

学习

vi



编辑器

O'REILLY®

机械工业出版社
China Machine Press



Linda Lamb & Arnold Robbins 著

莫蓉蓉 刘传昌 译

学习Vi 编辑器



对许多用户来说，在UNIX环境下工作就意味着使用*vi*，这是一个在大部分UNIX系统上可以使用的全屏幕文本编辑器。然而，即使了解*vi*的人通常也只使用了它的一小部分功能。

这本畅销书的最新修改版是使用*vi*进行文本编辑的完全手册。新主题涵盖了4种*vi*克隆版本：*nvi*、*elvis*、*vim*和*vile*，并且介绍了它们对*vi*的扩展功能，如多窗口编辑、GUI接口、扩展的正则表达式以及针对程序员的新增功能。新加的附录还描述了*vi*在UNIX和Internet文化中的地位。

这本书将带领读者快速地学习基本的编辑、光标移动以及全局查找与替换操作，然后是更灵活的*vi*功能，以及如何在*vi*中使用功能强大的*ex*行编辑器来提高编辑技巧。为了便于读者参考，本书第六版还在每个相应章节的最后增加了命令总结。

涵盖的主题有：

- 编辑基础
- 快速移动
- 基本编辑命令的扩展
- *ex*的更强大功能
- 全局查找与替换
- 定制*vi*和*ex*
- 命令的简化操作
- 介绍*vi*克隆版本的扩展
- *nvi*、*elvis*、*vim*和*vile*编辑器
- *vi*和*ex*命令总结
- *vi*和Internet

正如一个用户告诉我们的，“本书是我使用*vi*而不使用*emacs*的惟一原因。”

ISBN 7-111-10932-5

9 787111 109327 >



O'Reilly & Associates, Inc. 授权机械工业出版社出版

ISBN 7-111-10932-5

定价：43.00元

学习 vi 编辑器

第六版

Linda Lamb & Arnold Robbins 著

莫蓉蓉 刘传昌 译



O'REILLY®

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

O'Reilly & Associates, Inc. 授权机械工业出版社出版

机械工业出版社



0352180

~\$4

图书在版编目 (CIP) 数据

学习 vi 编辑器 (第六版) / (美) 拉姆 (Lamb, L.), (美) 罗宾斯 (Robbins, A.) 编著; 莫蓉蓉等译. - 北京: 机械工业出版社, 2003.1

书名原文: Learning the vi Editor, Sixth Edition

ISBN 7-111-10932-5

I. 学 ... II. ①拉 ... ②罗 ... ③莫 ... III. UNIX 操作系统—程序设计 IV. TP316.81

中国版本图书馆 CIP 数据核字 (2002) 第 070406 号

北京市版权局著作权合同登记

图字: 01-2002-1830 号

©1998 by O'Reilly & Associates, Inc.

Simplified Chinese Edition, jointly published by O'Reilly & Associates, Inc. and China Machine Press, 2003. Authorized translation of the English edition, 1998 O'Reilly & Associates, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly & Associates, Inc. 出版 1998。

简体中文版由机械工业出版社出版 2003。英文原版的翻译得到 O'Reilly & Associates, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly & Associates, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

书 名 / 学习 vi 编辑器 (第六版)

书 号 / ISBN 7-111-10932-5

责任编辑 / 贾梅, 徐申

封面设计 / Edie Freedman, 张健

出版发行 / 机械工业出版社

地 址 / 北京市西城区百万庄大街 22 号 (邮政编码 100037)

经 销 / 新华书店北京发行所发行

印 刷 / 北京牛山世兴印刷厂

开 本 / 787 毫米 × 1092 毫米 16 开本 22.75 印张 332 千字

版 次 / 2003 年 1 月第 1 版 2003 年 1 月第 1 次印刷

印 数 / 0001-4000 册

定 价 / 43.00 元 (册)

(凡购本书, 如有倒页、脱页、缺页, 由本社发行部调换)

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 公司授权机械工业出版社,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。



目录

前言	1
 第一部分 基本 vi 和高级 vi	
第一章 vi 文本编辑器	11
打开和关闭文件	13
不保存编辑而直接退出	18
 第二章 简单编辑	21
vi 命令	22
移动光标	23
简单编辑	27
插入文本的常用方法	42
使用 J 合并两行	44
回顾基本的 vi 命令	45

第三章 快速移动	48
按屏幕移动	49
按文本块移动	52
按搜索移动	53
按行号移动	57
回顾 vi 移动命令	59
第四章 基本编辑命令的扩展	61
更多的命令组合	61
启动 vi 时的选项	62
利用缓冲区	66
标记自己的位置	68
其他的高级编辑	69
回顾 vi 的缓冲区和标记命令	69
第五章 介绍 ex 编辑器	71
ex 命令	72
使用 ex 进行编辑	74
保存和退出文件	81
把文件复制到另一个文件中	83
编辑多个文件	84
第六章 全局替换	90
确认替换	91
上下文相关替换	93
模式匹配规则	94
模式匹配举例	102
总结模式匹配	111

第七章 高级编辑	118
定制 vi	119
执行 UNIX 命令	123
保存命令	128
使用 ex 脚本	142
编辑程序源代码	150

第二部分 扩展功能和克隆版本

第八章 vi 克隆版本的功能总结	159
vi 的各种克隆版本	159
多窗口编辑	161
GUI 接口	162
扩展的正则表达式	163
增强的标志	164
改进的功能	171
编程辅助	176
编辑器比较小结	178
后面内容预览	179

第九章 nvi — 新 vi	181
作者和历史	181
重要的命令行参数	182
联机帮助和其他的文档	183
初始化	184
多窗口编辑	185
GUI 接口	186
扩展的正则表达式	187
改进的编辑功能	188

编程辅助	191
令人感兴趣的功能	191
源代码和支持的操作系统.....	192
第十章 elvis	194
作者和历史	194
重要的命令行参数	195
联机帮助和其他的文档	196
初始化	196
多窗口编辑.....	198
GUI 接口	201
扩展的正则表达式	207
改进的编辑功能	208
编程辅助	213
令人感兴趣的功能	216
elvis 的未来	221
源代码和支持的操作系统	223
第十一章 vim — 改进的 vi	225
作者和历史	225
重要的命令行参数	226
联机帮助和其他的文档	228
初始化	228
多窗口编辑.....	231
GUI 接口	236
扩展的正则表达式	239
改进的编辑功能	241
编程辅助	251
令人感兴趣的功能	255
源代码和支持的操作系统	263

第十二章 vile—类 Emacs 的 vi.....	266
作者和历史	266
重要的命令行参数	267
联机帮助和其他的文档	268
初始化	270
多窗口编辑	270
GUI 接口	273
扩展的正则表达式	280
改进的编辑功能	282
编程辅助	289
令人感兴趣的功能	292
源代码和支持的操作系统.....	298

第三部分 附录

附录一 快速参考	301
附录二 ex 命令	307
附录三 设置选项	317
附录四 问题列表	336
附录五 vi 和 Internet	341

前言

文本编辑是所有计算机系统上最普通的应用之一，而*vi*又是用户系统上最有用的标准文本编辑器之一。你可以使用*vi*创建新文件或对任何现有的UNIX文本文件进行编辑。

本书的内容

本书由12章和5个附录组成，共分为三个部分。第一部分“基本*vi*和高级*vi*”可以使你迅速开始使用*vi*，并介绍了可以更有效地使用*vi*的高级技巧。

前两章“*vi*文本编辑器”和“简单编辑”介绍了一些简单的*vi*命令，你应该练习这些命令，直到掌握它们。由于已学到了一些基本的编辑方法，你甚至可以在第二章“简单编辑”的结尾停止对*vi*的学习。

但是*vi*所能做的却不只是基本的字处理，各种各样的命令和选项可以简化大量单调的编辑工作。第三章“快速移动”和第四章“基本编辑命令的扩展”集中介绍完成这些工作更简单的方法。在第一次阅读时，至少会对*vi*的功能以及为特殊应用选择适当的命令有个概念。以后，可以再返回这些章节进行进一步的学习。

第五章“介绍 ex 编辑器”、第六章“全局替换”和第七章“高级编辑”提供了许多帮助用户把更多的编辑任务转移给计算机的工具。这些章节介绍了 vi 底层的 ex 行编辑器，并展示了如何从 vi 中调用 ex 命令。

第二部分“扩展功能和克隆版本”介绍了那些在许多或所有 vi 克隆版本中共同使用的“标准” vi 的扩展功能。

第八章“vi 克隆版本的功能总结”介绍了多窗口编辑、GUI 接口、扩展的正则表达式、使编辑更加容易的工具和一些其他功能。

第九章“nvi——新 vi”到第十二章“vile——类 Emacs 的 vi”介绍了 vi 的各种克隆版本——nvi、elvis、vim 和 vile，展示了如何使用 vi 的扩展功能，并对每个编辑器特有的功能进行了讨论。

第三部分“附录”提供了许多有用的参考资料。

附录一“快速参考”列出了所有的 vi 和 ex 命令，并按功能对它们进行了分类。

附录二“ex 命令”是按字母表顺序排序的 ex 命令列表。

附录三“设置选项”列出了所有的 set 命令选项。

附录四“问题列表”对本书所提供的问题列表进行了总结。

附录五“vi 和 Internet”介绍了 vi 在众多 UNIX 和 Internet 文化中的位置。

内容组织

本书的基本出发点是使新用户对 vi 的丰富内容有个良好的概念。学习一种新编辑器，尤其是具有 vi 这么多选项的编辑器，可能会是一项无法完成的任务。我们尽量用易读、符合逻辑的方式来介绍 vi 的基本概念和命令。

vi 命令的介绍

 像左边那样的键盘按钮的图片表示主要对这个特殊的键盘命令或相关的命令进行介绍。你将在相关章节中找到对该主要概念的简要介绍，然后我们列出在每种情况下可以使用的合适命令，以及该命令的描述和使用它的正确语法。

约定

在语法描述和实例中，真正输入的内容将使用 **Courier** 字体显示，所有的命令名也是如此。变量（不能对它按照字面输入，只能在输入命令时用一个实际值替代它）将使用 **Courier** 斜体字体显示。中括号表示该变量是可选的。例如，在下面的语法规行中：

```
vi [filename]
```

filename 将由一个实际的文件名取代。中括号表示不指定文件名也可以调用该 vi 命令，中括号本身不用输入。

某些例子对在 UNIX shell 提示符下输入的命令的结果进行了显示。在这些例子中，真正输入的内容将使用 **Courier** 粗体字体进行显示，目的是把它与系统的响应相区别。例如：

```
$ ls  
ch01.sgm ch02.sgm ch03.sgm ch04.sgm
```

在这些例子中，斜体表示注释，不用输入。另外，斜体还用来强调专用术语和表示文件名。

按键

特殊的按键将显示在方框内，例如：

iWith a **ESC**

在本书中，还会找到 vi 命令以及它们的结果：

击键 结果

ZZ

"practice" [New file] 6 lines, 320 characters

给出写保存命令 ZZ，文件将保存为普通的 UNIX 文件。

在上面的例子中，ZZ 命令将显示在左边的列中，在右边的窗口中显示该命令结果的屏幕上的一行（或多行）。光标位置将用下划线表示。在这个例子中，由于 ZZ 命令进行保存并写文件，因此当写文件时只能看到状态行，而没有光标位置的指示。窗口的下面是对该命令及其结果的说明。

有时 vi 命令要通过同时按下 **CTRL** 键和另一个键进行调用。在本书中，这种组合键将写在方框内（例如，**CTRL-G**）。在一些例子中，写成在键名前加上插入符号 (^)。例如，^G 表示在按下 G 键时要按下 **CTRL** 键。

问题列表

问题列表出现在可能会遇到一些麻烦的章节中。当真正遇到问题时，可以浏览这些列表，然后再回到遇到问题的地方。为了便于参考，所有的问题列表都收集在附录四中。

在开始学习之前需要知道的

本书假定你已阅读过《学习 UNIX 操作系统》或其他一些关于 UNIX 的介绍。应该已经知道如何：

- 登录和退出
- 输入 UNIX 命令
- 改变目录
- 列出目录中的文件

- 创建、复制和删除文件

熟悉 *grep* (全局搜索程序) 和通配符字符也是很有帮助的。

虽然 *vi* 几乎可以运行在任何终端上，但是 *vi* 必须知道用户正在使用的终端的类型。通常将终端类型设置在 *.profile* 或 *.login* 文件中。如果不能确保是否正确地定义了终端类型，请询问系统管理员。这将避免在开始实践 *vi* 时可能遇到的一些麻烦。

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

要询问关于本书的技术问题或相关解释，请发送电子邮件：

bookquestions@oreilly.com

最后，你可以在 WWW 上找到我们：

<http://www.oreilly.com>

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

关于前面的版本

在本书的第五版中，首次对 *ex* 编辑器的命令进行了较为全面的讨论。在第五、六和七章中，通过增加更多的实例对 *ex* 和 *vi* 的复杂功能进行了阐述，其中的主题包括正则表达式语法、全局替换、*.exrc* 文件、单词缩写、键盘映射和编辑脚本。其中的一些例子来自《UNIX World》杂志中的论文。Walter Zintz 所著的关于 *vi* 的指南（注 1）介绍了许多新内容，也提供了许多巧妙的例子，我们已在本书中用它们来阐明各种功能。Ray Swartz 在他的专栏中也有一个有用的技巧（注 2）。

第六版的序言

《学习 vi 编辑器》第六版把本书带入 20 世纪 90 年代后期，除了“最初”版的 *vi* 成为每个 UNIX 系统的标准外，现在又有许多免费可用的“克隆”版本或相似的编辑器。它们中的许多都对最初的 *vi* 进行了完善，因此我们可以说现在有个 *vi* 编辑器“家庭”，而本书的目标就是教你学习如何使用它们。

新功能

对于本版本，下列功能是新加的：

- 对基本文本进行了许多修正和补充。

注 1： 1990 年 4 月《UNIX World》中“vi Tips for Power Users”和 1990 年 6 月《UNIX World》中的“Using vi to Automate Complex Edits”，这两篇论文都是 Walter Zintz 所编写的。

注 2： 1990 年 4 月《UNIX World》中的“Answers to UNIX”。

- 对于相关的章节，都在其结尾添加了命令总结。
- 新章节对每个*vi* 克隆版本、两个或多个克隆版本所共有的功能和/或扩展以及多窗口编辑进行了介绍。
- 每个*vi* 克隆版本的章节都对该编辑器的历史和目标、其独有功能以及获得它的地方进行了分段介绍。
- 新附录介绍了*vi* 在众多 UNIX 和 Internet 文化中的位置。

版本

使用下列编辑器对 *vi* 的各种功能进行了测试：

- Solaris 2.6 版的 *vi* 代表 UNIX *vi* 的“参考”版
- Keith Bostic 的 *nvi* 1.79 版
- Steve Kirkendall 的 *elvis* 2.0 版
- Bram Moolenaar 的 *vim* 5.0 版和 5.1 版
- Kevin Buettner、Tom Dickey 和 Paul Fox 的 *vile* 7.4 版和 8.0 版

感谢

首先要感谢我的妻子 Miriam，在我写这本书时是她照看孩子们，尤其是在进餐之前的“迷人时段”。我感谢她牺牲的大量安静时间和冰淇淋。

乔治亚计算技术学院的 Paul Manno 在解决我的打印软件方面提供了很大的帮助。O'Reilly & Associates 的 Len Muellner 和 Erik Ray 帮助我制做 SGML 软件，Jerry Peek 提供用于 SGML 的 *vi* 宏也给了我很大的帮助。

虽然在准备新内容和修正内容期间使用了所有的编辑器，但是大部分编辑操作都使用 GNU-Linux (Redhat 4.2) 下的 *vim* 4.5 和 5.0 版本进行。

感谢 Keith Bostic、Steve Kirkendll、Bram Moolenaar、Paul Fox、Tom Dickey 和 Kevin Buettner，是他们评审了该书。Steve Kirkendll、Bram Moolenaar、Paul Fox、Tom Dickey 和 Kevin Buettner 还提供了本书第八章到第十二章的重要内容。

没有电力公司产生的电，使用计算机进行各种处理是不可能的。但是当电存在时，你又不能停下来对它进行思考。因此很多时候在写书时，没有编辑，没有任何事发生，但是当编辑在那里做她的工作时，又很容易忘记她。O'Reilly 的 Gigi Estabrook 是个真正讨人喜欢的人，与她一起工作是种快乐，我感谢她已经为我做的和继续为我做的每件事。

最后，多谢 O'Reilly & Associates 的制作组。

Arnold Robbins

Ra'anana, ISRAEL

1998 年 6 月

第一部分

基本 vi 和高级 vi

第一部分可以使你快速开始使用*vi*, 并介绍了许多可以最有效地使用它的高级技巧。本部分包括以下几章:

- 第一章, vi 文本编辑器
- 第二章, 简单编辑
- 第三章, 快速移动
- 第四章, 基本编辑命令的扩展
- 第五章, 介绍 ex 编辑器
- 第六章, 全局替换
- 第七章, 高级编辑





本章内容：

- 打开和关闭文件
- 不保存编辑而直接退出

第一章

vi 文本编辑器

UNIX有许多可以处理文本文件内容的编辑器，不用考虑这些文件的内容是数据、源代码、还是语句。其中有行编辑器，如 *ed* 和 *ex*，它们只把文件的一行显示在屏幕上；还有全屏幕编辑器，如 *vi* 和 *emacs*，它们把文件的一部分显示在用户的终端屏幕上。基于 X Window 系统的文本编辑器也得到了广泛的使用，并且它们正变得越来越流行。GNU 的 *emacs* 及其派生的 *xemacs* 都提供了多个 X 窗口，另一个引人注目的替代工具是贝尔实验室的 *sam* 编辑器。本书第二部分介绍的 *vi* 克隆版本中的大部分都提供了基于 X 的接口。

vi 是用户系统上最为有用的标准文本编辑器 [*vi* 是可视编辑器 (*visual editor*) 的缩写，读做“vee-eye”]。与 *emacs* 不同，在大部分 UNIX 系统中，*vi* 使用的形式几乎完全相同，从而提供了一种文本编辑 *lingua franca* (注 1)。虽然在 *ed* 和 *ex* 编辑器中也存在这种情况，但是全屏幕编辑器使用起来通常比行编辑器要容易。使用全屏幕编辑器可以进行翻页、移动光标、删除行、插入字符等操作，并且在进行这些操作的同时还能看到编辑结果。由于全屏幕编辑器允许用户在浏览文件时就可以对其进行修改，如同快速地编辑一个打印复印件一样，因此它们是非常流行的一种编辑器。

注 1：实际上，目前 GNU 的 *emacs* 要比 *emacs* 的通用版本好，存在的惟一问题是其没有成为大部分商用 UNIX 系统中的标准编辑器，因此用户必须单独获取并安装它。

由于 vi 不是使用特殊的控制键来实现字符处理功能，而是使用所有常规的键盘键来触发命令，因此用户只能通过正常击键来输入命令，这使得许多初学者感到 vi 使用起来不太直观，并且比较麻烦。当键盘键触发命令时，vi 处于命令模式。只有 vi 处于特定的插入模式时，才可以在屏幕上键入真正的文本。除此之外，vi 命令似乎也太多了。

然而，一旦我们开始学习 vi，就会发现它设计得非常好，只需按下一些键，就可以让 vi 完成许多复杂的工作。我们还将了解到如何把更多的编辑工作提交给计算机（它所属的位置）去完成的快捷操作。

vi（同任何一个文本编辑器一样）不是“所见即所得”的字处理程序。如果想生成格式化的文档，就必须键入其他格式化程序所使用的代码来控制打印复印件的外观。例如，如果要缩进几个段落，则必须在缩进开始和结束的位置添加控制缩进的代码。用来进行格式化的代码允许用户对打印文件的外观进行实验或修改，并且与字处理程序相比，这种代码在许多方面都可以使用户对文档外观有更多的控制。UNIX 支持 *troff* 格式化包（注 2）。TeX 和 L^AT_EX 格式化程序也十分流行，通常可以交替使用。

（vi 支持一些简单的格式化机制。例如，可以让 vi 在到达行尾时自动换行，或是在每一段的开头自动缩进。）

与任何技巧一样，编辑得越多，基本操作就会变得越容易，所能完成的任务也就越来越多。一旦掌握了 vi 编辑器所有的功能，你就不会认为它是一个“简单”的编辑器。

编辑的基本功能是什么？首先，需要插入文本（遗忘的单词或漏掉的句子）和删除文本（零散的字符或整个段落）、修改字母和单词（更正错误拼写或反映术语内容的变化）。也可能需要把文本从文件的一个位置移动到另一个位置。而且，偶尔还需要进行文本复制以便把它复制到文件的其他位置。

注 2： troff 支持激光打印机和排字机，nroff 与其类似，支持行打印机和终端。这两个命令可以使用相同的输入语言，为了遵循共同的 UNIX 规则，我们也使用 troff 来指代这两种命令。

与许多字处理程序不同，*vi*的命令模式是初始的或是“默认的”。复杂的、交互式编辑通过一些击键就可以完成（如果要插入新文本，只需简单地给出许多“插入”命令中的一个，然后键入文本即可）。

*vi*通常使用一个或两个字母作为基本命令。例如：

i insert (插入)

cw change word (替换单词)

由于*vi*使用字母作为命令，这样就不必记忆成组的功能键或使用复杂的组合键。而大部分命令都很容易记忆（只需记住执行它们的字母），并且几乎所有的命令都遵循相同的模式并且相互联系，因此可以快速对文件进行编辑。

通常，*vi*命令：

- 是区别大小写的（键入大写字母与键入小写字母的含义是不同的；I与i是不同的）。
- 当键入该命令时，它并不显示（或回显）在屏幕上。
- 键入命令后不需要按下RETURN键（回车键）。

还有一组命令是显示在屏幕底行的。底行命令前要使用不同的符号，斜杠（/）和问号（?）放在搜索命令的开头，我们将在第三章“快速移动”对其进行讨论。冒号（:）引出所有的*ex*命令，*ex*命令是*ex*行编辑器所使用的命令。由于*ex*是底层的编辑器，而*vi*只是它的“可视”模式，因此在使用*vi*时*ex*也是可用的。*ex*命令及其概念将在第五章“介绍*ex*编辑器”中进行详细讨论，而本章只介绍不保存编辑而直接退出文件所使用的*ex*命令。

打开和关闭文件

你可以使用*vi*编辑任何文本文件。*vi*把需要编辑的文件复制到一个缓冲区（内存中临时留出来的区域）中，显示缓冲区的内容（尽管每次只能看到一整屏的文本

内容), 并且允许对文本进行添加、删除和修改等操作。当对所做的编辑进行保存时, **vi** 就把编辑过的缓冲区内容复制到永久文件中, 取代同名的旧文件。需要记住一点的是, 你一直是对在缓冲区里的文件副本进行处理, 只有保存缓冲区后, 编辑操作才会对初始文件有效。保存编辑结果也称为“写缓冲区”, 或更一般地称为“写文件”。

打开文件

  **vi** 是 UNIX 用来调用 **vi** 编辑器, 从而对现有文件或新文件进行编辑的命令。**vi** 命令的语法是:

```
$ vi [filename]
```

上面命令行中的方括号表示文件名是可选的, 这里的方括号不需要键入, \$ 是 UNIX 命令提示符。如果忽略文件名, 那么 **vi** 将打开一个未命名缓冲区。虽然可以在把缓冲区写入文件时指定名字, 但更好的做法则是在命令行上命名文件。

文件名在它所在的目录下必须是惟一的。在较老的 System V UNIX 系统中, 文件名的长度不能超过 14 个字符 (大部分常用的 UNIX 系统允许更长的文件名)。文件名可包含除斜杠 (/) 和 ASCII 的 NUL 之外的任何 8 位字符, 原因是保留斜杠作为路径中文件和目录之间的分隔符, 而 NUL 字符则所有位为零。甚至可以通过在空格前加反斜杠 (\) 而使文件名包含空格。然而实际上, 文件名通常由大小写字母、数字、字符点 (.) 和下划线 (_) 的任意组合形成。请记住, UNIX 是区分大小写的: 小写字母与大写字母的含义是截然不同的。还要记住, 必须按下 **RETURN** 键, 从而告诉 UNIX 已经结束了对命令的输入。

如果想打开某个目录下的一个新文件, 则要使用 **vi** 命令给出新文件名。例如, 如果想打开当前目录下的一个名为 “*practice*” 的新文件, 那么可以输入:

```
$ vi practice
```

由于这是个新文件, 因此缓冲区是空的, 屏幕显示如下:



0352180

15

```
~  
~  
~  
"practice" [New file].
```

屏幕左列的代字符 (~) 表示该文件中没有任何文本，甚至没有空行。屏幕底端的提示行（也称为状态行）显示文件的名字和状态。

也可以通过指定文件名来编辑任何目录下的任何已有文本文件。假定某个 UNIX 文件的路径为 /home/john/letter，如果已经在 /home/john 目录下，那么使用相对路径即可。例如：

```
$ vi letter
```

这样就把文件 “letter” 复制到屏幕上。

如果在其他目录下，则要使用完全路径名对其进行编辑：

```
$ vi /home/john/letter
```

打开文件时所遇到的问题

- ✓ 在调用 vi 时，出现 [open mode] 提示信息。

可能没有正确地标识终端类型。立即键入 :q 退出编辑状态，检查环境变量 \$TERM，该变量应该被设置为用户终端的名字。或者让系统管理员提供适当的终端类型。

- ✓ 如果看到下列信息中的一个：

```
Visual needs addressable cursor or upline capability
Bad termcap entry
Termcap entry too long
terminal: Unknown terminal type
Block device required
Not a typewriter
```

终端类型或者是没有定义，或者是 terminfo 或 termcap 条目有问题。输入 :q 退出，检查 \$TERM 环境变量，或者让系统管理员为你的环境选择一个终端类型。

- ✓ 如果认为文件存在时出现了[new file]提示信息。

你可能在一个错误的目录下。输入:q 退出, 然后检查是否在该文件所在的目录下(在UNIX命令提示符下输入pwd)。如果所在的目录是正确的, 检查该目录下的文件列表(使用ls), 确定该文件的名字是否有误。

- ✓ 调用vi, 但却出现了冒号提示符(表示正处于ex行编辑模式)。

可以在vi绘制屏幕之前输入了中断信号, 通过在ex提示符(:)后键入vi, 以进入vi模式。

- ✓ 出现下列提示信息中的一个:

```
[Read only]  
File is read only  
Permission denied
```

“Read only”表示只能查看该文件, 而不能保存对该文件所做的任何修改。可能是以“视图模式”(使用view或vi-R命令)调用vi, 或者是没有该文件的写权限, 请参考下面的“保存文件时所遇到的问题”小节。

- ✓ 出现下列提示信息中的一个:

```
Bad file number  
Block special file  
Character special file  
Directory  
Executable  
Non-ascii file  
file non-ASCII
```

所要编辑的文件不是普通的文本文件, 输入:q! 退出, 然后检查所要编辑的文件, 有可能需要使用file命令。

- ✓ 如果输入:q后, 上述问题中的任何一个, 可能会导致下面的提示信息:

```
No write since last change (:quit! overrides).
```

已经修改了文件而没有察觉。输入:q! 退出vi, 此次操作中所做的修改将不会保存在该文件中。

工作模式

正如前面所提到的，当前“模式”的概念是 vi 工作方式的基础。vi 有两种工作模式：命令模式和插入模式。如果以命令模式开始，这时的每次输入都表示一个命令。在插入模式下，输入的一切内容都将成为文件中的文本。

有时，你可能会无意中进入插入模式，或者无意中离开插入模式。在这两种情况下，所输入的内容可能不能满足对文件的处理要求。

按 [ESC] 键可以强迫 vi 进入命令模式。如果已经处于命令模式，按下 [ESC] 键时 vi 将会发出“嘟嘟”的声音。（因此，命令模式有时也称为“嘟嘟模式”。）

一旦处于命令模式，就可以对任何意外的修改进行处理，然后继续编辑文本。

保存和退出文件

你可以随时退出处理文件的过程，保存编辑操作并返回到 UNIX 命令提示符下。vi 用来退出并保存编辑的命令是 ZZ，要注意 ZZ 是大写的。

假设你创建了一个名为“*practice*”的文件来练习使用 vi 命令，并且输入了 6 行文本。如果要保存该文件，首先通过按 [ESC] 键确定处于命令模式，然后输入 ZZ。

击键	结果
ZZ	"practice" [New file] 6 lines, 320 characters
ls	ch01 ch02 practice

给出写保存命令 ZZ，文件将保存为常规的 UNIX 文件。

列出该目录下的文件来显示刚创建的新文件“*practice*”。

也可以使用 ex 命令来保存编辑操作。输入 :w 将保存文件但是并不退出 vi；如果没有进行任何编辑操作，那么输入 :q 就会退出 vi；输入 :wq 则保存编辑操作并退出 vi（:wq 命令等价于 ZZ 命令）。我们将在第五章“介绍 ex 编辑器”中详细说明如何使用这些命令。目前，只需记住一些用来写和保存文件的命令即可。

不保存编辑而直接退出

在开始学习 vi 时，还有两个 ex 命令是对编辑操作很有帮助的。

如果想放弃本次会话中所做的任何编辑，然后返回到初始文件，则命令：

```
:e! [RETURN]
```

可以返回到上一次保存的文件版本，因此可以重新对其进行编辑。

然而，假如想放弃编辑并退出 vi，那么所要使用的命令是：

```
:q! [RETURN]
```

该命令可使 vi 退出正在编辑的文件并返回到 UNIX 命令提示符下。使用这两个命令，都将失去自上次保存文件以来在缓冲区中所做的任何编辑操作。vi 通常不允许用户放弃他所做的编辑操作。将感叹号加在 :e 或 :q 命令的后面可使 vi 忽略这个限制，即使已经修改了缓冲区，vi 也会执行该操作。

保存文件时所遇到的问题

- ✓ 试着写文件，但却得到了下列提示信息中的一个：

```
File exists
File file exists - use w!
[Existing file]
File is read only
```

