在那里仍有意义。使用请求和宏产生垂直或水平的动作时要加以小心。

声明

pic 描述的开头，需要声明新的刻度和一些变量。pic 假设你需要 1 比 1 的比例，默认的单位是英寸。可以声明不同的刻度，如厘米，声明如下：

```
scale = 2.54
```

在描述中也可用变量代替数字，pic 会进行变量替换。替代：

```
line right n
```

可以使用变量，例如用 a，在描述的顶部声明：

```
a = n
```

以上命令写为：

```
line right a
```

变量名必须用小写字母开头，名称的其余部分可以使用大写或小写字母、数字和下划线。变量能在图与图之间保持它们的值。

图元

pic 认可几个基本图形对象或图元。图元由以下关键字指定：

```
arc     circle     move
arrow   ellipse    spline
box     line       "text"
```

语法

图元后面跟有相关的选项。本节后面再讨论选项。

arc [*cw*] [*options*]	弧线（默认是 1/4 圆周）。cw 选项指定一个顺时针方向的弧；默认值是反时针方向。
arrow [*options*]	画箭头。本质上与 line-> 相同。
box [*options*]	画方框。
circle [*options*]	画圆。
ellipse [*options*]	画椭圆。
line [*options*]	画线。
move [*options*]	移动。本质上是不可见的线。
spline [*options*]	光滑的曲线，用 then 选项可使方向呈梯度（斜的）变化；换句话说，用 line 画一条路线，当路线改变方向时，就会产生尖角。用 spline 会得到光滑的曲线。
"text"	在当前位置，文本居中放置。

options

下面的选项按功能分组。注意使用“ at ”、“ with ”和“ from ”等介词指定点。点以笛卡尔坐标系表示，或用相对于前面对象的值表示。

right [*n*] left[*n*] up [*n*] down [*n*]	图元的方向。默认方向是前面描述已经确定的方向。通过在选项线中使用两个方向来创建对角线的移动。每个方向后面都跟着长度 *n*。
rad *n* diam *n*	用半径或直径 *n* 创建图元。

`ht` *n* `wid` *n*	用高度或宽度 *n* 创建图元。对于箭头、线、曲线，高度和宽度等都指箭头的大小。
`same`	使用与最近匹配图元相同的尺寸来创建图元。
`at` *point*	在 *point* 处使图元居中放置。
`with .`*part* `at` *point*	将指定图元的一部分（如：顶或角）放在*point*位置。
`from` *point1* `to` *point2*	从第一个点 *point1* 至第二个点 *point2* 画图元。
`->`	箭头向前。
`<-`	箭头向后。
`<->`	箭头向两边。
`chop` *n m*	从图元的开头处切掉 *n*，结尾处切掉 *m*。如只有一个参数，两头切掉相同的值。如果没有参数，切掉默认的数量（通常是`circlerad`）。

`dotted`
`dashed`
`invis`
	用点线、虚线或不可见的线画图元（不可见的对象仍占输出空间）。默认是实线。

`then...`	在新的方向继续画图元。只与线、曲线、移动和箭头相关。可以放在文本之前或之后。
"*text*"	将文字居中放于对象的中间位置上。*text* 的选项在下节描述。

文本

文本必须用引号引起来。要断开一行，就要用引号分开引起来。如果没有给出下面的参数，文本总是在对象内居中。

`ljust` 文本靠左对齐，垂直居中。

`rjust` 文本靠右对齐，垂直居中。

`above` 文本置于中线以上。

`below` 文本置于中线以下。

对象块

几个图元可以合并成复杂的对象（例如，八边形）。声明为一个块后，这个复杂的对象可作为单个的对象处理：

```
Object: [
     description
      .
      .
      .
]
```

方括号用作定界符。注意对象作为一个位置的名字进行了声明，因此要用大写字母开头。

宏

使用宏可以重复相同的命令序列。语法为：

```
define sequence %
      description
           .
           .
           .
      %
```

这里，`%` 为定界符，也可使用描述里没有的任意字符。

宏可带参数，用 `$1` 至 `$9` 定义。用以下的语法格式调用宏：

```
sequence(value1,value2,...)
```

定位

在 `pic` 描述中，如果没有用坐标指定，开头的动作从（0,0）开始 。这样，当对象被置于第一个对象的左方或上方时，（0,0）点就会在图中向下和向右移动。

最后，所有的点都被格式化程序翻译为 x 和 y 坐标。增加或减少坐标值，可在图中指定一个特定的点。例如：

```
2nd ellipse + (.5,0)
```

这里指的是第二个椭圆中心右边 1/2 英寸处。

对象的 x 和 y 坐标是对象的中心位置。对象的 x 和 y 坐标，可在末尾用 .x 或 .y 指出。例如：

```
last box.x
```

表示最后画的方框的 x 坐标。可用相同的方法指出对象的物理属性。

.x	对象中心的 x 坐标。
.y	对象中心的 y 坐标。
.ht	对象的高度。
.wid	对象的宽度。
.rad	对象的半径。
.corner	对象的一个角。角在下面内容中描述。

除非指定其他位置，每个对象从最后一个对象的停止位置开始。但是，如果命令（命令序列）被花括号（{ }）围起来，则 pic 将返回到第一个括号前面的位置。

在对象之间定位

有两种方法指出前面的对象。

- 按顺序。例如：

```
1st box
3rd box
last box
2nd last box
```

- 用一个名字声明它，声明行用大写字母开头。例如：

```
Line1: line 1.5 right from last box.sw
```

要指出两个对象之间的点，或同一个对象内的两个点，可以写为：

fraction of the way between first.position and second.position

或（简写）：

fraction <*first.position, second.position*>

角

当指前一个对象时，pic 假设是指对象的中心，除非指定的是一个角。要指定角，用以下形式之一：

```
.corner of object
object.corner
```

例如：

```
.sw of last box
last box.sw
```

以下任一项都可指定有效的角：

n	北
s	南
e	东
w	西
ne	东北
nw	西北
se	东南
sw	西南
t	顶（同 n）
b	底（同 s）
r	右（同 e）
l	左（同 w）
start	画一个对象的起点
end	画一个对象的终点

也可用以下方法指定对象的一部分：

```
upper   right    lower    right
upper   left     lower    left
```

表达式

表达式用在 pic 需要使用数字值的地方（如当指定坐标或移动的数量时）。表达式由数字常量、变量和操作符组成。

pic 认可以下操作符。

+	加

- 减
* 乘
/ 除
% 取模（余数）
^ 指数

默认值

不同的系统可动态地控制对象的默认尺寸。按下面的形式输入描述行可改变这些默认值：

variable = value

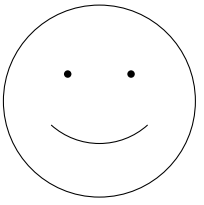
变量	默认值	变量	默认值
arcrad	0.25	ellipsewid	0.75
arrowwid	0.05	linewid	0.5
arrowht	0.1	lineht	0.5
boxwid	0.75	movewid	0.5
boxht	0.5	moveht	0.5
circlerad	0.25	scale	1
dashwid	0.05	textht	0
ellipseht	0.5	textwid	0

pic 的示例

输入：

```
.PS
define smile %
a = $1
circle radius a at 0,0
arc cw radius a*.75 from a*.5,-a*.25 to -a*.5,-a*.25
"\(bu" at a*.33,a*.25
"\(bu" at a*-.33,a*.25
%
smile(.5)
.PE
```

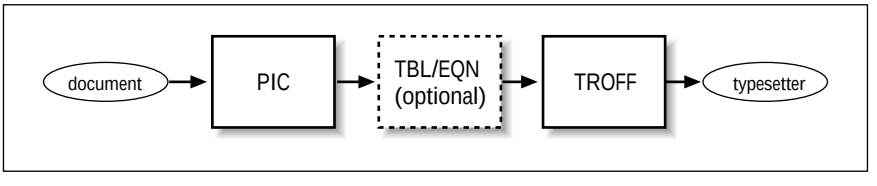
结果：



输入（从 CSTR#116）：

```
.PS
ellipse "document"
arrow
box "PIC"
arrow
box "TBL/EQN" "(optional)" dashed
arrow
box "TROFF"
arrow
ellipse "typesetter"
.PE
```

结果：



refer

`refer`是预处理程序，与一些相关的命令一起，管理参考文献数据库。数据库保存在独立的文件中，文档中简短的文献由扩充的正式版本替换掉。

本节最后提供了命令一览表，列出了 `refer` 的用法和选项，还列出了用于参考文献数据库的其他命令。

SVR4 不提供 `refer`，但 `refer` 是 Solaris 中的一个标准部分。

参考文献条目

参考文献数据库是文本文件，每个条目由一个或多个空行分隔。在一个条目内，每个字段由关键字母（用 *%letter* 给出）和相关的值组成。该值对相继的行发生作用，直到下一个由 `%` 开始的行。例如：

```
%T 5-by-5 Palindromic Word Squares
%A M.D. McIlroy
%J Word Ways
%V 9
%P 199-202
%D 1976
```

除 %A（作者）外，字段只提供一次，系统不提供不相关或无用的字段。

关键字母	意义
%A	作者名
%B	书中包含的文章
%C	城市（出版地）
%D	出版日期
%E	包含文章的书的编辑
%F	脚注号或标号（由 refer 提供）
%G	政府订购号
%H	题头注释，在参考之前输出
%I	出版商
%J	杂志包含的文章
%K	定位参考所用的关键字
%L	由 refer -k 所用的标号字段
%M	贝尔实验室备忘录
%N	卷内编号
%O	其他注释，在参考之后输出
%P	页号
%Q	公司或外国作者（姓名不颠倒）
%R	报告、文章或论文（未出版的）
%S	系列标题
%T	文章或书的标题
%V	卷号
%X	摘要（roffbib 使用，refer 不用）
%Y,%Z	refer 忽略

一般的代码方案

在文档中，refer 的使用看上去像这样：

```
Palindromes are fun.
Very large ones can be used to impress your friends.
Palindromic word squares
.[
%A McIlroy
.]
are even more amazing,
and should be reserved for impressing your boss.
...
.SH REFERENCES
.[
$LIST$
.]
```

这里显示的文档用了 refer 的集合模式（-e），所有的参考文献在文档的末尾输出，而不在列出文献的每个地方输出。

按字母顺序概述命令

addbib

addbib [*options*] *database*

交互地在数据库 *database* 中增加参考文献记录。

options

-a 不提示输入摘要。

-p *file*
用 *file* 作为提示梗概，每行都是提示、制表符，然后是要写的关键字母。

indxbib

indxbib *files*

为参考文献数据库文件 *files* 创建反向的索引。这些索引可为 lockbib 和 refer 所用。

产生的文件

对每个原始文件 *x*，indxbib 创建四个新文件：

x.ia 入口文件。

x.ib 登记文件。

x.ic 标记文件。

indxbib

x.ig　参考文件。

lookbib

lookbib *database*

查找由indxbib创建的参考文献数据库。lookbib用a>提示关键字并输出所有与关键字匹配的记录。如果未发现关键字，只会显示另一个提示符>。如果没有indxbib创建的反向索引文件，lookbib的操作会比较慢。参见 **addbib** 和 **indxbib**。

refer

refer [*options*] *files*

处理参考文献引用文件。除了用.[和.]括起来的行外，输入直接输出而不会改变。括起来的内容作为引用参考，并由独立的数据库保存。根据方括号提供的关键字，refer会产生troff.ds命令，该命令定义了包含相关信息的字符串。然后产生宏调用，对参考的内容进行适当的格式化。*ms* 和 *me* 宏软件包包含使用 refer 的宏定义。.[调用右边会添加合适的字符，表明参考内容的用法。使用-e选项，参考内容会被收集起来，作为一个整体放在最后。

options

-a[*n*]
　将前*n*个作者的名和姓换位（即，姓在前）。如果没有指定*n*，则所有名字都换位。

-b　空模式。不对文本增加内联参考内容。

-c*list*
　用 SMALL CAPS 将 *list* 列出的字母变为大写。

-e　收集参考内容，以便在结尾输出。参考的内容来自同一源，只输出一次。当遇到以下这些行，参考就会被输出出来。

```
.[
$LIST$
.]
```

-k*c*　用标号参考代替编号参考，数据库的%*c*字段提供数据。默认值为%L。

-l[*m*[,*n*]]
　用标号参考代替编号参考，标号由第一作者的姓氏和出版年份

refer

产生。如果提供了标号，*m*为作者姓氏的字母数，*n*为年份的最后几位。否则使用完整的姓名和年份。

`-n` 不搜索默认的文件（从`/usr/lib/refer/papers`中查找）。

`-p` *refsfile*
用*refsfile*作为参考列表。

`-s`*keylist*
根据*keylist*列出的字段对参考进行排序，隐含了`-e`。每个字母后面跟一个数字，表明将要用到多少个这样的字段。`A+`等于无限。默认为`-sAD`，按照第一作者和日期排序。

示例

先按所有的作者排序，再按日期排序，用`mybib`查看参考内容。

```
refer -sA+D -p mybib thesis.ms | tbl | eqn | troff -ms - | lp
```

roffbib

`roffbib` [*options*] [*files*]

输出参考文献数据库。`roffbib`是shell脚本，通过`refer`处理指定文件*files*（或如果没有文件，则处理标准输入），并输出参考文献。默认情况下，用`nroff`对参考文献进行格式化，用`-Q`选项来使用`troff`代替。

`roffbib`接受以下`nroff/troff`选项，并简单地传送给格式化程序：`-e`、`-h`、`-m`、`-n`、`-o`、`-q`、`-r`、`-s`和`-T`。详见第十二章。

options

`-H` *header*
设置标题为*header*。默认值是`BIBLIOGRAPHY`（该选项只用在脚本中，没有在文档中记载）。

`-Q` 用`troff`代替`nroff`。页面偏移量设为1英寸。

`-V` Versatec打印机/绘图仪的排版。该选项记载于参考页中，不在脚本中。

`-x` 格式化参考文献中`%X`字段的摘要或注释。形成带说明的资料目录时有用。`refer`不使用`%X`字段。

roffbib

示例

将数据库排序并在 PostScript 打印机上打印：

```
sortbib refs | roffbib -Q -x | /usr/lib/lp/postscript/dpost | lp
```

sortbib

sortbib [*option*] *files*

将一个或多个参考文献数据库排序。一般在使用 roffbib 输出时使用。最多可排 16 个数据库。记录长度不能超过 4096 字节。

option

-s *keys*

按照给定的键值 *keys* 排序。前四个 *keys* 影响排序，其余的会被忽略。*keys* 中的字母与参考文献条目中的关键字母相对应。为一个字母添加一个+，则在移到下一个之前，会按照关键字进行完全排序。

示例

先按作者排序，再按日期排序：

```
sortbib -sA+D myrefs | ...
```

按照作者、标题和日期排序：

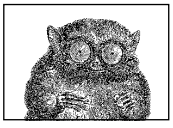
```
sortbib -sATD myrefs | ...
```

第四部分

软件开发

Unix操作系统通过提供极好的软件开发环境而赢得了广泛赞誉, SCCS、RCS和make在有效使用该开发环境的过程中起到了关键的作用。SCCS 和 RCS 允许一个源文件的多个版本保存在一个归档的文件中。make会自动更新一组关联的程序。

- 第十八章，SCCS
- 第十九章，RCS
- 第二十章，make 实用程序



第十八章

SCCS

本章将介绍以下主题：

- 简介
- 命令概述
- 基本操作
- 标识关键字
- 数据关键字
- 按字母顺序概述 SCCS 命令
- sccs 和伪命令

注意：对 RCS 比较熟悉的 SCCS（Source Code Control System, 源代码控制系统）用户，可从第十九章中的“ SCCS 用户转换指南 ”部分获得更多的知识，那里列出了 SCCS 命令和 RCS 等价命令。

要得到更多信息，参见列于参考文献中的《Applying RCS and SCCS》。

简介

SCCS 使你可以了解文档的每次修订情况，避免一个文档出现多个版本时造成的混乱。当程序功能增强而仍然需要最初版本时，SCCS 就特别有用。

一个文件的所有变化都保存在一个名为 `s.`*file* 的文件中，该文件称为一个 SCCS 文件。每当一个文件“ 调入 ”SCCS 中时，SCCS 会记录文件的哪一行在最新版本之后进行了修改或删除。从那些信息中，SCCS 可以根据需要重新产生该文件。每一组变化都依赖于所有以前各组的变化。

每组变化称为一个 *delta*，并分配一个 SCCS 标识字符串（sid）。*sid* 要么由两部分组成：版本及级别号（以 *a.b* 表示）；要么由四部分组成：版本、级别、分支和序列号（以 *a.b.c.d* 表示）。当同一个文件的两个运行版本要记录到 SCCS 中时，就要用到分支号和序列号。例如：*delta 3.2.1.1* 表示版本 3、级别 2、分支 1、序列号 1。

命令概述

SCCS 命令分成几大类。

基本设置和编辑命令

`admin` 创建新的 SCCS 文件并改变参数。

`get` 检索 SCCS 文件的版本。

`delta` 创建 SCCS 文件的一个新版本。

`unget` 取消一个 `get` 操作，不创建新的 delta。

修改 Delta 命令

`cdc` 修改与 delta 相关的注释。

`comb` 将几个连续的 delta 合并成一个。

`rmdel` 从 SCCS 中删除一个附属的 delta。

信息

`help` 输出一个命令摘要或阐明诊断消息。

`prs` 以一个指定的格式输出 SCCS 文件的几个部分。

`prt` 设定格式并输出一个或多个 SCCS 文件内容。只用于 Solaris。

`sact` 显示 SCCS 文件的编辑动作。

`what` 用 `get` 命令及通配符 `%z%` 查找某种模式的所有出现，并输出后续文本。

比较文件

`sccsdiff` 显示任意两个 SCCS 文件的差别。

`val` 验证一个 SCCS 文件。

基本操作

本节概述使用 SCCS 的步骤：

- 创建 SCCS 文件
- 检索文件
- 创建新的版本和分支
- 记录变化
- 警告

创建 SCCS 文件

`admin` 命令加一个 `-i` 选项可创建并初始化 SCCS 文件。例如：

```
admin -ich01 s.ch01
```

创建一个新的 SCCS 文件并用 `ch01` 的内容初始化，`ch01` 变为 *delta 1.1*. 如果不指定关键字，就会显示“ No id keywords (cm7) ”。一般情况下，“ id keywords ” 指文件中的变量，这些变量由 `get` 命令获得合适的值进行替换，标识创建的日期和时间、检索的版本号等等。本章的后面会列出标识关键字。

一旦创建了 `s.ch01` 文件，就会删除最初的 `ch01` 文件，可由 `get` 命令很容易地重新生成该文件。

检索文件

`get` 命令可以从 SCCS 中检索任意一个版本的文件。使用上面的示例，可以检索 `ch01`，只要键入：

```
get -e s.ch01
```

显示的信息是：

```
1.1
```

```
new delta 1.2
272 lines
```

这表明你“得到了”*delta 1.1*，文件有272行。当文件用`delta`命令再次进入SCCS文件`s.ch01`时，文件改为*delta 1.2*。

`-e`选项向SCCS表明准备对文件进行更多的修改，然后使其再重新进入SCCS。没有该选项，文件为只读的。`-e`选项除了产生读/写文件外，还创建一个`p.ch01`文件，当文件返回时，该文件记录了由SCCS使用的信息。