输入 :w! file 将覆盖现有的文件；或者输入 :w newfile，将把编辑后的版本保存到新文件中。

- ✓ 想写文件，但是如果没有写权限，就会得到 “*Permission denied*” 的提示信息。

使用 :w newfile 命令将缓冲区写到新文件中。如果对该目录具有写权限，就可以使用 mv 命令，利用该文件的副本取代它的原始版本；否则，输入 :w pathname/file，将缓冲区写到具有写权限的目录下（例如用户的主目录或 /tmp）。

- ✓ 试着写文件，但却得到文件系统已满的提示信息。

输入：`!rm junkfile`，删除不需要的（大）文件以释放一些空间（在 `ex` 命令前使用感叹号能使用用户访问 UNIX）。

或者输入：`!df`，查看其他的文件系统是否还有一些空间。如果有，选择该文件系统中的某个目录，并利用：`:w pathname` 命令把文件写到那个目录中（`df` 是 UNIX 用来检查磁盘剩余空间的命令）。

- ✓ 系统让用户进入开放模式并告诉用户文件系统已满。

带有 `vi` 的临时文件的磁盘已满，输入：`:!ls /tmp`，查看是否有可以删除的文件以获取一些磁盘空间（注 3）。如果有，创建一个临时的 UNIX shell 来删除文件或调用其他的 UNIX 命令。通过输入：`:sh` 可以创建 shell；输入 `[CTRL-D]` 或 `exit` 将终止 shell 执行并返回 `vi` 编辑环境（在大部分 UNIX 系统上，使用作业控制 shell 时，可以通过简单地输入 `[CTRL-Z]` 来挂起 `vi`，并返回到 UNIX 命令提示符下；输入 `fg` 即可返回到 `vi`）。一旦释放了一些空间，就可以使用 `:w!` 写文件。

- ✓ 试着写文件，但是却得到了磁盘空间配额已满的提示信息。

试着使用 `ex` 命令 `:pre (:preserve 的缩写)` 来强迫系统保存缓冲区内容。如果不可以，删除掉一些文件。使用 `:sh`（或 `[CTRL-Z]`，如果使用的是作业控制系统）离开 `vi` 并删除文件。删除文件后使用 `[CTRL-D]`（或 `fg`）返回 `vi`，然后使用 `:w!` 写文件。

练习

学习 `vi` 的惟一方法是实践。你现在已完全有能力创建新文件，然后返回到 UNIX 命令提示符下。请创建一个名为“`practice`”的文件，输入一些文本，然后保存该文件并退出。

注 3： `vi` 可能把临时文件保存在 `/usr/tmp`、`/var/tmp` 或当前目录下，可能需要到处寻找以确定所使用的磁盘空间的确切位置。

在当前目录下打开一个名为“*practice*”的文件: vi practice

插入文本: i any text you like

返回命令模式: [ESC]

退出 vi, 保存编辑: ZZ



本章内容：

- vi 命令
- 移动光标
- 简单编辑
- 插入文本的常用方法
- 使用J合并两行
- 回顾基本的 vi 命令

第二章

简单编辑

本章将向你介绍如何使用 *vi* 进行编辑，并且本章将以指导手册的形式帮助读者进行阅读。在本章中，我们将学习如何移动光标以及如何进行一些简单的编辑。如果你从没使用过 *vi*，那么应该通读全章。

后面的章节将展示如何通过扩展已掌握的技巧来进行更快、更有效的编辑。对于熟练的 *vi* 使用者，最大的好处之一是可以有许多选项进行选择（对于新的 *vi* 使用者，最大的不利之一是有那么多不同的编辑命令）。

你不可能通过记住每个单独的 *vi* 命令而学好 *vi* 的，要从学习本章介绍的基本命令着手，并注意各个命令的共同使用模式。

在学习 *vi* 时，要寻找更多能交给编辑器执行的任务，然后确定实现该操作的命令。在后面的章节中，我们将了解到更多有关 *vi* 的高级功能，但是在能使用那些高级功能之前，必须掌握好简单的基本命令。

本章包括：

- 移动光标

- 添加和修改文本
- 删除、移动和复制文本
- 进入插入模式的常用方法

vi 命令

vi 有两种工作模式：命令模式和插入模式。一旦进入文件就处于命令模式，编辑器正等着输入命令。命令可以使你移动到文件中的任何位置、进行编辑或进入插入模式以添加新文本，也可以使你退出文件（保存或忽略所做的编辑操作），然后返回 UNIX 命令提示符下。

可以把不同的模式看成代表两种不同的键盘。在插入模式下，你的键盘起着打字机的作用；而在命令模式下，每一个键都有一个新的含义或是发出某些指令。

如果想进入插入模式，可以有多种方法提示*vi*，其中最常用的一种方法是按下 `i` 键。虽然它并不显示在屏幕上，但是在按下 `i` 键之后，所键入的一切都将显示在屏幕上，并且会将其添加到缓冲区中。光标指示当前插入点的位置。如果想要告诉 *vi* 将停止输入文本，则可以按 `ESC`。按下 `ESC` 后，光标将后移一个空格（因此它位于输入的最后一个字符上），然后 *vi* 将返回命令模式。

例如，假设你已经打开了一个新文件并想插入单词“introduction”，如果输入的是“iintroduction”，则屏幕上将显示：

```
introduction
```

在你打开一个新文件时，*vi* 进入命令模式，并把第一次击键（`i`）解释为插入命令。插入命令之后的所有输入都将认为是文本，直到按下 `ESC` 键为止。如果你需要在插入模式下修改错误，退格后在错误处输入正确的文本即可。根据你所使用的终端类型，退格操作可能会删除前面输入的文本或只是沿着它后退。在上述任何一种情况下，退格所经过的所有文本都将删除。要记住，不能使用退格键移动到进入插入模式的位置之前的地方。

*vi*有一个选项，可以让你指定一个合适的右边距，并在到达该边界时就会自动进行回车操作。目前输入文本时，按下`RETURN`键来换行。

有时并不知道现在是处于插入模式下还是命令模式下，当*vi*不按预期的方式进行工作时，按一两次`ESC`键来检查所处的工作模式。当听到“嘟嘟”声时，实际上已处于命令模式下。

移动光标

在编辑会话中，可能只需少量的时间在插入模式下添加新文本，而需要把大量的时间花在对现有文本进行编辑上。

在命令模式下，可以把光标定位到文件中的任何位置。由于只有将光标定位到想要修改的文本位置，才能进行所有的基本编辑工作（修改、删除和复制文本），因此你希望尽可能快地把光标移动到那个位置。

有许多移动光标的*vi*命令：

- 上、下、左或右—每次移动一个字符
- 按照单词、句子或段落那样的文本块前移或后移
- 在文件中前移或后移，每次移动一屏

在图 2-1 中，下划线标识光标的当前位置，圆圈指示光标从其当前位置到不同的*vi*命令所产生的位置的移动。

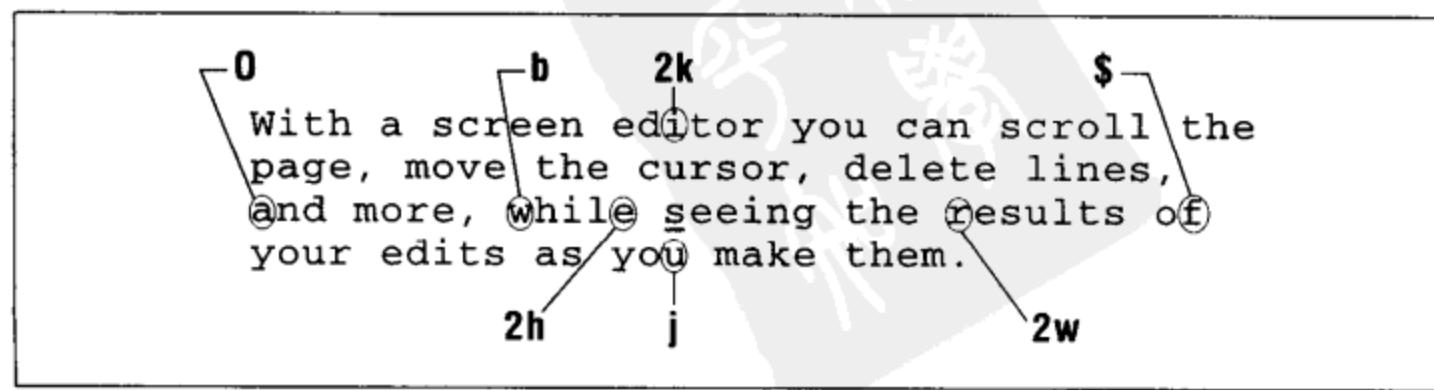


图 2-1 简单的移动命令

单一移动

在右手指下的 h、j、k 和 l 键可以用来移动光标：

h 左移，一个空格

j 下移，一行

k 上移，一行

l 右移，一个空格

也可以使用光标箭头键 (\leftarrow , \downarrow , \uparrow , \rightarrow)、+ 和 - 来上移和下移，或 **RETURN** 键和 **BACKSPACE** 键来移动光标，但它们都不是通用的方法，并且并不是所有的终端都支持箭头键。起初，使用字母键代替箭头键来移动光标可能会比较麻烦，但是经过一段时间的使用之后，你会发现这是 vi 的一个优点，因为不用让手指离开键盘中心就可以四处移动光标。

在移动光标之前，要按下 **ESC** 键以确保处于命令模式。然后在文件中用 h、j、k 和 l 键把光标从当前位置向前或向后移动。当向一个方向移动一定距离后，就会听到“嘟”的一声，然后光标停止移动。例如，一旦光标位于某行的开头或结尾，就不能使用 h 或 l 键把光标换行并移动到上一行或下一行；必须使用 j 或 k 键来完成上述操作（注 1）。同样，不能把光标移过表示没有文本的一行的代字符 (~)，也不能把光标移到第一行文本的前面。

数字参数

可以在移动命令的前面添加数字。图 2-2 展示了命令 “4l” 如何把光标向右移动 4 格，就像键入了 4 次 l 一样 (1111)。

复合命令的作用是，对于我们所学过的每个命令，可以有更多的选择和更强的功能，在学习其他命令时一定要牢记复合命令。

注 1： vim 4.x 版和带有不兼容设置的 vim 5.x 版允许使用 l 或空格键越过一行的结尾到达下一行。

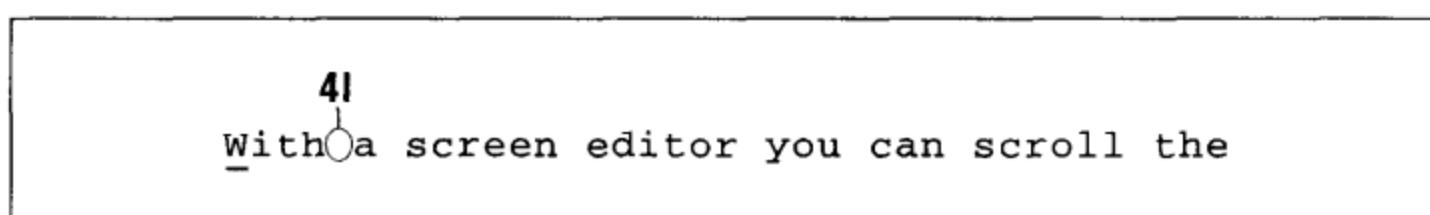


图 2-2 带数字的复合命令

行内移动

在保存文件“*practice*”时，*vi*会显示该文件的行数信息。行的长度并不一定必须和屏幕上显示的可见长度（通常是 80 个字符）相同。一行是在换行符之间输入的任何文本。（在插入模式下按下 RETURN 键时，就将换行符插入到文件中。）如果在按下 RETURN 键之前输入了 200 个字符，*vi*会把这 200 个字符看成单独的一行（尽管这 200 个字符在屏幕上看起来占据了几行的位置空间）。

正如前面所提到的，*vi*有设置页面右边距的选项，到达该位置后*vi*将自动插入换行符。该选项是 *wrapmargin*（它的缩写是 *wm*），我们可以把换行边界设置为 10 个字符：

```
:set wm=10
```

这个命令并不影响已经输入的行，我们将在第七章“高级编辑”详细讨论设置选项。

如果没有使用*vi*自动的 *wrapmargin* 选项，那么应该使用回车键进行断行，以保持每行的可控长度。

① *vi* 有两个实用的行内移动命令：

0 移动到行首

\$ 移动到行尾

在下面的例子中，将显示出行号。（通过使用 *number* 选项可以在*vi*中显示行号，通过在命令模式下输入 :set nu 即可。这个操作将在第七章讨论。）

- 1 With a screen editor you can scroll the page
- 2 move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.
- 3 Screen editors are very popular.

 逻辑行数（3）并不与屏幕上所看到的行数（6）相对应。如果光标位于单词“*delete*”的“d”位置，则输入\$后，光标将移动到单词“*them*”后面的句点上。如果你输入0，光标将移动到第二行的开始，即单词“*move*”的*m*字母处。

按文本块移动

 我们也可以按单词、句子、段落等等之类的文本块移动光标。**w**命令每次把光标向前移动一个单词，并把符号和标点作为单词处理。下面的这一行展示了命令w如何移动光标：

```
cursor, delete lines, insert characters,
```

如果使用**w**命令，也可以按单词移动光标，但此时并不把符号和标点作为单词计算（你可把这看作是一个“大的”或“大写”的Word）。

使用**w**移动光标如下所示：

```
cursor, delete lines, insert characters,
```

使用**b**命令可以按单词后移光标，大写**B**命令也可以按单词后移光标，但不计算标点。

正如前面所提到的，由于移动命令也可以带数字参数，因此在使用**w**或**b**命令时，可以利用数字来加速移动。例如，**2w**向前移动2个单词，**5B**不计标点向后移动5个单词。

我们将在第三章“快速移动”中讨论按句子和按段落移动光标。现在，请你用你知道的光标移动命令进行练习，并把它们与数字参数结合在一起使用。

简单编辑

当我们在文件中输入文本时，一定会产生一些问题。例如，发现了错误或想改善某个短语；有时程序也会产生错误。一旦我们输入了文本，就必须能够对其进行修改、删除、移动或复制。图 2-3 展示了可能要对文件进行的各种编辑，这些编辑都使用校对符进行表示。

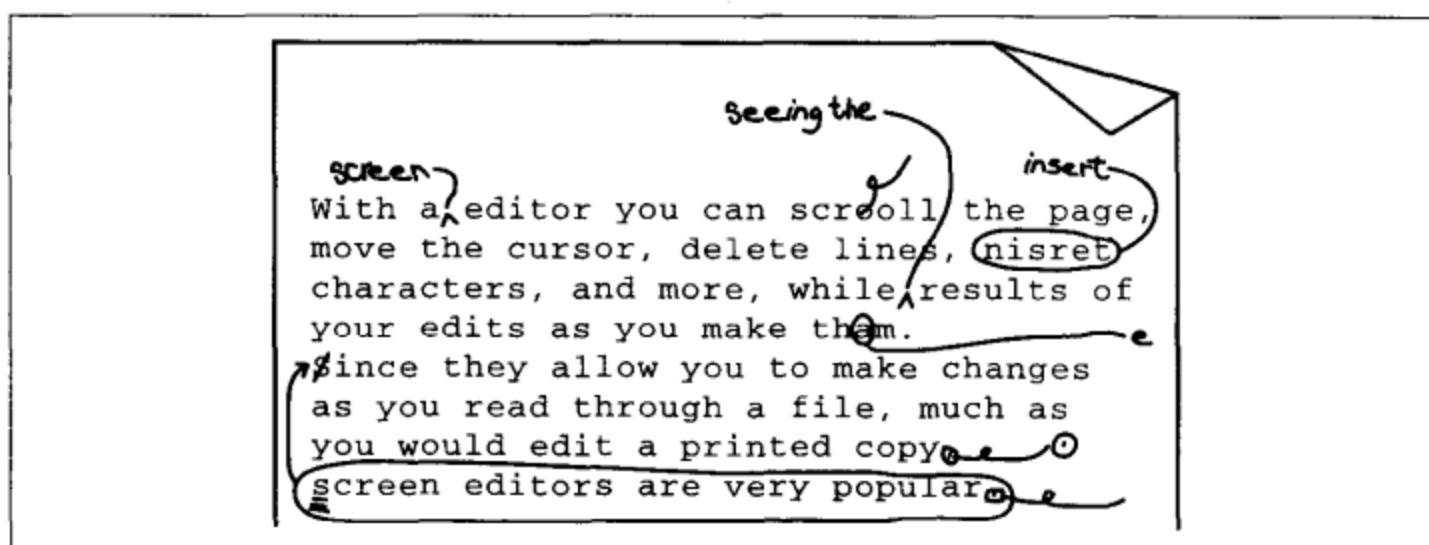


图 2-3 校对编辑

在 vi 中，可以使用几个基本的击键完成这些编辑中的任何一个：插入命令 i（这你已经看到了）；追加命令 a；修改命令 c 和删除命令 d。如果要移动或复制文本，则可以使用一对命令。先使用删除命令 d，然后再使用粘贴命令 p，即可移动文本；或是先使用复制命令 y，然后再使用粘贴命令 p，即可复制文本。本章会介绍所有类型的编辑。图 2-4 展示了如何使用 vi 命令完成图 2-3 中所标识的编辑。

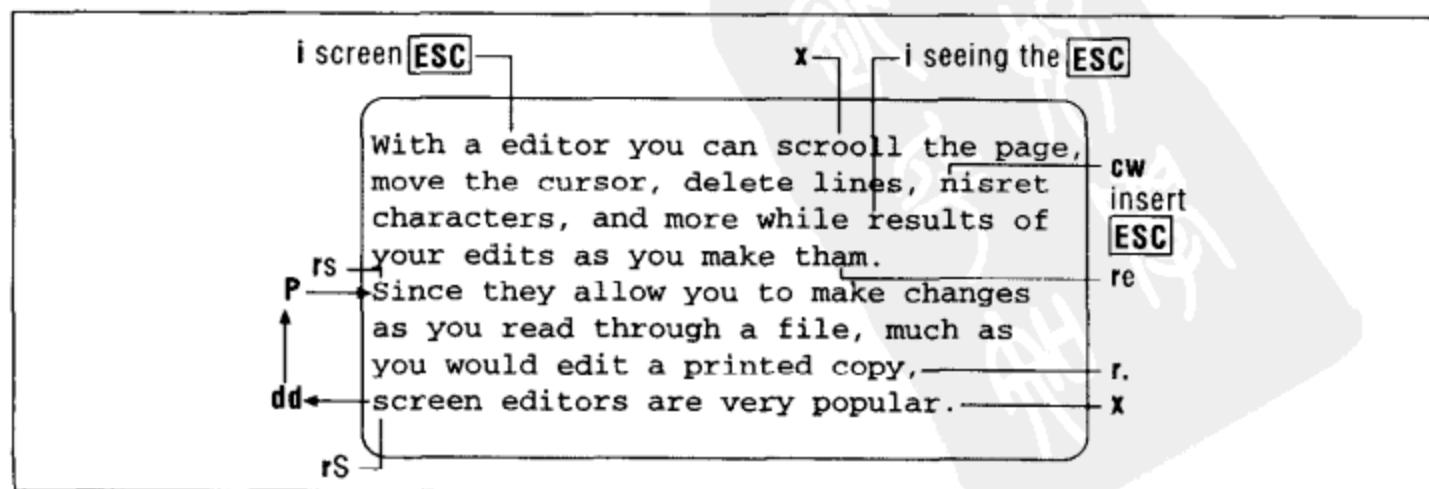


图 2-4 使用 vi 命令进行编辑

插入新文本

我们已经介绍了使用插入命令把文本输入到新文件中的方法。在编辑现有文本以添加遗忘的字符、单词和句子时，也需要使用插入命令。在文件“*practice*”中，假如已有句子：

```
you can scroll  
the page, move the cursor, delete  
lines, and insert characters.
```

光标的位置如上所示，为了在句子的开始插入“With a screen editor”，输入如下：

击键

2k

```
you can scroll  
the page, move the cursor, delete  
lines, and insert characters.
```

使用 k 命令把光标向上移动两行，达到想要插入文本的那一行。

iWith a

```
With a you can scroll  
the page, move the cursor, delete  
lines, and insert characters.
```

按 i 键进入插入模式，开始插入文本。

screen editor

ESC

```
With a screen editor you can scroll  
the page, move the cursor, delete  
lines, and insert characters.
```

完成文本输入后，按下 ESC 键结束输入，返回命令模式。

如上面的例子所示，在插入新文本时 vi 把现有的文本向右推，就如同插入新文本一样。这是因为我们假定你正在“智能”终端上使用 vi，该终端能在屏幕上重写你所输入的每个字符。在“哑”终端（如 *adm3a*）上进行的插入操作则会有很大不同。由于该终端本身不能在输入每个字符后刷新屏幕，因此直到你按下 ESC 键后 vi 才会重写屏幕。在哑终端上，同样的插入将显示如下：

击键**i**With a**结果**

With an scroll
the page, move the cursor, delete
lines, and insert characters.

按*i*键进入插入模式，开始插入文本。终端显示覆盖了该行上的现有文本。

screen editor

With a screen editor_
the page, move the cursor, delete
lines, and insert characters.

插入文本好像覆盖了现有文本。

ESC

With a screen editor you can scroll
the page, move the cursor, delete
lines, and insert characters.

在完成插入文本后，按下**ESC**键结束输入返回命令模式，这时*vi*才重写该行，因此可以看到所有的现有文本。

追加文本

a 你可以使用追加命令[a](#)在文件的任何位置追加文本。[a](#)命令除了把文本插入到光标之后而不是光标之前以外，它与*i*命令几乎以同样的方式进行工作。你可能已经注意到：当按*i*键进入插入模式时，光标直到输入某个文本以后才开始移动。相反，当按[a](#)键进入插入模式时，光标会立即向右移动一个空格，当你输入文本时，它会出现在光标起始位置的后面。

修改文本

c 你可使用修改命令[c](#)替换文件中的任何文本。为了告诉[c](#)命令要修改的文本的数量，可以把[c](#)命令与移动命令结合在一起使用。在这种情况下，移动命令用来指明[c](#)命令所作用的文本对象。例如，可以使用[c](#)命令修改远离光标的文本：

`cw` 到单词的尾部

`c2b` 后退两个单词

`c$` 到行尾

`c0` 到行首

发出修改命令以后，可以使用任何数量的新文本、零个字符、一个单词或上百行来替换指定的文本。类似于 `i` 和 `a`, `c` 命令也使你留在插入模式中，直到按下 `[ESC]` 键才会回到命令模式。

当修改只影响当前行时，`vi` 就用 `$` 来标记将修改的文本尾部，因此你可以看到这一行中已修改的部分（参看下面 `cw` 的例子）。

单词

 **c w** 如果要修改一个单词，可以把 `c`（修改）命令与表示单词的 `w` 结合在一起使用。可以用更长或更短的单词（或任何数量的文本）来取代一个单词 (`cw`)，可以把 `cw` 理解为“删除标识的单词，并插入新文本直到按下 `[ESC]` 键”。

假如文件 “*practice*” 含有以下行：

```
With an editor you can scroll the page,
```

并且要把 *an* 修改为 *a screen*，那么只需要修改一个单词：

击键	结果
<code>w</code>	With <u>an</u> editor you can scroll the page,

使用 `w` 把光标移动到要开始编辑的位置。

<code>cw</code>	With <u>a\$</u> editor you can scroll the page,
-----------------	---

给出修改单词命令，在要修改的文本尾部标上 `$`（美元符号）标记。

简单编辑

a screen

With a screen editor you can scroll the page,

输入完替换文本后，按下`ESC`键返回命令模式。

`cw`也可以作用于单词的一部分。例如，将 *spelling* 修改为 *spelled*，可以把光标定位在 *i* 上，输入 `cw`，然后输入 `ed` 并按`ESC`键结束。

行

 **C** **C** `cc` 是专门用来替换整个当前行的命令。`cc` 会对整行进行修改，并用按下`ESC`键之前所输入的任意数量的文本取代该行。不管该行中光标的位置，`cc` 都会对整行的文本进行替换。

`cw` 命令与 `cc` 命令的工作方式不同。使用 `cw` 时，原有文本将保存到覆盖它为止，当按下`ESC`键时，才将剩余的 (`$` 之前的) 文本删除。而使用 `cc` 时，首先删除原有文本，并留出一个空白行插入文本。

“覆盖”方法适用于任何其作用小于一整行的修改命令，而“空白行”方法则适用于任何作用一行或多行的修改命令。

 **C** `C` 用来替换从当前光标位置到行尾的所有字符。`C` 命令和 `c` 与专用的行尾标识符 `$` 的结合 (`c$`) 具有同样的效果。

由于命令 `cc` 和 `C` 实际上是其他命令的简化操作，因此它们不遵循 *vi* 命令的一般模式。当我们讨论删除和复制命令时，你会看到其他的快捷操作。

字符

 **r** `r` 是另一个用来替换文本的命令，`r` 命令利用一个单个字符替换另一个单个字符。在编辑完成后，不必按下`ESC`键就可以返回命令模式。下面的一行中有一处拼写错误：

Pith a screen editor you can scroll the page,

vi 命令的一般格式

到目前为止，在我们已经提到的所有修改命令中，你可能已经注意到下面的模式：

(命令) (文本目标)

其中命令是修改命令 c，文本目标是移动命令（不用输入圆括号）。但是 c 并不是唯一需要文本目标的命令，d 命令（删除）和 y 命令（复制）也遵循这个模式。

请记住移动命令可以带数字参数，因此也可以把数字加到 c、d 和 y 命令的文本目标的前面。例如 d2w 和 2dw 都是删除两个单词的命令。下面是大部分 vi 命令都遵循的一般模式：

(命令) (数字) (文本目标)

或者等同于：

(数字) (命令) (文本目标)

这就是 vi 命令的使用规则。数字和命令是可选的，如果没有它们，则只有一个移动命令。如果加上数字，就可以移动多次。相反，将命令 (c、d 或 y) 与文本目标结合就得到编辑命令。

当你认识到可以按照这种方式把多个命令结合在一起时，vi 才会真正变成一个功能强大的编辑器。

只有一个字母需要改正。由于我们不想重新输入整个单词，因此在这个句子中不必使用 cw。可以使用 r 来替换光标处的字符：

击键

rW

结果

With a screen editor you can scroll the page,

给出替换命令 r，紧接着是替换字符 W。

替换文本

S 假如只想修改几个字符而不是整个单词，替换命令（s）本身就只替换单个字符。使用前面的计数，则可以替换多个字符。与修改命令（c）一样，文本的最后一个字符也用 \$ 进行标记，这样就可以知道将要修改的文本的数量。

S 通常大写字母命令与小写字母命令是不同的，S命令允许改变整行。与从光标当前位置开始改变行的剩余部分的C命令相比，S命令将删除整行，而不管光标的位置。vi将在该行的开始位置进入插入模式，前面的计数表示将要替换的行数。

s 和 S 都会使你处于插入模式，当完成新文本的输入时，即可按下ESC键退出插入状态。

R 类似于与其对应的小写字母，R也是文本替换命令，区别在于它只进入改写模式。你输入的字符将逐个地替换屏幕上的文本，直到输入ESC键为止。最多只能改写一整行；当输入RETURN时，vi就会打开一个新行，并使你处于插入模式。

大小写转换

~ 改变字母的大小写是一种特殊形式的替换。代字符（~）命令将把小写字母变成大写，或把大写字母变成小写。把光标定位到想改变的字母上，然后输入~，该字母的格式将会转换，并且光标将移动到下一个字符。

在较老的 vi 版本中，你不能为~指定它要改变的数字前缀或文本目标，现在的版本允许使用数字前缀。

如果想一次改变多行的字母类型，那么你必须通过类似于 tr 的 UNIX 命令对文本进行筛选，这些内容将在第七章进行讨论。

删除文本

 也可以使用d命令删除文件中的冗余文本。类似于修改命令，删除命令也需要文本目标（将被删除的文本数量）。你可以按单词（dw）、按行（dd和D）或按后面将要学到的其他移动命令进行删除。

在删除文本时，要先把光标移动到要删除的文本位置，然后给出删除命令（d）和文本目标，例如w表示单词。

单词

  假如你的文件中有下列文本：

```
Screen editors are are very popular,  
since they allowed you to make  
changes as you read through a file.
```

光标的位置如上所示，你想删除第一行中的一个“are”。

击键

2w

结果

```
Screen editors are are very popular,  
since they allowed you to make  
changes as you read through a file.
```

把光标移动到要删除的位置（are）。

dw

```
Screen editors are very popular,  
since they allowed you to make  
changes as you read through a file.
```

给出删除单词命令（dw），删除单词“are”。

dw删除了光标定位在其首部的那个单词，要注意单词后面的空格也被删掉了。

dw也可被用来删除部分单词。在下面的例子中：

```
since they allowed you to make
```

我们要删除单词 “*allowed*” 尾部的 “*ed*”。

击键	结果
<code>dw</code>	since they allow <u>you</u> to make

给出删除单词命令 (`dw`)，从光标开始的位置对该单词进行删除。

`dw`总是删除本行中下一个单词前面的空格，但是在上面的例子中，我们并不想删除那个空格。为了保留单词之间的空格，可使用 `de`，它只删除到单词的尾部。输入 `dE` 将删除到单词的尾部，并且包括标点符号。

也可以向后删除 (`db`) 或者删除到一行的尾部或首部 (`d$` 或 `d0`)。

行

 `dd` 命令对光标所在的整行进行删除，它不能删除行的一部分。类似于它的互补命令 `cc`，`dd` 也是个专用命令。使用前面例子中同样的文本，光标将如下所示定位在第一行上：

```
Screen editors are very popular,
since they allow you to make
changes as you read through a file.
```

你可以删除前两行：

击键	结果
<code>2dd</code>	changes as you read through a file.

给出删除两行的命令 (`dd`)，注意尽管光标没有位于该行的首部，但还是删除了一整行。

如果你使用的是“哑”终端（注 2）（或非常慢的终端），行删除则会是不同的。直到移动光标到屏幕的底部，哑终端才会对屏幕进行重新刷新。在哑终端上，删除操作如下所示：

注 2：现在哑终端已非常稀少，很多时候，用户都在位图屏幕上的终端仿真程序中运行 `vi`。

击键 **结果**

2dd

```
@  
@  
changes as you read through a file.
```

给出删除两行的命令 (2dd), @ 符号占据着删除的行的空间, 直到 vi 重新刷新整个屏幕为止。

D D 命令用来删除从光标位置到行尾的文本 (D 是 d\$ 的缩写)。例如, 光标位置如下所示:

```
Screen editors are very popular,  
since they allow you to make  
changes as you read through a file.
```

你可以删除该行光标右边的部分。

击键 **结果**

D

```
Screen editors are very popular,  
since they allow you to make  
changes_
```

给出删除该行光标右边的部分的命令 (D)。

字符

x 用户经常只想删除一两个字符。就像 r 是用来替换单个字符的专用修改命令那样, x 是用来删除单个字符的专用删除命令。x 只删除光标所在位置的字符。在下面的一行中:

You can move text by deleting text and then

你可以通过按下 x 键来删除字母 z (注 3)。大写字母 X 将删除光标前面的那个字符。这些命令中的任何一个带上数字前缀, 将删除该数字所表示的数量的字符。例如, 5x 将删除光标右边的 5 个字符。

注 3: 记忆 x 的方法是把其想像为打字机用 x 划掉错误。当然, 谁还使用打字机呢?

删除操作中遇到的问题

- ✓ 删错了文本并想恢复它。

有几种方法可以恢复误删的文本。如果刚刚删除了一些内容并且要恢复它，那么只需输入 `u` 就能取消上次命令操作（如 `dd`）。由于 `u` 只能取消最近的一次操作，因此这只能在没有给出任何进一步的命令时起作用。相反，`U` 将能把该行恢复到其原来的状态，即对该行进行任何改变之前的状态。

然而，由于 `vi` 把最近的 9 次删除操作保存在 9 个已编号的删除缓冲区中，因此你也可以使用 `p` 命令恢复最近的删除操作。例如，如果你知道倒数第三次删除操作是想要恢复的，那么输入：

"3p

就会把 3 号缓冲区的内容输出到光标下面的行上。

这只对已删除的行起作用，单词或部分行都不能保存在缓冲区中。如果想恢复已删除的单词或部分行，而 `u` 又不起作用，那么就只能使用 `p` 命令了。这样将恢复最后一次所删除的内容。下面将详细讨论 `u` 和 `p` 命令。

移动文本

在 `vi` 中，可以通过删除文本然后再把所删除的文本粘贴到文件中的其他位置来移动该文本，如同“剪切和粘贴”那样。每次删除一个文本块，删除的部分都将暂时保存在一个专用缓冲区中。把光标移动到文件中的其他位置，并使用粘贴命令 (`p`) 将这些文本粘贴到新位置。虽然可以移动任何文本块，但是移动行比移动单词要更有用。

p 粘贴命令 (`p`) 把位于缓冲区中的文本粘贴到光标位置的后面。该命令的大写版本 `P` 把这些文本粘贴到光标的前面。如果删除了一行或多行，那么 `p` 将把已删除的文本粘贴到光标下面的新行中。如果删除的内容不到一整行，那么 `p` 将把已删除的文本粘贴到当前行中的光标的后面。

假如文件 “`practice`” 中有文本：

You can move text by deleting it and then,
like a "cut and paste",
 placing the deleted text elsewhere in
 the file.
 each time you delete a text block.

你想把第二行 *like a "cut and paste"*，移到第三行的下面。那么可以用删除操作实现。

击键

dd

结果

You can move text by deleting it and then,
 placing the deleted text elsewhere in
 the file.
 each time you delete a text block.

光标在第二行上，删除该行。删除的文本将保存在缓冲区（保留的内存）中。

p

You can move text by deleting it and then,
 placing that deleted text elsewhere in
 the file.
like a "cut and paste",
 each time you delete a text block.

给出粘贴命令（p）把已删除的一行恢复到光标下面的下一行中。为了完成该句子的重新排列，需要改变字母的大小写和标点（使用r）来匹配新结构。

注意：一旦删除了文本，则必须在下一个修改命令或删除命令之前恢复它。如果进行的其他编辑影响了缓冲区，那么所删除的文本就会丢失。只要没有进行新的编辑，就可以反复执行粘贴操作。在第四章“基本编辑命令的扩展”中，你将了解到如何把删除的文本保存到命名缓冲区中，以便在以后可以重新恢复它。

调换两个字母的顺序

可以使用xp（删除字符然后把它粘贴到光标的后面）来调换两个字母的顺序。例如，在单词“mvoe”中，字母“vo”的顺序颠倒了。为了改变顺序，只要把光标放置在v上并按下x键，然后再按下p键。同时，单词“transpose”会帮助你记住xp的顺序，x表示“trans”，p表示“pose”。

这里没有介绍调换单词的命令，第七章的“映射键的更多例子”一节中讨论了一系列调换两个单词的命令。

复制文本

y 通常，可以把文件的一部分复制到其他位置来进行新操作，这样可节省编辑时间（和按键）。使用 y（表示复制）和 p（粘贴）命令，可以复制任何数量的文本，然后把已复制的文本粘贴到文件的其他位置。复制命令把选中的文本复制到一个专用缓冲区中，文本将保存在那里直到出现另一个复制操作（或删除操作）。然后就可以使用粘贴命令把这个文本副本粘贴到文件的其他位置。

与修改和删除命令一样，复制命令也可以与任何移动命令结合在一起使用 (yw、y\$、4yy)。由于复制和粘贴单词要比只插入单词花费的时间长，因此复制命令经常用于一行（或多行）文本的操作。

类似于 dd 和 cc，快捷操作 yy 也作用于一整行。但是由于某些原因，快捷操作 Y 并不像 D 和 C 那样使用，Y 复制整行而不是复制从当前光标位置到行尾的部分。Y 和 yy 的作用相同。

假如你的文件 “*practice*” 中有文本：

```
With a screen editor you can
scroll the page.
move the cursor.
delete lines.
```

如果想将其改成 3 个完整的句子，每个句子都以 “*With a screen editor you can*” 开始，则可以使用复制和粘贴操作来复制要添加的文本。

击键

YY

结果

```
With a screen editor you can
scroll the page.
move the cursor.
delete lines.
```

把要复制的那行文本复制到缓冲区中，光标可以位于想要复制的行的任何位置（或一系列行的首行）。

2j

With a screen editor you can
scroll the page.
move the cursor.
delete lines.

把光标移动到想粘贴已复制文本的位置。

P

With a screen editor you can
scroll the page.
With a screen editor you can
move the cursor.
delete lines.

使用大写 P 命令把已复制的文本粘贴到光标所在行的上方。

jp

With a screen editor you can
scroll the page.
With a screen editor you can
move the cursor.
With a screen editor you can
delete lines.

把光标移动到下一行，然后用p命令把已复制的文本粘贴到光标所在行的上方。

复制操作与删除操作使用同一个缓冲区，每个新的删除操作或复制操作将取代复制缓冲区中先前的内容。正如我们将在第四章中看到的，可以使用粘贴命令恢复9个以前的复制或删除操作。也可以直接把文本复制或删除到26个已命名的缓冲区中，它们允许用户可以同时处理多个文本块。

重复或撤消最后的命令

在给出下一个命令之前，你所执行的每个编辑命令都将存储在一个临时缓冲区中。例如，如果在文件中的某个单词后插入*the*，那么用来插入文本的命令以及所输入的文本都将会暂时保存起来。

重复

任何时候需要重复同样的编辑命令，都可以通过使用重复命令：句点（.）来节省操作时间。把光标定位到想要重复编辑命令的位置，然后输入句点。

假如在你的文件中有下列行：

```
With a screen editor you can
scroll the page.
With a screen editor you can
move the cursor.
```

你可以删除一行，然后在删除另一行时只需输入句点即可。

击键	结果
dd	<pre>With a screen editor you can scroll the page. move the cursor.</pre>

使用命令 dd 删除一行。

```
With a screen editor you can
scroll the page.
```

重复删除操作。

较老的 vi 版本在重复命令时会有一些问题。例如，当设置 wrapmargin 时，这些版本在重复一个长的插入操作时可能会有些困难。如果你有那样的版本，这个缺陷可能迟早会影响你。虽然在那种情况发生后你对它也没有什么可做的，但是它有助于对你进行预先警告(近期的版本好像没有这个问题)。当你重复较长的插入操作时，可以有两种方法来预防潜在的问题。你可以在重复插入操作之前执行写文件 (:w) 操作（若插入操作不能正常进行则可以返回到这个副本），也可以关掉 wrapmargin，如下所示：

```
:set wm=0
```

在第七章的“映射键的更多例子”一节中，我们将介绍一种很容易重用该方案的办法。在某些 vi 版本中，**CTRL-@** 命令可重复最近的插入操作，在插入模式下输入 **CTRL-@** 后就会返回命令模式。

撤销

 正如前面所提到的，如果进行了错误操作，只要按下`u`键就可以撤销上次操作。光标也不需要位于上次操作进行时所在的行。

继续上面在文件“*practice*”中展示行删除的例子：

击键 **结果**

`u`

```
With a screen editor you can
scroll the page.
move the cursor.
```

`u` 撤消上次操作并恢复删除的行。

与`u`对应的大写字母`U`撤消对单行进行的所有编辑，前提是光标要保持在该行上。一旦把光标移到了其他行，就不能使用`U`命令了。

记住，可以使用`u`撤消上次的撤消操作，这样就可以在文本的两个版本之间进行切换。`u`也能撤消`U`，而`U`能撤消对一行进行的任何修改，包括使用`u`撤消过的行。（提示：`u`可以撤消本身操作的事实提供了一种在文件中到处移动的好办法。如果想回到上次的编辑状态，只需撤消它即可。然后将返回到对应的行。当撤消了撤消操作时，则将保持在那行。）

插入文本的常用方法

我们已经在光标前面顺序地插入过文本：

```
itext to be inserted[ESC]
```

我们也使用`a`命令在光标的后面插入过文本，这里还有一些在相对于光标不同位置的地方插入文本的其他命令：

- A 在当前行的尾部添加文本。
- I 在行首插入文本。

- 在光标所在行的下面新建一行，等待输入文本。
- 在光标所在行的上面新建一行，等待输入文本。
- s 使用输入的文本替换光标所在位置的字符。
- S 使用输入的文本替换当前行。
- R 使用新字符覆盖现有字符。

所有这些命令都会使你处于插入模式，在插入文本后，记得按下`ESC`键返回命令模式。

`A`（追加）和`I`（插入）使你在进入插入模式前不必把光标移到行首或行尾。`(A)`命令相对于`$a`来说省了一次击键。虽然一次击键可能不能节省很多时间，但是你对编辑器越熟练，所想忽略的击键就越多)。

- 和○（打开）使你不必输入回车键，可以在行内的任何位置执行这两个命令。
- s 和S（替换）允许删除一个字符或一整行，并用任意数量的新文本来替换所删除的文本。`s`等价于两次击键命令`c [SPACE]`，`S`与`cc`相同。`s`的最好用途之一是把一个字符改为多个字符。
- R 在你想修改文本但又不能确切知道文本数量时是非常有用的。例如，不用确定是`3cw`还是`4cw`，只要输入`R`然后输入替换字符即可。

插入命令的数字参数

除○和○以外，上面的插入命令（加上`i`和`a`）都可以带有数字前缀。通过数字前缀，可以使用`i`、`I`、`a`和`A`命令插入一排下划线或替换字符。例如，输入`50i*ESC`可以插入50个星号，输入`25a*-ESC`可以追加50个字符（25对星号和连字号）。不过最好只重复一小串字符（注4）。

注4：较老的vi版本在重复插入多行的文本值时有困难。

通过数字前缀，`r` 可以使用一个单个字符的重复实例替换许多字符。例如，在 C 或 C++ 代码中，要把 `||` 变成 `&&`，你可以把光标定位在第一个管道符上，然后输入 `2r&`。

可以使用带数字前缀的 `s` 命令来替换多行，这与使用带移动命令的 `c` 相比，既快又灵活。

适合使用带数字前缀的 `s` 命令的情况是想在一个单词的中间修改几个字符，这是因为输入 `r` 是不合适的，但输入 `cw` 又会改变太多的文本。使用带数字前缀的 `s` 通常与输入 `R` 是相同的。

这里还有一些可以一起正常工作的其他命令组合。例如，`ea` 对在单词的结尾追加新文本是非常有用的。这有助于训练你自己识别那些频繁使用的命令组合，以使你自觉使用它们。

使用 `J` 合并两行

J 有时在编辑文件时，想合并一系列很难浏览的短行。当要把两行合并为一行时，可以将光标定位在第一行上，然后按下 `J` 键来合并这两行。

假如文件 “*practice*” 中有：

```
With a  
screen editor  
you can  
scroll the page, move the cursor
```

击键

`J`

结果

```
With a screen editor  
you can  
scroll the page, move the cursor
```

`J` 把下面的行合并到光标所在的行。

With a screen editor you can scroll the page, move the cursor

使用 . 重复上次命令 (J)，把下一行合并到当前行。

使用带数字参数的 J 能合并该数量的连续行。在上面的例子中，你可以使用 3J 命令来合并 3 行。

问题列表

- ✓ 当输入命令时，文本显示在屏幕上但并没有按照期望的方式进行工作。

确保没有在想输入 j 命令时输入了 J。

你可能按下了 CAPS 键，而没有注意到。由于 vi 是区分大小写的，即大写命令 (I、A、J 等) 与小写命令 (i、a、j) 是不同的，因此你的所有命令都没有解释为小写，而是解释为大写。再次按下 CAPS 键可返回小写状态，并按下 ESC 键以确保处于命令模式，然后输入 U 恢复上次的行修改或者输入 u 撤消上次命令。你可能还要做一些额外的编辑以完全恢复文件的错误部分。

回顾基本的 vi 命令

表 2-1 列出了一些通过把 c、d、y 命令与各种文本目标结合在一起执行的命令。最后两行展示了其他的编辑命令。表 2-2 和表 2-3 列出了一些其他的基本命令。表 2-4 对本章所描述的其余命令进行了总结。

表 2-1 编辑命令

文本目标	修改	删除	复制
1 个单词	cw	dw	yw
2 个单词，不计标点	2cw 或 c2w	2dw 或 d2w	2yw 或 y2w
向后 3 个单词	3cb 或 c3b	3db 或 d3b	3yb 或 y3b
1 行	cc	dd	yy 或 Y

表 2-1 编辑命令（续）

文本目标	修改	删除	复制
到行尾	c\$ 或 C	d\$ 或 D	y\$
到行首	c0	d0	y0
单个字符	r	x 或 X	yl 或 yh
5 个字符	5s	5x	5yl

表 2-2 移动

移动	命令
←, ↓, ↑, →	h, j, k, l
到下一行的首字符	+
到上一行的首字符	-
到单词的尾部	e 或 E
按单词前移	w 或 W
按单词后移	b 或 B
到行尾	\$
到行首	0

表 2-3 其他操作

操作	命令
从缓冲区输出文本	p 或 p
启动 vi, 如果指定了文件, 就打开该文件	vi file
保存编辑、退出文件	ZZ
不保存编辑、退出文件	:q!

表 2-4 文本创建和操作命令

编辑行为	命令
在当前位置插入文本	i
在行首插入文本	I
在当前位置追加文本	a

表 2-4 文本创建和操作命令（续）

编辑行为	命令
在行尾追加文本	A
在光标所在行的下面新建一行，等待输入新文本	O
在光标所在行的上面新建一行，等待输入新文本	O
删除行并替换文本	S
使用新文本覆盖现有文本	R
合并当前行和下一行	J
转换大小写	~
重复上次操作	.
取消上次修改	u
恢复行到初始状态	U

虽然只使用这些表所列出的命令就可以运用 vi，但是为了掌握 vi 的真正功能（和提高自己的工作效率），将需要更多的工具。下面的章节将介绍这些工具。



第三章

快速移动

本章内容：

- 按屏幕移动
- 按文本块移动
- 按搜索移动
- 按行号移动
- 回顾 vi 的移动命令

你并不只是使用 *vi* 创建新文件，还会把许多时间花在用 *vi* 编辑现有的文件上。我们在处理文件时，需要直接到达文件的指定位置，然后开始工作。

所有的编辑操作都是从移动光标到达需要进行编辑的位置开始（或者使用 *ex* 行编辑命令指定将进行编辑的行号）。本章将向你展示如何考虑使用各种方式（按屏幕、按文本块、按模式或按行号）进行移动。由于在 *vi* 中，编辑速度取决于使用较少的击键到达目的位置，因此有许多种移动方法。

本章包括：

- 按屏幕移动
- 按文本块移动
- 按搜索模式移动
- 按行号移动

按屏幕移动

当我们读书时，一般按页考虑书中的位置：停止阅读的页或索引中的页码。但是在编辑文件的时候并没有这种便利。有一些文件只有几行，这样可以立刻看到整个文件，但是许多文件都有成百上千行，很难全部阅读。

你可以把文件看作长卷纸上的文本，屏幕是那个长卷纸上（通常）大小为24行文本的窗口。

在插入模式下，当使用文本填满屏幕时，将在该屏幕的最底行结束输入。当到达底部并按下 **RETURN** 时，最上面的行将会滚出视野，一个空行出现在屏幕的底部等待新文本的输入。这种操作就是滚动。

在命令模式下，通过在文件中向前或向后滚动屏幕，可以看到文件中的任何文本。而且，由于光标可以通过数字前缀加倍移动距离，因此可以快速地移动到文件中的任何位置。

滚动屏幕

CTRL **F** *vi* 有许多命令可以按全屏和半屏在文件中向前和向后滚动。

- ^F** 向前滚动一屏。
- ^B** 向后滚动一屏。
- ^D** 向前滚动半屏（下）。
- ^U** 向后滚动半屏（上）。

（在上面的命令列表中，**^** 符号表示 **CTRL** 键，**^F** 表示按下 **CTRL** 键后再同时按下 **f** 键。）

也有使屏幕向上（**^E**）和向下（**^Y**）滚动一行的命令。但是这两个命令并不把光标定位到行首，当调用这两个命令时，光标将保持在行中的原有位置。

使用 z 重新定位屏幕

z 如果想在向上或向下滚动屏幕时仍把光标保持在原来所在的行，则可使用 z 命令。

z [RETURN] 把当前行移动到屏幕顶部并滚动。

z . 把当前行移动到屏幕中央并滚动。

z - 把当前行移动到屏幕底部并滚动。

对于 z 命令，使用数字前缀作为倍数没有任何意义（毕竟你只需要重定位光标到屏幕顶部一次，重复同样的 z 命令将不进行任何移动）。相反，z 会把数字前缀理解为它将使用该行号表示的行取代当前行。例如，**z [RETURN]** 移动当前行到屏幕的顶部，但是 **200z [RETURN]** 将移动第 200 行到屏幕的顶部。

刷新屏幕

CTRL-L 有时你正在编辑时，一些来自计算机系统的信息将显示在屏幕上。虽然这些信息不能成为编辑缓冲区的一部分，但是它们会干扰你的工作。因此，当系统信息出现在你屏幕上时，你需要重新显示或刷新屏幕。

无论何时滚动屏幕，你都要刷新部分（或全部）屏幕。因此，你总是可以通过滚动把不需要的信息从屏幕上清除，然后再回到以前的位置。但是通过输入 **CTRL-L**，不用滚动也可以刷新屏幕。

在屏幕内移动

H 你也可以保持当前屏幕，或文件视图，使用下面的命令在屏幕内移动：

H 移动到起始点——屏幕首行。

M 移动到屏幕的中间行。

L 移动到屏幕的末行。

nH 移动到屏幕首行下面的第 n 行。

nL 移动到屏幕末行上面的第 *n* 行。

H 把光标从屏幕的任何位置移动到首行或“原点”，**M** 把光标移动到中间行，**L** 把光标移动到末行。使用 **2H** 可以把光标移动到首行下面一行。

击键 结果

L

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.
Screen editors are very popular, since they allow you to make changes as you read through a file.

使用 **L** 命令把光标移动到屏幕的末行。

2H

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.
Screen editors are very popular, since they allow you to make changes as you read through a file.

使用 **2H** 命令把光标移动到屏幕的第二行 (**H** 只把光标移动到屏幕的首行)。

按行移动

RETURN

也有许多命令可以在当前屏幕内按行移动。你已经见过 **j** 和 **k** 命令，其他可以使用的命令有：

RETURN

移动到下一行的首字符。

+

移动到下一行的首字符。

-

移动到前一行的首字符。

上面的三个命令会把光标移动到上一行或下一行的第一个字符，并忽略任何空格

或制表符 (tab)。相反, j 和 k 则把光标移动到上一行和下一行的第一个位置, 而不管那个位置是否为空 (假设光标开始于第一个位置)。

在当前行内移动

h 和 l 可向左和向右移动光标, 0 和 \$ 把光标移动到行首和行尾。也可以使用:

- ^ 移动到当前行的第一个非空格的字符。
- n | 移动到当前行的第 n 列。

与上面的行移动命令一样, ^ 把光标移动到当前行的第一个字符, 并忽略任何空格或制表符。相反, 0 则把光标移动到当前行的第一个位置, 而不管该位置是否为空格。

按文本块移动

 另一种可以在 vi 文件中移动的方式则是按文本块 (单词、句子、段落或节) 移动。

你已经学会按单词 (w、W、b 或 B) 向前或向后移动光标。另外, 也可以使用下面的命令:

- e 移动到词尾。
- E 移动到词尾 (忽略标点)。
- (移动到当前句子的开始。
-) 移动到下一句子的开始。
- { 移动到当前段落的开始。
- } 移动到下一段落的开始。
- [移动到当前节的开始。

]] 移动到下一节的开始。

为了找到句子的尾部, *vi* 将寻找标点符号?、. 和! 中的一个。当这些标点后至少有两个空格或作为一行的最后一个非空格字符出现时, *vi* 就认为找到了句尾的位置。如果仅在句号后跟一个空格, 或者句子以引号结束, *vi* 将不能识别该句子。

段落的定义是文本后为一空行, 或为 *troff* MS 宏包中的默认段落宏 (.IP、.PP、.LP 或.QP) 的一个。同样, 节的定义是文本后为默认的节宏 (.NH、.SH、.HI 和.HU)。标识段或节分隔符的宏可以用 :set 命令定义, 我们将在第七章“高级编辑”中介绍。

记住, 你可以把数字和移动命令结合在一起使用。例如, 3) 向前移动三个句子。还可以使用移动命令进行编辑, :d) 删除到当前句子结尾的文本, 2y} 复制前面的两个段落。

按搜索移动

 在大型文件中, 最有效的快速移动方法之一是按要搜索的文本(更确切地说是按字符模式)移动。有时可以使用搜索来寻找拼写错误的单词或程序中某个变量的实例。

搜索命令是专用字符 / (斜杠)。当输入斜杠时, 它将出现在屏幕的底行上, 然后就可以输入想要搜索的模式:/pattern。

模式可以是整个单词或任何其他的字符序列(也称为“字符串”)。例如, 如果搜寻字符 “red”, 可以将它作为整个单词进行匹配, 也可以对任何出现该字符中的地方进行匹配。如果在模式前后包含了空格, 那么空格将作为单词的一部分。如所有的底行命令一样, 按下 RETURN 键就会进行搜索。类似于所有其他的 UNIX 编辑器, *vi* 也有专用的模式匹配语言, 可以允许用户搜索不确定的文本模式, 例如, 任何以大写字母开始的单词或以 “The” 单词开始的行。

我们将在第六章“全局替换”中讨论更有用的模式匹配语法，这里我们只把模式看成单词或短语。

vi 从光标处开始向前搜索，如果需要，可以把光标移到文件的开始。光标将移动到模式第一次出现的位置，如果没有发现相匹配的文本，*vi* 就把“*Pattern not found*”信息显示在状态行上（注 1）。

下面使用文件“*practice*”来展示如何通过搜索移动光标：

击键 **结果**

/edits

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.

搜寻模式“*edits*”，按下 **RETURN** 键，光标将直接移动到该模式。

/scr

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.

搜寻模式“*scr*”，按下 **RETURN** 键。要注意“*scr*”后面没有空格。

搜寻折回到文件的前部。你可以使用字符的任何组合，一般并不要求搜索模式必须是完整的单词。

要向后搜索，可以使用?代替 /：

?pattern

在这两种情况下，如果需要，都将从文件的首部或尾部开始搜索。

注 1： 虽然确切的信息将随着不同的*vi* 克隆版本而改变，但是它们的意义却是相同的。通常，我们不用为在各处发现不同的文本信息而烦恼，因为在所有情况下它们所表达的意思都是相同的。

重复搜索

前面的搜寻模式在整个编辑会话中都是可用的。完成一次搜索后，可以对上一个模式使用再搜索命令，这样可以避免重复原来的击键。

n	同向重复搜索。
N	反向重复搜索。
/ <input type="button" value="RETURN"/>	向前重复搜索。
? <input type="button" value="RETURN"/>	向后重复搜索。

由于上次的搜索模式是可用的，因此可以在搜索某个模式后做些编辑工作，然后通过使用 n、N、/ 或?对同一模式进行再次搜索，而不用再次输入搜索命令。你的搜索方向（/ 向前，?向后）会显示在屏幕的左下角（注 2）。

继续使用上面的例子，由于模式 “scr” 仍可以使用，因此可以进行下列操作：

击键 结果

n

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.

使用 n（下一次）命令移动光标到模式 “scr”的下一个实例。

?you

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.

使用?从光标处向后搜寻 “you”的首次出现，在输入该模式后需要按下键。

N

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.

以相反的方向（向前）重复前面的搜索。

注 2： nvi 1.79 不显示 n 和 N 命令的方向。 vim 5.x 也将把搜索文本放到命令行上。

有时你只想向前搜索某个单词，而不愿使搜索折回到文件的前部。*vi* 有一个“*wrapscan*”选项，它可以控制是否需要折回搜索。可以按如下方式取消折回搜索：

```
:set nowrapscan
```

当设置为 *nowrapscan* 并且向前搜索失败时，状态行显示的信息为：

```
Address search hit BOTTOM without matching pattern
```

当设置为 *nowrapscan* 并且向后搜索失败时，状态行显示的信息用“TOP”取代了“BOTTOM”。

本节只是最简单地介绍搜索模式，第六章将介绍更多关于模式匹配的内容，以及它在文件中进行全局修改的用法。

通过搜索进行修改

你可以把 / 和 ? 搜索操作与 c 和 d 之类的文本修改命令结合在一起使用，继续使用上面的例子：

击键

结果

d?move

With a screen editor you can scroll the page, your edits as you make them.