创建新的版本和分支

`get`的`-r`选项告诉SCCS你需要的版本和级别编号，如果没有指定级别，默认得到最高级别。可用如下命令，*delta 3.2*是版本号：

```
get -r3.2 ch01
```

然而，如下命令：

```
get -r3 ch01
```

返回版本3的最高级别，例如`3.8`。忽略`-r`选项，可得到最高版本号。换句话说，可以获得最新版本。

当大部分改变保存到一个文件中时，可以通过“获得”文件的下一个最高版本号，开始该文件的一个新版本。例如：如果一个文件的最高版本为3.2，要从版本4开始，可键入：

```
get -e -r4 ch01
```

显示以下信息：

```
3.2
new delta 4.1
53 lines
```

如果希望对同一个文件的老版本进行修改，可键入：

```
get -e -r2.2 ch01
```

得到以下信息：

```
2.2
new delta 2.2.1.1
121 lines
```

你已经创建了一个从主干版本 2.2 产生的分支。这个 delta 的改变不会影响那些主干的 delta 值，如 2.3、3.1 等等。

记录变化

一旦对 SCCS 文件进行了修改，使用如下命令将其返回到 SCCS 中：

```
delta s.ch01
```

你会获得有关这些改变的注释提示。`delta`命令会执行自己的 `get`命令，并用 `diff` 比较文件的较新版本与最新版本，然后给出版本号及被插入、删除、未作改变的行号。

警告

当使用 SCCS 时，必须牢记以下事项：

- 不能将二进制数据保存到 SCCS 文件中。Solaris SCCS 可使用 `uuencode`命令进行编码。
- SCCS 不会从经过检查并存入的 files 的文件权限中保存执行位。这对 shell 脚本特别重要：当从 SCCS 中将其提取出后，必须明确使它们成为可执行的。用 `make` 命令可使其自动执行。
- 在 *printf*(3S)格式化字符串中使用 ID 关键字（见下节）会带来灾难。可以使用一些间接方法来产生这些用于输出的字符串。

标识关键字

一个 SCCS 文件可能用到以下关键字。`get` 命令将这些关键字扩展为描述的值。

`%A%` 用于为程序文件提供 `what` 字符串的简写

```
%A% = %Z%%Y% %M% %I%%Z%
```

`%B%` 分支号

`%C%` 当前行号，用于标识出错的地方

`%D%` 当前日期（YY/MM/DD）

`%E%` 创建 delta 的最新日期（YY/MM/DD）

`%F%` SCCS 文件名

%G% 创建 delta 的最新日期（MM/DD/YY）

%H% 当前日期（MM/DD/YY）

%I% 被检索文本的 *sid*（%R%.%L%.%B%.%S%）

%L% 分级编号

%M% 模块名（没有 s. 前缀的文件名）

%P% 完全合格的 SCCS 文件名

%Q% *string* 的值，由 admin -fq *string* 定义

%R% 版本号

%S% 顺序号

%T% 当前时间（HH:MM:SS）

%U% 创建 delta 的最新时间（HH:MM:SS）

%W% 像 %A%;%W% = %Z%%M% *tab*%I% 一样的缩写

%Y% 模块类型，由 admin -ft*type* 定义

%Z% 由 what 识别的字符串，即 @(#)

数据关键字

数据关键字指定 SCCS 文件的哪一部分使用 prs 命令的 -d 选项进行提取及输出。

:A: what 字符串的格式

:B: 分支号

:BD: 主体

:BF: 分支标志

:C: delta 的注释

:CB: 向上取整边界

:D: delta 创建的日期（:Dy:/:Dm:/:Dd:）

:Dd: delta 创建的工作日

:Dg: delta 忽略的值（顺序号）

:DI: delta 的顺序号（:Dn:/:Dx:/:Dg:）

:DL: 统计的 delta 行（:Li:/:Ld:/:Lu:）

:Dm: delta 创建月份

:Dn: delta 中包含的（顺序号）

:DP: 上一版 delta 的顺序号

:Ds: 默认的 sid

:DS: delta 顺序号

:Dt: delta 信息

:DT: delta 类型

:Dx: delta 中未包含的（顺序号）

:Dy: delta 创建年份

:F: SCCS 文件名

:FB: 向下取整边界

:FD: 文件描述文本

:FL: 标志列表

:GB: 获得的主体

:I: SCCS ID 字符串（sid）（:R:.:L:.:B:.:S:）

:J: 联合编辑标志

:KF: 关键字错误 / 警告标志

:KV: 关键字验证字符串（不用于 Solaris）

:L: 级别号

:Ld: 由 delta 删除的行

:Li: 由 delta 插入的行

:LK: 锁定版本

:Lu: 未被 delta 改变的行

:M: 模块名

:MF: 修改请求验证标志

:MP: 修改请求验证程序名

:MR: delta 的修改请求号

:ND: delta 为空的标志

:P: 创建 delta 的程序员用户名

:PN: SCCS 文件路径名

:Q: 用户自定义的关键字

:R: 版本号

:S: 顺序号

:T: delta 创建时间（:Th:::Tm:::Ts:）

:Th: delta 创建的小时值

:Tm: delta 创建的分钟值

:Ts: delta 创建的秒值

:UN: 用户名

:W: what 字符串形式（:Z::M:\t:I:）

:Y: 模块类型标志

:Z: what 字符串定界符（@(#)）

按字母顺序概述 SCCS 命令

SCCS 命令的文件参数可以是文件名或目录名。指定一个目录，会处理在这个目录下的所有文件，忽略不可执行或不可读的文件（不可读文件会产生错误信息）。如果文件参数是一个中划线（-），命令会从标准输入设备逐行读入要处理的文件名。

命令用*yy*[*mm*[*dd*[*hh*[*mm*[*ss*]]]]]形式得到时间和日期。左边外面的值默认是最高有效值。Solaris 将 69 年到 99 年当作 20 世纪的年份对待、21 世纪的年份为 0 到 68 年。

在 Solaris 中，所有的 SCCS 命令保存在 /usr/ccs/bin 目录下。要使用这些命令，可在 PATH 环境变量中添加这个目录。

admin	admin [*options*] *files* 向 SCCS 增加文件 *files* 或改变 SCCS 文件 *files* 的选项 *options*。 options -a[*user*|*groupid*] 指定可创建 delta 的 *user* 或 *groupid* ；不允许将一个 ! 置于 *user*

admin

或 *groupid* 前。如果未给出 *user* 或 *groupid* 列表，则任何人都可创建 delta。

`-b` 将文件编码为二进制数据文件。文件中包含的 ASCII NUL 或其他控制字符，或那些没有换行符的文字，会自动作为二进制文件进行编码。该选项一般与 `-i` 选项一起使用。仅用于 Solaris。

`-d`*flag*

用 `-f` 删除原先的 *flag* 设置。可用的 *flags* 有：

`b`	在 `get` 命令中使 `-b` 选项有效；这可使 delta 产生分支。
`c`*n*	设置最高版本号为 *n*（默认值是 9999）。
`d`*n*	设置 `get` 命令默认 delta 数为 *n*。
`f`*n*	设置最低版本号为 *n*（默认值是 1）。
`i[`*string*`]`	将“No id keywords（ge6）”提示视为致命错误。如果 *string* 项存在且关键字与之不匹配，会产生致命错误。Solaris 不允许提供 *string*。
`j`	允许多个并发的 `get` 命令。
`l`*list*	*list* 中的版本号不能改变，用字母 `a` 可指定所有的版本号。
`m`*name*	用模块名称 *name* 代替 `%M%` 关键字。
`n`	从一个拟分支的 delta 中创建一个空的 delta。
`q`*string*	用字符串 *string* 代替 `%Q%` 关键字。
`t`*type*	T 用模块类型 *type* 代替 `%Y%` 关键字。
`v[`*prog*`]`	迫使 delta 命令提示修改请求号，以此创建 delta。运行程序 *prog* 来检查有效的数字。

`-e[`*user*`|`*groupid*`]`

禁止每个 *user* 或 *groupid* 创建 delta。

`-f`*flag*

设置标志 *flag*（参见上面的 `-d` 选项）。

`-h` 检查已存在的 SCCS 文件有可能造成破坏的地方。

admin

-i[*file*]
用 *file* 的内容作为初始 delta 来创建新的SCCS 文件。如果忽略 *file*，则采用标准输入。这一选项隐含了 -n选项。

-m[*list*]
插入修改请求号列表 *list* 作为创建文件的原因。

-n 创建一个新的 SCCS 文件，该文件内容为空。

-r*n.n*
设置初始的 delta 版本号为 *n.n*。默认值为 1.1。只能与 -i一起使用。

-t[*file*]
用 *file* 内容替换 SCCS 文件的描述。如果没有 *file* 选项，则删掉已有的描述。

-y[*text*]
为初始的 delta 插入注释文本 *text*（只与 -i 或 -n一起使用时有效）。

-z 重新计算 SCCS 文件的校验和并保存在第一行。首先会对该文件进行验证。参见 **val**。

cdc

cdc -*rsid* [*options*] *files*

改变一个或多个SCCS 文件 *files* 指定的 *sid*（SCCS ID）的 delta 注释。

options

-m[*list*]
增加修改请求号的列表（在号码前用 a !表示删除该号码）。-m只有在 admin 已经设置了文件的 v 标志时才有用。如果省略 -m，终端上会显示 MRs?作为输入提示。

-y[*string*]
为指定的 delta 增加注释字符串 *string*。如果省略 -y，终端上会显示 comments?来提示输入。

cdc

示例

对于文件 s.prog.c 的 delta 1.3，增加修改号 x01-5 和 x02-8，并增加注释：

```
$  cdc -r1.3 s.prog.c
MRs?  x01-5 x02-8
comments?  this went out to review
```

comb

comb [*options*] *files*

减小指定 SCCS 文件 *files* 的大小。通过裁剪选定的 delta，并将那些剩余部分合并来重建 SCCS 文件。除保留特别分支中的最新 delta 和那些保存树结构所需要的祖先外，命令的默认动作是全部裁剪。comb在标准输出上产生一个shell脚本。运行脚本会重建SCCS 文件。

options

-c*list*
: 只保留那些在由逗号分隔的列表*list*中指定了SCCS ID的delta。用连字符“ - ”可指定一个范围，例如：1.3、2.1-2.5。

-o
: 用创建的 delta 的版本号访问重建的文件 *file*，而不使用最近的祖先。该选项可改变目录树的结构。

-p*sid*
: 重建文件时，丢弃那些 SCCS 标识字符串比 *sid* 还旧的 delta。

-s
: 产生一个 shell 脚本，统计文件减小了多少。使用 -s 预览 comb 实际运行时所做的工作是很有用的。

delta

delta [*options*] *files*

将改变合并到（增加一个delta）一个或多个SCCS 文件*files*中。delta 将所做的改变保存到一个文本文件中，该文件可由 get -e命令进行检索然后编辑。delta 通常会删除该文本文件。

options

-d
: 使用 diff 命令而非 bdiff 命令寻找改变的地方。只用于 Solaris。

delta

-g*list*

忽略由逗号分隔开的列表指定 SCCS ID（版本号）的 delta，用“ - ”指定范围，例如：1.3、2.1-2.5。

-m[*list*]

提供一个修改请求号的列表，以此创建新的 delta。-m 只在 admin 命令已经设置了文件的 v 标志时有用。如果忽略 -m，终端上会显示 MRs?来提示进行输入。

-n 执行 delta 后，不删除已编辑的文件（由 get -e 命令得到）。

-p 输出 diff 类型的 delta 变化列表到文件 *file*。

-r*SID*

标识文件 *file* 的 delta 版本号。在同时编辑一个 SCCS 文件的多个版本时，必须使用 -r。

-s 禁止输出新的 SID 和其他 delta 信息。

-y[*string*]

插入字符串注释，描述创建 delta 的原因。如果省略 -y，终端上显示 comments?来提示用户进行输入。

get

get [*options*] *files*

检索一个 SCCS 文件 *file* 的文本版本号。被检索的文本文件（也称 g-file）与 SCCS 文件有相同的文件名，但使用 s. 前缀。对于每个 SCCS 文件，get 命令会输出版本号和检索的行号。参见前面的“ 标识关键字 ”一节中可存入文本文件中的关键字列表。

options

-a*n* 检索 delta 的顺序号 *n*，不是非常有用（由 comb 使用）。

-b 创建新的分支（与 -e 一起使用）。

-c*date*

只检索在日期 *date* 之前修改的版本。*date* 是一系列的两位数字，指出了年份，后面跟着可选的月份、日期、小时、分钟和秒。非数字字符可作为字段分隔符，基本上都被忽略了。

-e 检索一个用于编辑的文本文件，是一个最常用的选项。隐含了 -k 选项。

get

`-g` 禁止使用文本，只检索SCCS ID（版本号），只进行一般的检查。

`-G`*name*

在文件*name*中（默认情况是加`s.`前缀）存入被检索的文本。只用于Solaris。

`-i`*list*

将任何SCCS ID（版本号）在使用逗号分隔的列表*list*中指定的delta合并到被检索的文本文件中。用连字符“ - ”提供一个范围，例如：`1.3`、`2.1-2.5`。

`-k` 不扩展ID 关键字为它们的值，用于代替`-e`重新产生（覆盖）在编辑过程中被破坏的文本文件。

`-l[p]`

创建一个delta的列表（存入一个文件或用`-lp`显示在标准输出上）。

`-m` 在每行文本之前加上与它相关的delta 的SCCS ID。

`-n` 在每行文本之前加上`%M%`关键字（通常是该文本的文件名）。

`-p` 向标准输出写被检索的文本，而不写入文件中。

`-r[`*sid*`]`

检索SCCS ID（版本号）*sid*。没有*sid*时，只检索最新版本或由SCCS 文件中的d标志指定的版本。

`-s` 禁止一般的输出（只显示错误信息）。

`-t` 检索最新发布的版本。

`-w`*string*

用*string*替代`%W%`关键字；`%W%`是由`what`使用的标题标号。

`-x`*list*

从被检索的文本文件中排除delta的列表；与`-i`相反。

示例

检索文件`prog.c`并进行编辑，随后的一个`delta`产生了一个版本1.3分支：

```
get -e -b -r1.3 s.prog.c
```

get

检索文件 prog.c，其内容不包括 1990 年 6 月 1 日下午 2：30 以后做的修改（除 delta 2.6 和 2.7 外）：

```
get -c'90/06/01 14:30:00' -i'2.6,2.7' s.prog.c
```

显示 s.text.c 的内容（除 1.1~1.7 之外的所有版本）：

```
get -p -x1.1-1.7 s.text.c
```

help

help [*commands* | *error_codes*]

联机帮助工具，用来解释SCCS 命令或错误消息。没有参数时，help 命令会提示命令名称和错误代码。要显示简要的语法，可只提供 SCCS 命令的名称。要显示错误消息的注释，可在SCCS 错误消息后提供错误代码。help 文件通常驻留在目录 /usr/ccs/lib 下。

由失败的 SCCS 命令产生的错误消息格式如下：

```
错误文件名:消息（代码）
```

错误代码对发现错误的本质是非常有用的。为此，可键入：

```
help code
```

示例

当所有操作都失败时，可以试试如下命令：

```
help stuck
```

prs

prs [*options*] *files*

输出一个或多个 SCCS 文件 *files* 的格式化信息。

options

-a 包含所有 delta 的信息，包括删除的 delta。

-c*date*

关闭由 -e 或 -l 使用的日期 *date*（参见 **get** 命令的 *date* 格式）。

-d[*format*]

用文本和（或）SCCS 关键字形式指定输出格式 *format*。见前节的“数据关键字”得到有效的关键字列表。可在格式 *format* 中分别用 \t 和 \n 创建一个制表符和换行符。

prs

`-e` 与 `-r` 一起列出比 *sid* 更早或包括 *sid* 的 delta 数据；与 `-c` 一起列出不比 *date* 更新的 delta 数据。

`-l` 与 `-e` 一样，但是列出比 *sid* 或 *date* 更晚或者包括 *sid* 或 *date* 的 delta 数据。

`-r[`*sid*`]`
指定 SCCS ID *sid*；默认值是最新的 delta。

示例

以下命令:

```
prs -d"program :M: version :I: by :P:" -r s.yes.c
```

会产生如下输出：

```
program yes.c version 2.4.6 by daniel
```

prt

`prt [`*options*`] `*files*

只用于Solaris。格式化并输出一个或多个SCCS文件的内容。默认情况下，`prt`会输出delta表（即版本日志）。*sccsfile*(4) 参考页详细描述了SCCS文件的内容。

options

`-a` 显示所有 delta 的条目，包括被删除的。

`-b` 输出 SCCS 文件的主体。

`-c`*date*
排除早于日期 *date* 的条目。每个条目输出一行，之前加上文件名。这使对多版本的日志分类变得更容易。

`-d` 输出 delta 表的条目。这是默认操作。

`-e` 输出所有内容。该选项隐含了 `-d`、`-I`、`-f`、`-t` 和 `-u` 选项。

`-f` 输出每个 SCCS 文件的标志。

`-i` 输出包含、不包含、忽略 delta 的 SID。

`-r`*date*
排除比日期 *date* 更新的 delta。

`-s` 只输出每个 delta 表的第一行（统计值）。

prt	`-t` 输出 SCCS 文件的描述文本。 `-u` 输出用户名和（或）用户的数字组 ID 号，允许这些用户进行修改操作。 `-y[`*sid*`]` 排除比 *sid* 更旧的 delta。如果表中没有与 *sid* 匹配的 delta，则输出表的全部内容。如果没有 *sid*，则输出当前 delta 的信息。
rmdel	`rmdel -`*r sid files* 从一个或多个 SCCS 文件 *files* 中删除 delta，文件的 *sid* 是 SCCS ID。在所在的分支中，delta 必须是最新的，但不能对其检验后再进行编辑。
sact	`sact` *files* 对于指定的 SCCS 文件 *files* 报告将要改动的 delta（即，哪些文件正被 `get -e` 命令编辑，但还未由 `delta` 更新）。`sact` 列表输出五个字段：当前被编辑 delta 的 SCCS ID 号、要创建的新 delta 的 SCCS ID 号、发出 `get -e` 命令的用户、发出命令的日期和时间。
sccsdiff	`sccsdiff -`*rsid1* `-`*rsid2* `[`*options*`]` *files* 报告一个 SCCS 文件 *file* 的两个版本的差异。*sid1* 和 *sid2* 指定要比较的 delta。该命令会调用 `bdiff`，并依次调用 `diff`。Solaris 中的 `sccsdiff` 调用 `diff` 而不是 `bdiff`。 options `-p` 通过 `pr` 进行管道输出。 `-s`*n* 使用文件段的大小 *n*（*n* 传递给 `bdiff`）。
unget	`unget [`*options*`]` *files* 取消前一次对一个或多个 SCCS 文件 *files* 执行的 `get -e` 命令。如文件正由 `get -e` 进行编辑，发出的 delta 会处理编辑内容（创建新的 delta），而 `unget` 会删除已编辑的版本（防止产生新的 delta）。

unget

options

`-n` 不删除由 `get -e` 检索到的文件。

`-r`*sid*

要删除的delta的SCCS ID。只有在对同一个SCCS文件多次进行 `get -e` 操作时有用。

`-s` 禁止显示某一 delta 的 *sid*。

val

`val [`*options*`] `*files*

验证 SCCS 文件 *files* 与选项中指定的字符相符。`val` 在标准输出上为每个文件产生消息并在退出时返回一个 8 位的码。在“返回值位”中对这些码进行了介绍，位按从左向右的顺序计算。

options

`-` 从标准输入中读入，并逐行解释作为 `val` 命令行参数。用 *EOF* 退出。该选项由它自己使用。

`-m`*name*

比较文件 *file* 中的名字 *name* 和关键字 `%M%`。

`-r`*sid*

检查 SCCS ID 是否不明确或无效。

`-s` 屏蔽任何错误消息。

`-y`*type*

比较文件中的类型 *type* 和关键字 `%Y%`。

返回值位	意义
0	缺少文件参数
1	未知或副本选项
2	被破坏的 SCCS 文件
3	不能打开文件，或文件不是 SCCS 文件
4	SID 非法或不明确
5	不存在的 SID
6	类型与 `-y` 参数不匹配
7	文件名与 `-m` 参数不匹配

what	what [*option*] *files* 搜索模式为@(#)的文件 *files*，输出模式之后的文本（*files* 通常是二进制可执行文件）。实际上，搜索的模式是%Z%的值，但是 get 命令将这个关键字扩展为@(#)。What 命令的主要目的是输出标识字符串。 option -s 查找到第一个模式后退出。

sccs 和伪命令

该兼容包包括了sccs（SCCS应用程序的前端）。这些命令提供了使用SCCS时更友好的接口，它的命令行语法如下：

命令行语法：

```
sccs [options] command [SCCS_flags] [files]
```

除了提供所有常规的SCCS命令，sccs还提供了伪命令。这些伪命令是易于使用的、常规 SCCS 命令的预建立组合。选项 *options* 只用于 sccs 接口；命令 *command* 是要运行的SCCS命令或伪命令；*SCCS_flags* 是传递至运行的SCCS 命令中的特定选项。