删除从光标前到单词 “move” 之间的所有文本。

要注意删除操作以字符为基础进行，并没有对整行进行删除。

当前行内搜索

也有作用在当前行内的微型版的搜索命令。命令 *fx* 把光标移动到字符 *x* 的下一个实例（这里 *x* 代表任何字符）。命令 *tx* 把光标移动到 *x* 的下一实例前面的字符。可以反复使用分号来搜索字符。

行内搜索命令的总结如下，所有这些命令都只在当前行内移动光标。

- | | |
|----|--|
| fx | 在行内搜索（移动光标到） <i>x</i> 的下一个实例，这里 <i>x</i> 代表任何字符。 |
| Fx | 在行内搜索（移动光标到） <i>x</i> 的上一个实例。 |
| tx | 在行内搜索（移动光标到） <i>x</i> 的下一个实例前面的字符。 |
| Tx | 在行内搜索（移动光标到） <i>x</i> 的上一个实例后面的字符。 |
| ; | 同方向重复前面的搜索命令。 |
| , | 反方向重复前面的搜索命令。 |

使用这些命令中的任何一个，数字前缀*n*将定位到实例第*n*次出现的位置。假如你正在编辑文件“*practice*”中的下列行：

击键	结果
fo	<pre>With a screen editor you can scroll the</pre>
;	<pre>With a screen editor you can scroll the</pre>

`dfx`将删除到指定字符*x*（包括*x*）之间的文本，该命令常用来删除或复制行中的部分文本。如果行内有符号或标点，统计单词就会变得很困难，那么可能需要使用`dfx`来代替`dw`。`t`命令除了把光标定位在搜索字符的前面以外，它的作用与`f`相同。例如，`ct.`命令可以修改到句子结尾的所有文本，并保留句点。

按行号移动

文件中的行都被顺序地编号，因此可以通过指定行号在文件中移动。

行号通常用来标识想要编辑的大型文本块的开始和结尾，由于编译程序的错误信

息指定了行号，因此它对程序员来说也非常有用。*ex* 命令也使用行号，我们将在下一章中介绍这些内容。

如果想要按行号移动，则必须有标识它们的办法。使用第七章介绍的 :set nu 选项可以把行号显示在屏幕上。在 *vi* 中，也可以把当前行的行号显示在屏幕底部。

CTRL-G 命令使下列信息显示在屏幕底部：当前行号、文件总行数和当前行在总行中的百分比。例如，对于文件 “*practice*”，**CTRL-G** 会显示：

```
"practice" line 3 of 6 --50%--
```

如果你已经离开编辑会话，那么 **CTRL-G** 对显示命令中使用的行号或确定你的位置是很有用的。

根据你正在使用的 *vi* 工具，可能还会看到其他的额外信息，例如光标所在的列和显示文件是否已修改但还没有保存。信息的确切格式也将发生变化。

G(转移) 命令

G 你可以使用行号在文件中移动光标。**G** (转移) 命令使用行号作为数字参数而直接移动到那行，例如 44G 把光标移动到第 44 行的开始。不带行号的 G 命令把光标移动到文件的最后一行。

如果在这段时期没有进行编辑，那么输入两个反引号 (` `) 可使你返回到原来的位置（上次调用 G 命令的位置）。如果已进行了编辑，然后使用了不同于 G 的命令移动过光标，那么 ` ` 将把光标返回到上次编辑的位置。如果已经调用了搜索命令 (/ 或 ?)，则 ` ` 将把光标返回到开始进行搜索的位置。一对单引号 (' ') 除了把光标返回到行首而不是光标以前所在的确切位置外，它与两个反引号的作用很相似。

使用 **CTRL-G** 显示的总行数可以提示你大致需要移动多少行，如果位于 1000 行文件的第 10 行：

```
"practice" line 10 of 1000 --1%--
```

并且知道想要编辑的文本在文件的尾部附近，那么就可以使用 800G 来给出大概的目标位置。

按行号移动可以使你在大型文件中快速地从一个位置移动到另一个位置。

回顾 vi 移动命令

表 3-1 是对本章所介绍的命令进行了总结。

表 3-1 移动命令

移动	命令
向前滚动一屏	$\wedge F$
向后滚动一屏	$\wedge B$
向前滚动半屏	$\wedge D$
向后滚动半屏	$\wedge U$
向前滚动一行	$\wedge E$
向后滚动一行	$\wedge Y$
把当前行移动到屏幕顶部并滚动	$z [RETURN]$
把当前行移动到屏幕中央并滚动	$z .$
把当前行移动到屏幕底部并滚动	$z -$
刷新屏幕	$\wedge L$
移动到起始点——屏幕首行	H
移动到屏幕的中间行	M
移动到屏幕的末行	L
移动到下一行的首字符	$[RETURN]$
移动到下一行的首字符	$+$
移动到上一行的首字符	$-$
移动到当前行的第一个非空格字符	\wedge
移动到当前行的第 n 列	$n !$

表 3-1 移动命令（续）

移动	命令
移动到词尾	e
移动到词尾（忽略标点）	E
移动到当前句子的开始	(
移动到下一句的开始)
移动到当前段落的开始	{
移动到下一段落的开始	}
移动到当前节的开始	[[
移动到下一节的开始]]
向前搜索模式	/pattern
向后搜索模式	?pattern
重复上次搜索	n
反方向重复上次搜索	N
向前重复上次搜索	/
向后重复上次搜索	?
移动到当前行中 <i>x</i> 的下一个实例	f <i>x</i>
移动到当前行中 <i>x</i> 的上一个实例	F <i>x</i>
移动到当前行中 <i>x</i> 的下一个实例的前面	t <i>x</i>
移动到当前行中 <i>x</i> 的上一个实例的后面	T <i>x</i>
同方向重复前面的搜索命令	;
反方向重复前面的搜索命令	,
移动到第 <i>n</i> 行	<i>n</i> G
移动到文件的尾部	G
返回到以前的标记或上下文	..
返回到包含以前标记的行的开始	..
显示当前行（不是移动命令）	^G

本章内容：

- 更多的命令组合
- 启动 vi 时的选项
- 利用缓冲区
- 标记自己的位置
- 其他的高级编辑命令
- 回顾 vi 的缓冲区和标记命令

第四章

基本编辑命令

的扩展

我们已经学习了基本的 *vi* 编辑命令 i、a、c、d 和 y，本章将对所学的编辑命令进行扩展，其内容主要包括以下方面：

- 介绍其他编辑工具，研究通用的命令格式
- 进入 *vi* 的其他方法
- 利用存储复制和删除内容的缓冲区
- 标记自己在文件中的位置

更多的命令组合

在第二章中，我们学习了编辑命令 c、d 和 y，以及它们与移动命令和数字的组合使用（如 2cw 或 4dd）。在第三章“快速移动”中，我们学会了更多的移动命令。虽然编辑命令与数字的组合已不是一个新概念，但是表 4-1 将给出许多新的编辑选项。

表 4-1 更多的编辑命令

修改	删除	复制	从光标到……
cH	dH	yH	到屏幕顶部
cL	dL	yL	到屏幕底部
c+	d+	y+	下一行
c5l	d5l	y5l	当前行的第 5 列
2c)	2d)	2y)	接下来的第二个句子
c{	d{	y{	上一段落
c/pattern	d/pattern	y/pattern	模式
cn	dn	yn	下一个模式
cG	dG	yG	文件尾部
c13G	d13G	y13G	第 13 行

注意，上面的所有命令遵循下列通用模式：

(数字) (命令) (文本目标)

其中，数字是可选的数字参数，这里的命令是 c、d 或 y 中的一个，文本目标是个移动命令。

第二章已对 vi 命令的通用格式进行了讨论，你可能需要再回顾一下表 2-1 和表 2-2。

启动 vi 时的选项

在本书中，我们已经使用过下面的命令调用 vi 编辑器：

```
$ vi file
```

这个 vi 命令的其他选项可能是比较有用的。通过使用这些选项，可以在打开文件时直接进入指定的行号或模式，也可以按只读方式打开文件。另一个选项可以把系统崩溃时所有的修改恢复到正在编辑的文件中。

前进到指定位置

在对现有文件进行编辑时，可以先打开该文件，然后把光标移动到模式首次出现的位置或指定的行号，也可以通过命令行中恰当的搜索或行号来指定首次移动（注 1）：

\$ vi +n file

打开文件到第 *n* 行。

\$ vi + file

打开文件到最后一行。

\$ vi +/pattern file

打开文件到模式首次出现的位置。

要在打开文件 “*practice*” 时直接进入包含单词 “*Screen*” 的行，可以输入：

击键	结果
----	----

vi +/Screen practice

With a screen editor you can scroll the page, move the cursor, delete lines, and insert characters, while seeing the results of your edits as you make them.
Screen editors are very popular, since they allow you to make changes as you read

使用带 *+/pattern* 选项的 *vi* 命令直接定位到包含 *Screen* 的行。

正如在上面的例子中所看到的，搜索模式并不是必须要位于屏幕的顶部。如果搜索模式中含有空格，就必须用单引号或双引号把整个模式括起来（注 2）：

+/ "you make"

或者使用反斜杠对空格进行转义：

注 1： 根据 POSIX 标准，*vi* 应该使用 *-c command* 来代替上面所显示的 *+command*。通常考虑到向后兼容性，两个版本都可以使用。

注 2： shell 需要使用引号，而不是 *vi*。

```
+/you\ make
```

另外，如果想使用第六章“全局替换”所描述的通用模式匹配语法，那么可能需要使用单引号或反斜杠来防止 Shell 解释一个或多个特殊字符。

如果在编辑过程中必须离开编辑会话，那么使用`+/pattern`是很有帮助的。可以通过插入像 ZZZ 或 HERE 的模式来标记所在的位置，然后在返回文件时只需记着`/ZZZ`或`/HERE`即可。

注意：通常，在 vi 中进行编辑时，wrapscan 选项是激活的。如果你已经定制了所处的环境，使 wrapscan 始终是禁用的（注 3），那么就可能无法使用`+/pattern`。如果以这种方式打开文件，那么 vi 将打开文件到最后一行，并显示“Address search hit BOTTOM without matching pattern”的提示信息。

只读方式

有时想浏览一个文件，但是又要防止该文件会被不经意地修改（你可能想打开一个长文件来练习 vi 的移动命令，或者想在文件或段落中进行滚动），那么就可以按只读方式进入文件，然后使用 vi 所有的移动命令，但是这样就不能对该文件进行修改操作。

要以只读方式浏览文件，可以输入：

```
$ vi -R file
```

或者：

```
$ view file
```

（与 vi 命令一样，view 命令可使用任何命令行选项把光标移动到文件中的指定位置，注 4）。如果你决定要对文件进行编辑，那么可以通过在 write 命令上添加感叹号来忽略只读方式：

注 3：见第三章中的“重复搜索”一节。

注 4：通常 view 只是到 vi 的链接。

```
:w!
```

或者：

```
:wq!
```

如果在写文件时遇到问题，请参考附录四“问题列表”中所总结的问题列表。

恢复缓冲区

有时系统会在你编辑文件时崩溃，通常上次写（保存）操作之后的所有编辑都会丢失，但是 -r 选项可使你恢复在系统崩溃时已编辑的缓冲区。

在系统重新运行后首次登录时，将会收到缓冲区已被保存的信息。另外，如果输入下面的命令：

```
$ ex -r
```

或者：

```
$ vi -r
```

将得到系统所有已保存的文件列表。

使用带文件名的 -r 选项恢复已编辑的缓冲区，例如，为了在系统崩溃后恢复文件“*practice*”的已编辑缓冲区，可以输入：

```
$ vi -r practice
```

为了防止不经意地对文件进行了修改，然后不得不解决已保存缓冲区和新近编辑的文件之间的版本差别，立即恢复文件是明智之举。

在系统没有崩溃之前，可以使用 :pre 命令来强迫系统保存缓冲区。如果已对文件进行了修改，然后发现由于没有写权限而不能保存修改，这时 :pre 命令是很有效的（也可以使用另一个名字或在有写权限的目录下保存文件的副本，请参第一章“vi 文本编辑器”中的“保存文件时所遇到的问题”一节）。

注意：不同版本的恢复可能会产生不同的结果，也可能会随着版本的不同而改变。最好检查一下自己的本地文档。*vile* 不支持任何方式的恢复，*vile* 文档推荐使用 `autowrite` 和 `autosave` 选项，第七章将介绍如何进行这些操作。

利用缓冲区

在进行编辑时，上一次的删除（`d` 或 `x`）或复制（`y`）操作都将保存在缓冲区（存储内存中的某个位置）中。你可以访问该缓冲区中的内容，也可以使用粘贴命令（`p` 或 `P`）把已保存的文本粘贴到自己的文件中。

vi 把前 9 次删除操作存储在编号缓冲区中，可以通过访问这些编号缓冲区中的任何一个来恢复前 9 次删除命令中的任何一个（或全部）。（但是，小型的删除操作将不会保存在编号缓冲区中，例如只删除行的某部分，这些删除只能在执行删除操作后立即使用 `p` 或 `P` 命令来恢复。）

vi 也允许把复制操作（复制文本）放置在字符标识的缓冲区中，可以用已复制的文本填满 26 个（`a-z`）缓冲区，然后在编辑会话中的任何时刻使用粘贴命令粘贴那些文本。

恢复删除操作

以简单的限制删除大型的文本块是非常好的，但是如果误删了有用的 53 行该怎么办？方法就是：由于前 9 次删除操作都保存在编号缓冲区中，因此可以恢复其中的任何一次操作。最后一次删除将保存在 1 号缓冲区中，倒数第二次在 2 号缓冲区，依次类推。

为了恢复删除操作，要输入”（双引号），并使用数字指定缓冲区中的文本，然后给出粘贴命令。从 2 号缓冲区中恢复第二次删除操作是：

“2p

2 号缓冲区中的删除内容将粘贴到光标的后面。

即使不能确定所要恢复的删除操作存放在哪个缓冲区中，也不必反复输入 "np。如果你对 p 使用跟在 u 后面的重复命令 (.)，那么系统将自动增加缓冲区编号。因此，可以按如下方式搜索编号缓冲区：

```
"1pu.u.u etc.
```

这样就会依次把所有缓冲区中的内容粘贴到文件中。每输入一次 u，恢复的文本就被删除；当输入圆点 (.) 时，下一个缓冲区的内容就会恢复到文件中。一直输入 u 和 .，直到恢复了在寻找的文本为止。

复制到命名缓冲区

在进行任何其他编辑之前，必须粘贴 (p 或 P) 未命名缓冲区的内容，否则缓冲区的内容将被覆盖。也可以把 y 和 d 与 26 个命名缓冲区一起使用，这些缓冲区专门用于复制和删除文本。如果指定一个缓冲区来保存已复制的文本，那么就可以在编辑会话的任何时刻恢复该命名缓冲区的内容。

如果要把文本复制到一个命名缓冲区中，可在复制命令的前面加上双引号 (") 和表示你想使用的缓冲区名字的字符。例如：

```
"dyy    复制当前行到 d 缓冲区中。  
"a7yy   复制接下来的 7 行到 a 缓冲区中。
```

在把文本装入到指定缓冲区中并把光标移到新位置以后，可以使用 p 或 P 把文本粘贴回来：

```
"dp    粘贴 d 缓冲区中的内容到光标前。  
"ap    粘贴 a 缓冲区中的内容到光标后。
```

这里没有办法把缓冲区的部分内容粘贴出来，只能全部粘贴或不粘贴。

在下一章，我们将学习同时编辑多个文件。一旦你知道了如何在 vi 编辑环境下的多个文件之间移动，就可以使用指定缓冲区来有选择地在多个文件之间移动文本。

使用同样的过程，也可以把文本删除到指定缓冲区中：

"a5dd 删除 5 行到缓冲区 a 中。

如果使用大写字母来指定缓冲区的名字，那么所复制或删除的文本将追加到该缓冲区当前内容的后面。这使你在剪贴或复制时可以有所选择。例如：

"zd)

删除从光标到当前句尾之间的文本，并将其保存到 z 缓冲区中。

2)

向前移动两个句子。

"zy)

把下一个句子添加到 z 缓冲区中。你可以把更多的文本添加到命名缓冲区中，但是要当心：如果忘记了在复制或删除到缓冲区时使用大写格式指定其名字，那么将覆盖该缓冲区，因此在该缓冲区中累加的文本就会全部丢失。

标记自己的位置

在 vi 会话期间，可以使用一个不可见的“书签”来标记自己在文件中的位置。在其他地方编辑后可返回到所标记的位置。在命令方式下输入：

mx

用 x (x 可为任何字符) 标记当前位置。

' x

(单引号) 移动光标到 x 所标记的行的首字符。

` x

(反引号) 移动光标到 x 所标记的字符。

``

(两个反引号) 返回到移动前的上一标记或上下文的确切位置。

''

(两个单引号) 返回到上一标记或上下文所在行的开始。

注意：位置标记只能在当前 *vi* 会话中设置，不能存储在文件中。

其他的高级编辑

虽然在 *vi* 中还有许多可以使用的高级编辑，但是要使用它们，必须首先阅读下一章以了解 *ex* 编辑器的更多知识。

回顾 *vi* 的缓冲区和标记命令

表 4-2 对所有 *vi* 版本的通用命令行选项进行了总结，表 4-3 和 4-4 对缓冲区和标记命令进行了总结。

表 4-2 命令行选项

选项	含义
<code>+n file</code>	打开文件到第 <i>n</i> 行
<code>+file</code>	打开文件到最后一行
<code>+/pattern file</code>	打开文件到模式首次出现的地方
<code>-c command file</code>	打开文件后运行命令；通常是行号或搜索（POSIX+ 版）
<code>-R</code>	以只读方式进行（与用 <i>view</i> 代替 <i>vi</i> 一样）
<code>-r</code>	系统崩溃后恢复文件

表 4-3 缓冲区名

缓冲区名	缓冲区用途
1-9	前 9 次删除操作，由最近的到最早的
a-z	需要时使用的命名缓冲区，大写字母表示添加到该缓冲区尾

表 4-4 缓冲区和标记命令

命令	含义
" <i>b</i> 命令	对缓冲区 <i>b</i> 执行命令
m <i>x</i>	用 <i>x</i> 标记当前位置
' <i>x</i>	移动光标到 <i>x</i> 所标记的行的首字符
` <i>x</i>	移动光标到 <i>x</i> 所标记的字符
``	返回到上一标记或上下文的确切位置
'''	返回到上一标记或上下文所在行的开始



本章内容：

- ex 命令
- 使用 ex 进行编辑
- 保存和退出文件
- 把文件复制到另一个文件中
- 编辑多个文件

第五章

介绍 ex 编辑器

如果本书是 *vi* 的使用手册，那么我们为什么还要安排一章有关另一个编辑器的内容呢？实际上，*ex* 并不是另一个编辑器，*vi* 是更一般、更基本的 *ex* 行编辑器的可视模式。由于一些 *ex* 命令可以节省大量的编辑时间，因此在使用 *vi* 时它们是非常有用的。这些命令中的大部分都可以在不离开 *vi* 的情况下使用（注 1）。

我们已经知道如何把文本看成一系列已编号的行。*ex* 可以使你更灵活、更有选择地执行编辑命令。使用 *ex*，可以容易地在文件之间移动，并以各种方式把文本从一个文件移动到另一个文件，还可以快速地对大于单个屏幕的文本块进行编辑。通过全局替换，可以在整个文件中使用给定的模式进行替换。

本章介绍了 *ex* 和它的命令，我们将学会：

- 使用行号在文件中移动
- 使用 *ex* 命令对文本块进行复制、移动和删除
- 保存文件和部分文本

注 1： *vile* 与其他克隆版本是不同的。许多更高级的 *ex* 命令在 *vile* 中是不可用的。我们在这里不介绍每个命令，第十二章中我们将详细介绍。

- 操作多个文件（读入文本或命令、在文件间切换）

ex 命令

在发明*vi*和其他全屏编辑器之前的很长一段时间内，人们一直在打印终端上与计算机进行通信，而不是在今天的CRT（或使用定位设备和终端仿真程序的位图屏幕）上。由于行号可快速确定将要编辑的文件部分，因此行编辑器逐步发展为用来编辑这些文件的工具。程序员或其他计算机用户通常愿意在打印终端上打印出一行（或多行），并给出修改该行的编辑命令，然后通过重新打印来检查被编辑的行。

虽然人们已经不再在打印终端上编辑文件，但是一些*ex*行编辑命令对那些熟悉建立在*ex*之上的可视编辑器的用户仍然是非常有用的。虽然使用*vi*进行的大部分编辑都是比较简单的，但是当你想对文件的多个部分进行大规模修改时，*ex*的行定位就会显示出它的优势。

注意：我们在本章看到的许多命令都有文件名参数。虽然文件名可以包含空格，但是这通常不是个好的方式。*ex*会对没有结尾感到困惑，而你将遇到的麻烦也不只是识别文件名的问题。请使用下划线、破折号或句点分隔文件名的组成部分，这是一种很好的文件名格式。

在开始简单地记忆*ex*命令（或更坏点说，忽略它们）之前，首先让我们了解一下行编辑器。当直接调用*ex*时，观察它的工作方式将有助于理解命令语法的含义。

打开一个熟悉的文件练习一些*ex*命令。如同对文件调用*vi*编辑器一样，也可以对文件调用*ex*行编辑器。如果调用了*ex*，那么将会看到有关文件总行数的信息和冒号命令提示符。

例如：

```
$ ex practice
"practice" 6 lines, 320 characters
:
```

如果不给出显示一行或多行的 *ex* 命令，那么将不能看到文件中的任何行。

ex 命令由行地址（行号）和命令组成，它们都以回车键结束。最基本的命令之一是用来打印（到屏幕）的 p。例如，如果在提示符后输入 1p，你将会看到文件的第一行：

```
:1p  
With a screen editor you can  
:
```

实际上可以去掉 p，因为行号本身就等价于打印该行的命令。如果要打印多行，则可以指定行号的范围（例如 1, 3 —— 两个数字用逗号分开，中间的空格可有可无）。例如：

```
:1,3  
With a screen editor you can  
scroll the page, move the cursor,  
delete lines, insert characters, and more,
```

没有行号的命令被认为作用于当前行。因此，例如用一个单词替换另一个单词的替换命令 (s)，可以输入如下：

```
:1  
With a screen editor you can  
:s/screen/line/  
With a line editor you can
```

注意，命令执行后已修改的行将会重新显示。下面的命令可以完成同样的修改：

```
:ls/screen/line/  
With a line editor you can
```

虽然在 vi 中调用 *ex* 命令并不是在直接使用它们，但在 *ex* 编辑器中花几分钟还是值得的，因为只有这样才能了解应该如何告诉编辑器要编辑的行（或多行）以及要执行的命令。

当在 “*practice*” 文件上使用一些 *ex* 命令后，应该对同一个文件调用 vi，这样就能看到 *ex* 的更为熟悉的可视方式。命令 :vi 将使你从 *ex* 转换到 vi 中。

如果要在 *vi* 中调用 *ex* 命令，则必须输入专用的底行字符 : (冒号)，然后输入命令并按下 **RETURN** 键执行它。例如在 *ex* 编辑器中，只要在冒号提示符后输入行号就会移动到该行。如果要在 *vi* 中使用该命令把光标移动到文件的第 6 行，可以输入：

:6

然后按下 **RETURN** 键。

我们将使用下面的练习来讨论在 *vi* 中执行 *ex* 命令。

练习： ex 编辑器

在 UNIX 提示符下，对 “*practice*” 文件调用 *ex* 编辑器：

显示信息：

ex practice

"practice" 6 lines, 320 characters

到第一行并打印（显示）：

:1

打印（显示）1 到 3 行：

:1, 3

使用 *line* 替换第一行中的 *screen*：

:ls/screen/line

对文件调用 *vi* 编辑器：

:vi

到第一行：

:1

问题列表

- ✓ 在 *vi* 中进行编辑时，无意中退出了 *ex* 编辑器。

在 *vi* 命令模式下，**Q** 命令可调用 *ex*，在 *ex* 中的任何时刻，命令 **vi** 可使你返回到 *vi* 编辑器。

使用 *ex* 进行编辑

许多完成正常编辑操作的 *ex* 命令在 *vi* 中都有更简单地完成相同工作的等价命令。很明显，你会用 **dw** 或 **dd** 而不是使用 *ex* 中的 **delete** 命令来删除单个单词或行。

但是，由于 *ex* 命令允许你使用单个命令对大的文本块进行修改，因此在想对许多行进行修改时，*ex* 命令会更有用。

这些 *ex* 命令将列在下面，后面是这些命令的缩写。记住，在 *vi* 中每个 *ex* 命令前必须有冒号。你可以使用完整的命令或该命令的缩写，关键是哪一个比较容易记忆。

delete	d	删除行
move	m	移动行
copy	co	复制行
	t	复制行（co 的同义词）

你可以使用空格来分隔 *ex* 命令中的不同元素，如果这样可以更容易地阅读命令的话。例如，你可以按这种方式分隔行地址、模式和命令，但是不能在模式里把空格作为分隔符或在替换命令的结尾使用空格。

行地址

对于每一个 *ex* 编辑命令，必须给出 *ex* 所要编辑的行。对于 *ex* 的 move 和 copy 命令，还需要给出 *ex* 移动或复制文本的目标位置。

可以使用下面几种方式来指定行地址：

- 使用明确的行号
- 使用能帮助你指定相对于文件中当前位置的行号的符号
- 使用搜索模式作为标识所要编辑的行的地址

下面让我们来看一些例子。

定义行的范围

你可以使用行号明确地定义一行或行的范围。使用明确数字的地址称为绝对行地址，例如：

:3,18d	删除 3~18 行
:160,224m23	移动 160~224 行到 23 行的下面（如同 vi 中的 delete 和 put）
:23,29co100	复制 23~29 行并把它们粘贴到 100 行的后面（如同 vi 中的 yank 和 put）

为了更容易地使用行号进行编辑，可以把所有的行号显示在屏幕的左边。命令：

```
:set number
```

或它的缩写：

```
:set nu
```

可以显示行号。文件 “*practice*” 将显示：

```
1 With a screen editor
2 you can scroll the page,
3 move the cursor, delete lines,
4 insert characters and more
```

显示的行号在写文件的时候并不能保存，如果打印文件，也不能打印它们。行号会一直显示直到退出 vi 会话或取消了 set 选项：

```
:set nonumber
```

或者：

```
:set nonu
```

为了暂时地显示部分行的行号，可以使用 # 符号。例如：

```
:1,10#
```

将显示从 1 到 10 的行号。

正如第三章 “快速移动” 中所描述的，也可以使用 **CTRL-G** 命令来显示当前行号。因此，通过移动到文本块的开始并输入 **CTRL-G**，然后再移动到文本块的结束并输入 **CTRL-G** 帮助你确定对应于文本块开始和结束的行号。

另一种指定行号的方法是使用 *ex* 的 = 命令：

:= 显示总行数。

:.= 显示当前行的行号。

:/*pattern*/=

显示第一个与模式相匹配的行的行号。

行地址符

也可以使用表示行地址的符号。点 (.) 代表当前行；\$ 代表文件的最后一行；% 代表文件中的每一行，它等同于 1, \$ 的组合。这些符号也可以与绝对行地址组合使用，例如：

:., \$d

删除从当前行到文件末尾之间的文本。

:20, .m\$

把从 20 行到当前行的文本移动到文件的结尾。

:%d

删除文件中的所有行。

:%t\$

复制所有行并把它们粘贴到文件的尾部（成为连续的副本）。

除了绝对行地址以外，也可以指定一个相对当前行的地址。符号 + 和 - 就像数学运算符一样，当将其放置在数字前面时，这些符号就会加上或减去跟在它们后面的数值。例如：

:., .+20d

删除从当前行开始向下的 20 行。

:226, \$m.-2

把 226 行到文件末尾的文本移动到当前行上面的第二行。

:., +20#

显示从当前行下面 20 行的行号。

实际上，由于当前行是假定的开始位置，因此你在使用 + 或 - 时不需要输入圆点 (.)。

如果在 + 和 - 的后面没有数字，它们就分别相当于 +1 和 -1 (注 2)。同样，++ 和 -- 也通过添加一行和减少一行来改变指定范围。如下一节所介绍的，+ 和 - 也可以用于搜索模式中。

数字 0 代表文件开头 (虚构的 0 行)。0 等同于 1-，这些命令都允许移动或复制行到文件的真正开始位置，即现有文本第一行的前面。例如：

: -, +t 0

复制三行 (从光标上面的行到光标下面的行)，并把它们粘贴到文件的开头。

搜索模式

ex 定位行的另一种方法是使用搜索模式。例如：

: /*pattern*/d

删除下一个包含模式 *pattern* 的行。

: /*pattern*/+d

删除下一个包含模式 *pattern* 的行的下面的行 (也可以使用 +1 来代替单独的 +)。

: /*pattern1*/, /*pattern2*/d

将从第一个包含模式 *pattern1* 的行到第一个包含模式 *pattern2* 的行之间的内容删除。

注 2： 在相对地址中，不能把加号或减号与其后面的数字分开。例如，+10 表示“接下来的 10 行”，而 + 10 则表示“接下来的 11 (1+10) 行”，这可能不是我们预计 (或期望) 的结果。

```
:., /pattern/m23
```

将从当前行(.)到第一个包含模式*pattern*的行之间的文本移动到23行的后面。

注意，模式的前面和后面要用斜杠来定界。

如果在*vi*和*ex*中使用模式进行删除操作，那么将会发现这两个编辑器的操作方式是不同的。假设你的“*practice*”文件中含有以下行：

With a screen editor you can scroll the page, move the cursor, delete lines, insert characters and more, while seeing results of your edits as you make them.

击键

d/while

结果

With a screen editor you can scroll the page, move the cursor, while seeing results of your edits as you make them.

*vi*的模式删除命令把从光标位置到单词*while*之间的文本删除掉，并保留两行的剩余部分。

:.,/while/d

With a screen editor you can scroll the of your edits as you make them.

*ex*命令会删除指定行的全部，在这种情况下是指当前行和包含模式的行。在它们范围之内的所有行都将删除。

重新定义当前行的位置

有时，在命令中使用相对行地址可能会带来意想不到的结果。例如，假设光标在第一行，我们想显示第100行和它下面的5行。如果输入：

```
:100,+5 p
```

我们将得到“First address exceeds second.”的错误信息。命令失败的原因是由于第二个行地址是相对于当前光标位置（第一行）计算的，因此命令实际上是：

```
:100,6 p
```

我们需要的是使该命令把第 100 行当做“当前行”的方法，即使光标位于第一行时也是如此。

ex 提供了这样的方法。当你使用分号代替逗号时，就会将第一个行地址当成当前行而进行重算。例如，命令：

```
:100;+5 p
```

就会显示所期望的行。现在 +5 是相对于第 100 行进行计算的。分号也可以用于搜索模式及相对地址中，例如，显示下一个包含模式 *pattern* 的行和它下面的 10 行，可以输入命令：

```
:/pattern/;+10 p
```

全局搜索

我们已经知道如何在 *vi* 中使用 / (斜杠) 来搜索文件中的字符模式。*ex* 也有全局命令 g，可以让你搜索模式并显示找出的所有包含该模式的行。命令 :g! 的作用与 :g 相反，使用 :g! (或它的同义词 :v) 可搜索不包含模式的行。

可以对文件中的所有行使用全局命令，也可以使用行地址把全局搜索限制在指定的行或行范围内。

```
:g/pattern
```

寻找（移动到）模式 *pattern* 在文件中最后出现的位置。

```
:g/pattern/p
```

寻找并显示文件中所有包含模式 *pattern* 的行。

```
:g!/pattern/nu
```

寻找并显示文件中所有不包含模式 *pattern* 的行，同时还显示这些行的行号。

```
:60,124g/pattern/p
```

寻找并显示第 60 行到 124 行之间所有包含模式 *pattern* 的行。

正如可能料到的，也可以将 g 用于全局替换，我们将在第六章“全局替换”中对此进行讨论。

组合 ex 命令

你不必一直输入冒号来开始新的 ex 命令。在 ex 中，竖直条 (|) 是命令分隔符，它允许用户把多个命令组合在同一个 ex 提示符下（与此类似的是分号分隔 UNIX shell 提示符下的多个命令）。在使用 | 时，要注意所指定的行地址。如果一个命令影响到文件中行的顺序，那么下一个命令将使用新的行位置进行工作。例如：

```
:1,3d | s/thier/their/
```

把第一行到第三行删除（保留文件中的顶行）；然后在当前行（该行是调用 ex 提示符以前的第 4 行）进行替换。

```
:1,5 m 10 | g/pattern/nu
```

把第 1 行到第 5 行移动到第 10 行的后面，然后显示所有包含模式 pattern 的行（和行号）。

注意，使用空格会使命令更容易阅读。

保存和退出文件

我们已经学习了退出并写（保存）文件的 vi 命令 ZZ。但是由于 ex 命令可以进行更好地控制，因此用户经常想使用它们退出文件。前面已经提到了这些命令，现在让我们来更系统地学习它们。

:w 把缓冲区写（保存）到文件中但不退出，可以在编辑会话期间使用 :w 来保护编辑操作以避免系统瘫痪或重大的编辑错误。

:q 退出编辑器（并返回 UNIX 提示符）。

:wq 写文件并退出编辑器，即使没有修改文件，写操作也会无条件的进行。

:x 写文件并退出编辑器，只有修改了文件写操作才能进行（注 3）。

vi 对缓冲区中的现有文件和你所做的编辑操作进行保护。例如，如果要把缓冲区写到现有文件中，*vi* 就会给出警告。同样，如果对文件调用 *vi* 并进行了编辑，但不保存编辑就退出，那么 *vi* 将给如下的错误信息：

```
No write since last change.
```

虽然这些警告能防止严重的错误，但是有时可能需要执行这些命令，在命令的后面加上感叹号（!）将忽略这些警告：

```
:w!  
:q!
```

:w! 也可以把编辑保存到使用 *vi -R* 或 *view* 以只读方式打开的文件中（假设你对该文件具有写权限）。

:q! 命令是个基本的编辑命令，它允许你在退出时不影响原来的文本，忽略在本次会话中进行的所有编辑。缓冲区中的内容将被删除。

重命名缓冲区

也可以使用 :w 将整个缓冲区（正在编辑的文件的副本）保存到新文件中。

假设打开了包含 600 行的“*practice*”文件，并且进行了大量的编辑。如果要退出但又想同时保存老版本的“*practice*”和新编辑的版本以进行比较，使用下面的命令可以把已编辑的缓冲区保存到名为“*practice.new*”的文件中：

```
:w practice.new
```

“*practice*”文件中的老版本保持不变（假如以前没有使用 :w），现在可以通过输入 :q 来退出对新版本的编辑。

注 3：当编辑源代码和使用根据文件的修改时间来执行操作的 *make* 时，:wq 和 :x 之间的区别是很重要的。

保存部分文件

在编辑时，有时只想把文件的一部分保存为一个单独的新文件。例如，你可能输入了作为几个文件开头的格式化代码和文本。

为了保存文件的部分文本，可以把 *ex* 行地址和写命令 *w* 结合起来使用。例如，如果要把文件 “*practice*” 中的一部分保存到新文件 “*newfile*” 中，可以输入：

```
:230,$w newfile
```

将从第 230 行到文件末尾的文本保存到 “*newfile*” 中。

```
:.,600w newfile
```

将从当前行到第 600 行的文本保存到 “*newfile*” 中。

添加到现有文件中

可以使用 UNIX 的重定向添加符 (>) 和 *w* 把缓冲区的全部或部分内容添加到现有文件尾。例如，如果输入：

```
:1,10w newfile
```

然后再输入：

```
:340,$w >>newfile
```

那么 “*newfile*” 文件将包含从 1 到 10 行和从 340 行到缓冲区尾部的内容。

把文件复制到另一个文件中

有时想把系统中已有的文本或数据复制到正在编辑的文件中，在 *vi* 中可以使用 *ex* 命令读取另一个文件的内容：

```
:read filename
```

或者使用缩写形式：

```
:r filename
```

该命令把*filename*文件的内容插入到本文件中光标位置后面的行的开始。如果想指定不是光标所在的行，那么只需在read和r命令之前输入行号（或其他行地址）即可。

假设你正在编辑文件“*practice*”并想从名为“/home/tim”的目录下读取文件“*data*”。将光标定位到插入数据的行的前面一行，并输入：

```
:r /home/tim/data
```

“/home/tim/data”的全部内容就将读取到文件“*practice*”中光标所在行下一行的开始位置。

如果要读取同一个文件并把它放在第185行的后面，可以输入：

```
:185r /home/tim/data
```

读取文件的其他方法有：

```
:$r /home/tim/data
```

将要读取的文件放到当前文件的尾部。

```
:0r /home/tim/data
```

将要读取的文件放到当前文件的开始。

```
:/pattern/r /home/tim/data
```

将要读取的文件放到当前文件中包含模式*pattern*的行的后面。

编辑多个文件

*ex*命令可在多个文件之间进行切换。同时编辑多个文件的好处是速度快。如果正在与其他的用户共享系统，那么退出当前文件然后再以要编辑的文件进入*vi*是需要时间的。在同一编辑会话中，在文件之间切换不仅能加速访问，而且还能保留

已指定的缩写和命令序列（参考第七章“高级编辑”），以及复制缓冲区，这样可以在文件之间复制文本。

调用 vi 打开多个文件

在首次调用 vi 时，可以指定要编辑的多个文件，然后使用 ex 命令在文件之间进行切换。例如：

```
$ vi file1 file2
```

首先编辑 *file1*。在完成第一个文件的编辑后，*ex* 的 :w 命令写（保存）*file1*，:n 命令调用下一个文件 (*file2*)。

假如要编辑 “*practice*” 和 “*note*” 两个文件。

击键	结果
-----------	-----------

vi practice note

With a screen editor you can scroll
the page, move the cursor, delete lines,
insert characters, and more, while seeing

打开文件 “*practice*” 和 “*note*”，指定的第一个文件 “*practice*” 出现在屏幕上，然后进行一些编辑。

:w

"practice" 6 lines, 328 characters

使用 *ex* 的 w 命令保存编辑后的文件 “*practice*”。按下 [RETURN] 键。

:n

Dear Mr.
Henshaw:
Thank you for the prompt . . .

使用 *ex* 的 n 命令调用下一个文件 “*note*”，按下 [RETURN] 键，进行一些编辑。

:x

"note" 23 lines, 1343 characters

保存第二个文件 “*note*”，退出编辑会话。

使用参数列表

参数列表中, *ex* 命令不仅仅是使用:*n* 移动到下一个文件。*:args* 命令(缩写:*:ar*)列出在命令行上指定的文件, 当前文件使用方括号括起来。

击键 结果

vi practice note

With a screen editor you can scroll
the page, move the cursor, delete lines,
insert characters, and more, while seeing

打开文件 “*practice*” 和 “*note*”, 指定的第一个文件 “*practice*” 出现在屏幕上。

:args

[*practice*] *note*

vi 在状态行上显示参数列表, 当前文件名使用方括号括起来。

:rewind(*:rew*) 命令把当前文件重新设置为命令行上指定的第一个文件。*elvis* 和 *vim* 提供了相应的*:last* 命令, 用来移动到命令行上的最后一个文件。

调用新文件

你不必在编辑会话的开始调用多个文件, 可以使用 *ex* 的*:e* 命令在任何时候切换到另一个文件。如果想在 *vi* 中编辑另一个文件, 那么首先需要保存当前文件(*:w*), 然后给出命令:

:e filename

假如你正在编辑文件 “*practice*” 的时候想去编辑文件 “*letter*”, 然后再返回到文件 “*practice*”。

击键 结果

:w

“*practice*” 6 lines, 328 characters

使用 *w* 命令保存文件 “*practice*” 并按下 **RETURN** 键, 该文件被保存并显示在屏幕上。由于已经保存了编辑操作, 因此现在可以切换到另一个文件。

```
:e letter
```

"letter" 23 lines, 1344 characters

使用 e 命令调用文件 “letter” 并按下 RETURN 键，进行一些编辑。

在一个时刻，vi 把这两个文件名记为当前和备用文件名，也可以通过使用符号 % (当前文件名) 和 # (备用文件名) 来指定。由于 # 允许你在两个文件之间来回切换，因此在使用 :e 时 # 尤其有用。在刚才的例子中，可以通过输入 :e # 返回到第一个文件 “practice” 中，也可以通过输入 :r # 把文件读取到当前文件中。

如果没有首先保存当前文件，并且没有在 :e 和 :n 后添加感叹号来命令 vi，那么将不允许使用上述命令在文件之间进行切换。

例如，如果在对文件 “letter” 进行一些编辑后想放弃这些编辑并返回到文件 “practice” 文件中，那么可以输入 :e! #。

下面的命令也很有用，它将放弃编辑操作并返回到当前文件上次保存过的版本中：

```
:e!
```

与 # 号相比，% 主要用于把当前缓冲区的内容写到新文件中。例如，在前面 “重命名缓冲区” 中，我们展示了如何使用命令来保存文件 “practice”的第2个版本：

```
:w practice.new
```

由于 % 代表当前文件名，因此也可以输入：

```
:w %.new
```

在 vi 中切换文件

CTRL **^** 由于切换到前面的文件会经常发生，因此不必移动到 ex 命令行下进行这种操作。vi 的 ^ 命令 (“control” 键加插入符号键) 可以实现这种功能。使用这个命令与输入 :e # 相同。与 :e 命令相同，如果没有保存当前缓冲区，那么 vi 将不允许切换到前面的文件中。

在文件之间进行编辑

当为复制缓冲区指定了单字母名字时，就拥有了把文本从一个文件移动到另一个文件的便捷方法。当使用:`e`命令把新文件加载到`vi`缓冲区时，将不会清除命名的缓冲区。因此，通过从一个文件中复制或删除文本（如果需要，可以到多个命名缓冲区中），使用:`e`调用新文件，然后把命名缓冲区粘贴到新文件中，这样就可以在文件之间传输文本了。

下面的例子展示了如何把文本从一个文件传递到另一个文件。

击键 结果

"f4YY

```
With a screen editor you can scroll  
the page, move the cursor, delete lines,  
insert characters, and more, while seeing  
the results of the edits as you make them
```

复制 4 行到缓冲区 f。

:w

```
"practice" 6 lines, 238 characters
```

保存文件。

:e letter

```
Dear Mr.  
Henshaw:  
I thought that you would  
be interested to know that:  
Yours truly,
```

使用:`e`进入文件“letter”中，移动光标到要放置复制的文本的位置。

"fp

```
Dear Mr.  
Henshaw:  
I thought that you would  
be interested to know that:  
With a screen editor you can scroll  
the page, move the cursor, delete lines,  
insert characters, and more, while seeing  
the results of the edits as you make them  
Yours truly,
```

把缓冲区 f 中已复制的文本粘贴到光标的下面。

另一个把文本从一个文件移动到另一个文件的方法是使用 *ex* 的 :ya (复制) 和 :pu (粘贴) 命令。虽然这些命令与等价的 *vi* 命令: y 和 p 的工作方式相同，但是它们要与 *ex* 的行地址范围和命名缓冲区一起使用。

例如：

```
:160,224ya a
```

将把第 160 行到 224 行复制到缓冲区 a 中，接下来可以使用 :e 命令移动到想要放置这些行的文件、把光标定位到放置已复制行的某一行上，然后输入：

```
:pu a
```

把缓冲区 a 中的内容粘贴到当前行的后面。



第六章

全局替换

本章内容：

- 确认替换
- 上下文相关替换
- 模式匹配规则
- 模式匹配举例
- 总结模式匹配

有时，在文档的中间或者草稿的尾部，可能会发现存在涉及的某样东西不统一的问题。或者，在一个手册中，遍布整个文件的某个产品的名字突然改变。这种事情经常发生，以致于不得不返回去修改已经写好的东西，并且需要在多处进行修改。

完成这些修改的方法是使用称为全局替换的有效修改命令。使用这个命令，可以自动地替换单词（或字符串），无论它出现在文件的什么位置。

在全局替换中，*ex* 编辑器使用指定的字符模式对文件中的每行进行检查。在所有行中发现模式的位置，*ex* 使用新字符串来替换模式。从现在开始，我们将把搜索模式看成是一个简单的串，在本章的后面我们将学习强大的模式匹配语言——正则表达式。

全局替换真正使用的是两个 *ex* 命令：`:g`（全局）和：`s`（替换）。由于全局替换命令的语法可以变得相当复杂，因此我们对它进行分阶段学习。

替换命令的语法如下：

```
:s/old/new/
```

这将把当前行中模式 *old* 的第一次出现修改为 *new*。/（斜杠）是命令不同部分之间的分隔符（当斜杠为该行的最后一个字符时，它是可选的）。

下面这种形式的替换命令：

```
:s/old/new/g
```

把当前行 *old* 的每次出现改为 *new*，而不只是该行的第一个 *old*。`:s` 命令允许替换串后面带有选项，上面语法中的 `g` 选项代表全局（`g` 选项影响一行中的每个模式，不要把它与影响文件中所有行的 `:g` 命令相混淆）。

通过在 `:s` 命令前面加上地址前缀，可以把它的范围扩展到多行。例如，下面这行命令将把第 50 行到第 100 行的 *old* 的每次出现改为 *new*：

```
:50,100s/old/new/g
```

下面的命令将把整个文件中的 *old* 的每次出现改为 *new*：

```
:1,$s/old/new/g
```

也可以使用 `%` 代替 `1, $` 来指定文件中的每一行，因此上面的命令也可以变为：

```
:%s/old/new/g
```

全局替换要比分别寻找并替换字符串的每个实例快的多。由于该命令可以进行不同类型的修改，并且它的功能非常强大，因此，我们将首先介绍简单的替换，然后逐步上升到复杂的、上下文相关的替换。

确认替换

使用搜索替换命令时一定要非常仔细，有时得到的结果并不是想要的。可以通过输入 `u` 来取消任何搜索替换命令，前提是该命令是最后进行的编辑操作。但是并不是总能在还可以恢复的时候就发现意外的修改。另一个保护所编辑的文件的办

法是在进行全局替换前使用:w保存文件。这样至少可以不保存编辑而退出文件，然后再返回文件修改前的状态。你还可以使用:e!读取缓冲区的先前版本。

小心并确定文件中需要修改的内容是非常明智的。如果想在替换前看到搜索结果和确认每个替换，则可以在替换命令的尾部加上c选项（用于确认）：

```
:1,30s/his/the/gc
```

它将显示字符串所在的一整行，并且该字符串将由一系列的插入符号(^^^^)所标记：

```
copyists at his school
      ^^^_
```

如果想替换该字符串，必须输入y（代表是）并按下[RETURN]键；如果不想要替换，则只需按下[RETURN]即可（注1）。

```
this can be used for invitations, signs, and menus.
      ^^^_
```

*vi*的n（重复上次搜索）和点(.)（重复上次命令）命令的结合也是一种极为有用的方法，它可以快速地在文件中翻页来执行不想全部替换的修改。例如，如果在该使用*that*的时候使用了*which*，那么可以抽查*which*的每次出现，只修改那些不正确的：

/which	搜索 <i>which</i>
cwthat[ESC]	修改为 <i>that</i>
n	重复搜索，跳过一次修改
n	重复搜索
.	重复修改（如果正确）
.	
.	

注1：*elvis* 2.0不支持这个用法。在其他的克隆版本中，虽然实际的外观和提示不同，但结果是相同的，在每种情况下都允许用户选择是否进行替换。

上下文相关替换

最简单的全局替换是使用一个单词（或短语）替换另一个。如果文件中有一些拼写错误（*editor* 误写为 *editer*），那么可以进行全局替换：

```
:%s/editer/editor/g
```

这样就把文件中 *editer* 的每次出现替换为 *editor*。

还有稍微有点复杂的全局替换语法。这些语法可以对一个模式进行搜索，一旦找到含有模式的行，就可以使用不同于模式的串进行替换。我们可以把这种替换看作上下文相关替换。

其中的语法如下所示：

```
:g/pattern/s/old/new/g
```

第一个 g 是在文件的所有行上执行的命令，模式 *pattern* 识别要发生替换的行。在那些包含模式 *pattern* 的行上，*ex* 将把 *old* 替换 (s) 为 *new*。最后的 g 表示在该行上进行全部替换。

例如在本书中，SGML 指示要把 <keycap> 和 </keycap> 放在 **ESC** 键的周围来表示 ESCAPE 键，如果想让所有的 **ESC** 键都在键帽中，但是又不想改变文本中的任何 *Escape* 实例。为了只在含有 <keycap> 指示的行上把 *Esc* 的实例修改为 *ESC*，可以输入：

```
:g/<keycap>/s/Esc/ESC/g
```

如果用来进行搜索的模式与想要修改的模式相同，那么就不必重复它。命令：

```
:g/string/s//new/g
```

将搜索包含 *string* 的行并对 *string* 进行替换。

注意：

```
:g/editer/s//editor/g
```

和

```
:%s/editer/editor/g
```

的效果相同。

可以使用第二种形式少输入一些字符。也可以把:`g`命令与:`d`、`:m o`、`:co`和包括:`s`在内的其他`ex`命令结合在一起使用。正如我们将要介绍的，可以进行全局删除、移动和复制。

模式匹配规则

在进行全局替换时，类似于`vi`的UNIX编辑器允许你搜索的不只是固定的字符串，还允许搜索由正则表达式指代的可变的单词模式。

当你指定一个字面的字符串时，搜索可能会找到不想匹配的其他实例。例如在文件中搜索单词时，单词可以有不同的使用方式。正则表达式有助于在上下文中搜索单词。要注意正则表达式可以与`vi`的搜索命令`/`、`?`以及`ex`中的`:g`、`:s`命令一起使用。

在最大程度上，相同的正则表达式都可以在其他的UNIX程序中使用，如`grep`、`sed`和`awk`（注2）。

正则表达式由普通字符和许多称为元字符的专用字符结合在一起组成（注3）。元字符及其使用方法如下所示。

注2：有关正则表达式的更多信息可以在Dale Dougherty和Arnold Robbins所著的《sed & awk》和Jeffrey E.F. Friedl所著的《Mastering Regular Expressions》中找到。

注3：科学地说，我们也许应该把这些字符称为元字符序列，因为有时两个字符合在一起才有特殊含义，而不只是单个字符。但是，由于术语“元字符”在UNIX著作中普遍使用，因此我们在这里也遵循了这种习惯。

元字符在搜索模式中的使用

· 匹配除换行符之外的任何一个单个字符。空格也将作为字符。例如，`p.p` 可以匹配 `pep`、`pip` 和 `pcp` 那样的字符串。

* 匹配其前面的单个字符的0个或多个实例。例如，`bugs*` 将匹配 `bugs`（一个 `s`）或 `bug`（没有 `s`）。

* 可以跟元字符。例如，由于 `.`（点）代表任何字符，因此 `.*` 表示“匹配任何数量的任何字符”。

这里有一个关于它的特殊例子。命令：`s/End.*/End/` 会删除 `End` 后面的所有字符（使用空来替换该行的剩余部分）。

^ 当用于正则表达式的开始时，它要求后面的正则表达式要出现在一行的开始。例如，当 `Part` 出现在一行的开始时，`^Part` 才匹配它；`^. . .` 匹配一行的前三个字符。当 `^` 不在正则表达式的开始时，`^` 只代表本身。

\$ 当用于正则表达式的结尾时，它要求前面的正则表达式要出现在一行的结尾。例如，当 `here:` 出现在一行的结尾时，`here:$` 才匹配它。当 `$` 不在正则表达式的结尾时，`$` 只代表本身。

\ 它将后面的专用字符看成普通字符。例如，`\.` 与一个真正的点匹配，而不是“任何单个字符”；`*` 与一个真正的星号匹配，而不是“一个字符的任意多个”。`\`（反斜杠）会防止对专用字符的解释，这种解释称为“转义该字符”（使用 `\\` 可得到字面的反斜杠）。

[] 匹配方括号所包括的字符中的任何一个。例如，`[AB]` 匹配 `A` 或 `B`，`p[aeiou]t` 匹配 `pat`、`pet`、`pit`、`pot` 或 `put`。可以通过使用连字符分开该范围的首尾字符来指定连续的字符范围。例如，`[A-Z]` 将匹配 `A` 到 `Z` 之间的任何一个大写字母，`[0-9]` 将匹配 `0` 到 `9` 之间的任何一个数字。

可以在方括号内包括多个范围，也可以把范围和单独的字符混合在一起。例如 `[: ; A-Z a-z ()]` 将匹配四种不同的标点符号和所有的字母。

大部分元字符都将在方括号内失去它们的特殊含义，因此如果想把它们作为普通字符使用，就不需要对它们进行转义。在方括号内，仍需要转义的三个

元字符是 \、- 和]。连字符 (-) 的含义是范围说明符，如果要使用真正的连字符，也可以将其放在方括号内第一个字符的位置。

插入符号 (^) 只有它在方括号内为第一个字符时才有特殊含义，但是在这种情况下其含义与通常 ^ 元字符的含义不同。作为方括号内的第一个字符，^ 使它们的含义反过来：方括号将匹配不在列表内的任何一个字符。例如，[^a-z] 匹配任何不是小写字母的字符。

\(\)

把 \(和 \) 之间的模式保存到专门的存储空间或“存储缓冲区”中。这种方式最多可以把 9 个模式保存到单一行中。例如，模式：

```
\(That\)\ or \(\this\ \)
```

把 *That* 保存到 1 号存储缓冲区，把 *this* 保存到 2 号存储缓冲区。可以在替换中使用序列 \1 到 \9 来重新引用存储的模式。例如，如果要用 *this or That* 修改 *That or this*，则可以输入：

```
:%s/\(That\)\ or \(\this\)\/\2 or \1/
```

也可以在搜索或替换串中使用 \n 符号：

```
:s/\(abcd\)\1/alphabet-soup/
```

把 *abcdabcd* 改为 *alphabet-soup*（注 4）。

\< \>

在单词的开始 (\<) 或结尾 (\>) 匹配字符。单词的结束或开始由标点符号或空格决定。例如，表达式 \<ac 将只匹配以 ac 开头的单词，如 *action*。表达式 ac\> 将只匹配以 ac 结尾的单词，如 *manica*。两个表达式都不匹配 *react*。要注意与 \(...\)` 不同的是，这些命令不必成对使用。

`~` 匹配在上次搜索中使用的任何正则表达式。例如，如果搜索过 *The*，那么可以使用 /~n 来搜索 *Then*。要注意只能在普通的搜索（使用 /）中使用这种模

注 4：这可以在 vi、nvi 和 vim 中使用，但是不能在 elvis 2.0、vile 7.4 和 vile 8.0 中使用。

式（注 5）。在替换命令中它将不能起到模式的作用。但是，在替换命令的替换部分则具有同样的含义。

一些克隆版本支持可选的、扩展后的正则表达式语法。要得到更多的信息，请参看第八章的“扩展的正则表达式”一节。

POSIX 的方括号表达式

我们刚刚介绍了方括号的用途是用来匹配它所包围的字符中的任何一个，如 [a-z]。POSIX 标准对匹配不在英语字母表中的字符引入了其他工具。例如，法语 è 是字母表中的字符，但是普通的字符类 [a-z] 不能匹配它。而且，该标准还提供在匹配和整理（排序）字符串数据时应该被看成单一单元的字符序列。

POSIX 同时也对术语进行了形式化。在 POSIX 标准中，方括号内的字符组称为“括号表达式”。在括号表达式中，除了像 a,! 等那样的字面字符以外，还可以有其他的组成部分。它们是：

- **字符类。** POSIX 字符类由 [: 和 :] 包围的关键字组成。关键字描述了不同的字符类，如字母表字符、控制字符等等（参考表 6-1）。
- **整理符号。** 整理符号是应该被看成单一单元的多字符序列，它由 [. 和 .] 所包围的字符组成。
- **等价类。** 等价类列出了一组应将其看成等价的字符，如 e 和 è。它由 [= 和 =] 包围的指定元素组成。

这三个概念必须出现在括号表达式的方括号里。例如，[[:alpha:]!] 匹配任何一个单个字母字符或感叹号，[[.ch.]] 匹配整理元素 ch，但是不再与字母 c 或 h 相匹配。在法语中，[[=e=]] 可能匹配 e、è 或 é 中的任何一个。类和匹配字符如表 6-1 所示。

注 5：这是最初 vi 的古怪功能。使用这种模式后，保存的搜索模式将被赋值为 ~ 后输入的新文本，而不是人们认为的新的组合模式。而且，所有的克隆版本都不这样处理。因此，当存在该功能时，最好少用。

表 6-1 POSIX 字符类

类	匹配字符
[:alnum:]	字母数字字符
[:alpha:]	字母字符
[:blank:]	空格和制表符
[:cntrl:]	控制字符
[:digit:]	数字字符
[:graph:]	可打印和可见（非空格）字符
[:lower:]	小写字符
[:print:]	可打印字符（包括空白）
[:punct:]	标点字符
[:space:]	空白字符
[:upper:]	大写字符
[:xdigit:]	十六进制数字

你必须进行调查以确定使用的 vi 版本是否具有这种功能。可能需要使用一个专门的选项来启动 POSIX 命令，对某个特殊环境变量进行设置，或者使用特殊目录下的 vi 版本。

在 HP-UX 9.x（和更新）系统上的 vi 支持 POSIX 括号表达式，Solaris 系统上的 /usr/xpg4/bin/vi（不是 /usr/bin/vi）也支持它。该功能在 nvi 和 elvis 2.1 中也是可用的。随着越来越多的商用 UNIX 提供者遵循标准，该功能将变得更加普遍。

元字符在替换串中的使用

当进行全局替换时，上面的正则表达式只有在命令的搜索部分（第一部分）时才具有特殊含义。

例如，在输入这样的内容时：

```
:%s/1\. Start/2. Next, start with $100/
```

要注意，替换串按照字面意思处理字符 . 和 \$，而不对它们进行转义。同样，可以输入：

```
:%s/[ABC]/[abc]/g
```

如果想把 A 替换为 a、B 替换为 b 和 C 替换为 c，由于括号在替换串中与普通字符相同，因此该命令将把出现的每个 A、B 或 C 改为 5 个字符的 [abc] 串。

要解决这样的问题，需要一种指定可变替换串的方法。幸运的是，有许多其他的元字符在替换串中具有特殊含义。

- \n 它被前面用 \ (和 \) 保存的第 n 个模式相匹配的文本替换，这里 n 是从 1 到 9 的数字，先前保存的模式（保存在存储缓冲区中）从行的左边开始计数。请参考本章前面对 \ (和 \) 的解释。
- \ 它把其后的专用字符看成普通字符。与在搜索模式中一样，反斜杠在替换串中也是元字符。为了指定一个真正的反斜杠，要连续输入两个 (\ \)。
- & 当用于替换串中时，它代表与搜索模式相匹配的整个文本。这在试图避免重复输入文本时很有用：

```
:%s/Yazstremski/&, Carl/
```

替换文本将是 “Yazstremski, Carl”。& 也可以替换可变模式（由正则表达式所指定）。例如，要使用圆括号包围从第 1 行到第 10 行的每一行，可以输入：

```
:1,10s/.*/(&)/
```

该搜索模式匹配整个行，& 对该行进行“重现”，紧接着是你的文本。

- 它与在搜索模式中的含义相同，使用上次替换命令中指定的替换文本替换找到的串。这对重复编辑很有用，例如，可以在一行中指定 :s/thier/their/，然后在另一行中使用 :s/thier/~/ 来重复修改。搜索模式也不需要是同一个。例如，你可以在一行中指定 :s/his/their/，然后在另一行中使用 :s/her/~/。注 6 进行重复替换。

注 6：ed 编辑器的最新版本把 % 用做替换文本中的专用字符，用来指代“上次替换命令中的替换文本”。

\u 或 \l

把替换串中的下一个字符分别变为大写或小写。例如，为了把 *yes*, *doctor* 变为 *Yes*, *Doctor*, 可以输入：

```
:%s/yes, doctor/\uyes, \udoctor/
```

然而，由于只在第一个位置使用首字母大写键输入替换串比较容易，因此这是个没有意义的例子。与任何一个正则表达式一样，\u 和 \l 的最大用途是与可变串一起使用。例如，采用我们前面使用过的命令：

```
:%s/\(That\)\ or \\(this\)/\2 or \1/
```

结果是 *this* 或 *That*，但是需要调整大小写。我们将使用 \u 把 *this* 中的第一个字母变为大写（当前保存在 2 号存储缓冲区中）；使用 \l 把 *That* 中的第一个字母变为小写（当前保存在 1 号存储缓冲区中）：

```
:%s/\(That\)\ or \\(this\)/\u\2 or \l\1/
```

结果是 *This* 或 *that*。（不要把数字 1 与小写 l 混淆，数字 1 在后面。）

\U 或 \L 和 \e 或 \E

除了将从它们后面开始到替换串结束或到 \e 或 \E 出现为止的所有字符都转换为大写或小写外，\U 和 \L 与 \u 和 \l 相似。如果没有 \e 或 \E，替换文本中的所有字符都要受到 \U 或 \L 的影响。例如，要把 *Fortran* 变为大写，可以输入：

```
:%s/Fortran/\UFortran/
```

或者使用 & 字符来重复搜索串：

```
:%s/Fortran/\U&/
```

所有的模式搜索都是区分大小写的，即对 *the* 的搜索不会发现 *The*。可以通过在模式中指定大写和小写来处理这种问题：

```
/[Tt]he
```

也可以通过输入 :set ic 让 vi 忽略大小写。如果要想了解更多的细节，请参考第七章“高级编辑”。

更多的替换技巧

我们应该知道一些有关替换命令的其他重要内容：

1. 简单的 :s 与 :s//~/ 相同，即重复上次替换。当在整个文档中重复相同的修改而又不想使用全局替换命令时，这样可以节省大量的时间和输入。
2. 如果把 & 认为是“同一事情”的意思（与匹配的文本），那么该命令相对来说就比较容易记忆了。可以把 g 跟在 & 的后面，从而在整行内进行替换，甚至还可以把它与行范围一起使用：

:%&g 在各处重复上次替换

3. & 键也可以作为 vi 命令来执行 :& 命令，即重复上次替换。这可以比 :s [RETURN] 节省更多的输入，一次击键相当于三次击键。
4. 除了有些细微的差别外，:~ 命令与 :& 命令相似。:~ 所使用的搜索模式是上次任何命令中所使用的正则表达式，而不局限于上次替换命令中所使用的正则表达式。

例如（注 7），在以下序列中：

```
:s/red/blue/  
:/green  
:  
:
```

:~ 等价于 :s/green/blue/。

5. 除 / 字符外，还可以使用除反斜杠、双引号和竖直条 (\、" 和 |) 之外的任何非字母表、非空白字符作为分隔符。在对路径名进行修改时，这一点尤其便利。

```
:%s;/user1/tim;/home/tim;g
```

6. 当启动 edcompatible 选项时，vi 将记住上次替换中使用的标志（g 代表全局、c 代表确认），然后把它们应用到下次替换中。

当你正在文件中移动并想进行全局替换时，这将会非常有用。你可以进行第一次修改：

注 7： 在 nvi 文档中，Keith Bostic 提供了很好的例子。

```
:s/old/new/g  
:set edcompatible
```

然后，接下来的替换命令将是作用于全局的。

尽管名字如此，但没有一种常用的 UNIX *ed* 版本真正按这种方式工作。

模式匹配举例

如果你对正则表达式不熟悉，那么上面对特殊字符的讨论可能会相当复杂。更多的实例会使这些事情变得较为清楚。在下面的例子中，方框（□）表示空格而不是一个特殊字符。

下面我们将介绍如何在替换中使用一些特殊字符。假如有一个长文件，如果想把整个文件中的单词 *child* 替换为单词 *children*。那么首先使用:w 保存已编辑的缓冲区，然后试着进行全局替换：

```
:%s/child/children/g
```