使用sccs指定一个文件非常容易，因为它自动将SCCS/s.添加到任意一个文件参数前面。例如：

```
sccs get -e file.c
```

会翻译为：

```
get -e SCCS/s.file.c
```

这样，当使用sccs时，必须先创建一个名字为SCCS的子目录，以便保存所有的s.SCCS 文件。

options

-d*prepath*

在 *prepath* 中定位文件，而不用当前目录。例如：

```
sccs -d/home get file.c
```

会翻译为：

```
get /home/SCCS/s.file.c
```

`-p`*`endpath`*

从目录 *endpath* 而非 SCCS 中访问文件。例如：

```
sccs -pVERSIONS get file.c
```

会翻译为：

```
get VERSIONS/s.file.c
```

`-r` 作为真实的用户而非有效的用户调用 `sccs`。

伪命令

与 SCCS 操作等同，在下面介绍。

`check`

与 `info` 一样，但是返回非零值的退出代码，而非文件名。

`clean`

从当前的目录中删除未被 SCCS 编辑的任意文件（如通过 `get -e` 命令）。

`create`

创建 SCCS 文件。（使用了 `get` 后，再用 `admin -I` 命令。）

`deledit`

与 delta 后面再使用 `get -e` 作用相同。

`delget`

与 delta 后面再使用 `get` 作用相同。

`diffs`

比较文件的当前版本与 SCCS 版本（与 `sccsdiff` 相同）。

`edit`

编辑文件（即 `get -e`）。

`enter`

与 `create` 相似，但不随后执行 `get`（`admin -i`）命令。

`fix` 与 `rmdel` 相同（但是必须跟 `-r`）。

`info`

列出正在编辑的文件（与 `sact` 相似）。

`print`
: 输出信息（与 `prs -e` 后面再使用 `get -p -m` 相似）。

`tell`
: 与 `info` 相似，但一行只列一个文件名。

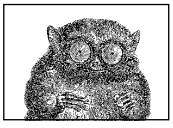
`unedit`
: 与 `unget` 相同。

Solaris 中的注意事项

- 只有安装了开发系统，SCCS 才可用。
- 环境变量 PROJECTDIR 指定了 sccs 搜索 SCCS 文件的地方。有两种值可用：

绝对路径名
: `sccs` 在 $PROJECTDIR 指定的目录名下搜索 SCCS 文件。

用户名
: `sccs` 在给定用户主目录下的 `src` 或 `source` 子目录中搜索 SCCS 文件。



第十九章

RCS

本章将介绍以下主题：

- 命令概述
- 基本操作
- 一般 RCS 规范
- SCCS 用户转换指南
- 按字母顺序概述命令

RCS（Revision Control System，版本修订控制系统）和前章介绍的SCCS一起，都是为跟踪文件的多次修订而设计的，用于减少所需的存储空间。通过RCS命令，可以自动地保存和检索修订、合并或比较修订，保存完整的修改历史（日志），并用符号关键字标识所做的修订。通常RCS比SCCS更有效。与SCCS不同，RCS保留了它所管理的文件的可执行权限，并可在RCS文件中存储二进制数据。

标准的SVR4或Solaris中没有RCS。可从自由软件基金会（见*http://www.gnu.org*）获取它。本章将介绍RCS5.7版本。要获得更多的信息，可参见参考文献中的《Applying RCS and SCCS》。

命令概述

三个最重要的RCS命令是：

`ci` 登记修订文件（将一个文件置于 RCS 控制下）。

`co` 取出修订文件。

`rcs` 设置或改变 RCS 文件属性。

提供 RCS 文件信息的两个命令：

`ident` 从 RCS 文件中提取关键字的值。

`rlog` 在 RCS 文件中显示修订汇总（日志）。

用以下命令比较 RCS 文件：

`merge` 将两个文件中的修改保存到第三个文件中。

`rcsdiff` 报告修订版本间的差异。

`rcsmerge` 将两个 RCS 文件中的修改保存到第三个 RCS 文件中。

下列命令有助于配置管理，但是作为选件并不总是安装。

`rcsclean` 删除未做修改的工作文件。

`rcsfreeze` 为进行配置的文件加标号。

基本操作

通常，在 RCS 子目录下维护 RCS 文件，所以使用 RCS 的第一步是：

```
mkdir RCS
```

第二步，运行以下登记命令，将已存在的文件置于 RCS 控制之下：

```
ci file
```

该命令在 RCS 目录下创建一个名为 *file*，`v` 的文件，*file*，`v` 称为一个 RCS 文件，它保存以后对文件 *file* 修订的所有版本。如果第一次对文件进行 `ci` 操作，系统提示输入描述内容，`ci` 会将 *file* 存入 RCS 文件，RCS 的文件修订版本号为 1.1。

若要编辑新的修订版本，需取出文件的副本，可用命令：

```
co -l file
```

从 RCS 文件中提取 *file* 的一个副本。这个副本必须用 `-l` 锁定，以便可以修改文件。这个副本文件就叫工作文件。编辑完成后，可将工作文件重新拷贝回 RCS，以便将产生的变化记录下来：

```
ci file
```

这次，系统提示输入修改的日志信息，文件作为修订版1.2保存起来。注意，登记命令通常会将工作文件删除。要取回文件的只读副本，用取出修订文件命令，不要加锁：

```
co file
```

当需要保留一个副本文件进行编译或搜索时，这个命令很有用。作为前面介绍的`ci/co`命令的快捷方法，可键入：

```
ci -u  file
```

这个命令登记文件*file*，但是立即取出*file*的只读副本文件（不锁定）。实际上，可能需要对工作文件标记一个“检查点”，然后继续进行修改，像这样：

```
ci -l  file
```

这个命令将登记文件*file*，并取出*file*的副本进行锁定，然后继续编辑。要比较工作文件与它的最新修订版之间所做的修改，可键入：

```
rcsdiff file
```

另一个有用的命令是`rlog`，它显示日志的汇总信息。系统管理员可以用`rcs`命令建立RCS的默认操作。

一般RCS规范

本节讨论以下内容：

- 关键字替换
- 关键字
- 示例值
- 修订版本号
- 指定日期
- 指定状态
- 标准选项和环境变量

关键字替换

RCS 允许在工作文件中使用关键字变量。这些变量随后会被扩展到修订注释中。这些注释可作为输入文件自带的说明或打印输出文件时出现的文本字符串使用。要通过关键字替换创建修订注释，可按以下步骤操作：

1. 在工作文件中输入下面关键字列表中的关键字。
2. 登记文件。
3. 取出文件。在取出时，`co` 命令扩展每个关键字以包含关键字的值。即，`co` 将：
   $*keyword*$
   替换为：
   $*keyword*: *value* $.
4. 随后的登记和取出命令更新所有存在的关键字的值。除非有其他注解，否则总是使用新值替换已存在的值。

许多命令都有 `-k` 选项，以便在替换关键字时有更好的灵活性。

关键字

`$Author$` 登记修订文件的用户名。

`$Date$` 登记日期和时间。

`$Header$` 包含 RCS 文件完整的路径名、修订版本号、日期、作者、状态和锁定文件者姓名（如果文件被锁定）的标题。

`$Id$` 与 `$Header$` 相同，但是不包括 RCS 文件完整的路径名。

`$Locker$` 锁定修订文件者的用户名。如果文件未被锁定，该值为空。

`$Log$` 该信息是在登记时键入的对文件的描述，前面是 RCS 文件名、修订版本号、作者、日期等。日志信息添加时不覆盖以前的信息。RCS 为文件留下的日志信息使用 `$Log$` 行的“注释前导字符”。只在与老版本的 RCS 交换文件时，保存在 RCS 文件中的注释前导字符才有用。

`$Name$` 如果有该选项，则是登记 RCS 文件修订版本时使用的文件符号名。

`$RCSfile$` RCS 文件名，不带路径名。

`$Revision$` 指定的版本号。

`$Source$` RCS 文件名，包括路径名。

`$State$` 由 `ci` 或 `rcs` 的 `-s` 选项指定的状态。

示例值

假设名字为 `daniel` 的一个用户登记、取出了文件 `/projects/new/chapter3`，在文件的第二个修订本中，对每个关键字进行了关键字替换：

```
$Author: daniel $

$Date: 1992/03/18 17:51:36 $

$Header: /projects/new/chapter3,v 1.2 92/03/18 17:51:36 daniel \
    Exp Locker: daniel $

$Id: chapter3,v 1.2 1992/03/18 17:51:35 daniel Exp Locker: daniel $

$Locker: daniel $

$Log: chapter3,v $
# Revision 1.2 92/03/18 17:51:36 daniel
# Added section on error-handling
#
# Revision 1.1 92/03/18 16:49:59 daniel
# Initial revision
#

$Name: Alpha2 $

$RCSfile: chapter3,v $

$Revision: 1.2 $

$Source: /projects/new/chapter3,v $

$State: Exp $
```

修订版本号

除非另外声明，RCS命令一般只对最新的修订版本进行操作。有些命令带有 `-r` 选项，可指定修订版本号。另外，许多选项接受修订版本号作为可选参数。（在命令汇总表里，该参数表示为[*R*]）。修订版本号由四部分组成：发布号、分级号、分支号及顺序号；但绝大多数的修订版本号只由发布号和分级号组成。例如：将修订版本号为1.4的文件取出：

```
co -l -r1.4 ch01
```

当再次登记时，新的修订版本号被标为1.5。假设需要以下一个发布号将已编辑过的副本文件登记，可输入:

```
ci -r2 ch01
```

这个命令创建修订版本号为 2.1 的文件。也可从较早的修订文件中创建一个分支文件，以下命令创建修订版本号为 1.4.1.1 的文件：

```
ci -r1.4.1 ch01
```

开头为句点的数字被认为是与 RCS 文件的默认分支相对应的。通常情况下是树状修订版本号的树干。

数字不是指定修订版本号的惟一办法。用 `ci` 命令或 `rcs` 命令加 `-n` 选项，可以指定文本标号作为修订名。凡是接受数字修订版本号为参数的选项都可指定该名。例如：不管当前文件的修订版本号是什么，都可用相同的标号登记每个 C 文件：

```
ci -u -nPrototype *.c
```

另外，可用 `$` 从工作文件的关键字中提取修订版本号，例如用：

```
rcsdiff -r$ ch01
```

比较 `ch01` 和登记的修订文件。也可以将名字与符号合并，用命令：

```
rcs -nDraft:$ ch*
```

可为与几个章文件关联的修订版本号分派一个名字。

指定日期

文件登记时，为修订本添加时间和日期等时间信息。有几个关键字串符值包含了日期值。日期由 `ci`、`co` 或 `rlog` 加选项给出。RCS 使用以下默认格式的日期：

```
2000/01/10 02:00:00 Year/month/day time
```

默认时区是格林威治标准时间（GMT），也称为协调世界时间（UTC）。给出的日期是自由格式的，可以指定成不同的风格，以下一些较为通用，仍以上面的示例为例：

```
6:00 pm lt                          假设今天是 Jan. 10, 2000
2:00 AM, Jan. 10, 2000
Mon Jan 10 18:00:00 2000 LT
Mon Jan 10 18:00:00 PST 2000
```

大写或小写的“ lt ”表示当地时间(这里为太平洋标准时间)。第三行表示 ctime 格式(加上“ LT ”)；第四行为 `date` 命令格式。

指定状态

有些情况下，特别是在编程环境下，需要了解一系列修订文件的状态。所有RCS文件由一个文本字符串标记它们的状态。默认状态为Exp（实验的）。其他的常用状态为Stab（稳定的）或Rel（发布的）。这些单词由用户定义，没有特别的含义。有些关键字符串包含状态值。另外，状态值可由ci、co、rcs和rlog加选项给出。

标准选项和环境变量

RCS定义了一个环境变量RCSINIT，为RCS命令设置默认的选项。如果将RCSINIT设为由空格分隔的多个选项的列表，在给出RCS命令时，这些选项就会添加到命令行选项的前面。

RCSINIT中有6个有用的选项：-q、-V、-Vn、-T、-x和-z。由于绝大多数RCS都接受这些选项，它们可视为标准选项。

-q[*R*]
: 安静模式。不输出诊断信息。*R*指定一个文件的修订版本。

-T 如果新修订文件的更新时间比RCS文件要晚，则会更新RCS文件的修改时间。否则，保留RCS文件原来的修改时间。使用该选项时要慎重。更详细的内容可参见ci参考页。

-V 输出RCS修订版本号。

-V*n*模拟RCS文件版本*n*。当在运行不同版本的系统间传递文件时非常有用。*n*可为3、4或5。

-x*suffixes*
: 为RCS文件指定一个可替换的后缀*Suffixes*的列表。每个后缀由/分隔。在Unix系统中，RCS文件名通常以字符，v结束。-x选项提供了系统的工作区，该工作区不允许在文件名中使用逗号。

-z*timezone*
: *timezone*在关键字替代中控制日期的输出格式。*timezone*应从下表列出的值中取其一：

值	作用
empty	默认格式：UTC，没有时区，日期的每一部分由“ / ”分隔。
LT	本地时间和日期，使用 ISO-8601 格式，用时区指示（*YYYY-MM-DD HH*:*MM*:*SS-ZZ*）。
± *hh*:*mm*	相对于 UTC 的数字偏移量，输出使用 ISO-8601 格式。

例如，当把工作文件存入 RCS 文件时，命令为：

```
ci -x,v/ ch01     第二个后缀为空
```

按顺序查找 RCS 文件名：

```
RCS/ch01,v
ch01,v
RCS/ch01
```

RCS 允许为临时文件指定位置。它按顺序检查环境变量 TMPDIR、TMP 和 TEMP。如果都不存在，使用默认的位置，如 `/tmp`。

SCCS 用户转换指南

SCCS 命令功能上等价于 RCS 命令。下表给出了对 SCCS 用户通用的指南。

SCCS	RCS
admin	rcs
admin -i	ci
cdc	rcs -m
delta	ci
get	co
prs	ident or rlog
rmdel	rcs -o
sact	rlog
sccsdiff	rcsdiff
unget	co (覆盖)，或与 rcs-o 一起使用的 ci
what	ident

按字母顺序概述命令

关于关键字语法、修订版本号、日期、状态和标准选项的细节，参考前面的讨论。

ci

`ci [`*`options`*`] `*`files`*

登记修订版本。`ci` 将指定工作文件 *files* 的内容保存到相应的 RCS 文件中。通常，`ci` 保存工作文件后就将其删除了。如果 RCS 文件不存在，工作文件就是原始文件。这种情况下，会创建 RCS 文件，系统会提示输入关于文件的描述。如果 RCS 文件已存在，`ci` 会递增修订版本号并提示输入关于修改的日志信息。如果工作文件登记后未做修改，文件会恢复到原来的版本。

两个相互对立的选项 `-u` 和 `-l`，以及 `-r` 是最常用的。用 `-u` 保存工作文件的只读副本（例如，文件可被编译和搜索）。用 `-l` 更新修订本后立即取出，并加以锁定，这可保存中间修改的过程并可继续进行编辑（例如，在很长的编辑会话中）。用 `-r` 登记不同修订版本号的文件。`ci` 接受标准选项 `-q`、`-V`、`-Vn`、`-T`、`-x` 和 `-z`。

options

`-d[`*`date`*`]`
: 登记文件并带上日期时间信息；如果没指定日期，则用最后修改的时间。

`-f[`*`R`*`]`
: 即使没有差别，也强制进行一次登记。

`-i[`*`R`*`]`
: 初始化登记。如果 RCS 文件已存在，则报告错误信息。

`-I[`*`R`*`]`
: 交互模式，即使标准输入不是终端，也向用户提示信息（例如，当 `ci` 是命令管线的一部分时）。

`-j[`*`R`*`]`
: 只登记而不进行初始化。如果 RCS 文件不存在，则报告错误信息。

ci

-k[*R*]
分配修订版本号、创建日期、状态和作者名，这些内容从工作文件关键字的取值中获得，而不是取自本地环境。-k 对软件的发布是非常有用的：预先设置的关键字可作为时间信息，由所有发布站点共享。

-l[*R*]
登记后进行一次 co -l 操作，将修订的文件副本锁住。

-m*msg*
使用 *msg* 字符串作为所有登记文件的日志信息。当登记多个文件时，ci 通常会提示是否重新使用文件原先的日志信息。-m 则不进行提示。

-M[*R*]
将工作文件的修改时间设置为被检索的版本的修改时间。-M 的使用会与 make 弄混，需加以小心。

-n*name*
关联一个文本文件名 *name* 和一个新的修订版本号。

-N*name*
与 -n 相同，但取代前一个文件名 *name*。

-r*R* 用修订版本号 *R* 登记文件。

-r 没有修订版本号，-r 恢复释放锁和删除工作文件的默认行为。目的在于取代其他别名文件或脚本默认设置的 -l 或 -u。ci 中的 -r 与大多数其他的 RCS 命令不同。

-s*state*
设置登记修订版本的状态 *state*。

-t*file*
用文件 *file* 的内容取代 RCS 文件的描述。只在初始登记时进行。

-t-*string*
用字符串 *string* 取代 RCS 文件的描述。只在初始登记时进行。

-u[*R*]
登记后进行一次 co -u 操作，产生一个只读的文件副本。

ci

`-w`*user*

在登记修订本时为用户 *user* 设置作者字段。

示例

用相同的日志信息登记章文件：

```
ci -m'First round edits' chap*
```

登记编辑 `prog.c` 文件，产生一个只读的副本文件：

```
ci -u prog.c
```

开始修订级号 2，将版本 2.1 作为原型文件：

```
ci -r2 -nPrototype prog.c
```

co

`co [`*options*`] `*files*

取回（输出）一个先前登记的修订文件到相应的工作文件中（如果指定了 `-p`，则在标准输出设备上输出）。如果打算编辑工作文件并再登记回去，则指定 `-l` 来锁定文件。`co` 接受标准选项 `-q`、`-V`、`-Vn`、`-T`、`-x` 和 `-z`。

options

`-d`*date*

取回最新修订的文件，该文件登记时间信息是 *date* 表示的日期或该日期之前。

`-f[`*R*`]`

强迫覆盖工作文件。

`-I[`*R*`]`

交互模式。即便标准输入不是终端，也向用户进行提示。

`-`*jR2*`:`*R3*`[,...]`

与 `rcsmerge` 类似。*R2* 和 *R3* 指定两个修订的文件，它们所修改的内容会合并到第三个文件中：该文件可以是相应的工作文件，也可以是第三个修订文件（由其他 `co` 选项指定的任意 *R*）。用多个逗号对作为分隔符，第一个组的输出作为下一个组的输入。参见 `co` 参考页。

co

-k*c* 根据标志 *c* 扩展关键字符号。*c* 可为：

- b 与 -ko 类似，但用二进制的输入/输出。对于非 Unix 系统很有用。
- kv 关键字和其值的扩展符号（默认）。只在用到 ci -l 或 co -l 时插入锁定者的名字。
- kvl 与 kv 类似，总是插入锁定者的名字。
- k 关键字扩展符号（值没有扩展号符）。在进行文件比较时，可以忽略细微的差异。
- o 以前修订文件的关键字和其值的扩展符号。对于不允许改变子串的二进制文件很有用。
- v 值的扩展符号（关键字没有扩展符号）。不允许进行关键字替代，不推荐使用该选项。

-l[*R*]
与 -r 相同，但是锁定取回的修订文件。

-M[*R*]
将工作文件的修改时间设置为取回的修订文件的修改时间。-M 容易与 make 混淆，需加以小心。

-p[*R*]
将取回的文件输出到标准设备上，而不产生工作文件。输出重定向或过滤时有用。

-r[*R*]
取回最新的修订文件；如果给出了 *R*，则取回等于或低于 *R* 的最新修订文件。如果 *R* 为 $，则取回由工作文件中的关键字指定的修订文件。

-s*state*
取回具有给定状态 *state* 的最新修订文件。

-u[*R*]
与 -r 相同。如果前面锁定了取回的文件，用此命令解除锁定。

-w[*user*]
取回最新修订的文件，这些文件由调用的用户或指定的用户 *user* 登记过。

co

示例

对文件 *file* 的最新存储版本进行排序：

```
co -p file | sort
```

输出（并锁定）所有大写字母的文件名用于编辑：

```
co -l [A-Z]*
```

注意，如果工作文件的副本没有保存在当前的目录下，文件名扩展就会失败。因此，只有这些文件在以前使用 `ci -u` 登记过，此示例才正确。最后，可用不同的方法提取一系列 RCS 文件的工作文件（在当前的目录下）。

```
co -r3 *,v                      发布号为 3 的最新修订本
co -r3 -wjim *,v                同上，只有通过 jim 登记过才正确
co -d'May 5, 2 pm LT' *,v       在此日期当天或在此日期之前修改过的
                                最新修订版本
co -rPrototype *,v              名字为 Prototype 的最新修订版本
```

ident

`ident [`*options*`] [`*files*`]`

从文件 *files* 中提取关键字 / 值符号。*files* 可以是文本文件、目标文件或哑文件。`ident` 接受标准选项 `-V`。

options

`-q` 当未找到关键字模式时，不显示警告信息。

`-V` 输出 `ident` 的版本号。

示例

如果已编译过文件 `prog.c`，并包含下面的一行代码：

```
char rcsID[] = "$Author: arnold $";
```

则会产生以下的输出内容：

```
$  ident prog.c prog.o
prog.c:
     $Author: arnold $
prog.o:
     $Author: arnold $
```

显示所有 RCS 文件的关键字（不显示警告）：

ident

```
co -p RCS/*,v | ident -q
```

merge

merge [*options*] [*diff3 options*] *file1 file2 file3*

执行三种合并文件的方法（通过 diff3），并把改变保存在 *file1* 中。*file2* 为原始文件。*file1* 是 *file2* 修改好的文件。*file3* 是与 *file2* 进行了不一致修改的文件。merge 从 *file2* 和 *file3* 中寻找差异，并将其合并后保存到文件 *file1* 中。如果 *file1* 和 *file3* 都对共同的行做了修改，merge 会对共同修改的地方进行告警，并在 *file1* 中插入两种选择。插入的内容如下：

```
<<<<<<< file1
lines from file1</>
========
lines from file3
>>>>>>> file3
```

需要对 *file1* 进行编辑时，可删除一种选择。merge 在退出时状态为 0（没有相同处）、1（有一些相同）或 2（未知的问题）。参见 **rcsmerge**。

merge 的 diff3 选项接受 -A、-e 和 -E，并简单地让其通过，使 diff3 执行某些相应的合并动作。参见第二章的 **diff3** 一项（-A 选项用于 diff3 的 GUN 版本）。

options

-L *label*
: 该选项最多可用三次，分别在 *file1*、*file2* 和 *file3* 的文件名中提供不同的标号。

-p 将合并文件发送到标准输出上，不发送到 *file1* 中。

-q 将共同修改的内容插入文件，但不警告。

rcs

rcs [*options*] *files*

设置或修改 RCS 文件默认属性的管理命令。rcs 要求至少提供一个选项（主要用于“将来扩展”）。

其他方面，rcs 可进行严格的锁定（-L），删除修订文件（-o），取代由 co（-l 和 -u）设置的锁定。RCS 文件有一个访问列表（通过

rcs

-a创建），在列表中列出的用户名可以运行rcs。访问列表经常是空的，意味着任何人都可使用rcs。另外，如果你拥有文件，就可以调用rcs。特权用户可以添加-i运行rcs，rcs接受标准选项-q、-V、-V*n*、-T、-x和-z。

options

-a*users*
向访问列表中追加由逗号分隔的用户列表。

-A*otherfile*
向文件*files*的访问列表中追加其他文件*otherfile*的访问列表。

-b[*R*]
设置默认的分支为*R*；如果忽略*R*，则设置为树干中的最高分支。

-c′*s*′
将Log关键字的注释前导字符设置为字符串*s*。例如，在troff文件中可将*s*设为.\"，或在C程序中将*s*设为*。（需要手动地在Log前后插入由/*和*/包围的内容。）

-c选项已经淘汰。RCS在文件中将字符置于Log前面，作为日志信息的注释前导字符。如果用旧版本的RCS存取RCS文件，可以这样设置。

-e[*users*]
从访问列表中删除所有的用户（或只删除指定的用户）。

-i 创建（初始化）RCS文件，但不保存修订文件。

-I 交互模式。即使标准输入不是终端，也向用户进行提示。

-k*c* 用*c*作为关键字替换的默认格式（*c*的值见**co**）。-kkv恢复默认的替换格式。

-l[*R*]
锁定修订版本*R*或最新的修订版本。-l会“追溯锁定”一个文件。如果错误地使用co而不是co -l输出文件，rcs会询问如果其他人已经锁定了文件，是否需要解除锁定，在这种情况下该命令很有用。

rcs

`-L` 打开严格的锁定（默认）。这意味着任何人，包括RCS文件的属主都必须使用 `co -l`编辑文件。文件共享时，推荐使用严格锁定（参见 `-U`）。

`-m`*R*`:`*msg*
使用 *msg* 字符串代替修订版本 *R* 的文件日志信息。

`-M` 当打破锁定时不发送邮件。只由 RCS 前端使用，并不直接为用户使用！

`-n`*flags*
增加或删除修订文件和文件名的关联。标志 *flags* 可以是：

name:R 将 *name* 与修订文件 *R* 关联。

name: 将 *name* 与最新修订文件关联。

name 删除 *name* 的关联。

`-N`*flags*
与 `-n` 相同，但覆盖已存在的 *names*。

`-o`*R_list*
删除（过期）由 *R_list* 列出的修订文件。 *R_list* 可指定为：*R1*、*R1*:*R2*、 *R1*:或 :*R2*。当给出分支号时，`-o` 只删除它的最新修订文件。范围分隔符 - 在 5.6 版以前的 RCS 中仍然合法。

`-s`*state*`[:`*R*`]`
设置修订文件 *R* 的状态（或最新修订文件）为 *state*。

`-t[`*file*`]`
用文件*file*的内容替换RCS文件的描述；如果没有给出文件，则用标准输入。

`-t-`*string*
用字符串 *string* 替换 RCS 文件的描述。

`-u[`*R*`]`
`-l`的补充：解除前面用`co -l`输出的锁定文件。如果别人要输出文件，会向你提示解除锁定的理由。这条消息会由邮件发送给最初的锁定者。

`-U` 打开非严格的锁定。除文件属主本人外，所有人都必须使用`co -l` 编辑文件（参见 `-L`）。

rcs

示例

将标号 *To_customer* 与所有 RCS 文件的最新修订文件关联：

```
rcs -nTo_customer: RCS/*
```

在文件 beatle_deals 的访问列表中追加三个用户：

```
rcs -ageorge,paul,ringo beatle_deals
```

删除修订文件 1.2 至 1.5:

```
rcs -o1.2:1.5 doc
```

用变量的内容替换 RCS 文件的描述：

```
echo "$description" | rcs -t file
```

rcsclean

rcsclean [*options*] [*files*]

尽管 RCS 中包括这个命令，但是该命令是可选的，可能没有安装到你的系统中。rcsclean 按照最新修订文件或修订版本号为 R 的文件（由选项给出）来比较输出文件。如果没有发现区别，就删除工作文件（使用 rcsdiff 发现差异）。rcsclean 在 make 程序的描述文件中很有用。例如，可以指定一个“ 干净的 ”目标来更新目录。rcsclean 在运行 rcsfreeze 之前也很有用。rcsclean 接受标准选项 -q、-V、-Vn、-T、-x 和 -z。

options

-k*c* 当比较修订文件时，用格式 *c* 扩展关键字（*c* 的值参见 **co**）。

-n[*R*]
显示要出现的情形，但不实际执行。

-r[*R*]
与修订版本号为 *R* 的文件进行比较。*R* 由其他选项的参数指定，所以 -r 是冗余的。

-u[*R*]
如果与工作文件相同，则解除对修订文件的锁定。

示例

删除程序文件和头文件未做修改的副本：

rcsclean

```
rcsclean *.c *.h
```

rcsdiff

`rcsdiff [options] [diff_options] files`

通过 `diff` 比较修订文件。用以下的 `-r` 指定修订文件：

修订文件数	所做的比较
无	工作文件与最新修订文件比较
1 个	工作文件与指定的修订文件比较
2 个	一个修订文件与另一个修订文件比较

与接受 *diff_options* 选项一样，`rcsdiff` 也接受标准选项 `-q`、`-V`、`-Vn`、`-T`、`-x` 和 `-z`，*diff_options* 选项可以是任何有效的 `diff` 选项。`rcsdiff` 在退出时状态值为 0（没有差异）、1（有一些差异）或 2（未知的问题）。用于 `diff` 的 `-c` 选项在与 `rcsdiff` 一起使用时是非常有用的。

`rcsdiff` 会在标准错误中显示“检索修订版本...”信息，以及用于分离多个文件的一行等号。在将标准错误和标准输出重定向到同一个文件时，`rcsdiff` 很有用。

options

`-k`*c* 当比较修订文件时，用 *c* 格式扩展关键字（*c* 的值参见 **co**）。

`-r`*R1*

用修订版本号为 *R1* 的文件进行比较。

`-r`*R2*

用修订版本号为 *R2* 的文件进行比较（必须指定 `-r`*R1*）。

示例

比较当前的工作文件与最后一次的登记文件：

```
rcsdiff -c ch19.sgm 2>&1 | more
```

比较当前的工作文件与第一个修订文件：

```
rcsdiff -c -r1.1 ch19.sgm 2>&1 | more
```

比较较早的文件的两个不同版本：

```
rcsdiff -c -r1.3 -r1.4 ch19.sgm 2>&1 | more
```

rcsfreeze

rcsfreeze [*name*]

尽管在RCS中包括了该命令，但是这个命令是可选的，可能没有安装在你的系统中。rcsfreeze为整个系列的RCS文件指定文件名，这些文件必须已经登记过。标记一组文件为一个配置文件是有用的。默认的*name*是C_*n*，*n*在每次运行rcsfreeze时递增。

rcsmerge

rcsmerge [*options*] [*diff3 options*] *file*

用三种方法合并修订文件，从两个不同版本中提取修改的内容合并到工作文件中。必须给出要合并的一个或两个文件（一般用-r）。对两个文件中同一位置进行修改时，其处理方法与merge一样，即在合并的文件中进行警告。rcsmerge接受标准选项-q、-V、-Vn、-T、-x和-z。rcsmerge在退出时状态值为0（没有差异）、1（有一些差异）或2（未知的问题）。

rcsmerge的diff3接受-A、-e和-E选项。rcsmerge会简单地使其通过，由diff3执行相应的合并动作。参见**merge**，也可参见第二章中的**diff3**命令（-A选项用于diff3的GUN版本）。

options

-k*c* 当比较修订文件时，用格式*c*扩展关键字（*c*的值参见**co**）。

-p[*R*]
: 将合并文件发送到标准输出上，而不覆盖文件*file*。

-r[*R*]
: 合并修订文件*R*；如果未给出*R*，则合并最新修订文件。

示例

假设要更新prog.c的旧版本（1.3），但是当前文件的版本已是1.6，可将改变进行合并：

```
co -l prog.c        得到最新修订文件
(更新文件1.3得到最新修订文件，然后:)
rcsmerge -p -r1.3 -r1.6 prog.c > prog.updated.c
```

撤消3.5版与3.2版之间做的修改，并覆盖工作文件：

```
rcsmerge -r3.5 -r3.2 chap08
```

rlog

rlog [*options*] *files*

显示RCS文件*files*的标识信息，包括与每个修订文件相关的日志信息、增加或删除的行号、最后登记的日期等。没有选项时，rlog显示所有信息。可使用选项显示指定的条目。rlog接受标准选项 -q、-V、-Vn、-T、-x 和 -z。

options

-b 清除显示。只输出默认的分支。

-d*dates*

显示修订文件的信息，这些文件的登记时间信息在日期*dates*范围内（由分号分隔的列表）。一定要用引号。每个日期可指定为：

d1 < *d2*

选择日期在 *d1* 和 *d2* 之间的修订文件，包括 *d1* 和 *d2*。

d1 <

选择日期为 *date1* 或晚于 *date1* 的修订文件。

d1 >

选择日期为 *date2* 或早于 *date2* 的修订文件。

时间信息的比较是严格的。如果两个文件的时间完全相同，＜和＞不起作用，要用＜＝和＞＝。

-h 显示一般 rlog 列表的开头。

-l[*users*]

只显示锁定的修订文件的信息；如果指定了用户 *users*，则只显示由 *users* 的列表锁定的修订文件 。

-L 跳过未锁定的文件。

-N 不输出符号名。

-r[*list*]

显示用逗号分隔的修订版本号列表*list*中的修订版本信息。如果未给出列表 *list*，则使用最新修订版本。项目可以指定为：

R1 选择修订版本号为 *R1* 的文件。如果 *R1* 为分支号，则选择该修订版本号的所有文件。

R1. 如果 *R1* 为分支号，则选择它的最新版本。

rlog

R1:*R2* 选择修订版本号为 *R1* 至 *R2* 的所有文件。

:*R1* 选择修订版本号从分支开头至 *R1* 的文件。

R1: 选择修订版本号从 *R1* 至分支结尾的文件。

来自 RCS 5.6 以前版本的范围分隔符“ - ”仍然有效。

`-R` 只显示 RCS 文件名。

`-s`*states*

显示修订版本信息，这些修订版本的状态与由逗号分隔的状态 *states* 列表中的状态相匹配。

`-t` 与 `-h` 相同，但只显示文件的描述。

`-w[`*users*`]`

显示修订文件信息，这些文件已经由用逗号分隔的用户 *users* 列表中的用户登记过。如果未指定 *users*，则假设使用调用者的名字。

示例

显示所有 RCS 文件的修订历史：

```
rlog RCS/*,v | more
```

显示由用户 `daniel` 锁定的 RCS 文件名：

```
rlog -R -L -ldaniel RCS/*
```

显示工作文件的“ 标题 ”部分（没有修订历史）：

```
rlog -t calc.c
```

第二十章

make 实用程序

本章将介绍以下主题：

- 概念综述
- 命令行语法
- 描述文件行
- 宏
- 特殊目标名
- 写命令行
- 默认宏、后缀和规则的示例

更多的信息，可以参见参考文献中的《Managing Projects with make》一书。

概念综述

make程序生成一系列可由 Unix shell 执行的命令。它使用由程序员提供的文件相关表完成这些工作，通过这些信息，可以使用户自动更新任务。它保存了创建某些文件的命令序列的记录，及文件或程序的列表，在有效执行这些文件或程序前，需要其他文件处于当前状态。当程序进行了修改时，make只需要很少的工作即可创建合适的文件。

每条相关语句称为一条规则（rule）。规则定义了一个或多个目标（target），目标是

要生成的文件或依赖的文件、先决条件（prerequisite）或依赖条件（dependency）。例如：prog.o 是依赖于 prog.c 的目标文件；每次修改了 prog.c，就会重新生成 prog.o。这就是 make 自动执行的任务，对于由很多块程序组成的大程序也是很重要的。

本章将介绍 SVR4 中的 make。许多 Unix 厂商以不同方式增强了 make 的功能，但往往相互不兼容。要了解最后的信息，可以检查本地的文档。

对于 Solaris，/user/lib/svr4.make 是 SVR4 的通用版本。如果在运行环境中设置了 USE_SVR4_MAKE，/usr/ccs/bin/make 或 /usr/xpg4/bin/make 就会运行该版本。

命令行语法

```
make [options] [targets] [macro definitions]
```

选项、目标和宏定义（macro definition）可以按任意顺序出现。宏定义使用如下格式：

```
name=string
```

如果 makefile 或 Makefile 不存在，make 将试图从 SCCS 中提取一个已存在的最新版本的文件（有些版本认识 RCS）。

options

-e 环境变量，可替代描述文件中定义的任意一个宏。

-f *file*
使用 *file* 作为描述文件；文件名 - 表示标准输入。-f 可以使用多次，用于连接多个描述文件。如果没有 -f 选项，make 首先查找名字为 makefile 的文件，然后查找名字为 Make-file 的文件。

-i 忽略命令中的错误代码（同 .IGNORE 命令）。

-k 放弃当前出错的目标文件，继续处理与之无关的目标文件。

-n 只输出而不执行命令（用于测试）。即使在描述文件的开头使用了 @，-n 也只是输出命令。

以 $(MAKE) 开始的行是一个异常。这些行将会执行。但是，由于 -n 要传递给 MAKEFLAGS 环境变量中后续的 make，make 也只输出其执行的命令。这就可以在全部软件层次中，不用进行实际的操作，就可测试所有的 makefile 文件。