在继续编辑时，你会发现出现了类似 *childrenish* 这样的单词，原来无意中匹配了单词 *childish*。使用:e! 返回到上次保存过的缓冲区，并输入：

```
:%s/child□/children□/g
```

（注意 *child* 后也有空格。）但是该命令会忽略出现的 *child.*, *child,* 和 *child:* 等等。回忆一下，方括号允许从列表中指定一个字符，于是我们找到了一个解决方案：

```
:%s/child[□,.;:!?]/children[□,.;:!?]/g
```

这样将对后面跟有空格（由□表示）或标点字符 . ; : ! ? 中任何一个的 *child* 进行搜索。虽然我们想使用后面跟有同样空格或标点符号的 *children* 替换它，但是由于已经在每个 *children* 后使用一串标点符号作为结束，因此需要把空格和标点符号保存到 \ (和 \) 里，然后用 \1 重现它们。这是另一个尝试：

```
:%s/child\([□,.;:!?\]\)/children\1/g
```

当搜索匹配到 \(\backslash\) (和 \(\backslash\)) 中的字符时，右边的 \(\backslash1\) 将恢复同样的字符。虽然上述语法好像有点复杂，但是这条命令序列可以节省许多工作和时间。

然而，该命令仍不是完美的。你已经注意到了 *Fairchild* 的出现已被修改，因此你需要一种在它不是其他单词的一部分时才对它进行匹配的办法。

结果证实，*vi*（并不是所有其他的编辑器都使用正则表达式）有一个专门用来表示“只有该模式是个完整的单词”的语法。字符序列 \< 要求模式匹配单词的开头，而 \> 要求模式匹配单词的结尾。使用这两个语法就会把匹配限制为完整的单词。因此，在上面指定的任务中，\<child\> 将发现单词 *child* 的所有实例，而不管它后面跟的是空格还是标点符号。下面是应该使用的替换命令：

```
:%s/\<child\>/children/g
```

搜索单词的常规类

假设你的子函数名以前缀 *mgi*、*mgr* 和 *mga* 开头。

```
mgibox routine,  
mgrbox routine,  
mgabox routine,
```

如果既想保留前缀，又想把名字 *box* 改为 *square*，那么下面两个替换命令中的任何一个都能完成该任务。第一个例子说明了如何使用 \(\backslash\) (和 \(\backslash\)) 保存实际已匹配的模式，第二个例子展示如何使用一个模式进行搜索而使用另一个模式进行替换：

```
:g/mg\([ira]\)box/s//mg\1square/g
```

```
mgisquare routine,  
mgrsquare routine,  
mgasquare routine,
```

该全局替换实现了无论是 *i*、*r* 还是 *a* 都需要保留。按照这种方法，只有当 *box* 为例程名的一部分时它才会被修改为 *square*。

```
:g/mg[ira]box/s/box/square/g
```

```
mgisquare routine,  
mgrsquare routine,  
mgasquare routine,
```

虽然该命令与上面的命令有同样的效果，但是由于它可以改变同一行中其他的 *box* 实例，而不只是修改例程名中的 *box* 实例，因此它具有较小的安全性。

按模式移动块

你也可以按模式移动确定的文本块。例如，假设有一个 150 页的参考手册，每页都有三段，每段都有同样的标题：SYNTAX、DESCRIPTION 和 PARAMETERS。一个参考页的例子如下所示：

```
.Rh 0 "Get status of named file" "STAT"  
.Rh "SYNTAX"  
.nf  
integer*4 stat, retval  
integer*4 status(11)  
character*123 filename  
...  
retval = stat (filename, status)  
.fi  
.Rh "DESCRIPTION"  
Writes the fields of a system data structure into the  
status array.  
These fields contain (among other  
things) information about the file's location, access  
privileges, owner, and time of last modification.  
.Rh "PARAMETERS"  
.IP "\fBfilename\fR" 15n  
A character string variable or constant containing  
the UNIX pathname for the file whose status you want  
to retrieve.  
You can give the ...
```

假如要把 DESCRIPTION 移动到 SYNTAX 段的前面。通过模式匹配，可以使用一个命令移动 150 页中的所有文本块。

```
:g /SYNTAX/.,/DESCRIPTION/-1 move /PARAMETERS/-1
```

该命令的工作过程如下：首先，*ex* 寻找并标记所有与第一个模式相匹配的行（即包含单词 *SYNTAX* 的行）。其次，把每个已标记的行设置成 .（点，当前行），然后执行命令。通过 *move* 命令，将从当前行（点）到含有单词 *DESCRIPTION* (/DESCRIPTION/-1) 的行的前一行之间的行块移动到含有 *PARAMETERS* 的行的前一行 (/PARAMETERS/-1)。

注意，*ex* 只能把文本放到指定行的下面。如果要使 *ex* 把文本放到行的上面，则首先要使用 -1 减去一行，然后 *ex* 才会把文本放到前一行的下面。在这样的例子中，一个命令就可节省几小时的工作（这是一个实际的例子，我们曾经使用这样的模式来重新调整上百页的参考手册）。

使用模式定义的块也同样可以与其他的 *ex* 命令一起使用。例如，如果想删除参考章节中所有的 *DESCRIPTION* 段，那么可以输入：

```
:g/DESCRIPTION/,/PARAMETERS/-1d
```

虽然这种非常有效的修改是 *ex* 的行地址语法所固有的，但即使对于熟练的用户，其作用也不是非常明显。鉴于这个原因，在遇到了复杂、重复的编辑任务时，都要花时间来对问题进行分析，是否可以使用模式匹配工具来处理相关内容。

更多的例子

由于通过实例学习模式匹配是最好方法，因此我们在这里列出了许多模式匹配的例子及其解释。仔细地研究这些语法，就可以理解它们的工作原理。然后你应该把这些例子应用到自己的操作环境中。

1. 把 *troff* 斜体字代码放到单词 *RETURN* 的四周：

```
:%s/RETURN/\\"fI&\\fP/g
```

注意，在替换中需要用两个反斜杠 (\\"), 因为 *troff* 斜体字代码中的反斜杠将解释为特殊字符（单独的 \fI 将解释为 fI，必须输入 \\fI 才能得到\fI）。

2. 修改文件中的路径名列表：

```
:%s/\\home\\tim/\\home\\linda/g
```

当斜杠作为全局替换序列中的分隔符时，必须使用反斜杠对它进行转义，即使用 \/ 才能得到 /。具有同样效果的可替换办法是使用不同的字符作为模式分隔符。例如，在上面的替换中，可以使用冒号作为分隔符。如下所示：

```
:%s:/home/tim:/home/linda:g
```

该方法的可读性较强。

3. 把 HTML 斜体字代码放到单词 *RETURN* 的四周：

```
:%s:RETURN:<I>&</I>;g
```

注意，这里使用 & 表示真正匹配的文本，并且正如刚才所描述的，冒号可用做分隔符来代替斜杠。

4. 把 1 到 10 行内的所有点号都改为分号：

```
:1,10s/\./;;g
```

点在正则表达式语法中有特殊含义，必须使用反斜杠（\）对它进行转义。

5. 把单词 *help*（或 *Help*）的所有出现改为 *HELP*：

```
%s/[Hh]elp/HELP/g
```

或者：

```
:%s/{Hh}elp/\U&/g
```

\U 把它后面的模式全部变为大写。后面的模式是对搜索模式 *help* 或 *Help* 的重复。

6. 使用单个空格替换一个或多个空格：

```
:%s/□□*/□/g
```

这将确保你理解星号作为一个特殊字符是如何工作的。跟在任何字符后面（或者跟在任何匹配单一字符的正则表达式后面，如 . 或 [a-z]）的星号都与该字符的零个或多个实例相匹配。因此，必须指定两个空格跟一个星号来匹配一个或多个空格（一个空格加零个或多个空格）。

7. 使用两个空格替换冒号后的一个或多个空格：

```
:%s/:□□*/:□□/g
```

8. 使用两个空格替换句号或冒号后的一个或多个空格：

```
:%s/ \ ([.:]\)\*\*/ \1\0/g
```

可以对方括号内两个字符中的任何一个进行匹配。该字符将通过使用 \ (和 \) 而保存到存储缓冲区中，并使用 \1 将其恢复到右边。要注意方括号内像点那样的特殊字符不需要对其转义。

9. 对单词或标题的各种用法进行标准化：

```
:%s/^Note\s*:*/Notes:/g
```

由于方括号内有三个字符：空格、冒号和字母 s，因此模式 Note\s*: 将会匹配 Note\s:、Note: 或 Notes。由于星号位于模式的后面，因此它也可匹配 Note（在它后有零个空格）和 Notes:（恰好是正确的拼写）。如果没有星号，则将完全忽略 Note，Notes: 也将被错误地改为 Notes:\s:。

10. 删除所有空行：

```
:g/^$/d
```

这里真正需要匹配的是行尾 (\$) 紧跟着行首 (^)，它们之间什么也没有。

11. 删除所有空行和任何只包含空白的行：

```
:g/^[\t ]*\$/d
```

（在上一行中，制表符将显示为 tab。）一行可能显示为空，但实际上含有空格或制表符，但前一个例子不删除这样的行。类似于前一个例子，本例子搜索行的开始和结尾。但是除了在它们之间没有任何内容以外，该模式还试着发现任何数量的空格和制表符。如果没有相匹配的空格或制表符，那么该行就是空行。为了删除含有空白的非空行，必须至少使用一个空格或制表符来匹配该行：

```
:g/^[\t ]*\t */d
```

12. 删除每行中所有的前导空格：

```
:%s/^[\t ]*\(.*\)/\1/
```

使用 ^[\t]* 搜索每行开始的一个或多个空格，然后使用 \(.*\) 保存行的其余部分到第一个存储缓冲区中。使用 \1 恢复不带前导空格的行。

13. 删除每行尾部的所有空格：

```
:%s/\(.*\)\口\口*$/\1/
```

对于每一行，使用 \(.*)\口\口 保存该行所有的文本，但是不保存行尾的一个或多个空格。恢复所保存的不带空格的文本。

由于本例和上例中的替换在任何给定的行上只发生一次，因此g选项不需要跟在替换串的后面。

14. 在文件中每行的开始插入 >\口\口：

```
:%s/^/>\口\口/
```

这里我们只是使用 >\口\口 替换行的开始内容。当然，并没有真正替换行的开始（是逻辑结构，不是真正的字符）。

该命令在回复邮件或USENET消息记录时很有用。人们通常希望在回复时包含部分的原始信息。按照惯例，要在行的开始使用一个右尖括号和两个空格标识包含的文本，从而把包含的部分与你的回复区别开。这种处理可以按照上面所展示的那样很容易地实现。（通常，只需要包含部分的原始信息。可以在上面的替换前后把不需要的文本删除。）先进的邮件系统可以自动完成这些功能。但是，如果你正在使用 vi 编辑邮件，那么可以使用该命令来完成它。

15. 在接下来的 6 行的尾部添加一个句号：

```
:.,+5s/$/. /
```

行地址指示当前行和下面的 5 行，\$ 指示行的结尾。类似于前面的例子，\$ 是个逻辑结构，因此并没有对行的结尾进行替换。

16. 交换列表中所有连字符分隔的条目的顺序：

```
:%s/\(.*\)\口-\口\(.*\)/\2\口-\口\1/
```

使用 \(.*)\口-\口 之前的文章保存到第一个存储缓冲区中，然后使用 \(.*)\口 该行的其余部分保存到第二个存储缓冲区中。恢复行的已保存部分，并交换两个存储缓冲区的顺序。该命令对多个条目的处理结果如下所示。

```
more - display files
```

变为：

```
display files - more
```

以及：

```
lp - print files
```

变为：

```
print files - lp
```

17. 把文件中的每个单词变为大写：

```
:%s/.*/\U&/
```

或者：

```
:%s/.*/\U&/g
```

在替换串开头的 \U 标志告诉 vi 要把替换字符变为大写， & 字符把与搜索模式相匹配的文本作为替换字符进行重现。这两个命令是等价的；但是由于第一种形式对每行只进行一次替换（.* 匹配整行，每行一次），而第二种形式对每行都要进行重复替换（. 只匹配一个单个字符，由于尾部的 g 才进行重复替换），因此第一种形式要快得多。

18. 颠倒文件中的行序（注 8）：

```
:g/.*/m00
```

该搜索模式匹配所有的行（包含零个或多个字符的行）。一行接一行，每行都将移动到文件的头部（即，移动到虚构的第 0 行的后面）。由于每个匹配的行都将放置在顶部，因此它将把前面移动过的行向下移，这样一行接一行，直到将最后一行移到文件的顶部。由于每行都有开头，因此也可以用下面的方式更简单地实现这种处理：

```
:g/^/m00
```

19. 对数据库中所有没有标记 “Paid in full” 的行追加短语 “Overdue”：

```
:g!/Paid\in\full/s/$/Overdue/
```

或者等价于：

```
:v/Paid\in\full/s/$/Overdue/
```

要影响所有与模式不匹配的行，可在 g 命令后加上!，或者只使用 v 命令。

注 8： 摘自 Walter Zintz 于 1990 年 6 月在《UNIX World》发表的一篇论文。

20. 寻找所有不以数字开始的行，然后把该行移到文件的尾部：

```
:g!/^[0-9]/m$
```

或者：

```
:g/^[^0-9]/m$
```

由于^作为方括号里的第一个字符起到了否定的作用，因此这两个命令有相同的效果。第一个命令的意思是“不匹配以数字开始的行”，第二个命令的意思是“匹配不以数字开始的行”。

21. 把人工编号的节标题（如1.1、1.2等）修改为*troff*宏（即.Ah代表A级标题）：

```
:%s/^*[1-9]\.[1-9] /.Ah/
```

该搜索串所匹配的串为非零数字加句号再加非零数字。要注意替换串中的句号不需要进行转义（尽管加上\也不会产生任何影响）。该命令不能发现含有两个或更多数字的章号。如果要搜索章号，可以把该命令修改为：

```
:%s/^*[1-9][0-9]*\.[1-9] /.Ah/
```

现在它将匹配10到99章（数字1到9，后面再跟一个数字）、100到999章（数字1到9，后面再跟两个数字）等等。该命令也能找到1到9章（数字1到9，后面不跟数字）。

22. 去掉文档中节标题的编号。如果想把下一行：

```
2.1 Introduction  
10.3.8 New Functions
```

修改为：

```
Introduction  
New Functions
```

那么实现命令如下：

```
:%s/^*[1-9][0-9]*\.[1-9][0-9]* //
```

该搜索模式类似于上例中的搜索模式，只是现在数字的长度在不断变化。以最低限度为例，标题要包含数字、句号和数字，因此可以按上例中的搜索模式开始：

```
[1-9][0-9]*\.[1-9]
```

但是在该例中，标题可能延续任何数量的数字或句号：

[0-9.]*

23. 把单词*Fortran*修改为短语*FORTRAN (acronym of FORMula TRANslatiOn)*:

```
:%s/\\(For\\)\\(tran\\)/\\U\\1\\2\\E\\(acronym\\of\\U\\1\\Emula\\U\\2\\Eslation\\)/g
```

首先，由于我们注意到单词*FORMula*和*TRANslatiOn*含有部分的初始单词，因此决定分两块来保存搜索模式：`\(For\)`和`\(tran\)`。在第一次恢复它时，我们同时使用这两块，并把所有的字符转换为大写：`\U\1\2`。接下来，我们使用`\E`取消大写，否则余下的替换文本都将变成大写。替换过程以实际输入的单词继续，然后我们恢复第一个存储缓冲区。由于该缓冲区包含的仍是*For*，因此我们要再次把它转换为大写：`\U\1`。紧接着我们把单词的其余部分转换为小写：`\Emula`。最后，我们恢复第二个存储缓冲区。由于该缓冲区中的内容为*tran*，因此我们先按大写形式对其进行“重现”，接着又使用小写输出单词的余下部分：`\U\2\Eslation`。

总结模式匹配

我们通过列举一些与复杂的模式匹配概念有关的实际例子来对本章进行总结。这里我们并没有立即解决问题，而是逐步寻找问题的解决方案。

删除未知的文本块

假如有几行通用格式的文本：

```
the best of times; the worst of times: moving
The coolest of times; the worst of times: moving
```

我们所关心的行都以*moving*结尾，但是不知道每行的前两个单词是什么。而我们想把任何以*moving*结尾的行修改为：

```
The greatest of times; the worst of times: moving
```

由于修改必须在特定的行进行，因此需要指定一个上下文相关的全局搜索。使用 :g/moving\$/对以*moving*结尾的行进行匹配。然后，我们发现搜索模式可以是任意数量的任何字符，因此可以使用元字符`.*`。但是，如果并没有以某种方式对匹配进行限制，那么将会对整行进行匹配。下面是第一次处理：

```
:g/moving$/s/.*of/The□greatest□of/
```

我们使用的搜索串将会从行的开始匹配到第一个*of*。由于需要指定单词*of*来对搜索进行限制，因此我们只在替换过程中对它进行重复。下面是最后的结果：

```
The greatest of times: moving
```

这里有些错误，替换进行到该行的第二个*of*而不是第一个。原因是在默认情况下，“匹配任意数量的任何字符”的行为将匹配尽可能多的文本。在这种情况下，由于单词*of*在行中出现了两次，因此搜索串找到的是：

```
the best of times; the worst of
```

而不是：

```
the best of
```

需要进一步限制搜索模式：

```
:g/moving$/s/.*of times;/The greatest of times;/
```

现在`.*`将对短语*of times;*的实例之前的所有字符进行匹配；由于行中只有一个实例，因此它必定是第一个。

然而在一些情况下，使用`.*`元字符是有困难的，或者是不正确的。例如，你可能发现自己输入了太多的单词来限制搜索模式，或者可能无法通过特殊的单词来限制模式（如果行中的文本变化非常大）。下一节将列举这方面的例子。

交换数据库中的条目

假如想交换（文本）数据库中所有姓与名的顺序。这些行如下所示：

```
Name: Feld, Ray; Areas: PC, UNIX; Phone: 123-4567
```

```
Name: Joy, Susan S.; Areas: Graphics; Phone: 999-3333
```

每个字段名都以冒号结尾，各字段用分号隔开。以第一行为例，我们要把 *Feld, Ray* 修改为 *Ray Feld*。下面将列出一些看上去可以但实际上不能完成工作的命令，在每个命令的后面，我们将给出该行在修改前与修改后的情况。

```
:%s/: \(.*\), \(.*\);/: \2 \1;/
```

Name: Feld, Ray ; Areas: PC, UNIX; Phone: 123-4567	修改前
Name: <i>UNIX Feld, Ray</i> ; Areas: PC; Phone: 123-4567	修改后

我们已使用粗体强调了第一个存储缓冲区中的内容，并用斜体字强调了第二个存储缓冲区中的内容。注意第一个存储缓冲区中包含的内容比我们需要的多。由于存储缓冲区没有被它后面的模式完全限制，因此可以保存到第二个逗号。现在，试着限制第一个存储缓冲区中的内容：

```
:%s/: \(\.\.\.\), \(.*\);/: \2 \1;/
```

Name: Feld, Ray ; Areas: PC, UNIX; Phone: 123-4567	修改前
Name: <i>Ray</i> ; Areas: PC, UNIX Feld ; Phone: 123-4567	修改后

我们已经设法把姓保存到第一个存储缓冲区中，但是第二个存储缓冲区还会把该行最后一个分号之前的文本保存起来。现在对第二个存储缓冲区也进行限制：

```
:%s/: \(\.\.\.\), \(\.\.\.\);/: \2 \1;/
```

Name: Feld, Ray ; Areas: PC, UNIX; Phone: 123-4567	修改前
Name: <i>Ray Feld</i> ; Areas: PC, UNIX; Phone: 123-4567	修改后

这个结果是我们想要的，但是这只能在有四个字母的姓和三个字母的名的特定情况下才可以实现（前面的尝试包含同样的错误）。为什么不返回到第一次尝试的方法，对搜索模式的结尾加以更多的限制呢。

```
:%s/: \(.*\), \(.*\); Area/: \2 \1; Area/
```

Name: Feld, Ray ; Areas: PC, UNIX; Phone: 123-4567	修改前
Name: <i>Ray Feld</i> ; Areas: PC, UNIX; Phone: 123-4567	修改后

虽然这样可以完成任务，但是我们将通过引入一个额外的担忧继续讨论这个问题。假如*Area*字段不是一直存在或不是一直处于第二个字段，那么上面的命令就不会对这样的行起作用。

我们引入这个问题是为了说明一点，即无论什么时候重新考虑模式匹配时，选择合适的变量（元字符）通常要比使用特殊文本限制模式好。在模式中使用变量越多，命令就会越有效。

在当前例子中，再考虑一下想要匹配的模式。每个单词都以大写字母开始然后是任意数量的小写字母，因此可以按如下方式对名进行匹配：

```
[A-Z] [a-z]*
```

姓可能有多个大写字母（如*McFly*），因此要在第二个和随后的字母中搜索这种可能性：

```
[A-Z] [A-Za-z]*
```

使用该模式也不会对名有任何影响（不知道何时出现*McGeorge Bundy*），命令现在变成：

```
:%s/: \(([A-Z][A-Za-z]*\), \(([A-Z][A-Za-z]*\));/: \2 \1;/
```

满足充分限制的条件吗？上述命令还没有包含像*Joy, Susan S*这种名字的情况。由于名字段可能在中间有大写字母，因此需要在方括号的第二部分中加上空格和句号。但是要适可而止，有时确切指定匹配的内容比指定不想匹配的内容要难。由于在实例数据库中姓以逗号结束，因此可以把姓字段看成不包含逗号的字符串：

```
[^,]*
```

该模式要把字符匹配到第一个逗号。同样，名字段也是一个没有分号的字符串：

```
[^;]*
```

把这些更有效的模式放到前面的命令中，可以得到：

```
:%s/: \([^\,]*\), \([^\;]*\);/: \2 \1;/
```

同样的命令也可以作为一个上下文的相关替换。如果所有的行都以*Name*开始，则可以输入：

```
:g/^Name/s/: \([^\,]*\), \([^\;]*\);/: \2 \1;/
```

为了匹配带有额外空格（或没有空格）的冒号，也可以在第一个空格后加上一个星号：

```
:g/^Name/s/: *\([^\,]*\), \([^\;]*\);/: \2 \1;/
```

使用:g 来重复命令

正如我们通常所看到的:*:g* 命令的应用，它所选择的行通常都要由该行上的后继命令进行编辑。例如，我们使用*g* 选择行，然后对它们进行替换，或选择和删除它们：

```
:g/mg[ira]box/s/box/square/g  
:g/^$/d
```

然而，Walter Zintz 在《UNIX World》（注 9）使用指南的第二部分对*g* 命令提出了一个非常有趣的观点。该命令选择行，但是相关的编辑命令不需要真正作用于那些选择的行。

相反，他示范了一个可以对*ex* 命令重复任意次的技巧。例如，假設想把文件中从第 12 行到第 17 行的内容复制 10 次放到当前文件的尾部，则可以输入：

```
:1,10g/^/ 12,17t$
```

虽然这是意想不到的*g* 的用法，但是它是有效的。*g* 命令选择第一行，执行指定

注 9： 第一部分“vi Tips for Power Users”刊登在 1990 年 4 月份的《UNIX World》杂志上，第二部分“Using vi to Automate Complex Edits”刊登在 1990 年 6 月份的《UNIX World》杂志上。所列举的例子出自第二部分。

的 `t` 命令，然后到第二行执行下一次复制命令。当到达第 10 行时，`ex` 将完成 10 次复制。

搜集行

这是 `g` 的另一个高级应用实例，也是根据 Zintz 的论文中提供的建议编写的。假如你正在编辑由几个部分组成的文档，该文件的第二部分如下所示，其中省略号表示忽略的文本，并显示行号作为参考。

```
301 Part 2
302 Capability Reference
303 .LP
304 Chapter 7
305 Introduction to the Capabilities
306 This and the next three chapters ...

400 ... and a complete index at the end.
401 .LP
402 Chapter 8
403 Screen Dimensions
404 Before you can do anything useful
405 on the screen, you need to know ...

555 .LP
556 Chapter 9
557 Editing the Screen
558 This chapter discusses ...

821 .LP
822 Part 3:
823 Advanced Features
824 .LP
825 Chapter 10
```

其中章号占一行，它们的标题在下面的一行，本章文本（已被着重显示）从再下一行开始。你想做的第一件事情就复制每章的开始行，并把它放到名字为“`begin`”的现有文件中。

下面是完成该工作的命令：

```
:g /^Chapter/ .+2w >> begin
```

调用该命令前光标必须位于文件的开始，首先搜索位于行开始的 *Chapter*，然后在每章的开始行——*Chapter*下面的第二行执行该命令。由于以 *Chapter* 开始的行现在被选为当前行，因此行地址 *.+2* 将指示它下面的第二行。等价的行地址 *+2* 或 *++* 的含义也一样。由于想把这些行写到名为“begin”的现有文件中，因此调用带有追加运算符 *>>* 的 *w* 命令。

假如只想发送第二部分各章的开始内容，那么需要使用 *g* 对所选择的行进行限制，因此要把命令修改为：

```
:/^Part 2/, /^Part 3/g /^Chapter/ .+2w >> begin
```

其中，虽然 *g* 命令选择以 *Chapter* 开始的行，但是它只对文件中从以 *Part 2* 开始的行到以 *Part 3* 开始的行之间的部分进行搜索。如果调用上面的命令，那么文件“begin”的最后几行将是：

```
This and the next three chapters ...
Before you can do anything useful
This chapter discusses ...
```

这些行是 *Chapter 7*、*8* 和 *9* 的开始行。

除了刚发送的行以外，还想把各章标题复制到该文档的尾部，为建立内容列表做准备。你可以使用竖直条把第二个命令追加在第一个命令的后面，如下所示：

```
:/^Part 2/, /^Part 3/g /^Chapter/ .+2w >> begin | t$
```

要记住一点的是，无论使用什么子序列命令，行地址都相对于前面的命令。第一个命令已标记了那些以 *Chapter* 开始的行（第二部分内），各章标题位于那些行的下面一行。因此，如果要在第二个命令中访问各章标题，则行地址应该是 *+*（或者等价的 *+1* 或 *.+1*）。然后使用 *t\$* 把各章标题复制到该文件的尾部。

正如这些例子所展示的，思考和经验会使我们有一些不同寻常的编辑方案。不要害怕试着做一些事情！只要确保首先对文件进行了备份！

第七章

高级编辑

本章内容：

- 定制 vi
- 执行 UNIX 命令
- 保存命令
- 使用 ex 脚本
- 编辑程序源代码

本章将介绍一些 *vi* 和 *ex* 编辑器的高级性能。在开始学习本章所阐述的概念之前，你应该相当熟悉本书前面几章的内容。

本章共分为五个部分。第一部分讨论了一些设置选项的方法，可以定制编辑环境。你将学会如何使用 `set` 命令和 `.exrc` 文件创建不同的编辑环境。

第二部分讨论了如何在 *vi* 中执行 UNIX 命令以及 *vi* 如何通过 UNIX 命令过滤文本。

第三部分讨论了通过使用缩写的命令，或甚至缩减到只用一个按键的命令（这被称为映射键）来存储长命令序列的各种方法。本部分还包括一节有关 @ 功能的内容，可以将命令序列存储到缓冲区中。

第四部分讨论了在 UNIX 命令行或 shell 脚本中使用 *ex* 脚本。脚本提供了一种有效的重复编辑的方法。

第五部分讨论了 *vi* 对程序员非常有用的一些功能。*vi* 有许多控制行缩进的选项和一个显示不可见字符（特别是制表符和换行符）的选项。*vi* 的许多搜索命令对程序代码块或 C 函数都很有用。

定制 vi

我们已经看到 *vi* 在不同终端上运行的情况（例如，在“哑”终端上，*vi* 用 @ 符号代替已删除的行；在智能终端上每进行一次编辑操作 *vi* 都要刷新屏幕）。在现在的 UNIX 系统中，*vi* 从 *terminfo* 终端数据库中获得有关终端类型的操作指令。（在较老的系统中，*vi* 使用原始的 *termcap* 数据库，注 1）。

vi 还有许多可以在其内部进行设置的选项。这些设置影响它的操作方式。例如，可以设置使 *vi* 自动换行的右边距，这样就不用输入 **RETURN** 键了。

可以在 *vi* 中使用 *ex* 的 :set 命令修改这些选项。而且，无论 *vi* 什么时候启动，它都要读取主目录下的 *.exrc* 文件以获得进一步的操作指令。通过在该文件中放置 :set 命令，就可以在使用 *vi* 的任何时候改变它的操作方式。

也可以在本地目录下建立 *.exrc* 文件来初始化需要在不同环境中使用的各个选项。例如，可以定义一组选项来编辑英文文本，而定义另一组选项来编辑源代码。*vi* 首先执行主目录下的 *.exrc* 文件，然后才执行当前目录下的 *.exrc* 文件。

最后，存储在 shell 变量 EXINIT 中的任何命令都将在 *vi* 启动时执行，EXINIT 中的设置将覆盖主目录 *.exrc* 文件中的设置。

:set 命令

有两种类型的选项可以使用 :set 命令进行修改：不是开就是关的触发选项和带数字或字符串值的选项（如页边距的位置和文件名）。

触发选项可以默认设置为开或关。开启触发选项的命令为：

```
:set option
```

关闭触发选项的命令为：

注 1：这两个数据库的位置随着厂商的不同而变化。试着使用 man *terminfo* 和 man *termcap* 命令获得更多有关特定系统的信息。

```
:set nooption
```

例如，要指定模式搜索忽略大小写，可以输入：

```
:set ic
```

如果要 *vi* 在搜索时不忽略大小写，可以使用命令：

```
:set noic
```

一些选项具有分配给它们的值。例如，*window* 选项设置屏幕“窗口”上所显示的行数。可以使用等号 (=) 对这些选项进行赋值：

```
:set window=20
```