`-p` 输出宏定义、后缀和目标描述。

`-q` 查询；若已经更新文件，则返回 0；否则返回非零值。

`-r` 不使用默认的规则。

`-s` 不显示命令行（同 .SILENT）。

`-t` 接触目标文件，导致这些文件更新。

描述文件行

描述文件中的指令被逐行解释。如果指令必须跨越一个输入行，则在行尾使用反斜杠 `\`，使得将下一行当作是延续行。描述文件可能包含以下几种类型的行。

空行

空行会被忽略掉。

注释行

用井字符（#）置于一行的开头或一行中间的其他位置。`make` 忽略该行井字符（#）之后所有的内容。

相关行

取决于一个或多个目标，将会执行其后的命令，可能的格式包括：

```
targets : prerequisites
targets :: prerequisites
```

第一种形式中，后续的命令如符合该先决条件，则执行；第二种形式是变体，用于在多个相关行中指定相同的目标。这两种形式中，如果未给定先决条件，后续的命令也照样执行（无论何时指定目标）。*targets* 前不需要用制表符。（在相关行行尾，在分号之后可写一个命令；但是，命令往往单独使用一行，前面需要加制表符。）

库（成员）形式的目标表示存档库的成员，例如：`libguide.a(dontpanic.o)`。

后缀规则

该规则规定：以第一个后缀结尾的文件是以第二个后缀结尾文件的先决条件（假定根文件名相同）。以下两种形式都可使用：

```
.suffix.suffix:
.suffix:
```

第二种形式表示根文件名取决于有对应后缀的文件名。

宏定义

宏定义有以下几种形式：

name = *string*

等号前后可为空格。

include 语句

与 C 语言的 include 命令相同，使用方式如下：

include *file*

make 在打开文件之前，先为宏扩展处理文件 *file* 的值。

命令行

在这些行给出命令重建过期文件。命令分组列在相关行之后，在前面加制表符。如果命令前有连字符“ - ”，make 会忽略返回的所有错误。如果命令前有符号 @，命令行不会回显出来（除非调用 make -n）。下面将进一步说明命令行。

宏

本节总结内部宏、修饰符、字符串替换和特殊的宏。

内部宏

$? 比当前目标文件更新得更晚的先决条件列表。可用于正规的描述文件条目，即不是后缀规则。

$@ 当前目标名，除了用于 make 库的描述文件条目以外（此时目标名会变为库的名字），可用于正规的描述文件条目和后缀规则中。

$$@ 当前目标名。只用于相关行分号的右边（不是所有的 make 版本都能用）。

$< 比当前目标文件更新得更晚的当前先决条件名标。只用于后缀规则和 .DEFAULT: 条目。

$* 比当前目标文件更新得更晚的先决条件名称，该名称没有后缀。只用于后缀规则。

$% 如果当前目标是库模块，则表示相应的 .o 文件名。可用于正规描述文件条目和后缀规则。

宏修饰符

宏修饰符不一定在所有的 make 变体中都有效。

D 除 $?外的内部宏名称的目录部分。有效的用法为：

```
$(*D)   $$(@D)
$(<D)   $(%D)
$(@D)
```

F 除$?外内部宏名称的文件部分。有效的用法为：

```
$(*F)   $$(@F)
$(<F)   $(%F)
$(@F)
```

宏字符串替换

字符串替换不一定在所有的make变体中都有效。

$(*macro*:*s1=s2*)

计算当前的$（*macro*）定义，所有*s1*出现的地方用字符串*s2*代替（*s1*出现在空格或标记前或宏定义结尾）。

特殊处理的宏

MAKEFLAGS 包含通过继承环境变量MAKEFLAGS得到的标志，加上任意命令行选项。用于向后面的make调用传递标志。通常通过makefile中有$(MAKE)的命令行实现。

SHELL 设置用于命令注释的shell。如果在描述文件中未定义此宏，则取值取决于系统。有些Unix实现使用来自用户环境的shell（也用其他宏）。还有一些实现（包括SVR4）在/bin/sh目录下设置了默认的SHELL。

VPATH （不是在所有make的变体中都适用。）在找不到当前目录时，根据先决条件到指定目录列表中查找。

特殊目标名

.DEFAULT: 如果make找不到描述文件条目或后缀规则以建立请求的目标，则执行与该目标关联的命令。

.IGNORE: 忽略错误代码。同 -i 选项。

.PRECIOUS: 当发出一个信号（如中断）中止make的执行，或当描述文件中的命令行返回了出错信息时，不删除为该目标指定的文件。

.SILENT: 执行命令但不回显它们。同 -s 选项。

`.SUFFIXES:` 与该目标关联的后缀在后缀规则中是有意义的。如未列出后缀，则有效地“关闭”已有的后缀规则列表。

写命令行

写一个好的、可移植的`Makefile`文件是一门艺术。实践和经验可提高编写的技巧。以下是开始进行此项工作的一些提示：

- 自己命名一个`Makefile`文件，而不要用`makefile`程序产生的文件名，因为这些文件先用ls命令才能列出。这使得在一个有很多文件的目录中可以容易地找到它。
- 记住命令行前一定有一个前导制表符。不能用空格使其缩进，即使是8个空格。如果用了空格，`make`会退出并提示“丢失分隔符”。
- 记住`$`符号对`make`非常特殊。在命令行中用`$$`得到字符`$`。这对访问不是用`make`宏得到的环境变量是尤其重要的。如果当前的进程ID用到shell的`$$`，则需键入`$$$$`。
- 写多行的shell语句，如条件语句和循环语句，用分号和反斜杠结尾：

```
if [ -f specfile ] ; then \
... ; \
else \
... ; \
fi
```

  注意shell关键字`then`和`else`之后不需用分号。（这样`make`会将反斜杠和换行符传递给shell。转义的换行符在语法上不重要，所以用分号分开命令的不同部分。这有点令人糊涂。如果在shell脚本中用分号代替换行符，该做的工作会正确地执行。）
- 记住每行命令都运行在单独的shell下。这意味着改变shell环境的命令（如`cd`）在跨多行时无效。正确的方法是在同一行写一组用分号隔开的命令：

```
cd subdir; $(MAKE)
```

- 为确保可移植性，总是设置`SHELL`为`/bin/sh`。若`make`使用的值未在`Makefile`中显式设置，有些版本的`make`使用为`SHELL`提供的环境值。
- 使用宏代替标准命令。`make`帮助使用这种方法，提供了如`$(CC)`、`$(YACC)`之类的宏。
- 当删除文件时，在命令行开头用`-$(RM)`命令而不要用`$($RM)`命令（“-”使`make`

忽略命令的退出状态)。这样，如果要删除的文件不存在，rm会带着出错信息退出，make将继续执行。

- 一般在主程序目录树的子目录下，运行make的辅助调用时，总是使用$(MAKE)，并且不执行make。即使提供了-n选项，也总是执行有$(MAKE)的行，这样可以测试Makefile文件全部的分层结构。直接调用make的行不会有这样的结果。
- 将大型软件项目分解成子项目，每个子项目有一个子目录会很方便。顶级的Makefile只调用每个子目录下的make。以如下方式：

```
SUBDIRS = proj1 proj2 proj3
...
projects: $(SUBDIRS)
          for i in $(SUBDIRS); \
           do \
                    echo ====== Making in $$i ; \
                    ( cd $$i ; $(MAKE) $(MAKEFLAGS) $@ ) ; \
           done
```

默认宏、后缀和规则的示例

```
.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~ \
.C .C~ .Y .Y~ .L .L~

MAKE=make
BUILD=build
AR=ar
ARFLAGS=rv
AS=as
ASFLAGS=
CC=cc
CFLAGS=-O
F77=f77
FFLAGS=-O
GET=get
GFLAGS=
LD=ld
LDFLAGS=
LEX=lex
LFLAGS=
YACC=yacc
YFLAGS=
C++C=CC
C++FLAGS=-O

.c:
        $(CC) $(CFLAGS) $< -o $@ $(LDFLAGS)
```

```
.c~:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) $*.c -o $@ $(LDFLAGS)
        -rm -f $*.c
.f:
        $(F77) $(FFLAGS) $< -o $@ $(LDFLAGS)
.f~:
        $(GET) $(GFLAGS) $<
        $(F77) $(FFLAGS) $*.f -o $@ $(LDFLAGS)
        -rm -f $*.f
.s:
        $(AS) $(ASFLAGS) $< -o $@ $(LDFLAGS)
.s~:
        $(GET) $(GFLAGS) $<
        $(AS) $(ASFLAGS) $*.s -o $* $(LDFLAGS)
        -rm -f $*.s
.sh:
        cp $< $@; chmod 0777 $@
.sh~:
        $(GET) $(GFLAGS) $<
        cp $*.sh $*; chmod 0777 $@
        -rm -f $*.sh
.C:
        $(C++C) $(C++FLAGS) $< -o $@ $(LDFLAGS)
.C~:
        $(GET) $(GFLAGS) $<
        $(C++C) $(C++FLAGS) $*.C -o $@ $(LDFLAGS)
        -rm -f $*.C
.c.a:
        $(CC) $(CFLAGS) -c $<
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.o
.c.o:
        $(CC) $(CFLAGS) -c $<
.c~.a:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) -c $*.c
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.[co]
.c~.c:
        $(GET) $(GFLAGS) $<
.c~.o:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) -c $*.c
        -rm -f $*.c
.f.a:
        $(F77) $(FFLAGS) -c $*.f
        $(AR) $(ARFLAGS) $@ $*.o
```

```
        -rm -f $*.o
.f.o:
        $(F77) $(FFLAGS) -c $*.f
.f~.a:
        $(GET) $(GFLAGS) $<
        $(F77) $(FFLAGS) -c $*.f
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.[fo]
.f~.f:
        $(GET) $(GFLAGS) $<
.f~.o:
        $(GET) $(GFLAGS) $<
        $(F77) $(FFLAGS) -c $*.f
        -rm -f $*.f
.h~.h:
        $(GET) $(GFLAGS) $<
.l.c:
        $(LEX) $(LFLAGS) $<
        mv lex.yy.c $@
.l.o:
        $(LEX) $(LFLAGS) $<
        $(CC) $(CFLAGS) -c lex.yy.c
        -rm lex.yy.c; mv lex.yy.o $@
.l~.c:
        $(GET) $(GFLAGS) $<
        $(LEX) $(LFLAGS) $*.l
        mv lex.yy.c $@
        -rm -f $*.l
.l~.l:
        $(GET) $(GFLAGS) $<
.l~.o:
        $(GET) $(GFLAGS) $<
        $(LEX) $(LFLAGS) $*.l
        $(CC) $(CFLAGS) -c lex.yy.c
        -rm -f lex.yy.c $*.l
        mv lex.yy.o $@
.s.a:
        $(AS) $(ASFLAGS) -o $*.o $*.s
        $(AR) $(ARFLAGS) $@ $*.o
.s.o:
        $(AS) $(ASFLAGS) -o $@ $<
.s~.a:
        $(GET) $(GFLAGS) $<
        $(AS) $(ASFLAGS) -o $*.o $*.s
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.[so]
.s~.o:
        $(GET) $(GFLAGS) $<
```

```
        $(AS) $(ASFLAGS) -o $*.o $*.s
        -rm -f $*.s
.s~.s:
        $(GET) $(GFLAGS) $<
.sh~.sh:
        $(GET) $(GFLAGS) $<
.y.c:
        $(YACC) $(YFLAGS) $<
        mv y.tab.c $@
.y.o:
        $(YACC) $(YFLAGS) $<
        $(CC) $(CFLAGS) -c y.tab.c
        -rm y.tab.c
        mv y.tab.o $@
.y~.c:
        $(GET) $(GFLAGS) $<
        $(YACC) $(YFLAGS) $*.y
        mv y.tab.c $*.c
        -rm -f $*.y
.y~.o:
        $(GET) $(GFLAGS) $<
        $(YACC) $(YFLAGS) $*.y
        $(CC) $(CFLAGS) -c y.tab.c
        -rm -f y.tab.c $*.y
        mv y.tab.o $*.o
.y~.y :
        $(GET) $(GFLAGS) $<
.C.a:
        $(C++C) $(C++FLAGS) -c $<
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.o
.C.o:
        $(C++C) $(C++FLAGS) -c $<
.C~.a:
        $(GET) $(GFLAGS) $<
        $(C++C) $(C++FLAGS) -c $*.C
        $(AR) $(ARFLAGS) $@ $*.o
        -rm -f $*.[Co]
.C~.C:
        $(GET) $(GFLAGS) $<
.C~.o:
        $(GET) $(GFLAGS) $<
        $(C++C) $(C++FLAGS) -c $*.C
        -rm -f $*.C
.L.C:
        $(LEX) $(LFLAGS) $<
        mv lex.yy.c $@
.L.o:
```

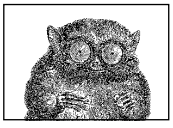
```
	$(LEX) $(LFLAGS) $<
	$(C++C) $(C++FLAGS) -c lex.yy.c
	-rm lex.yy.c; mv lex.yy.o $@
.L~.C:
	$(GET) $(GFLAGS) $<
	$(LEX) $(LFLAGS) $*.L
	mv lex.yy.c $@
	-rm -f $*.L
.L~.L:
	$(GET) $(GFLAGS) $<
.L~.o:
	$(GET) $(GFLAGS) $<
	$(LEX) $(LFLAGS) $*.L
	$(C++C) $(C++FLAGS) -c lex.yy.c
	-rm -f lex.yy.c $*.L
	mv lex.yy.c $@
.Y.C:
	$(YACC) $(YFLAGS) $<
	mv y.tab.c $@
.Y.o:
	$(YACC) $(YFLAGS) $<
	$(C++C) $(C++FLAGS) -c y.tab.c
	-rm y.tab.c
	mv y.tab.o $@
.Y~.C:
	$(GET) $(GFLAGS) $<
	$(YACC) $(YFLAGS) $*.Y
	mv y.tab.c $*.C
	-rm -f $*.Y
.Y~.o:
	$(GET) $(GFLAGS) $<
	$(YACC) $(YFLAGS) $*.Y
	$(C++C) $(C++FLAGS) -c y.tab.c
	-rm -f y.tab.c $*.Y
	mv y.tab.o $*.o
.Y~.Y :
	$(GET) $(GFLAGS) $<
markfile.o:	markfile
	echo "static char _sccsid[] = \"`grep @'(#)' markfile`\";" > markfile.c
	$(CC) -c markfile.c
	-rm -f markfile.c
.SCCS_GET:
	$(GET) $(GFLAGS) s.$@
```

第五部分

附录

本部分包括ASCII字符的附录、一个描述已淘汰命令的附录和一个Unix参考文献的目录。

- 附录一，ASCII 字符集
- 附录二，已淘汰的命令
- 参考文献



附录一

ASCII 字符集

本附录介绍了 ASCII 字符集及其十进制、八进制和十六进制值。第一个表格列出了非打印字符。在需要打印时，可用八进制码将其打印出来。例如，`echo`命令和`tr`命令允许指定形式为 `\`*nnn* 的八进制值的字符。也可用 `od` 命令以多种不同的形式显示这些字符。

第二个表格列出了可打印字符。前面介绍的命令不仅都适用，而且可以使用通配符形式指定字符的使用范围。

表 A-1：非打印字符

十进制	八进制	十六进制	字符	说明
0	000	00	CTRL-@	NUL（空提示符）
1	001	01	CTRL-A	SOH（标题开始字符）
2	002	02	CTRL-B	STX（本文开始字符）
3	003	03	CTRL-C	ETX（本文结束字符）
4	004	04	CTRL-D	EOT（传输结束字符）
5	005	05	CTRL-E	ENQ（询问符）
6	006	06	CTRL-F	ACK（确认符）
7	007	07	CTRL-G	BEL（报警符）
8	010	08	CTRL-H	BS（退格符）
9	011	09	CTRL-I	HT（横向制表符）
10	012	0A	CTRL-J	LF（换行符）

表 A-1：非打印字符（续）

十进制	八进制	十六进制	字符	说明
11	013	0B	CTRL-K	VT（纵向制表符）
12	014	0C	CTRL-L	FF（进纸符）
13	015	0D	CTRL-M	CR（回车符）
14	016	0E	CTRL-N	SO（移出符）
15	017	0F	CTRL-O	SI（移入符）
16	020	10	CTRL-P	DLE（数据链路转义符）
17	021	11	CTRL-Q	DC1（第 1 类设备控制字符，XON）
18	022	12	CTRL-R	DC2（第 2 类设备控制字符）
19	023	13	CTRL-S	DC3（第 3 类设备控制字符，XOFF）
20	024	14	CTRL-T	DC4（第 4 类设备控制字符）
21	025	15	CTRL-U	NAK（否定符）
22	026	16	CTRL-V	SYN（同步空闲符）
23	027	17	CTRL-W	ETB（传输块结束符）
24	030	18	CTRL-X	CAN（作废符）
25	031	19	CTRL-Y	EM（媒体用毕符）
26	032	1A	CTRL-Z	SUB（置换符）
27	033	1B	CTRL-[	ESC（转义符）
28	034	1C	CTRL-\	FS（文件分隔符）
29	035	1D	CTRL-]	GS（组分隔符）
30	036	1E	CTRL-^	RS（记录分隔符）
31	037	1F	CTRL-_	US（单元分隔符）
127	177	7F		DEL（删除符）

表 A-2：可打印字符

十进制	八进制	十六进制	字符	说明
32	040	20		空格
33	041	21	!	感叹号
34	042	22	"	双引号
35	043	23	#	井字符
36	044	24	$	美元符号
37	045	25	%	百分号
38	046	26	&	and 符号

表 A-2：可打印字符（续）

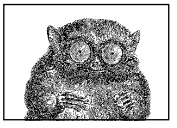
十进制	八进制	十六进制	字符	说明
39	047	27	′	撇号
40	050	28	(	左括号
41	051	29	)	右括号
42	052	2A	*	星号
43	053	2B	+	加号
44	054	2C	,	逗号
45	055	2D	—	连字符
46	056	2E	.	句号
47	057	2F	/	斜杠
48	060	30	0	
49	061	31	1	
50	062	32	2	
51	063	33	3	
52	064	34	4	
53	065	35	5	
54	066	36	6	
55	067	37	7	
56	070	38	8	
57	071	39	9	
58	072	3A	:	冒号
59	073	3B	;	分号
60	074	3C	<	左尖括号
61	075	3D	=	等号
62	076	3E	>	右尖括号
63	077	3F	?	问号
64	100	40	@	At 符号
65	101	41	A	
66	102	42	B	
67	103	43	C	
68	104	44	D	
69	105	45	E	

表 A-2：可打印字符（续）

十进制	八进制	十六进制	字符	说明
70	106	46	F	
71	107	47	G	
72	110	48	H	
73	111	49	I	
74	112	4A	J	
75	113	4B	K	
76	114	4C	L	
77	115	4D	M	
78	116	4E	N	
79	117	4F	O	
80	120	50	P	
81	121	51	Q	
82	122	52	R	
83	123	53	S	
84	124	54	T	
85	125	55	U	
86	126	56	V	
87	127	57	W	
88	130	58	X	
89	131	59	Y	
90	132	5A	Z	
91	133	5B	[	左括号
92	134	5C	\	反斜杠
93	135	5D	]	右括号
94	136	5E	^	脱字符号
95	137	5F	_	下划线
96	140	60	、	反引号
97	141	61	a	
98	142	62	b	
99	143	63	c	
100	144	64	d	

表 A-2：可打印字符（续）

十进制	八进制	十六进制	字符	说明
101	145	65	e	
102	146	66	f	
103	147	67	g	
104	150	68	h	
105	151	69	i	
106	152	6A	j	
107	153	6B	k	
108	154	6C	l	
109	155	6D	m	
110	156	6E	n	
111	157	6F	o	
112	160	70	p	
113	161	71	q	
114	162	72	r	
115	163	73	s	
116	164	74	t	
117	165	75	u	
118	166	76	v	
119	167	77	w	
120	170	78	x	
121	171	79	y	
122	172	7A	z	
123	173	7B	{	左花括号
124	174	7C	|	竖线
125	175	7D	}	右花括号
126	176	7E	~	波浪号



附录二

已淘汰的命令

本附录包括的命令项仍与 SVR4 和（或）Solaris 一起发布，但功能上已经由其他命令或技术替代，这里将有：

- 介绍
- 按字母顺序概述命令

介绍

本附录的命令分成几类，这个列表对命令及淘汰的原因进行了说明。

存档维护

`lorder` 和 `tsort` 用于将目标文件放置在档案库中。`ar` 的现代版本维护一个符号表，允许加载程序 `ld` 在需要时查找目标文件。

通信

`cu`、`uucp`、`uuglist`、`uulog`、`uuname`、`uupick`、`uustat`、`uuto`、`uux` 和 `write`。这些命令用于拨号式的交互或系统间的通信。Internet 广泛有效的连通性使这些命令被淘汰了，`talk` 是 `write` 的比较好的替代。

压缩

`pack`、`pcat` 和 `unpack` 已经被 `compress/uncompress` 和 `gzip/gunzip` 淘汰了。

文件处理

- `bfs`旨在于处理大文件，最多为1兆字节。在现代系统中，`vi`很容易处理相当大的文件。
- `crypt`提供对文件加密，算法很弱，现在已可以得到更好的工具了。
- `newform`旨在于数据再格式化，用`sed`或`awk`很容易处理。
- `red`是`ed`的受限版本。在实践中，命令的不同限制版本很少有用，它们很难设置也不容易正确使用。`ed`现在已很少用了。
- `sum`仅仅是将文件中的字节合计起来，对有问题的值求校验和。现由`cksum`代替。
- `tabs`在可再编程的终端上控制设置制表位。但是Unix系统很少用来在其所处理的编程序语言中编程。
- `vc`提供非常简单的版本控制形式。RCS和SCCS是较好的代替。

层

`ismpx`、`jterm`、`jwin`、`layers`、`relogin`和`shl`。

除了`shl`都特定于现在已淘汰的AT&T Teletype 5620 DMD窗口终端。X Window系统在现代的Unix系统中提供窗口功能。`shl`企图提供与BSD作业控制相似的功能控制终端，但始终没有实现。

网络状态

`ruptime`、`rwho`和`whois`。

前两个程序使用经常过载本地网络的端口监控程序。`whois`登记程序过大而不适于Internet使用，这个用于在中心跟踪每个使用的人员的程序大太了。

简单菜单

`face`和`fmli`提供了简单的方法创建CRT终端的菜单驱动程序。始终没有做到，特别是在基于X Window的系统日益普及的情况下。

UPAS

`mailalias`、`notify`和`vacation`与UPAS邮件系统一起使用，该邮件系统与SVR4是标准的。现代Unix系统使用`sendmail`。

窗口系统

OpenWindows（由`openwin`命令开始）是SunOS和Solaris多年来默认的窗口系统。CDE现在是Sun为Solaris提供的首选窗口环境。 OpenWindows不再被Solaris 7以后的版本支持。

其他混杂的命令

- `cof2elf`将COFF格式目标文件和档案转换到ELF格式。ELF格式至少有10年了，这个程序已没有太大必要。
- `fmtmsg`旨在于提供从shell脚本中产生错误消息的标准化方法，从没有广泛地使用过。
- `fold` 进行转行，以满足特定的宽度。`fmt`总地来说工作得更好。
- `lptest`为行式打印机产生波动的图案。现在，激光打印机和喷墨打印机很普遍了。
- `newgrp`表示Unix系统允许一个用户一次只能在一个组里。现代Unix系统允许用户同时在多个组里。
- `news`提供了系统用户感兴趣的项目。是针对每个机器的。Usenet news软件是个好得多的替代。
- `pg`是个简单的翻页程序，用`more`代替。

按字母顺序概述命令

bfs

`bfs [`*`option`*`] `*`file`*

大文件扫描器。读一个大文件*file*，使用与`ed`类似的语法。在扫描非常大的文件时，因为文件不读入缓冲区，所以这个命令比`ed`更有效。文件可大到1024K字节。`bfs`可用于查看大型文件，并识别由`csplit`划分的节。不太有用。

option

`-` 不输出文件大小。

cof2elf

`cof2elf [`*`options`*`] `*`files`*

转换一个或多个COFF文件*files*为ELF格式，覆盖原来的内容。输入可为目标文件或档案文件。

options

`-i` 忽略未识别的数据，无论如何都进行转换。

cof2elf

`-q` 安静模式，运行时不显示消息。

`-Q`*c* 输出时输出 `cof2elf` 信息（如果 *c* = `y`）或不输出信息（如果 *c* = `n`，默认）。

`-s`*dir*
将原始文件保存在已有的目录 *dir* 下。

`-V` 在标准错误中输出 cof2elf 版本。

crypt

`crypt [`*password*`] < `*file*` > `*encryptedfile*

加密文件，阻止未授权的访问。*password* 可以是选择的字符串，也可以是选项 `-k`，`-k` 为环境变量 CRYPTKEY 赋值（Solaris: CrYpTkEy）作为*password*。用相同的*password*加密或解密文件。如果未给出*password*，`crypt`会进行提示。`crypt`只限于美国使用（由于出口限制）。

使用的算法相当弱，这个命令不能用于重大内容的加密。参见《PGP：Pretty Good Privacy》，列于参考文献中。

cu

`cu [`*options*`] [`*destination*`] [`*command*`]`

通过专线或调制解调器呼叫另一个Unix系统或终端。非Unix系统也可被呼叫到。

options

`-b`*n* 用 *n* 位字符（7 或 8）处理线路。

`-c`*name*
搜索UUCP设备文件并选择与*name*匹配的本地网络（假设连接到一个系统）。

`-C` 不进入交互模式，通过连接到远程系统的标准输入、输出运行命令行的命令。只用于 Solaris。

`-d` 输出诊断信息。

`-e` 向远程系统发送偶校验。

`-h` 模拟本地响应，并支持向其他系统呼叫，期望终端使用半双工模式。

cu

`-H` 忽略中断。当呼叫的远程系统断开连接并用登录提示回叫时有用。只用于 Solaris。

`-l`*line*
在名为 *line* 的设备（例如，`/dev/tty001`）上通信。

`-L` 使用由 `/etc/uucp/Systems` 指定的谈话顺序。只用于 Solaris。

`-n` 向用户提示电话号码。

`-o` 使用奇校验（与使用 `-e` 相反）

`-s`*n* 设置传输速率为 *n*（例如，1200、2400、9600 bps）。默认值是 Any。

`-t` 拨打有自动应答装置的 ASCII 终端。

destination

telno 调制解调器连接的电话号码。

system 呼叫 uucp 知道的系统（运行 uuname 列出合法的系统名）。

addr 本地网络专用的地址。

cu 运行两个进程：发送和接收。发送从标准输入上读入并将字符发给远程系统；接收从远程系统上读入数据，并将行传送给标准输出。由波浪号（~）开始的行被作为命令处理，不能通过。

传输选项

`~.` 结束会话。

`~!` 退出到本地系统的交互 shell 中。

`~!`*cmd* `...`
在本地系统运行命令（通过 `sh -c`）。

`~$`*cmd* `...`
在本地运行命令，输出到远程系统上。

`~%`*cd*
改变本地系统的目录。

`~%take` *file* [*target*]
从远程系统将文件 *file* 拷贝到本地系统的目标 *target* 中。如果 *target* 被省略，文件在两处都可使用。远程系统必须运行 Unix 以使此命令可以工作。不提供传输数据的校验和。

cu

~%put *file* [*target*]
从本地系统将文件 *file* 拷贝到远程系统的目标 *target* 中。如果 *target* 被省略，文件在两处都可使用。远程系统必须运行 Unix 以使此命令可以工作。不提供传输数据的校验和。

~~ ...
当要传送由波浪号开头的行时，用两个波浪号。这允许向 cu 链中的多个系统发出命令。例如：用 ~~ 终止第二个系统 cud 到第一个系统的会话。

~%b 向远程系统发送一个 BREAK 序列。

~%d 关闭或打开 debug 模式。

~t 为本地终端输出 termio 结构（用于调试）。

~l 为通信线路输出 termio 结构（用于调试）。

~%ifc
为剩下的会话打开 / 关闭 DC3/DC1 XOFF/XON 控制协议（字符 ^S、^Q）（以前的 ~%nostop 仍然有效）。

~%ofc
设置输出流控制开或关。

~%divert
允许 / 禁止不是由 ~%take 指定的转移。

~%old
为接收的转移允许 / 禁止旧格式的语法。

示例

以 9600 baud 连接终端线路 /dev/ttya：

```
cu -s9600 -l/dev/ttya
```

连接电话号为 555-9876 的调制解调器：

```
cu 5559876
```

连接名为 usenix 的系统：

```
cu usenix
```

face

face [*options*] [*files*]

调用加框的存取命令环境界面（Framed Access Command Environment Interface）并打开文件 *files*。习惯上，每个文件名必须用 Menu.*string*、Form.*string* 或 Text.*string* 形式之一，具体取决于被打开的目标类型。如果没有指定 *files*，face 打开 FACE 菜单，同时也打开由环境变量 LOGINWIN 指定的默认目标。

options

-a *afile*
: 装入路径名的别名列表，别名由文件 *afile* 指定。列表项的形式为 *alias=pathname*。这个文件一旦装入，就可以用简写表示法 $*alias* 指向长路径名。

-c *cfile*
: 装入命令别名列表，别名由文件 *cfile* 指定。这个文件允许修改 FACE 命令默认的动作或创建新的命令。

-i *ifile*
: 装入文件 *ifile*，指定启动功能如：导入帧、标题信息、屏幕颜色和标号。

fmli

fmli [*options*] *files*

调用 Form 和 Menu 语言解释程序并打开文件。习惯上，每个文件名必须采用 Menu.*string*、Form.*string* 或 Text.*string* 形式之一，具体取决于打开目标的类型。

options

-a *afile*
: 装入路径名的别名列表，别名由文件 *afile* 指定。列表项的形式为 *alias=pathname*。这个文件一装入，就可以用简写表示法 $*alias* 指向长路径名。

-c *cfile*
: 装入命令别名列表，别名由文件 *cfile* 指定。这个文件允许修改 FMLT 命令默认的动作或创建新的命令。

fmli

`-i` *ifile*

装入文件*ifile*，指定启动功能如：导入帧、标题信息、屏幕颜色和标号。

fmtmsg

`fmtmsg [`*options*`]` *text*

在标准错误（或系统控制台上）输出*text*，作为格式化的错误消息的部分。*text*作为单个参数必须用引号引起来。`fmtmsg`用于shell脚本，以标准格式输出消息。

消息显示如下：

```
label:    severity:     text
TO FIX:   action        tag
```

可定义MSGVERB变量，选择要输出的消息，用下面的选项说明。

如果提供附加条件，SEV_LEVEL环境变量允许增加附加服务和要输出的关联字符串。

options

`-a` *action*

字符串描述了恢复错误采取的第一个动作。字符串“TO FIX：”在*action*字符串之前。

`-c` *source*

问题根源，*source*是`hard`（硬件）、`soft`（软件）或`firm`（固件）之一。

`-l` *label*

通过文本标号*label*识别信息来源，通常形式为：*file*:*command*。

`-s` *severity*

情况的严重性。*severity*是`halt`（停机）、`error`（错误）、`warn`（警告）或`info`（通知）之一。

`-t` *tag*

消息的另一个字符串标识符。

fmtmsg

`-u` *types*

将消息分为一个或多个类型 *types*（由逗号分隔）。*types* 可为关键字 `appl`、`util` 或 `opsys`（意为问题分别来自应用程序、实用程序或内核）之一、关键字 `recov` 或 `nrecov`（应用程序可恢复或不可恢复）之一、`print`（在标准错误上显示的消息）、`console`（在系统控制台上显示的消息）。

fold

`fold [`*options*`] [`*files*`]`

为文件 *files* 断行，使之不比指定的宽度宽。即使在单词的中间，`fold` 也会准确地在指定的宽度断行。

options

`-b` 行宽指定为字节，而不是字符。只用于 Solaris。

`-s` 在第一个 *width* 字符内的最后的空白字符之后断行。只用于 Solaris。

`-w` *n*

创建宽度为 *n* 的行（默认值为 80）。（为与 BSD 兼容，可用 *-n* 调用）。

ismpx

`ismpx [`*option*`]`

测试标准输入是否运行在 `layers` 下（命令名由“ **Is** the **m**ulti**p**le**x**or running（多路转换器是否正在运行）? ”而来）。输出是 `yes`（退出状态 0）或 `no`（退出状态 1）。在 shell 脚本中下载程序到一个层窗口终端或依赖于屏幕大小的终端时有用。

option

`-s` 禁止输出，只返回到退出状态。

示例

```
if ismpx -s
then jwin
fi
```

jterm

jterm

在程序改变了终端的层属性后，重置窗口终端层。只在层下使用。成功时返回 0，不成功时返回 /。

jwin

jwin

按字节输出当前窗口的尺寸。只用在 layers 下。

layers

layers [*options*] [*layers_program*]

用于 DMD 窗口终端的层多路转换器。layers 在窗口终端上管理异步窗口。*layers_program* 是一个文件，包含固化的补丁程序，由 layers 下载到终端上（在层创建之前或启动命令执行前）。

options

-d 输出文本、数据和下载的固件补丁程序的bss部分的大小到标准错误上。

-D 在标准错误上输出调试消息。

-f *file*
用 *file* 给定的配置初始化 layers。*file* 的每行是要被创建的一层，格式为 *x1 y1 x2 y2 commands*，指定来源、对角和启动命令。例如：

```
10 10 800 240 date; who; exec $SHELL
```

-h *list*
提供推进层的由逗号分隔的 STREAMS 模块列表 *list*。

-m *size*
设置 xt 包的数据部分为最大尺寸（32~252）。

-p 在标准错误上输出下载协议统计和下载固件补丁程序的记录。

-s 退出层后，在标准错误上报告协议统计。

-t 打开 xt 驱动程序包追踪并在退出 layers 后将追踪记录转储到标准错误上。

lorder

/usr/ccs/bin/lorder *objfiles*

取目标文件名（例如带.o后缀的文件）并输出一个相关对的列表。列出的第一个文件包括外部标识符的引用，标识符由第二个文件定义。lorder的输出可发送给tsort，在档案中建立文件的顺序以便更有效地载入。

示例

产生一个有顺序的目标文件列表并由它们替换库libmyprog.a中的内容（倘若它们更新）：

```
ar cru libmyprog.a `lorder *.o | tsort`
```

lptest

/usr/ucb/lptest [*length* [*n*]]

在标准输出上显示所有96个可打印的ASCII字符。每个位置上输出的字符形成了“波纹图案”。可指定输出行的长度*length*（默认值是79）和输出的行数（默认值是200）。lptest在测试打印机和终端或运行有虚拟输入的shell脚本时有用。

mailalias

mailalias [*options*] *names*

显示与一个或多个别名*names*相关的邮件地址。mailalias显示的地址在文件/var/mail/*name*、$HOME/lib/names和在/etc/mail/namefiles中的列表指向的文件中。mailalias被mail调用。

注意

这个命令是UPAS邮件系统软件的一部分。商业Unix系统都用sendmail，这个命令不适用。

options

-s 禁止输出名字*names*，只显示相应的邮件地址。

-v 详细模式，显示调试信息。

newform

newform [*options*] *files*

根据指定的选项格式化文件*files*。newform类似于cut和paste，可以过滤文本输出。选项可多次出现，可以散布在*files*间（-s除外，必须先出现）。

options

-a[*n*]
每行的最后添加*n*个字符，或如果没有指定*n*，则向每行添加字符，直到行达到-l指定的长度为止。

-b[*n*]
从每行的开头删除*n*个字符，或如果没有指定*n*，则删除字符，直到行达到-l指定的长度为止。

-c*m* 当-a或-p填充行时，用字符*m*（代替空格）。-c必须在-a或-p之前。

-e[*n*]
同-b，但从结尾处开始删除。

-f 显示由最后一个-o选项使用的*tabspec*格式。

-i′*tabspec*′
用*tabspec*转换将制表符扩展为空格（默认值是8个空格）；*tabspec*是在**tabs**下列出的选项之一。

-l[*n*]
行长度为*n*（默认值为72）。如果未指定-l，默认行长度为80。-l通常在其他修改行长度的选项（-a、-b、-c、-e或-p）之前。

-o′*tabspec*′
用*tabspec*转换将空格转换为制表符。

-p[*n*]
同-a，但填充行的开头。

-s 从每行中剥去引导字符（直到并包括第一个制表符）。前7个字符移到行尾（不包括制表符）。所有行必须包含至少一个制表符。

newform

示例

从一个 COBOL 程序中删除顺序号：

```
newform -l1 -b7 file
```

newgrp

newgrp [-] [*group*]

登录到组 *group*。如果没有指定 *group* 名，则恢复原来的组。如果给出了 -，则采用与 *group* 登录相同的环境进行登录。

这个命令也内置在 Bourne 和 Korn shell 中。在现代的 Unix 系统中，允许用户同时在多个组内，这个命令淘汰了。

news

news [*options*] [*item_files*]

向 news 目录询问当前事件的信息。如果没有参数，news 输出所有当前 item_files。条目通常驻留在 /usr/news 或 /var/news 中。

options

-a 输出所有 news 条目，不论是不是当前的条目。

-n 输出 news 条目的名字，而不是它们的内容。

-s 报告当前 news 条目数。

notify

notify [*options*]