在 *vi* 会话期间，可以检查 *vi* 所使用的选项。命令：

```
:set all
```

会显示所有的选项，其中包括你设置的和 *vi* “选择”的默认值。显示结果应该如下所示（注 2）：

autoindent	nomodelines	noshowmode
autoprint	nonumber	noslowopen
noautowrite	nonovice	tabstop=8
beautify	nooptimize	taglength=0
directory=/var/tmp	paragraphs=IPLPPPQPP LIpplpipnppbp	tags=tags /usr/lib/tags
noedcompatible	prompt	tagstack
errorbells	noreadonly	term=vt102
noexrc	redraw	noterse
flash	remap	timeout
hardtabs=8	report=3	ttytype=vt102
noignorecase	scroll=11	warn
nolisp	sections=NHSHH HUuhsh+c	window=23
nolist	shell=/bin/ksh	wrapscan
magic	shiftwidth=8	wrapmargin=0
nomesg	showmatch	nowriteany

注 2：:set all 的结果在很大程度上依赖于使用的 *vi* 版本。这里给出的是 UNIX *vi* 典型的结果，不同的克隆版本将会出现不同的结果。

可以按名字找到任何单个选项的当前值，使用的命令是：

```
:set option?
```

命令：

```
:set
```

显示在 *.exrc* 文件或当前会话期间明确修改或设置过的选项。

例如，显示结果有可能如下所示：

```
number sect=AhBhChDh Window=20 wrapmargin=10
```

.exrc 文件

控制 *vi* 环境的 *.exrc* 文件位于主目录下（首次登录时所在的目录）。可以用 *vi* 编辑器修改 *.exrc* 文件，就像修改其他文本文件一样。

如果主目录下还没有 *.exrc* 文件，那么简单地使用 *vi* 创建一个即可。把将会在使用 *vi* 或 *ex* 时产生影响的 *set*、*ab* 和 *map* 命令输入到该文件中（*ab* 和 *map* 将在本章的后面进行讨论）。简单的 *.exrc* 文件可能如下所示：

```
set nowrapscan wrapmargin=7
set sections=SeAhBhChDh nomesg
map q :w^M:n^M
map v dwElp
ab ORA O'Reilly & Associates, Inc.
```

由于该文件是由 *ex* 在进入可视模式之前读入的，因此在 *.exrc* 文件中的命令前面不需要有冒号。

替换环境

除了读取主目录下的 *.exrc* 文件文件外，还可以让 *vi* 读取当前目录下名为 *.exrc* 的文件。这样可以对那些适用于特殊项目的选项进行设置。

例如，可能需要在主要用于编程的目录下设置一组选项：

```
set number autoindent sw=4 terse  
set tags=/usr/lib/tags
```

而在主要用于文本编辑的目录下设置另一组选项：

```
set wrapmargin=15 ignorecase
```

注意，可以在主目录下的`.exrc`文件中设置某些选项，而在本地目录中取消这些设置。

还可以通过把选项设置保存到不同于`.exrc`文件的文件中，然后再使用`:so`命令（`so`是`source`的缩写）读取该文件来定义另外的`vi`环境。

例如：

```
:so .progoptions
```

本地`.exrc`文件对定义缩写和键映射（将在本章的后面介绍）也很有用。当我们编写书籍或手册时，我们要把那本书中使用的所有缩写保存到创作该书的目录下的`.exrc`文件中。

注意：所有`vi`的现有版本中，在`vi`读取当前目录下的`.exrc`文件之前，必须先在主目录下的`.exrc`文件中设置`exrc`选项：

```
set exrc
```

这种机制可防止其他人在用户的工作目录下放置可能会影响系统安全的`.exrc`文件（注3）。

一些有用的选项

就像输入`:set all`时看到的一样，可以设置众多的选项。它们中的许多将由`vi`

注 3：如果这两个文件存在，`vi`的最初版本会自动读取它们。`exrc`选项关闭了潜在的安全漏洞。

内部使用，通常不需要修改。其他选项在某些场合很重要，但在另一些场合又会变得不重要（例如，`noredraw`和`window`在低波特率的拨号线路上是很有用的）。附录三中“Solaris 2.6 vi 的选项”一节中的表格包含了对每个选项的简要描述。我们建议你花些时间熟悉这些设置选项。如果你对某个选项感兴趣，可以试着对这个选项进行设置（或取消对它的设置），并在编辑时观察产生的结果。你可能会找到一些意想不到的有用的工具。

正如本章前面所描述的，选项`wrapmargin`对编辑非程序文本是必需的。`wrapmargin`所指定的右边页边距的大小会对输入的文本进行自动换行（这省去了人工输入回车键）。通常该选项的值为 7 到 15：

```
:set wrapmargin=10
```

还有三个其他选项控制着`vi`在搜索时的行为。通常，搜索操作区别大写字母和小写字母（*foo* 不与 *Foo* 相匹配），会绕回到文件的开始进行（也就是说你可以在文件的任何位置开始搜索，直到找到所有的出现），在模式匹配时识别通配符。这些选项的默认设置分别为`noignorecase`、`wrapscan`和`magic`。要改变这些默认设置中的任何一个，则可以设置相反的触发选项：`ignorecase`、`nowrapscan`和`nomagic`。

对程序员可能会有特殊影响的选项包括：`autoindent`、`showmatch`、`tabstop`、`shiftwidth`、`number`、`list`以及它们的相反触发选项。

最后，要记住使用`autowrite`选项。对其设置后，`vi`在调用`:n`（下一个）命令移动到下个文件进行编辑时和使用`:!`运行 shell 命令之前将自动写出已修改的缓冲区内容。

执行 UNIX 命令

在`vi`中进行编辑时，可以显示或读取任何 UNIX 命令的执行结果。感叹号（`!`）可指定`vi`创建一个 shell 并把它后面的内容看成 UNIX 命令：

```
:!command
```

因此，如果你正在进行编辑并想在没有退出 vi 的情况下看一下时间或日期，则可以输入：

```
:!date
```

时间和日期就会显示在屏幕上，按下 [RETURN] 键可继续在文件中的同一位置进行编辑。

如果想在一行中给出多个 UNIX 命令而又不想在中间返回 vi 编辑环境，则可以使用 ex 命令创建一个 shell：

```
:sh
```

想退出 shell 并返回 vi 时，请输入 [CTRL-D]。

为了把 UNIX 命令的执行结果读取到文件中，可以把 :read 和 UNIX 调用结合在一起使用。一个非常简单的例子是：

```
:r !date
```

它将把系统的日期信息读取到文件中。通过在 :r 命令前加上行地址，可以把命令的执行结果读取到文件中任何想要的位置。默认情况下，将读取到当前行的后面。

假如你正在编辑一个文件，并想从名为 “phone” 的文件中按字母顺序读取 4 个电话号码。“phone” 中的内容为：

```
Willing, Sue 333-4444  
Walsh, Linda 555-6666  
Quercia, Valerie 777-8888  
Dougherty, Nancy 999-0000
```

命令：

```
:r !sort phone
```

会读取出 “phone” 排序后的内容：

```
Dougherty, Nancy 999-0000
Quercia, Valerie 777-8888
Walsh, Linda 555-6666
Willing, Sue 333-4444
```

假如你正在编辑一个文件并想从该目录下的另一个文件中插入文本，但是却忘记了新文件的名字。那么可以使用较慢的方法来完成该任务：退出文件，执行ls命令，记下正确的文件名，重新进入文件并搜索你刚才的位置。

也可以使用较少的步骤完成该任务：

击键	结果
:!ls	file1 file2 letter newfile practice

显示当前目录下的文件列表，记下正确的文件名字，按下[RETURN]键继续编辑。

:r newfile	"newfile" 35 lines, 949 characters
------------	------------------------------------

读取新文件。

使用命令过滤文本

你也可以把文本块作为标准输入发送给UNIX命令，该命令的输出将取代缓冲区中的文本。你可以使用ex或vi的命令对文本进行过滤，两种方法之间的主要区别是在ex中使用行地址指示文本块，而在vi中使用文本目标（移动命令）指示文本块。

使用 ex 过滤文本

第一个例子展示了如何使用ex过滤文本。假如前面例子中的文件列表不是位于名为“phone”的单个文件中而是位于当前文件的第96到第99行中，那么只需输入所要过滤的行地址、感叹号以及所要执行的UNIX命令。例如，命令：

```
:96,99!sort
```

将使用 *sort* 对第 96 行到第 99 行进行过滤，并用 *sort* 的输出取代这些行。

使用 vi 过滤文本

在 *vi* 中，通过依次输入感叹号、任何指示文本块的 *vi* 移动键和所要执行的 UNIX 命令，即可使用该 UNIX 命令对文本进行过滤。例如：

```
!) command
```

将使 *command* 作用于下一个句子。

当使用该功能时，*vi* 的行为会出现一些不寻常的特征：

- 感叹号并不立即显示在屏幕上。当为要过滤的文本目标输入按键时，感叹号才显示在屏幕底部，但是输入的用来指定目标的字符并没有显示出来。
- 文本块必须大于一行，因此可以只使用移动多行的按键 (*G*, { }, (), [[]], +, -)。为了重复结果，可以把数字加在感叹号或文本目标的前面。（例如，!10+ 和 10!+ 都指示下面的 10 行。）*w* 在这里不能正常工作，除非指定了足够的数量使其超过了一行。也可以使用带有模式和回车键的斜杠 (/) 来指定目标，这将把模式之前的文本作为输出发送给命令。
- 所有的行都要受到影响。例如，如果光标位于行的中间并且调用了到下一句尾部的命令，那么将修改所有包含该句开头和结尾的行，而不只是该句本身（注 4）。
- 有一个特殊的文本目标，它只能与该命令语法一起使用：可以通过输入另一个感叹号来指定当前行：

```
!! command
```

无论是整个序列还是文本目标，都可以在前面加上数字来重复影响。例如，在上面的例子中要修改第 96 行到第 99 行，则可以把光标定位到第 96 行，然后输入：

注 4：当然，总会有意外情况。在这个例子中，*vim* 5.0 只修改当前行。

```
4!!sort
```

或者：

```
!4!sort
```

另一个例子是，假設想把文件中的部分文本从小写变为大写，则可以使用改变大小写的 tr 命令对该部分文本进行处理。在本例中，第二句是通过命令将要过滤的文本块。

One sentence before.
With a screen editor you can scroll the page move the cursor, delete lines, insert characters, and more, while seeing the results of your edits as you make them.
One sentence after.

击键

结果

!)

One sentence after.
~
~
~
!_

感叹号出现在最后一行上，提示输入 UNIX 命令。“)”指示句子是要过滤的文本单元。

```
tr '[a-z]' '[A-Z]'
```

One sentence before.
WITH A SCREEN EDITOR YOU CAN SCROLL THE PAGE MOVE THE CURSOR, DELETE LINES, INSERT CHARACTERS, AND MORE, WHILE SEEING THE RESULTS OF YOUR EDITS AS YOU MAKE THEM.
One sentence after.

输入 UNIX 命令并按下 **RETURN** 键，输入将被输出替换。

重复前面命令的语法是：

```
! object !
```

有时把已编码的文档发送给 *nroff*，并使用格式化的输出进行替换则是很有用的（或者在编辑电子邮件时，也可以在发送消息前使用 *fmt* 程序对文本进行“美化”）。

记住，“原始”输入将被输出替换。幸运的是，如果出现错误，那么发送的将是错误信息而不是预期的输出，这时可以取消命令来恢复那些行。

保存命令

我们经常会在文件中反复输入同样的长的短语。无论在命令模式还是在插入模式下，*vi* 和 *ex* 都有许多不同的用来保存长命令序列的方法。当调用这些已保存的序列中某一个时，只需输入几个字符（或者甚至一个字符）就会执行整个命令序列。

单词缩写

你可以定义一些缩写，无论什么时候在插入模式下输入缩写，*vi* 将自动把它扩展为完全的文本。使用 *ex* 命令定义缩写：

```
:ab abbr phrase
```

其中 *abbr* 表示指定的 *phrase*（短语）的缩写。只有在插入模式下把组成缩写的字符序列作为完整的单词输入，它才会被扩展；单词内部的 *abbr* 将不会被扩展。

假如在文件“*practice*”中输入含有经常出现的短语的文本，例如不容易记忆的产品或公司名字。那么命令：

```
:ab imrc International Materials Research Center
```

会把 *International Materials Research Center* 缩写为多个首字符的组合 *imrc*。无论什么时候，只要在插入模式下输入 *imrc*，就会将其扩展为完整的文本。

击键
i the imrc

结果

```
the International Materials Research Center
```

你一按下非字母数字字符（如标点符号）、空格、回车或 **ESC** 键（返回命令模式）时，就会对缩写进行扩展。当你选择缩写时，要选择在你输入文本时不经常出现

的字符组合。如果你创建了一个缩写，并且要在不需要它扩展的时候停止扩展，那么可以输入：

```
:unab abbr
```

使缩写失效。

如果要列出当前定义的缩写，则可以输入：

```
:ab
```

组成缩写的字符不能出现在短语的最后。例如，如果调用命令：

```
:ab PG This movie is rated PG
```

将得到“*No tail recursion*”的信息，因此不能设置缩写。该信息的意思是你企图定义将重复进行自我扩展的缩写，这样会产生死循环。如果调用命令：

```
:ab PG the PG rating system
```

可能会也可能不会产生一个死循环，但是在这两种情况下，你都不会得到警告信息。例如，当上面的命令在 UNIX 的 System V 上进行测试时，将正常进行扩展。大约 1990 年在 UNIX 的 Berkeley 版本上，缩写会进行重复扩展，如下所示：

```
the the the the the ...
```

直到发生内存错误并退出 vi 为止。

在进行测试时，我们在这些 vi 版本上获得了下面的结果：

Solaris 2.6 vi

不允许尾部循环的情况，对名字位于扩展体中间的情况只扩展一次。

nvi 1.79

两种形式都超出了内部扩展的限制，会停止扩展，*nvi* 生成错误信息。

elvis 2.0

尾部循环形式会一直运行到中断编辑器。对名字位于扩展体中间的形式最终会停止扩展，但是没有任何错误信息。

vim 5.0 和 5.1

两种形式都可以识别并且只扩展一次。

vile 7.4 和 8.0

两种形式都可以识别并且只扩展一次。

我们建议你要避免把缩写作为要定义短语的一部分进行重复。

使用映射命令

当你正在进行编辑时，可能会发现正在频繁地使用某个命令序列，或者可能偶然使用一个非常复杂的命令序列。为了节省按键或记忆该序列所花费的时间，可以使用 `map` 命令把该序列指定为一个不使用的键。

除了是为 `vi` 的命令模式而不是插入模式定义了一个宏外，`map` 命令的行为与 `ab` 极为相似。

`:map x sequence`

把字符 `x` 定义为一系列编辑命令。

`:unmap x`

取消为 `x` 定义的序列。

`:map`

列出当前映射的所有字符。

在开始创建自己的映射前，你需要知道哪些键是命令模式下用而用户定制的命令可用的：

字母

`g K q V v`

控制键

`^A ^K ^O ^W ^X`

符号

`_ * \ =`

注意：如果设置了Lisp模式，那么vi将使用`=`，一些克隆版本也使用`=`对文本进行格式化。在许多vi的现在版本中，`_`等同于`^`命令，elvis和vim也有使用`v`、`V`和`^V`键的“可视模式”。原则是要仔细地测试所使用的vi版本。

根据你的终端，你也可以把映射序列与特殊的功能键相关联。

使用映射，你可以创建简单或复杂的命令序列。作为一个简单的例子，你可以定义一个反转单词顺序的命令。在vi中，把光标定位到下列位置：

```
you can the scroll page
```

把单词*the*放到*scroll*后面的命令序列是`dwe1p:`，`dw`删除单词，`e`把光标移到下一个单词的尾部，`1`向右移动一个空格，`p`把删除的单词粘贴在那里。保存该命令序列：

```
:map v dwelp
```

然后就可以使用单个按键`v`在编辑会话的任何时候反转两个单词的顺序。

防止ex解释键

要注意在定义映射时，不能只输入`RETURN`、`ESC`、`BACKSPACE`和`DELETE`之类的键作为映射的命令的一部分，因为这些键已在ex内有含义了。如果想在命令序列中包含这些键中的某个，则必须在该键之前使用`CTRL-V`来避开它的正常含义。按键`^V`在映射中显示为`^`字符，`^V`后面的字符也不按想要的方式显示。例如，回车键显示为`^M`，`ESC`键显示为`^[`，退格键显示为`^H`等等。

相反，如果想用控制键作为映射的字符，那么在大多数情况下必须同时按下 **[CTRL]** 键和该字母键。例如，要映射 ^A，需要输入：

```
:map [CTRL-A] sequence
```

但是有三个控制字符必须要使用 ^V 进行转义，它们是：^T、^W 和 ^X。因此，如果想映射 ^T，则必须输入：

```
:map [CTRL-V] [CTRL-T] sequence
```

[CTRL-V] 的这种用法适用于任何 *ex* 命令，不只是映射命令。这说明可以在缩写或替换命令中输入回车键。例如，缩写：

```
:ab 123 one^Mtwo^Mthree
```

展开是：

```
one  
two  
three
```

(这里我们把序列 **[CTRL-V] [RETURN]** 显示为 ^M，它也将按这种方式显示在屏幕上。)

也可以在某些位置同时添加多行。命令：

```
:g/^Section/s//As you recall, in^M&/
```

在所有以单词 *Section* 开头的行的前面插入一个短语作为独立行。& 用来重现搜索模式。

遗憾的是，一个字符在 *ex* 命令中总是有特殊含义，即使使用 **[CTRL-V]** 引用它也是如此。由于竖直条 (|) 有作为多个 *ex* 命令分隔符的特殊含义，因此不能在插入模式下的映射中使用它。

既然我们已经了解了如何使用 **[CTRL-V]** 来保护 *ex* 命令中的某些键，那么现在就可以定义一些强大的映射序列了。

复杂的映射例子

假如有含有多个条目的术语表，如下所示：

```
map - an ex command which allows you to associate  
a complex command sequence with a single key.
```

如果想把该术语表转换为 *troff* 格式，即如下所示：

```
.IP "map" 10 n  
An ex command...
```

那么定义复杂映射的最好方法是手工编辑一次，在此期间记下要输入的每个按键，然后再作为映射重建这些键。需要：

1. 为每个首行缩排的段在行的开始插入 MS 宏。同时也插入第一个引号 (I.IP ")。
2. 按下 **ESC** 键结束插入模式。
3. 移动到第一个单词的尾部 (e) 添加第二个引号和一个空格以及缩进的大小 (a" 10n)。
4. 按下 **RETURN** 键插入一个新行。
5. 按下 **ESC** 键结束插入模式。
6. 删除连字符和两个空格 (3x) 并使下个单词的首字母大写 (~)。

如果需要对该过程重复多次，那么这将是相当麻烦的编辑工作。

可以使用 :map 把整个命令序列保存起来，然后使用单个按键对其重复执行：

```
:map g I.IP "^[ea" 10n^M^ [3x~
```

注意，必须使用 **CTRL-V** “引用” **ESC** 和 **RETURN** 字符。**^ [** 是输入 **CTRL-V** 和 **ESC** 后出现的序列，**^ M** 是输入 **CTRL-V** **RETURN** 后显示的序列。

现在，只输入 **g** 就可以完成所有一连串的编辑操作。在低波特率的情况下，你可

以看到所有编辑操作都在一个一个地进行；在高波特率的情况下，很快就可以完成。

如果你的首次映射键失败了，别灰心。定义映射时出现的小错误可能会有完全不同的结果，与预期结果差别很大。输入 `u` 取消这次编辑，然后再试。

映射键的更多例子

这些例子将使你在定义键映射时，对尽可能巧妙的简化操作有个概念：

1. 移动到单词的尾部添加文本：

```
:map e ea
```

很多时候，移动到词尾的惟一原因是添加文本。该映射序列会自动使你处于插入模式。记住，映射的键 `e` 在 `vi` 中也有含义。虽然允许对 `vi` 已经使用的键进行映射，但是只要该映射起作用，该键的正常功能就不能使用。由于 `E` 命令通常与 `e` 相同，因此在这种情况下也不是什么问题。

2. 调换两个单词：

```
:map K dwElp
```

虽然我们在本章前面讨论过该序列，但是现在需要使用 `E`（假如这里和其他例子中 `e` 命令已经映射为 `ea`）。要记住光标要位于两个单词的第一个单词的开始处。不幸的是，由于 `l` 命令，如果这两个单词位于一行的结尾，则该序列（和前面的版本）将不起作用：在该序列中，光标将位于行尾，`l` 命令不能实现向右移动。较好的解决办法是：

```
:map K dwwP
```

也可以使用 `w` 代替 `w`。

3. 保存当前文件，编辑序列中的下一个文件：

```
:map q :w^M:n^M
```

虽然可以对 `ex` 命令进行映射，但是要确保使用回车键结束每个 `ex` 命令。该命令序列可以很容易地从一个文件移动到另一个文件，这在已使用一个 `vi` 命

令打开多个短文件时很有用。映射字母q有助于记住该命令序列同“quit”相似。

4. 在单词周围添加*troff*来加粗代码：

```
:map v i\fB^[e\fp^[
```

该命令序列假设光标位于单词的开始。首先，进入插入模式，然后输入表示粗体的代码。在映射命令中，不需要输入两个反斜杠来表示一个反斜杠。接下来，通过输入一个“引用的”**ESC**键返回命令模式。最后，在单词的结尾添加结束*troff*的代码并返回命令模式。要注意在词尾扩展时，不需要使用ea，因为该序列本身已映射为单个字母e。这说明映射序列也可以包含其他已映射的命令。（使用嵌套映射序列的能力由vi的remap选项控制，该选项通常是激活的。）

5. 在单词周围添加*troff*来加粗代码，不管光标是否位于单词的开始处：

```
:map V lbi\fB^[e\fp^[
```

除了使用l b来完成把光标定位到单词开始处的额外任务以外，该序列与前面的序列相同。由于光标有可能位于单词的中间，因此你想使用b命令把光标移向词首。但是，如果已位于词首，那么b命令将会把光标移动到前面的单词上。为了防止这种情况，要在使用b回移光标前输入l，这样光标就不可能位于该单词的第一个字母上。也可以使用B代替b和Ea代替e，来定义该序列的变形。但是在所有情况下，如果光标位于一行的尾部，l命令将阻止该序列进行工作（可以在周围添加空格）。

6. 重复搜索并删除单词或短语周围的圆括号（注5）：

```
:map = xf)xn
```

该序列假定首先通过输入/（和**RETURN**键发现了左圆括号。

如果选择删除圆括号，那么则使用映射命令：使用x删除左圆括号；使用f)搜索右圆括号；使用x删除左圆括号，然后再使用n重复对左圆括号的搜索。

如果不想要删除这个圆括号（例如，如果是正确地使用它们），也不想使用映射命令：按下n继续搜索下一个左圆括号。

注5： 摘自1990年4月的《UNIX World》中Walter Zintz发表的论文。

还可以通过修改上面的映射序列来处理成对的引号。

7. 把 C/C++ 注释放到整行的周围：

```
:map g I/* ^[A */^]
```

该序列在行的开始插入 /*，在行的结尾添加 */。也可以映射一个替换命令来进行同样的处理：

```
:map g :s;.*;/ * & */;^M
```

其中对整行进行了匹配（使用 .*），当重复使用该操作（使用 &）时，则在其周围加上注释符号。记住分号分隔符的用法，要避免对注释中的 / 进行转义。

8. 安全地重复长插入：

```
:map ^J :set wm=0^M.:set wm=10^M
```

在第二章“简单编辑”中我们已经提到，当设置 wrapmargin 时，*vi* 在处理长文本的插入时偶而会出现问题。该映射命令是个非常有用的解决办法，它暂时取消了 wrapmargin（通过将其设置为 0），给出了重复命令，然后又恢复了 wrapmargin 的设置。注意，一个映射序列可以包含 *ex* 和 *vi* 命令。

在上面的例子中，虽然 ^J 是 *vi* 命令（它沿着行移动光标），但是由于它与 j 命令非常相似，因此该键可以安全地映射该键。虽然有许多键与其他键完成的任务相同，也有许多键极少使用，但是，通过在映射定义中使用它们来取消它们的正常用法之前，你应该对 *vi* 命令非常熟悉。

用于插入模式的映射键

通常，映射只适用于命令模式，因为在插入模式下各键都代表其本身，不应将其映射为命令。但是，通过在 map 命令前加上感叹号 (!)，就可以强迫它忽略正常含义，从而在插入模式下产生映射。这种用法在你处于插入模式但又需要暂时返回命令模式运行一个命令、然后又返回插入模式时非常有用。

例如，假如你只输入了某个单词而忘记了使用斜体字对它排版（或在它周围加引号，等等）。则可以定义下面的映射：

```
:map! + ^[bi<I>^ [ea</I>
```

现在，当在单词结尾输入+时，就可以使用HTML的斜体字代码包围该单词。+并不在文本中显示出来。

上面的序列退出到命令模式 (^[]), 后退插入第一个代码 (bi<I>), 然后再返回到命令模式 (^[]), 前移后添加第二个代码 (ea</I>)。由于映射序列开始和结束时都位于插入模式下，因此在对单词标记后可以继续输入文本。

下面是另一个例子。假如已经输入了文本，并且意识到前面的行应该以冒号结尾。则可以通过定义下面的映射序列来改正它（注 6）：

```
:map! % ^[kA:^ [jA
```

现在，如果在当前行的任何位置输入%，那么将会在上一行的尾部添加一个冒号。该命令进入命令模式，向上移动一行并添加冒号 (^[kA:), 然后再返回到命令模式，下移到刚才所在的行并进入插入模式 (^[jA)。

注意，我们要为前面的映射命令使用不常用的字符（% 和 +）。在将一个字符映射为用于插入模式时，就不能再把该字符作为文本进行输入了。

如果要恢复字符的正常输入，可以使用命令：

```
:unmap! x
```

其中x是前面映射为用于插入模式的字符。（虽然在输入x时vi会在命令行上对它进行扩展，使它看起来好像正在取消扩展后的文本的映射，但是实际上vi正确地取消了对该字符的映射。）

通常插入模式下的映射更适合于把字符串与不想使用的特殊键结合在一起，这一点对可编程功能键尤其有用。

注 6： 摘自 1990 年 4 月的《UNIX World》中 Walter Zintz 发表的论文。

映射功能键

许多终端有可编程功能键(今天的终端仿真程序已经在位图化的工作站上实现了该功能键)。通常可以设置这些键来使用终端上的特殊设置模式，并且显示任何想要的字符或字符序列。但是，使用终端的设置模式进行编程的功能键只能在特定终端上使用；它们也可能会限制那些要对功能键进行设置的程序的行为。

ex 允许使用数字映射功能键，使用语法为：

```
:map #1 commands
```

代表 1 号功能键，依次类推（由于编辑器可以访问在 *terminfo* 或 *termcap* 数据库中找到的终端入口，并且知道功能键正常输出的转义序列，因此它能完成上面的功能）。

与其他的键一样，在默认情况下映射适用于命令模式，但是通过使用 *map!* 命令，也可以为一个功能键定义两个不同的值：一个用于命令模式下，另一个用于插入模式下。例如，如果我们是 HTML 用户，可能想在功能键上加入字体转换代码。例如：

```
:map #1 i<I>^{  
:map! #1 <I>
```

如果在命令模式下，则第一个功能键将进入插入模式，输入三个字符 <I> 后返回命令模式。如果已处于插入模式，则该功能键将只输入三个字符的 HTML 代码。

注意：如果功能键在终端的设置模式下重新定义，由于这些功能键不再输出预期的控制或对它的终端数据库入口中所描述的序列进行了转义，因此 #n 语法可能会不起作用。需要对用于终端的 *terminfo* 源（或 *termcap* 条目）进行测试，并检查功能键的定义。除此之外，还有一些终端的功能键只执行本地操作，不能真正地把任何字符发送给计算机。这样的功能键将不能被映射。

终端功能 k1、k2 到 k0 描述前 10 个功能键，功能 11、12 到 10 描述余下的功能键。使用终端的设置模式，则可以改变控制或使功能键的输出序列与 *terminfo* 或

termcap 入口相一致（如果要了解更多的信息，请参考 O'Reilly & Associates 出版的《*termcap & terminfo*》一书）。

如果序列含有回车键 ^M，则要按下 **CTRL-M**。例如，为了使功能键 1 可用于映射，则用于终端的终端数据库入口必须有 k1 的定义，例如：

```
k1 = ^A@^M
```

接下来，定义：

```
^A@^M
```

必须是按下那个键时的输出。

如果要查看功能键的输出结果，则可以使用带有 -c 选项（显示每个字符）的 *od*（八进制转储）命令。并且需要在功能键后输入 **RETURN** 键，然后输入 **CTRL-D** 将使 *od* 显示信息。例如：

```
$ od -c
[[[A
^D
0000000 033      [      A      \n
0000005
```

这里，功能键输出 Esc 键、两个左方括号和一个 A。

映射其他的特殊键

许多键盘带有与 *vi* 中的命令功能相同的特殊键，如 **HOME**、**END**、**PAGE UP** 和 **PAGE DOWN**。如果终端的 *terminfo* 或 *termcap* 描述完整，那么 *vi* 将能识别这些键。但是如果描述不完整，则可以使用 *map* 命令使它们在 *vi* 中使用。这些键通常给计算机发送一个转义序列，即一个转义字符，后面是包含一个或多个其他字符的串。为了限制转义，应该在映射中特殊键的前面按下 ^V。例如，要把 IBM PC 键盘上的 **HOME** 键映射为适当的 *vi* 命令，则可以定义下面的映射：

```
:map [CTRL-V] [HOME] 1G
```

这将在屏幕上显示为：

```
:map ^[[H 1G
```

类似的映射命令显示如下：

:map [CTRL-V] [END] G	显示为	:map ^[[Y G
:map [CTRL-V] [PAGE UP] ^F	显示为	:map ^[[V ^F
:map [CTRL-V] [PAGE DOWN] ^B	显示为	:map ^[[U ^B

你也许想把这些映射放在 *.exrc* 文件中。如果特殊键生成了长转义字符（包含多个非显示字符），那么 ^V 将只引用最初的转义字符，因此映射将不起作用。你必须手工寻找整个转义字符（也许来自终端手册）并输入它，应该在合适的位置引用，而不是只输入 ^V 和该键。