当邮件到达时，通知用户。没有选项表示自动通知是允许还是禁止。

注意

这个命令是 UPAS 邮件系统软件的一部分。商业 Unix 系统都使用 sendmail，因此，这个命令没有用。

options

-m *file*

保存 mail 消息到文件 *file*（默认为 $HOME/.mailfile）。只用于允许自动通知的情况（-y 选项）。

notify

`-n` 禁止邮件通知。单独使用 `-n`。

`-y` 允许邮件通知。

openwin

`/usr/openwin/bin/openwin [`*`options`*`]`

开始OpenWindows图形用户界面环境。这个环境已经淘汰了，比较好的环境是CDE，OpenWindows不再被Solaris 7以后的版本支持。见第二章。

有用的OpenWindows命令

以下OpenWindows命令可能还有意思，详见参考页。

`calctool`	屏幕上的科学计算器
`clock`	时钟
`cm`	日历管理器
`cmdtool`	终端仿真器
`iconedit`	图标编辑器
`mailtool`	邮件阅读器
`oclock`	圆形钟
`pageview`	PostScript浏览器
`perfmeter`	系统性能测试器
`printtool`	打印管理器
`shelltool`	另一个终端仿真器（不妨碍`stty`设置）
`snapshot`	保存X显示的位置
`textedit`	简单文本编辑器
`xbiff`	图形邮件到达看家狗程序
`xcalc`	简单的屏幕计算器
`xditview`	与设备无关的`troff`输出浏览器
`xedit`	简单文本编辑器
`xhost`	控制可连接到显示设备人员的权限
`xload`	系统装入监视器

openwin	`xlock` 屏幕保护 / 锁 `xmag` 放大显示的部分 `xman` 参考页浏览器 `xterm` 标准 X window 系统终端仿真器
pack	`pack [`*`options`*`] `*`files`* 压缩每个文件 *files*，把结果放入 *file*`.z`。原始文件被替换了。要将打包的文件恢复为原来的形式，参见 **pcat** 和 **unpack**。 `compress` 和 `gzip` 命令压缩得较好。推荐使用它们。（详见第二章里的 **compress** 和 **gzip**）。 options `-` 输出每个字节使用的次数、相对频率和字节代码。 `-f` 即使磁盘不能保存，也进行压缩。
pcat	`pcat `*`files`* 显示（像用 `cat` 那样）一个或多个打包文件 *files*。参见 **pack** 和 **unpack**。
pg	`pg [`*`options`*`] [`*`files`*`]` 在终端上显示指定文件 *files*，一次一页。每屏显示后，会提示用户“ 按回车键显示下一页 ”。按 `h` 键得到另外命令的帮助，按 `q` 键退出。参见第二章中的 **more**。 options `-c` 清屏（与 `more` 的 `-c` 相同）。 `-e` 文件之间不停顿。 `-f` 不拆分长的行。 `-n` 在提示时发出一个 `pg` 命令，不等换行回车（`more` 也是这样工作）。 `-p`*`str`* 用字符串 *str* 作为命令提示。特殊变量 `%d` 显示页码。

pg

`-r` 受限制模式。不允许 shell 转义。

`-s` 以突出模式显示消息（反白显示）。

`-`*n* 每个窗口使用 *n* 行（默认为全屏幕）。

`+`*num*

从行号为 *num* 的行开始显示。

`+/`*pat*

从包括模式 *pat* 的第一行开始显示。

示例

```
pg -p 'Page %d :' file
```

red

`red [`*options*`] [`*file*`]`

`ed` 的受制版本。只能编辑在当前工作目录下的文件。使用 ! 的 shell 命令不被允许。

relogin

`relogin [`*option*`] [`*terminal*`]`

改变登录入口，反映出运行在 `layers` 下的当前窗口。这保证了像 `who` 和 `write` 这样的命令正确使用登录信息。`layers` 自动调用 `relogin`，但有时想用 `relogin` 改变 `write` 消息的目标窗口。*terminal* 是要改变的终端文件名，例如：`ttyp0`。

option

`-s` 不输出错误消息。

ruptime

`ruptime [`*options*`]`

显示本地网络机器的状态（与 `uptime` 类似）。这个命令不再使用了，因为支持端口监控程序产生了大量不必要的网络流量。

options

`-a` 包括用户，即使他们处于空闲状态已有多个小时。一般不对这种用户计数。

`-l` 按照负载的平均数排列。

ruptime

`-r` 反序排列。

`-t` 按照最高时间排列。

`-u` 按照用户数排列。

rwho

`rwho [`*`option`*`]`

报告在本地网络的所有机器上登记的用户（与 `who` 类似）。

这个命令不再使用，因为支持端口监控程序产生了大量不必要的网络流量。

option

`-a` 列出用户，即使他们处于空闲状态已有多个小时。

shl

`shl`

从一个终端上控制多个 shell（层）。从 `shl` 提示层，可发出以下列出的命令（如果需要，可将它们简写成任意一个惟一的前缀）。*name* 文本字符串名不能超过 8 个字符。参见 `layers`。

`block` *name* `[`*name2* `...]`
将每个层 *name* 合成块输出（同 `stty loblk`）。

`create [`*name*`]`
创建层 *name*（总共不超过 7 个）。

`delete` *name* `[`*name2* `...]`
删除层 *name*。

`help` 或?
提供 `shl` 命令语法。

`layers [-l] [`*name* `...]`
输出关于 layers 的信息。`-l` 提供的显示与 `ps` 类似。

name
使层 *name* 处于当前层。

`quit`
退出 `shl` 命令，关闭所有层。

shl

`resume [`*name*`]`
返回最近的层或层 *name*。

`toggle`
倒回到前一个层。

`unblock` *name* `[`*name2* `...]`
不将每个层 *name* 合成块输出（与 `stty -loblk` 相同）。

sum

`sum [`*option*`]` *file*

计算、输出文件 *file* 的校验和及块数（512 字节）。校验数据传送时有用。参见第二章的 **cksum**。

注意：`/usr/ucb/sum` 报告的大小以千字节为单位，而 `/usr/bin/sum` 报告的大小以 512 字节的块为单位，即使有 `-r` 选项。

option
`-r` 使用另一种校验算法，与 `sum` 的 BSD 版的结果相同。

tabs

`tabs [`*tabspec*`] [`*options*`]`

根据 *tabspec* 设置终端制表位。*tabspec* 的默认值 `-8` 给出了标准 Unix 制表符的设置。可将 *tabspec* 预定义为某些特别语言的制表符集。例如：`a`（IBM 汇编程序）、`c`（COBOL 语言），`f`（FORTRAN 语言）、p（PL/1 语言）、s（SNOBOL 语言）和 u（UNIVAC 汇编程序）。*tabspec* 可以是重复的数字、任意数字或从文件调用的结果。

tabspec
`-n` 每隔 *n* 列置一个制表符(例如：1+*n*、1+2***n* 等等)。

n1, *n2*, . . .
任意以升序排列的值。如果 *n* 前有 `+`，就是增加（即制表符相对于前一个位置）。

`-a` 1, 10, 16, 36, 72。

`-a2` 1, 10, 16, 40, 72。

`-c` 1, 8, 12, 16, 20, 55。

`-c2` 1, 6, 10, 14, 49。

tabs

`-c` 3 1, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 67。

`-f` 1, 7, 11, 15, 19, 23。

`-p` 1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61。

`-s` 1, 10, 55。

`-u` 1, 12, 20, 44。

`--`*`file`*
读文件第一行的制表符。

options

`+m`*`n`* 设置左边空白为 *n*（默认值为 10）。

`-T`*`type`*
设置终端类型 *type*（默认为 $TERM）。

tsort

`/usr/ccs/bin/tsort [`*`file`*`]`

对文件*file*执行拓扑逻辑排序。一般与`lorder`一起使用识别档案库，以便由`ar`或`ld`更有效地处理这些文件。不是太有用。参见**lorder**。

示例
查找所有目标文件的顺序关系，并按`ld`访问的顺序排序：

```
ld -o myprog `lorder *.o | tsort`
```

unpack

unpack *`files`*

解压由`pack`压缩的文件*files*，恢复到原来的形式。参见**pcat**和**pack**。

uucp

uucp [*`options`*] [*`source`*!]*`file`*[*`destination`*!]*`file`*

将文件（或文件组）从源地址拷贝到目的地址。*source*和*destination*可以是远程系统。目的地址文件可以是一个目录。

options

`-c` 不将文件拷贝到假脱机目录下（默认）。

`-C` 将文件拷贝到假脱机目录下以便传输。

uucp

`-d` 当拷贝的目录不存在时，创建目录（默认）。

`-f` 当拷贝的目录不存在时，不创建目录。

`-g`*x* 设置作业的等级（优先级）。*x* 一般是单个字母或数字，`a` 和 `1` 是最高传输优先级。用 `uuglist` 可显示 *x* 的值。

`-j` 输出 uucp 作业号。

`-m` 当拷贝完成后，发邮件给发 uucp 命令的人。

`-n`*user*
当拷贝完成后，发邮件给（通知）用户。

`-r` 排队作业，但不启动传送程序（uucico）。

`-s`*file*
将传输状态发送给文件 *file*（完整的路径名），代替 `-m`。出于安全原因，Solaris 接受但忽略这个选项 。

`-x`*n* 在第 *n*（0~9）层进行调试，更高的层次有更多的输出。

示例

这个 shell 脚本向系统 `orca` 发送一个压缩的文件：

```
$   cat send_it
#! /bin/sh
compress $1
uucp -C -n$2 -m $1.Z orca!/var/spool/uucppublic
uncompress $1
```

使用 `-c`，从假脱机目录里将拷贝传送出去。（通常，uucp 从最初的位置上取文件，所以在调用结束前不能重命名或解压文件。）这个命令在传输结束后会向发送者和接收者发出通知。这有个例子运行：

```
send_it chapter1 bob
```

uuglist

uuglist [*option*]

使用 uux 和 uucp 的 `-g` 选项，列出所有有效的服务等级。服务等级定义了数据传输的优先级，通常用单个字符或字符串表示。

option

`-u` 为当前的用户列出有效的服务等级。

uulog

uulog [*options*]

输出 uucp 或 uuxqt 日志文件信息，日志信息存在 /var/uucp/.Log 目录下（uucico 或 uuxqt 子目录）。参见第二章的 **tail**。

options

-f*sys*
对给出的系统发出 tail -f 命令输出最新动作。

-s*sys*
对给出系统输出所有动作。

-x 检查给出系统的 uuxqt 日志文件（与 -f 或 -s 一起使用）。

-*n* 执行 *n* 行的 tail 命令（与 -f 一起使用）。

uuname

uuname [*options*]

输出 uucp 知道的系统名。

options

-c 输出 cu 知道的系统名（通常相同）。

-l 输出本地系统节点名。

uupick

uupick [*option*]

询问由 uuto 发送给用户的文件状态。

option

-s*system*
只搜索由系统 *system* 发送的文件。

交互响应

a[*dir*]
将从系统 *system* 发送来的文件全部移到指定的目录 dir 下。

d 删除项目。

m[*dir*]
将文件移到目录 *dir* 下。

uupick

p 打印文件。

q 退出 uupick。

* 打印命令汇总。

!*cmd*
执行 shell 命令 *cmd*。

EOF 退出 uupick。

RETURN
移到下一项。

uustat

uustat [*options*]

提供关于 uucp 请求的信息。这个命令也可以取消 uucp 请求。选项 -a、-j、-k、-m、-p、-q 和 -r 不能彼此一起使用。

options

-a 报告所有排队作业。

-c 当与 -t 一起用时，报告排队平均时间而不是平均传输速率。

-d*n* 当与 -t 一起用时，报告过去 *n* 分钟的平均数，而不是小时的平均数。

-j 报告所显示作业的总数（只与 -a 或 -s 一起用）。

-k*n* 取消工作请求 *n*，自己必须拥有它。

-m 报告其他系统的可访问性。

-n 禁止标准输出，但不禁止标准错误。

-p 在活动 UUCP 进程上执行 ps -flp。

-q 报告所有系统排队的作业。

-r*n* 通过向相关的文件发出 touch 命令，重新开始作业 *n*。

-s*system*
报告 *system* 的作业状态。

-S*x* 报告类型为 *x* 的作业的状态：

c 完成的作业。

i 中断的作业。

uustat

q 排队的作业。

r 正在运行的作业。

-ts*system*

报告 *system* 在过去时间的平均传输速率（字节 / 秒）。

-u*user*

报告用户 *user* 的作业状态。

uuto

uuto [*options*] *sourcefiles destination*

将源文件发送到目的地，*destination* 是 *system*!*user* 形式。目的系统的用户可用命令 uupick 提取文件。

options

-m 当拷贝结束时发出邮件。

-p 将文件拷贝到假脱机目录下。

uux

uux [*options*] [[*sys*]!*command*]

从不同系统收集文件，并在指定的机器 *sys* 上执行命令 *command*。uux 识别 uucp 选项 -c、 -C、-g、 -r、 -s 和 -x。

options

- 与 -p 相同（将标准输入传送给命令 *command*）。

-a*user*

在完成时通知用户 *user*（参见 -z）。

-b 当退出状态提示错误时，输出标准输入。

-j 输出 uux 作业号。

-n 如果命令 *command* 失败，则不发送邮件。

-p 将标准输入传送给命令 *command*。

-z 在成功地完成时，通知调用用户。

vacation

vacation [*options*]

UPAS的SVR4版本（参见第二章的**vacation**）。自动向发送者返回邮件消息，通知发送者接收者正在休假。要禁用此功能，键入`mail -F ""`。

options

`-d` 在*logfile*后添加日期（参见`-l`）。

`-F` *user*
当不能把邮件发到*mailfile*时，将其发给用户*user*（参见`-m`）。

`-l` *logfile*
将接收到自动答复的发送者名记入*logfile*（默认是`$HOME/.maillog`）。

`-m` *mailfile*
将接收到的消息保存在*mailfile*中（默认为`$HOME/.mailfile`）。

`-M` *msg_file*
用*msg_file*作为对邮件的自动答复（默认为`/usr/lib/ mail/std_vac_msg`）。

vc

/usr/ccs/bin/vc [*options*] [*keyword=value* ...]

“Version control（版本控制）”。在由标准输入中的参数和`vc`关键字的控制下，从标准输入拷贝文字行到标准输出上。

这个命令与RCS和SCCS完全无关。基本上没用了。

options

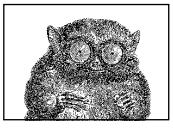
`-a` 替换所有行的控制关键字，包括文本行。

`-c`*k* 用*k*代替:作为控制字符。

`-s` 禁止警告消息。

`-t` 如果在文件的第一个制表符前发现了控制字符，则删除第一个制表符前的所有字符。

whois	whois [*option*] *name* 在Internet目录上搜索由*name*指定的人、登录、联系信息或组织。在*name*前单独或组合使用修饰符！、.或*限制查找（1）人名或用户名、（2）联系信息和（3）组织三者之一。 option -h *host* 在主机*host*上搜索。
write	write *user* [tty] *message* *EOF* 开始或响应与用户*user*的交互式会话。一个write会话由*EOF*终止。如果用户登录到多个终端上，要指定*tty*。参见第二章的**talk**。



参考文献

有许多讲述Unix及相关内容的书籍已经出版，但是将所有的这些书都列出来是不可能的，这样也不是很有帮助。本章我们将展示经典的书籍 —— 这些书籍是真正的Unix魔术师们的架上秘籍（唉，只是其中的一些书已经绝版了，因此，也只有那些Unix魔术师会有这些书☺）。

因为Unix在其历史发展过程中已经影响到了计算机科学的各个方面，所以你不仅能从列出的书籍中了解到操作系统本身，还能了解到其他一些知识。

本章包括如下内容：

- Unix基本描述和程序员指南
- Unix内核
- 使用Unix工具编程
- 编程语言
- TCP/IP组网
- 排版
- Emacs
- 标准
- O'Reilly出版的书籍

Unix 基本描述和程序员指南

1. *The Bell System Technical Journal*, Volume 57 Number 6, Part 2, July-August 1978. AT&T Bell Laboratories, Murray Hill, NJ, USA. ISSN 0005-8580. 专门为 Unix 编写的特殊书籍，由 Unix 系统的初创人员编写。

2. *AT&T Bell Laboratories Technical Journal*, Volume 63 Number 8, Part 2, October 1984. AT&T Bell Laboratories, Murray Hill, NJ, USA. 另一个专门为 Unix 编写的特殊书籍。

   这两卷按照下列名称再版：

3. *UNIX System Readings and Applications*, Volume 1, Prentice-Hall, Englewood Cliffs, NJ, USA, 1987. ISBN 0-13-938532-0.

4. *UNIX System Readings and Applications*, Volume 2, Prentice-Hall, Englewood Cliffs, NJ, USA, 1987. ISBN 0-13-939845-7.

5. *UNIX Time-sharing System: UNIX Programmers Manual*, Seventh Edition, Volumes 1, 2A, 2B. Bell Telephone Laboratories, Inc., January 1979.

   这些是参考手册（卷 1）及讲述具有里程碑意义的第 7 版 Unix 系统的一些论文集，是当前所有商业 Unix 系统的祖先。

   这些书籍由 Holt Rinehart & Winston 进行了再版，但是现在已经绝版很长时间了。但是，可以在线通过贝尔实验室的网址获得，它们使用 `troff` 源代码、PDF 和 PostScript 格式。网址如下：*http://plan9.bell-labs.com/7thEdMan*。

6. *UNIX Research System: Programmer's Manual*, Tenth Edition, Volume 1, AT&T Bell Laboratories, M.D. McIlroy and A.G. Hume editors, Holt Rinehart & Winston, New York, NY, USA, 1990. ISBN 0-03-047532-5.

7. *UNIX Research System: Papers*, Tenth Edition, Volume 2, AT&T Bell Laboratories, M.D. McIlroy and A.G. Hume editors, Holt Rinehart & Winston, New York, NY, USA, 1990. ISBN 0-03-047529-5.

   这些书籍是关于第 10 版 Unix 系统的一些使用指南及论文集。尽管这些系统在贝尔实验室以外的地方很少使用，但是其中的许多观念及其以前的思想都融合到 System V 的各种不同版本中，并且无论在何种情况下，这些使用指南都能使读者有兴趣阅读。

8. *4.4BSD Manuals*, Computing Systems Research Group, University of California

at Berkeley. O'Reilly & Associates, Sebastopol, CA, USA, 1994. ISBN: 1-56592-082-1.

该使用指南用于4.4BSD。已绝版。

9. 你的Unix程序员使用指南。你可以做的一件最重要的事情是从前向后阅读你的使用指南电子文档（注1）。（随着Unix系统的不断发展，做到这一点可能比较难。）如果你的Unix系统厂商提供了使用指南的打印版本，就比较容易阅读。否则，就从第7版使用指南开始阅读，然后根据需要阅读本地的文档。

10. *A Quarter Century of Unix*, Peter H. Salus, Addison Wesley, Reading, MA, USA, 1994. ISBN: 0-201-54777-5.

这是一本令人满意的图书，其中讲述了Unix的历史，从其出现到本书编写时为止。该书读起来就像一本好的小说，只是其内容都是真实的。

11. *The Unix Philosophy*, Mike Gancarz, Digital Equipment Corp, USA, 1996. ISBN: 1-55558-123-4.

12. *Plan 9: The Manuals, The Documents, The System*, AT&T Bell Laboratories, Harcourt Brace and Company, Boston, MA, USA, 1995. ISBN: 0-03-017143-1 for the full set. ISBN: 0-03-01742-3 for just the manuals. *http://plan9.bell-labs.com/plan9 /distrib.html.*

这些书籍记录并提供了“来自贝尔实验室的计划9”的系统和源代码，由贝尔实验室创建Unix系统的人员开发的下一代操作系统。它包括了许多有趣及令人激动的思想，这些书籍带有一张包括了所有源代码的CD-ROM，你也可以只购买使用指南。

Unix内核

真正的Unix魔术师不但知道如何使用Unix系统，还要知道Unix系统的工作原理。

1. *Lions' Commentary on UNIX 6th Edition, with Source Code*, John Lions, Peer- to-Peer Communications, San Jose, CA, USA, 1996. ISBN: 1-57398-013-7. *http://www.peer-to-peer.com/catalog/opsrc/lions.html.*

注1： 一个夏天，在我作为一位合同程序员进行工作的同时，利用午餐时间逐页阅读了Unix的System III（这是很长时间以前的事情了）。我都无法想像当时我在这么短的时间内是如何学习这么多内容的。

该经典著作对第 6 版 Unix 系统内核进行了讲述。（有中文版，机械工业出版社 —— 编注）

2. *The Design of the UNIX Operating System*, Maurice J. Bach, Prentice-Hall, Englewood Cliffs, NJ, USA, 1986. ISBN: 0-13-201799-7.

   这本书非常透彻地讲述了 System V 版本 2 的设计思想，并对 System V 版本 3 中的一些重要功能进行了讨论，比如 STREAMS（流）和文件系统的转换。（有中文版，机械工业出版社 —— 编注）

3. *The Magic Garden Explained: The Internals of Unix System V Release 4: An Open Systems Design*, Berny Goodheart, James Cox, John R. Mashey, Prentice-Hall, Englewood Cliffs, NJ, USA, 1994. ISBN: 0-13-098138-9.

4. *Unix Internals: The New Frontiers*, Uresh Vahalia, Prentice-Hall, Englewood Cliffs, NJ, USA, 1996. ISBN: 0-13-101908-2.

5. *Unix Internals: A Practical Approach*, Steve D. Pate, Addison Wesley, Reading, MA, USA, 1996. ISBN: 0-201-87721-X.

6. *The Design and Implementation of the 4.3BSD UNIX Operating System*, Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels and John S. Quarterman,Addison Wesley, Reading, MA, USA, 1989. ISBN: 0-201-06196-1.

   该书讲述了Unix系统的4.3BSD版本。商业Unix系统中见到的许多重要功能首先是在 BSD Unix 系统中创建的，比如长文件名、作业控制及网络等。

7. *The Design and Implementation of the 4.4 BSD Operating System*, Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, John S. Quarterman, Addison Wesley Longman, Reading, MA, USA, 1996. ISBN 0-201-54979-4.

   该书是以前版本的更新版，主要讲述来自UCB的最后的Unix系统版本4.4BSD。引用发行人的话，“ 该书详细讲述了进程及内存管理方面的主要改变，描述了新的可扩展及可堆叠的文件系统接口，包括了在新的网络文件系统中至关重要的一章，以及在网络和进程间通信方面的一些更新信息。”

使用 Unix 工具编程

Brian Kernighan编写的任何书籍都值得认真阅读，通常应读上好几遍。最前面的两本书讲述了Unix“ 工具箱 ”编程的方法学，将帮助你学会如何使用 Unix 思考问题。

第三本书更明确地讲述了Unix系统的关键问题，第四和第五本书讲述了一般的编程问题，也是非常有价值的。

1. *Software Tools*, Brian W. Kernighan and P. J. Plauger, Addison Wesley, Reading, MA, USA, 1976. ISBN: 0-201-03669-X.

   是讲述Unix的`grep`、`sort`、`ed`和其他工具的设计及编程代码的一本非常优秀的图书（注 2），程序使用的语言是RATFOR（Rational FORTRAN），一个用于FORTRAN的预处理程序，并使用与C语言相似的控制结构。

2. *Software Tools in Pascal*, Brian W. Kernighan and P. J. Plauger, Addison Wesley, Reading, MA, USA, 1981. ISBN: 0-201-10342-7.

   将以前书籍转换成使用Pascal语言的书籍。仍值得阅读。Pascal提供了FORTRAN无法实现的许多功能。

3. *The Unix Programming Environment*, Brian W. Kernighan and Rob Pike, Prentice-Hall, Englewood Cliffs, NJ, USA, 1984. ISBN:0-13-937699-2 (hardcover), 0-13-937681-X (paperback).

   该书重点讲述了在特定环境中使用Unix的各种工具。特别是，该书增加了有关shell、`awk`及`lex`和`yacc`用法等方面的一些重要材料。参见 *http://cm.bell-labs.com/cm/cs/upe*.。

4. *The Elements of Programming Style*, Second Edition. Brian W. Kernighan and P. J. Plauger, McGraw-Hill, New York, NY, USA, 1978. ISBN: 0-07-034207-5.

   该书是Strunk & White编写的著名的 *The Elements of Style* 之后的又一本重要著作，该书主要讲述了可以应用到任何环境中的良好的编程方法。

5. *The Practice of Programming*, Brian W. Kernighan and Rob Pike, Addison Wesley Longman, Reading, MA, USA, 1999. ISBN: 0-201-61586-X.

   与前面的书籍相似，不过该书更注重讲述技术，参见 *http://cm.bell-labs.com/cm /cs/tpop*。（有中文版，机械工业出版社 —— 编注）

6. *Writing Efficient Programs*, Jon Louis Bentley, Prentice-Hall, Englewood Cliffs, NJ, USA, 1982. ISBN: 0-13-970251-2 (hardcover), 0-13-970244-X (paperback).

   尽管该书与Unix无关，但是这是一本适用于任何希望提高编程效率人员的优秀书籍。

注 2： 该书永远改变了我的生活。

7. *Programming Pearls*, Second Edition. Jon Louis Bentley, Addison Wesley, Reading, MA, USA, 2000. ISBN: 0-201-65788-0.

8. *More Programming Pearls: Confessions of a Coder*, Jon Louis Bentley, Addison Wesley, Reading, MA, USA, 1988. ISBN: 0-201-11889-0.

   这是两本优秀的书籍，引用 Nelson H. F. Beebe 的话为：“优化了 Unix 系统的智能组件，提供了许多关于语言片段、算法设计及其他内容的精彩小例子。”所有认真的程序员都应该拥有这些书籍。

9. *Advanced Programming in the Unix Environment*, W. Richard Stevens, Addison Wesley, Reading, MA, USA, 1992. ISBN: 0-201-56317-7.

   一本比较厚但是非常优秀的图书，主要讲述了在现代Unix系统中如何使用丰富的系统调用。（有中文版，机械工业出版社 —— 编注）

编程语言

大量重要的编程语言首先在Unix系统下进行了开发。请再一次关注Brian Kernighan编写的书籍。

1. *The C Programming Language*, Brian W. Kernighan and Dennis M. Ritchie, Prentice-Hall, Englewood Cliffs, NJ, USA, 1978. ISBN: 0-13-110163-3.

   C 语言的最初经典著作，Dennis Ritchie 发明了 C 语言，并且是 Unix 的两个创始人之一。该版本已经不再发行。

2. *The C Programming Language*, Second Edition. Brian W. Kernighan and Dennis M. Ritchie, Prentice-Hall, Englewood Cliffs, NJ, USA, 1988. ISBN: 0-13-110362-8.

   这是对原来 ANSI C 语言一书的修订版，它保持并增强了第一版的质量，参见 *http://cm.bell-labs.com/cm/cs/cbook*。（国内有影印版和中文版 —— 编注）

3. *C: A Reference Manual*, Fourth Edition, Samuel P. Harbison and Guy L. Steele, Prentice-Hall, Englewood Cliffs, NJ, USA, 1994. ISBN: 0-13-326224-3.

   该书深入探讨了我们需要知道的一些细节问题。

4. *The C++ Programming Language*, Third Edition, Bjarne Stroustrup, Addison Wesley, Reading, MA, USA, 1997. ISBN: 0-201-88954-4.

由C++语言的发明者及ANSI C++委员会主席规定的C++语言的定义语句，参见 *http://www.awl.com/cseng/titles/0-201-88954-4/*。（国内有影印版和中文版——编注）

5. *C++ Primer*, Third Edition, Stanley B. Lippman and Josée Lajoie. Addison Wesley Longman, Reading, MA, USA, 1998. ISBN: 0-201-82470-1.

   该书非常详尽地介绍了C++语言。参见*http://www.awl.com/cseng/ titles/0-201-82470-1/*。（中文版已由中国电力出版社出版 —— 编注）

6. *The Java Programming Language*, Ken Arnold and James Gosling. Addison Wesley, Reading, MA, USA, 1997. ISBN: 0-201-31006-6.

   该书主要用于学习Java，由Java语言的两名设计者编写。

7. *The Java Language Specification*, James Gosling, Bill Joy, Guy L. Steele Jr. Addison Wesley, Reading, MA, USA, 1996. ISBN: 0-201-63451-1.

8. *The AWK Programming Language*, Alfred V. Aho and Brian W. Kernighan and Peter J. Weinberger, Addison Wesley, Reading, MA, USA, 1987. ISBN:0-201-07981-X.

   awk编程语言的原始定义，非常有价值。参见 *http://cm.bell-labs.com/cm/cs/awkbook*。

9. *Effective AWK Programming*, Arnold Robbins, Specialized Systems Consultants, Seattle, WA, USA, 1997. ISBN: 1-57831-000-8.

   该书提供了很多awk的向导语句，讲述了awk的POSIX标准。也可以作为用户学习gawk的使用指南，参见 *http://www.ssc.com/ssc/eap/*。

10. *Tcl and the Tk Toolkit*, John K. Ousterhout. Addison Wesley, Reading, MA, USA, 1994. ISBN: 0-201-63337-X.

11. *Practical Programming in Tcl & Tk*, Brent B. Welch. Prentice-Hall, Englewood Cliffs, NJ, USA, 1997. ISBN: 0-13-616830-2.（新版本的中文版《Tcl/TK编程权威指南》已由中国电力出版社出版 —— 编注）

12. *Effective Tcl/Tk Programming: Writing Better Programs in Tcl and Tk*, Mark Harrison and Michael J. McLennan. Addison Wesley, Reading, MA, USA, 1997.ISBN: 0-201-63474-0.

13. *The New Kornshell Command and Programming Language*, Morris I. Bolsky and David G. Korn, Prentice-Hall, Englewood Cliffs, NJ, USA, 1995. ISBN:0-13-182700-6.

关于 Korn shell 并由其作者撰写的权威内容。

14. *Hands-On KornShell 93 Programming*, Barry Rosenberg, Addison Wesley Longman, Reading, MA, USA, 1998. ISBN: 0-201-31018-X.

15. *CompilersPrinciples, Techniques*, and Tools, Alfred V. Aho and Ravi Sethi and Jeffrey D. Ullman, Addison Wesley Longman, Reading, MA, USA, 1986.ISBN: 0-201-10088-6.

    这是编译器构造方面著名的“龙书”，该书讲述了 `lex` 和 `yacc` 操作后面的许多理论。（国内有中文版和影印版 —— 编注）

TCP/IP 组网

由 Comer 编写的这些书籍非常优秀，它们对 TCP/IP 协议进行了权威性的描述。由 Stevens 编写的书籍也是非常值得推崇的。

1. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Third Edition, Douglas E. Comer, Prentice-Hall, Englewood Cliffs, NJ, USA, 1995.ISBN: 0-13-216987-8.

2. *Internetworking With TCP/IP: Design*, Implementation, and Internals, Third Edition, Douglas E. Comer and David L. Stevens, Prentice-Hall, Englewood Cliffs, NJ, USA, 1998. ISBN: 0-13-973843-6.

3. *Internetworking With TCP/IP: Client-Server Programming and Applications: BSD Socket Version*, Second Edition, Douglas E. Comer and David L. Stevens, Prentice-Hall, Englewood Cliffs, NJ, USA, 1996. ISBN: 0-13-260969-X.

4. *Internetworking With TCP/IP: Client-Server Programming and Applications: AT&T TLI Version*, Douglas E. Comer and David L. Stevens, Prentice-Hall,Englewood Cliffs, NJ, USA, 1993. ISBN: 0-13-474230-3.

5. *TCP/IP Illustrated, Volume 1: The Protocols*, W. Richard Stevens, Addison Wesley Longman, Reading, MA, USA, 1994. ISBN: 0-201-63346-9.

6. *TCP/IP Illustrated, Volume 2: The Implementation*, W. Richard Stevens and Gary R. Wright, Addison Wesley Longman, Reading, MA, USA, 1995. ISBN:0-201-63354-X.

7. *TCP/IP Illustrated, Volume 3: TCP for Transactions*, HTTP, NNTP, and the Unix

Domain Protocols, W. Richard Stevens, Addison Wesley Longman, Reading,MA, USA, 1996. ISBN: 0-201-63495-3.

8. *Unix Network Programming, Volume 1: Networking APIs: Sockets and XTI*, W. Richard Stevens, Prentice-Hall, Englewood Cliffs, NJ, USA, 1997. ISBN:0-13-490012-X.

9. *Unix Network Programming, Volume 2: Interprocess Communications*, W.Richard Stevens, Prentice-Hall, Englewood Cliffs, NJ, USA, 1998. ISBN:0-13-081081-9.
   这一卷和以前的一些卷是对第一版的修订, 而第一版已经成为多年来关于Unix网络编程的一流书籍。(以上九本书国内均有中文版和影印版 —— 编注)

10. *Unix System V Network Programming*, Steven A. Rago, Addison Wesley Longman, Reading, MA, USA, 1993. ISBN: 0-201-56318-5.

排版

1. *Document Formatting and Typesetting on the Unix System*, Second Edition, Narain Gehani, Silicon Press, Summit, NJ, USA, 1987. ISBN: 0-13-938325-5.

2. *Typesetting Tables on the Unix System*, Henry McGilton and Mary McNabb, Trilithon Press, Los Altos, CA, USA, 1990. ISBN: 0-9626289-0-5.
   该书讲述了你可能希望知道的任何事情, 并讲述了使用`tbl`来格式化表格方面的内容。

Emacs

1. *GNU Emacs Manual*, for Version 20.1, Thirteenth Edition, The Free Software Foundation, Cambridge, MA, USA, 1998. ISBN: 1882114 06X.

2. *GNU Emacs Lisp Reference Manual*, for Emacs Version 20, Edition 2.4, The Free Software Foundation, Cambridge, MA, USA, 1998. ISBN: 1882114 728.

3. *Writing GNU Emacs Extensions*, Bob Glickstein, O'Reilly & Associates, Sebastopol, CA, USA, 1997. ISBN: 1-56592-261-1.

参见“ O'Reilly 出版的书籍 ”部分的 *Learning GNU Emacs* 一书。

标准

有许多官方的标准规定了在Unix系统和类似Unix的系统之间移植应用程序的做法。最前面的两项是标准自身 ;接下来的一项是使用第一种标准的学习指南 ;最后两项规定了 C 和 C++ 编程语言的通用标准。

1. *ISO/IEC Standard 9945-1: 1996 [IEEE/ANSI Std 1003.1, 1996 Edition] Information Technology —Portable Operating System Interface (POSIX) —Part 1: System Application: Program Interface (API) [C Language]*. IEEE, 1996. ISBN: 1-55937-573-6.

   该版图书增加了实时应用程序（1003.1b-1993,1003.1i-1995）和线程（1003.1c-1995）等扩充内容。也可以订阅其电子版，参见 *http://www.standards.ieee.org*。

   该书描述了 C 和 C++ 程序员熟悉的操作系统接口。

2. *ISO/IEC Standard 9945-2: 1993 [IEEE/ANSI Std 1003.2-1992 & IEEE/ANSI 1003.2a-1992] Information Technology —Portable Operating System Interface (POSIX) —Part 2: Shell and Utilities* IEEE, 1996. ISBN: 1-55937-406-3.

   该标准包括及增加了 1003.2d-1994。

   该标准与本书读者的关系非常大, 因为它从shell和实用程序的角度描述了操作系统。

3. *Posix Programmer's Guide: Writing Portable Unix Programs*, Donald A.Lewine. O'Reilly & Associates, Sebastopol, CA, USA, 1991. ISBN:0-937175-73-0.

4. X3 Secretariat: *Standard —The C Language*. X3J11/90-013. ISO Standard ISO/IEC 9899. Computer and Business Equipment Manufacturers Association.Washington DC, USA, 1990.

5. X3 Secretariat: *International Standard — The C++ Language*. X3J16-14882. Information Technology Council (NSITC). Washington DC, USA, 1998.

O'Reilly 出版的书籍

下面是本书引用过的 O'Reilly & Associates 书籍的一个列表，当然，还有其他的 O'Reilly 书籍与 Unix 有关，参见 *http://www.oreilly.com/catalog*。

1. *Advanced Perl Programming*, Sriram Srinivasan, O'Reilly & Associates,

Sebastopol, CA, USA, 1997. ISBN: 1-56592-220-4.（本书中文版《高级 Perl 编程》已由中国电力出版社出版 —— 编注）

2. *Applying RCS and SCCS*. Don Bolinger and Tan Bronson, O'Reilly & Associates, Sebastopol, CA, USA, 1995. ISBN: 1-56592-117-8.

3. *Checking C Programs with lint*. Ian F. Darwin, O'Reilly & Associates, Sebastopol, CA, USA, 1988. ISBN: 0-937175-30-7.

4. *Learning GNU Emacs*, Second Edition, Debra Cameron, Bill Rosenblatt, and Eric Raymond, O'Reilly & Associates, Sebastopol, CA, USA, 1996. ISBN:1-56592-152-6.

5. *Learning Perl*, Second Edition, Randal L. Schwartz and Tom Christiansen, O'Reilly & Associates, Sebastopol, CA, USA, 1997. ISBN: 1-56592-284-0.（本书中文版《Perl 语言入门》已由中国电力出版社出版 —— 编注）

6. *Learning the Korn Shell*, Bill Rosenblatt, O'Reilly & Associates, Sebastopol, CA,USA, 1993. ISBN: 1-56592-054-6.

7. *Learning the Unix Operating System*, Fourth Edition, Jerry Peek, Grace Todino, and John Strang, O'Reilly & Associates, Sebastopol, CA, USA, 1997. ISBN: 1-56592-390-1.（本书中文版《Unix操作系统》已由中国电力出版社出版 —— 编注）

8. *Learning the vi Editor*, Sixth Edition, Linda Lamb and Arnold Robbins, O'Reilly & Associates, Sebastopol, CA, USA, 1998. ISBN: 1-56592-426-6.

9. *lex & yacc*, Second Edition, John Levine, Tony Mason, and Doug Brown,O'Reilly & Associates, Sebastopol, CA, USA, 1992. ISBN: 1-56592-000-7.

10. *Managing Projects with make*, Second Edition, Andrew Oram and Steve Talbott, O'Reilly & Associates, Sebastopol, CA, USA, 1991. ISBN: 0-937175-90-0.

11. *Mastering Regular Expressions*, Jeffrey E. F. Friedl, O'Reilly & Associates, Sebastopol, CA, USA, 1997. ISBN: 1-56592-257-3.

12. *PGP: Pretty Good Privacy*, Simson Garfinkel, O'Reilly & Associates, Sebastopol, CA, USA, 1994. ISBN: 1-56592-098-8.

13. *Programming Perl*, Second Edition, Larry Wall, Tom Christiansen, and Randal L. Schwartz, O'Reilly & Associates, Sebastopol, CA, USA, 1996. ISBN: 1-56592-149-6.（本书中文版《Perl 语言编程》已由中国电力出版社出版 —— 编注）

14. *sed & awk*, Second Edition, Dale Dougherty and Arnold Robbins, O'Reilly & Associates, Sebastopol, CA, USA, 1997. ISBN: 1-56592-225-5.

15. *termcap & terminfo*. Third Edition, John Strang, Linda Mui, and Tim O'Reilly, O'Reilly & Associates, Sebastopol, CA, USA, 1988. ISBN: 0-937175-22-6.

16. *Using csh & tcsh*, Paul DuBois, O'Reilly & Associates, Sebastopol, CA, USA, 1995. ISBN: 1-56592-132-1.

索引

符号

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

词汇表

associative array
　　关联数组

core image
　　核心映像

CRC（Cyclic Redundancy Check）
　　循环冗余检验

demangle
　　反改编

environment variable
　　环境变量

EUC（Extended Unix Code）
　　扩展的 Unix 代码

GUI（Graphical User Interface）
　　图形用户界面

hard link
　　硬链接

JNI（Java Native Interface）
　　Java 本地接口

macro definition
　　宏定义

private key
　　私钥

public key
　　公钥

RCS（Revision Control System）
　　版本修订控制系统

SCCS（Source Code Control System）
　　源代码控制系统

token
　　标记