映射多个输入键

映射的多个击键并不局限于功能键。也可以对常规的击键序列进行映射，这将使输入某些类型的文本（如 SGML 或 HTML）变得容易。

这里有一些可使输入 SGML 标记变得容易的 :map 命令，为此要感谢 O'Reilly 出版的《Learning the UNIX Operating System》一书的合著者 Jerry Peek（以双引号开始的行是注释，这将在下面的“ex 脚本中的注释”一节中进行讨论）。

```
" ADR: need this
:set noremap
" bold:
map! =b </emphasis>^[[F<i<emphasis role=bold>
map =B i<emphasis role=bold>^[
map =b a</emphasis>^|
" Move to end of next tag:
map! =e ^[f>a
map =e f>
" footnote (tacks opening tag directly after cursor in text-input mode):
map! =f <footnote>^M<para>^M</para>^M</footnote>^[[kO
```

```
" Italics ("emphasis"):
map! =i </emphasis>^[F<i<emphasis>
map =I i<emphasis>^[
map =i a</emphasis>^[
" paragraphs:
map! =p ^[jo<para>^M</para>^[O
map =P O<para>^[
map =p o</para>^[
" less-than:
map! *l &lt;;
...
...
```

使用这些命令，如果要输入在插入模式下才可输入的脚注，则只需输入 =f 即可。然后 vi 将插入开始和结束标志符，并进入插入模式：

```
All the world's a stage.<footnote>
<para>
-
</para>
</footnote>
```

这些宏是非常有用的。

@ 函数

命名缓冲区还提供了另一种创建“宏”的方法，可以只使用几个击键就能重复复杂的命令序列。

如果在文本中输入命令行 (vi 序列或前面带有冒号的 ex 命令)，然后把它删除到一个命名缓冲区中，那么就可以使用 @ 命令执行该缓冲区中的内容。例如，打开新行并输入：

```
cwgadfly [CTRL-V] [ESC]
```

这将在屏幕上显示为：

```
cwgadfly^[
```

按下`ESC`键退出插入模式，然后通过输入`"gdd`把该行删除到`g`缓冲区中。现在，无论什么时候只要把光标放在一个单词的开头并输入`@g`，文本中的那个单词就将改为`gadfly`。

由于`@`被解释为`vi`命令，因此即使缓冲区包含的是`ex`命令，点`(.)`也将重复整个序列。`@@`重复上一个`@`，可以使用`u`或`U`取消`@`的结果。

这是个简单的例子。由于`@`函数适合于非常特殊的命令，因此它们是非常有用的。当你在文件之间进行编辑时它们尤其有用，这是因为可以把命令存储到以它们的名字命名的缓冲区中，然后在编辑的任何文件中访问它们。`@`函数在与第六章“全局替换”中所讨论的全局替换命令一起使用时也非常有用。

从`ex`中执行缓冲区

也可以从`ex`模式中执行保存在缓冲区中的文本。在这种情况下应该输入`ex`命令，并将其删除到一个命名缓冲区中，然后在`ex`冒号提示符下使用`@`命令。例如，输入下面的文本：

```
ORA publishes great books.  
ORA is my favorite publisher,  
1,$s/ORA/O'Reilly \& Associates/g
```

让光标定位到最后一行，把命令删除到`g`缓冲区中：`"gdd`。把光标移动到第一行：`kk`，然后在冒号提示符下执行缓冲区：`:@g [RETURN]`。现在屏幕上将显示：

```
O'Reilly & Associates publishes great books.  
O'Reilly & Associates is my favorite publisher.
```

在`ex`命令中使用`*`时，一些`vi`版本会把`*`当成`@`。而且，如果缓冲区字符支持`@`或`*`命令后面的`*`，那么该命令将来自默认（未指定）缓冲区。

使用`ex`脚本

某些`ex`命令只能在`vi`中使用，如映射、缩写等等。如果把这些命令存储在`.exrc`

文件中，那么在调用 *vi* 时这些命令将会自动执行。任何包含可执行命令的文本都称为脚本。

在通常的 *.exrc* 脚本中的命令不能在 *vi* 之外使用。但是，可以把其他的 *ex* 命令保存到脚本中，然后在一个或多个文件中执行该脚本。大多数情况下在这些外部脚本中将使用替换命令。

对于一个作者，*ex* 脚本的有效使用可以保证整个文档集中术语或是拼写的一致性。例如，假如已经在两个文件中执行了 UNIX *spell* 命令，并且该命令已显示出下面的拼写错误列表：

```
$ spell sect1 sect2
chmod
ditroff
myfile
thier
writeable
```

通常情况下，虽然 *spell* 会标识出一些它不能识别的技术术语和特殊格式，但是 *spell* 也会识别出两个真正的拼写错误。

由于同时检查两个文件，因此我们不知道错误所发生的是哪个文件以及它们在文件中的位置。虽然有许多找到错误的办法，而且在两个文件中只寻找两个错误的工作不太难，但是，对于一个质量较低的课本或一个需要同时校验多个文件的打字员，这项工作将会花费很多时间。

为了使这项工作变得容易，我们可以编写一个包含下列命令的 *ex* 脚本：

```
%s/thier/their/g
%s/writeabel/writable/g
wq
```

假设已经把这些行保存到名为“*exscript*”的文本中，则可以在 *vi* 中使用下面的命令执行该脚本：

```
:so exscript
```

或者从命令行直接把该脚本应用到文件中。可以对文件 *sect1* 和 *sect2* 进行编辑：

```
$ ex - sect1 < exscript  
$ ex - sect2 < exscript
```

ex 调用后面的减号会让它取消正常的结束信息（注 7）。

如果脚本比我们例子中的要长，我们可能会节省大量的时间。但是，你可能想知道是否有一些方法可以避免对每个要编辑的文件重复执行该过程。确切地说，由于我们可以编写包含 *ex* 调用的 shell 脚本，因此它可以用于任何数量的文件上。

shell 脚本中的循环

shell 是一种可编程语言，也是一个命令行解释器。为了在多个文件上调用 *ex*，我们可以使用一种称为“*for* 循环”的简单类型的 shell 脚本命令。*for* 循环允许你把一个命令序列应用到脚本中给定的每个参数（对于初学者而言，*for* 循环也许是 shell 编程中最有用的一个部分，即使你不写任何 shell 程序，你都应该记住它）。

for 循环的语法如下：

```
for variable in list  
do  
    command(s)  
done
```

例如：

```
for file in $*  
do  
    ex - $file < exscript  
done
```

（该命令不需要缩进，为了清晰起见，我们对它进行了缩进。）创建完 shell 脚本后，我们把它保存到名为“*correct*”的文件中，并使用 *chmod* 命令使它变为可执

注 7： 根据 POSIX 标准，*ex* 应该使用 *-s* 来代替这里显示的 *-*。通常，为了向后兼容性，两个版本都是通用的。

行的。(如果你不熟悉*chmod*命令以及把命令添加到UNIX搜索路径中的过程,请参考O'Reilly & Associates出版的《Learning the UNIX Operating System》。)现在输入:

```
$ correct sect1 sect2
```

“*correct*”文件中的for loop将把每个参数(由\$*指定列表中的每个文件,\$*代表所有的参数)赋给变量*file*,然后在那个变量内容所标识的文件上执行*ex*脚本。

利用一个输出比较明显的例子,可以更容易地掌握for loop的工作原理。请查看对文件重命名的脚本:

```
for file in $*
do
    mv $file $file.x
done
```

假如该脚本在名为“*move*”的可执行文件中,则输入:

```
$ ls
ch01  ch02  ch03  move
$ move ch??
$ ls
ch01.x  ch02.x  ch03.x  move
```

可以更精确地重新编写用于重命名文件的脚本:

```
for nn in $*
do
    mv ch$nn  sect$nn
done
```

引用利用这种方法编写的脚本,可以在命令行中通过指定数字来代替文件名:

```
$ ls
ch01  ch02  ch03  move
$ move 01 02 03
$ ls
```

```
sect01 sect02 sect03 move
```

`for` 循环不是必须要把 `$*`（所有的参数）当成要替换的值的列表，也可以指定明确的列表。例如：

```
for variable in a b c d
```

将依次把 *a*、*b*、*c* 和 *d* 赋给 *variable*。或者也可以把它替换为命令的输出，例如：

```
for variable in `grep -l "Alcuin" *`
```

将依次把 *grep* 所找到的每个含有字符串 *Alcuin* 的文件的名字赋给 *variable*。

如果没有指定列表：

```
for variable
```

就依次把每个命令行参数赋给 *variable*，这与我们最初的例子比较相似。实际上这并不等价于：

```
for variable in $*
```

而是等价于：

```
for variable in "$@"
```

其中的含义略有不同。符号 `$*` 扩展为 `$1`、`$2`、`$3` 等等，而四字符序列 `"$@"` 扩展为 `"$1"`、`"$2"`、`"$3"` 等等。引号进一步防止了对特殊字符的解释。

现在返回到我们的初始脚本：

```
for file in $*
do
    ex - $file < exscript
done
```

这仿佛有些粗糙，必须使用两个脚本——`shell` 脚本和 `ex` 脚本。但是实际上，`shell` 提供了一种在 `shell` 脚本里包含编辑脚本的方法。

嵌入文档 (here documents)

在 shell 脚本中，操作符 `<<` 意思是将其之后到指定的串之前的所有行作为命令的输入（通常称为嵌入文档）。使用这个语法，我们可以按下面的方式在 “*correct*” 文件中包含编辑命令：

```
for file in $*
do
ex - $file << end-of-script
g/thier/s//their/g
g/writeable/s//writable/g
wq
end-of-script
done
```

字符串 *end-of-script* 完全是任意的，它只需是一个未出现在输入中的字符串，可以被 shell 用于识别嵌入文档结束位置。按照表示文件结束的习惯，许多用户使用字符串 *EOF* 或 *E_O_F* 来表示嵌入文档的结束。

上面展示的每种方法都有优缺点。如果想一次进行一系列的编辑而又不想每次重新编写脚本，那么嵌入文档则提供了一种有效地完成该工作的方法。

但是，在不同于 shell 脚本的单独文件中输入编辑命令更具有灵活性。例如，你应养成把编辑命令放到名为 “*exscript*” 文件中的习惯。这样只需要编写一次 *correct* 脚本，把它存储到个人的 “*tools*” 目录下（该目录已经添加到搜索路径中），然后无论什么时候都可以使用它。

文本块排序：简单的 ex 脚本

假如你想对 *troff* 编码的术语定义文件按字母排序。每个术语都以 *.IP* 宏开头，并且每个条目都由 *.KS/.KE* 宏对包围着（这样可确保术语及其定义将作为块进行显示，而且不会被拆分到一个新页中）。文件的内容如下所示：

```
.KS
.IP "TTY_ARGV" 2n
The command, specified as an argument vector,
```

```

that the TTY subwindow executes.

.KE
.KS
.IP "ICON_IMAGE" 2n
Sets or gets the remote image for icon's image.

.KE
.KS
.IP "XV_LABEL" 2n
Specifies a frame's header or an icon's label.

.KE
.KS
.IP "SERVER_SYNC" 2n
Synchronizes with the server once.

Does not set synchronous mode.

.KE

```

虽然可以通过在所有行中执行 UNIX 的 sort 命令来对文件按字母排序，但并不一定是想对所有行进行排序。可能仅仅想排序术语条目，使每个定义与其所对应的术语一起移动。通过把块连接为一行，即可把每个文本块作为一个单元进行处理。下面是 *ex* 脚本的第一个版本：

```

g/^\.KS/,/^\.KE/j
%!sort

```

每个术语条目都在 .KS 和 .KE 宏之间。*j* 是 *ex* 中合并行的命令（等同于 *vi* 中的 *J*）。这样，第一个命令把每个术语条目连接为一行，然后第二个命令对文件进行排序。生成的行如下所示：

```

.KS .IP "ICON_IMAGE" 2n Sets or gets ... image. .KE
.KS .IP "SERVER_SYNC" 2n Synchronizes with ... mode. .KE
.KS .IP "TTY_ARGV" 2n The command, ... executes. .KE
.KS .IP "XV_LABEL" 2n Specifies a ... icon's label. .KE

```

这些行现在已按术语条目进行排序，但每一行都把宏和文本混合在一起（使用省略号[...]来显示忽略的文本）。由于某种原因，需要插入换行符来分解各行。我们可以通过修改 *ex* 脚本来完成任务：在连接文本块之前标记它们的连接点，然后使用新行代替那些标记。下面是扩展的 *ex* 脚本：

```

g/^\.KS/,/^\.KE/-1s/$@@/

```

```
q/^\.KS/,/^\.KE/j
%!sort
%s/@@ /^M/g
```

前三个命令生成的行如下所示：

```
.KS@@ .IP "ICON_IMAGE" 2nn@@ Sets or gets ... image. @@ .KE
.KS@@ .IP "SERVER_SYNC" 2nn@@ Synchronizes with ... mode. @@ .KE
.KS@@ .IP "TTY_ARGV" 2nn@@ The ... vector, @@that ... . @@ .KE
.KS@@ .IP "XV_LABEL" 2nn@@ Specifies a ... icon's label. @@ .KE
```

要注意 @@ 后面的特殊空格。这些空格是由 `j` 命令产生的，因为 `j` 把每个换行符转换为一个空格。

第一个命令使用 @@ 标记原始行的断点。由于第一个命令使用 -1 返回到位于每块尾部的行，因此不需要标识块的结尾（在 .KE 之后）。第四个命令通过使用换行符替换标记（加额外的空格）来恢复行的断点。现在文件将按块排序。

ex 脚本中的注释

有时需要重用上面那样的脚本，使它适用于新的情况。在使用复杂的脚本时，对它加上注释是非常明智的，这将使其他人（甚至你自己）更容易理解脚本的工作原理。在 `ex` 脚本中，双引号后的一切都将在执行期间忽略，因此双引号可以标记注释的开始。注释可以有它们自己的行，也可以在任何没有把引号作为其组成部分的命令的结尾（例如，由于引号表示映射命令和 shell 转义，因此不能把注释放在那些行的结尾）。

除了使用注释外，还可以用它的全名来指定命令，通常这些处理在 `vi` 中将是非常费时的。最后，如果添加一些空格，上面的 `ex` 脚本将会更具有可读性：

```
" Mark lines between each KS/KE block
global /^\.KS/,/^\.KE/-1 s /$/@@/
" Now join the blocks into one line
global /^\.KS/,/^\.KE/ join
" Sort each block--now really one line each
%!sort
```

```
" Restore the joined lines to original blocks  
% s /@@ /^M/g
```

注意，即使可以执行其他命令的全名，`substitute`命令在 `ex` 中也没有任何作用。

ex 之外

UNIX 提供了比 `ex` 更强的编辑器：`sed` 流编辑器和 `awk` 数据操作语言，还有极流行的 Perl 编程语言。要知道有关这些编辑器的更多信息，请参考 O'Reilly 出版的《`sed & awk`》、《`Learing Perl`》和《`Programming Perl`》。

编辑程序源代码

无论正在编辑的是英文文本还是程序源代码，前面讨论的所有功能都是非常重要的。但是，也有许多其他主要针对程序员的重要功能，它们包括缩进控制、搜索过程的开始和结尾以及使用 `ctags`。

下面的讨论摘自 Mortice Kern Systems 使用 `vi` 工具提供的文档，该工具可用于 DOS 和基于 Windows 的系统，也可用做 MKS 工具箱的一部分或单独作为 MKS Vi 使用。这些讨论在 Mortice Kern Systems 的授权下使用。

缩进控制

程序源代码在许多方面都与普通文本不同，其中最重要的区别之一是源代码使用缩进的方式。缩进显示了程序的逻辑结构：在这种方式下语句被分为许多块。`vi` 提供自动的缩进控制，如果要使用这种功能，可以调用命令：

```
:set autoindent
```

现在，如果使用空格或制表符对一行进行了缩进，接下来的行将会自动按同样的大小进行缩进。在输入完第一个缩进的行按下 `RETURN` 键以后，光标就会到达下一行并自动缩进与上一行同样的距离。

作为一个程序员，你将发现这会节省大量使缩进正确的工作，尤其文档中有一些不同级别的缩进时更是如此。

在你使用自动缩进输入代码时，在行的开始输入 **CTRL-T** 就会实现另一个级别的缩进，输入 **CTRL-D** 则会取消一个缩进级别。

我们应该指出，**CTRL-T** 和 **CTRL-D** 要在插入模式下输入，而不同于大部分其他的命令，这些命令都在命令模式下输入。

还有其他两种不同的 **CTRL-D** 命令（注 8）。

^ ^D

当输入 **^ ^D** (**】 [CTRL-D]**) 时，*vi* 把光标移回行的开始，但是只针对当前行。输入的下一行将以当前的自动缩进级别开始。这一点对在输入 C/C++ 源代码时输入 C 预处理命令尤其有用。

0 ^D

当输入 **0 ^D** 时，*vi* 把光标移回行的开始，并且当前的缩进级别还将重新设置为 0；输入的下一行将不再自动缩进（注 9）。

在输入源代码时要试着使用 *autoindent* 选项。它简化了正确缩进的工作，有时甚至还能避免程序错误（例如，在 C 源代码中，通常需要一个右大括号 **}** 来表示返回的每一个缩进级别）。

<< 和 **>>** 命令在缩进源代码时也很有用。默认情况下，**>>** 把行向右移动 8 个空格（即，添加 8 个空格的缩进），**<<** 把行向左移动 8 个空格。例如，把光标移动到行的开始并按两次**>** 键(**>>**)，你将看到行向右移动了。如果你现在按两次**<** 键(**<<**)，该行就会向原始位置移动。

可以通过在**>>** 或 **<<** 前面输入数字来移动多行。例如，移动光标到大型段落的首行并输入 **5>>**，将会移动该段落中的前 5 行。

注 8： 这些操作在 *elvis 2.0* 中不起作用。

注 9： *nvi 1.79* 文档中不仅有这两个移动命令，而且编辑器的真正行为也与这里描述的相同。

默认情况下移动 8 个空格（向左或向右）。该默认可以使用下面的命令进行修改：

```
:set shiftwidth=4
```

注意，将 shiftwidth 与制表位之间的宽度大小设置为相同是很方便的。

通常，当每次按 8 个空格缩进的文本时，*vi* 实际上是把制表符插入到文件中，这是因为通常将制表符扩展为 8 个空格。这是 UNIX 的默认值，在正常输入期间输入一个制表符时通常都能发现这一点，当打印文件时，UNIX 使用 8 个空格的制表位对它们进行扩展。

可以通过修改 tabstop 选项来改变 *vi* 在屏幕上显示制表符的方式。例如，如果需要缩进很多，可以使用每 4 个字符的制表位设置，以至于不会换行。下面的命令将完成这个改变：

```
:set tabstop=4
```

注意：我们不推荐修改制表位。虽然 *vi* 能使用任意的制表位设置来显示文件，但是其他的 UNIX 编辑器仍然会使用 8 字符的制表位对文件中的制表符进行扩展。8 字符的制表位是 UNIX 文件中的常见规格。

由于我们常会将一个或多个空格误认为制表符，因此有时缩进不能按期望的方式进行。通常，屏幕把制表符和空格都显示为空白，使得这两者看起来好像没有区别。但是，可以调用命令：

```
:set list
```

这将改变屏幕显示，使制表符显示为控制字符 ^I，行尾显示为 \$。通过这种方式可以识别真正的空格，还能看到行尾的额外空格。临时的等价命令是 :l。例如，命令：

```
:5,20 l
```

会显示第 5 行到第 20 行，并显示出制表符和行尾符。

一个特殊的搜索命令

字符 (、[、{ 和 < 都可以称为开括号，当光标位于这些字符中的一个时，按下 % 键就会把光标从开括号向前移动到对应的闭括号 ——)、}、] 或 >，请记住这个嵌套括号的规则（注 10）。例如，如果光标在下面语句中的第一个(上：

```
if ( cos(a[i]) > sin(b[i]+c[i]) )
{
    printf("cos and sin equal!\n");
}
```

按下%后，光标将跳到行尾的圆括号上。这就是与该开圆括号相匹配的闭圆括号。

同样，如果光标位于这些闭括号字符中的一个，按下%将把光标向后移动到对应的开括号字符上。例如，把光标移到上面printf行后面的闭括号上然后按下%。

vi 在搜索括号字符时十分灵活。如果光标不在括号字符上，在按下%后，*vi* 将会在当前行中向前寻找它发现的第一个开或闭括号字符，然后移动到与之相匹配的那个字符。例如，假如光标位于上例中第一行的 > 上，% 将会发现开圆括号，然后把光标移到与之对应的闭圆括号。

这个搜索字符不仅可以在程序中跳跃着向前和向后移动，而且还有助于检查源代码中的方括号和大括号的嵌套。例如，如果把光标定位到 C 函数开始的第一个 { 上，按下%应该把光标移动到该函数结尾的那个 } 上。如果没有移动到那个大括号上，那么肯定出现了错误。如果在文件中没有相匹配的 }，*vi* 将会发出嘟嘟的声音。

另一个发现匹配括号的技巧是启用下面的选项：

```
:set showmatch
```

与%不同，在插入模式时设置 showmatch（或它的缩写 sm）将十分有用。当输

注 10： 在测试过的*vi*版本中，只有*nvi*支持使用%来匹配<和>。*vile*允许你设置用%进行匹配的一对字符集的选项。

入)或}时(注11),光标在返回当前位置之前将会短暂地移回相匹配的(或{。如果相匹配的括号字符不存在,终端就会发出嘟嘟的声音。如果相匹配的括号字符不在当前屏幕上,vi就会继续移动。

使用标志

大型C或C++程序中的源代码通常会分布在几个文件中。有时,要记住哪个文件包含哪个函数定义是很困难的。为了简化这些处理,可以把称为ctags的UNIX命令与vi的:tag命令结合在一起使用。

注意: ctags的UNIX版本可处理C语言、常用的Pascal和Fortran 77,有时它们甚至能处理汇编语言。但是,它们普遍都不能处理C++。也有一些可用来为C++、其他的语言和文件类型生成tags文件的其他版本。

可在UNIX命令行上调用ctags命令,它的目的是创建信息文件,vi可以使用该文件来确定哪个文件定义了哪个函数。默认情况下,该文件称为tags。在vi内部,其格式为:

```
:!ctags file.c
```

该命令将在当前目录下创建一个名为“tags”的文件,该文件包含有关在file.c中定义的函数的信息。下面的命令:

```
:!ctags *.c
```

将创建描述该目录下所有C源文件的tags文件。

现在假设tags文件包含能组成一个C程序的所有源文件的信息,并且我们想查看或编辑程序中的某个函数,但是并不知道该函数的位置。在vi内部,下述命令:

```
:tag name
```

注11: 在elvis、vim和vile中,showmatch也将显示相匹配的方括号([和])。

将查看 *tags* 文件来寻找定义 *name* 函数的文件，然后将读取该文件并把光标定位到定义 *name* 函数的行上。这样，只需要确定你要编辑的函数而不是文件。

也可以在 *vi* 的命令模式下使用 *tag* 功能。把光标定位到所要查找的标识符上，然后输入 ^]，*vi* 将会搜索 *tag* 并移动到定义该标识符的文件。要当心光标放置的位置，*vi* 使用光标下面从当前光标位置开始的“单词”，而不是包含光标的整个单词。

注意：如果使用:*tag* 命令来读取新文件并且自上次修改以来没有保存当前文件，那么 *vi* 将不允许访问新文件。必须使用:w 命令保存当前文件然后再调用:*tag*，或者输入:

```
:tag! name
```

忽略 *vi* 的一些编辑操作。

Solaris 2.6 中的 *vi* 版本真正支持标志栈，但是在 Solaris 的帮助页中并没有对其完全文档化。由于许多 UNIX 的 *vi* 版本不支持标志入栈，因此我们把对该功能的讨论放在第八章的“标志栈”一节，那里引入了标志入栈。





第二部分

扩展功能和克隆版本

第二部分对各种用来扩展*vi*性能的新功能以及它们在*vi*四种克隆版本中的可用性进行了介绍。本部分包含下列章节：

- 第八章，*vi* 克隆版本的功能总结
- 第九章，*nvi*——新*vi*
- 第十章，*elvis*
- 第十一章，*vim*——改进的*vi*
- 第十二章，*vile*——类*Emacs* 的*vi*





本章内容：

- vi 的各种克隆版本
- 多窗口编辑
- GUI 接口
- 扩展的正则表达式
- 增强的标志
- 改进的功能
- 编程辅助
- 编辑器比较小结
- 后面内容预览

第八章

vi 克隆版本的 功能总结

vi 的各种克隆版本

vi 编辑器有许多免费可用的“克隆”版本。附录五“vi 和 Internet”提供了列出所有知名的*vi* 克隆版本的 Web 站点，我们选择了其中最流行的四个进行介绍。它们是：

- Keith Bostic 的 *nvi* 1.79 版
- Steve Kirkendall 的 *elvis* 2.0 版
- Bram Moolenaar 的 *vim* 5.0 版
- Kevin Buettner、Tom Dickey 和 Paul Fox 的 *vile* 7.4 版

编写这些克隆版本的原因是由于*vi* 的源代码不是免费可用的，这使得把*vi* 移植到非 UNIX 环境或研究它的源代码变得不可能，或者是由于 UNIX 的*vi*（或其他的克隆版本）不能提供想要的功能。例如，UNIX 的*vi* 通常对行的最大长度有限制并且不能编辑二进制文件（与各种编辑器相关的章节提供了更多有关它们各自的历史信息）。

每个编辑器都对UNIX的*vi*进行了大量扩展；虽然一些克隆版本提供了相同的扩展，但是它们的扩展方式不同。本章我们不是对在每个编辑器的章节中所讨论的每个普通功能进行重复，而是集中讨论它们的实现方法。你可以把本章认为是对“各个克隆版本做什么”进行讨论，而每个克隆版本的章节则讨论“该版本如何工作”。

本章包括下列主题：

多窗口编辑

该功能把屏幕分割为多个“窗口”（注1），你可以在每个窗口编辑不同的文件，或者在多个窗口编辑同一个文件。这也许是常规*vi*的惟一最重要的扩展。

GUI 接口

除*nvi*之外的所有克隆版本都可以将其编译以支持X Window接口。如果是在运行X的系统中，GUI版本的使用可能比分割*xterm*（或其他的终端仿真程序）的屏幕效果更好。GUI版本通常提供像滚动条和多字体那样的优秀功能。也可能支持其他操作系统自带的GUI。

扩展的正则表达式

所有的克隆版本都使用与UNIX*egrep*(1)命令提供的正则表达式相似或相同的正则表达式进行文本匹配。

增强的标志

正如第七章“高级编辑”中“使用标志”一节所介绍的，可以使用*ctags*程序来构建文件的可搜索数据库。通过在进行标志搜索时保存当前位置，克隆版本使标志入栈成为可能。以后还可以返回到那个位置。多个位置可以按后进先出(LIFO)的顺序进行保存，从而形成一个位置栈。

一些*vi*克隆版本的作者和*ctags*克隆版本的作者一起为*ctags*格式的增强版定义了一个标准形式。特别是现在使用允许对函数名重载的C++编写的编辑器，可以使标志功能的使用变得更为容易。

注1：要注意这些窗口并不是读者在基于X Window的UNIX工作站上或者在MS-Windows或Apple Macintosh下所发现的窗口。

改进的编辑功能

所有的克隆版本都提供了编辑 *ex* 命令行的能力、“无限撤消”能力、任意长度的行和 8 位数据、增量搜索、（至少一个选项）利用对长行从左到右滚动来取代对长行折行、模式指示以及其他功能。

编程辅助

一些编辑器提供了在典型的“编辑 – 编译 – 调试”软件开发循环期间允许用户保持在编辑器中的功能。

语法高亮显示

在 *elvis*、*vim* 和 *vile* 中，可以使用不同的颜色和 / 或字体显示文件的不同部分。这在编辑程序源代码时尤其有用。

克隆版本中的另外一个功能我们没有进行讨论，那就是扩展语言。在 1998 年 6 月，*nvi* 对 Perl 和 Tcl 集成有了基础的支持；*elvis* 有它自己的类似 C 的表达式计算程序（注 2）；*vim* 有类似 C 的表达式计算程序，并且支持 Perl、Python 和 Tcl 集成；一直拥有自己内置扩展语言的 *vile* 支持了 Perl 集成。扩展语言的集成和支持对于所有的编辑器还将继续变化。由于这个原因，任何对扩展语言工具的讨论几乎在该书出版的时候就会变得过时。

如果你对用扩展语言对编辑器进行编程感兴趣，我们建议你检查克隆版本的联机文档（注 3）。扩展语言是个值得关注的功能，它们承诺要带给 *vi* 用户全新的感受。由于对用户来说，熟悉 Perl、Python 和 Tcl 那样众所周知的编程语言中的一种或多种是可能的，因此使用它们是很有帮助的。

多窗口编辑

也许各种克隆版本为标准 *vi* 提供的惟一一个最重要的功能就是在多个“窗口”编

注 2： *elvis* 2.0 文档提到 *elvis* 将拥有真正的扩展语言，它很像 Perl 但是很可能不适用于 2.1 版本。Steve Keikendall 并没有真正把表达式计算程序看做扩展语言。

注 3： 从一开始，*emacs* 用户就一直在做这项工作，这也是它的许多用户对他们的编辑器相当狂热的一个原因。

辑多个文件的能力。这使得同时容易地编辑多个文件成为可能，也使得通过复制和粘贴从一个文件到另一个文件“剪切和粘贴”文本成为可能（注 4）。

支持每个编辑器的多窗口功能的两个基本概念是：缓冲区和窗口。

缓冲区存储将要编辑的文本。这些文本可能来自某个文件，或者可能是最后要写到文件中的新文本。任何给定的文件都只有一个缓冲区与它相关联。

窗口把文件视图放入缓冲区中，这使得你可以在缓冲区中浏览和修改文本。一个缓冲区可能会与多个窗口相关联。在一个窗口中对缓冲区的修改将会反映到在同一缓冲区打开的任何其他窗口中。缓冲区也可能没有与它相关联的窗口，在这种情况下，虽然可以在以后为它打开一个窗口，但是不能使用该缓冲区进行许多处理。关闭在缓冲区上打开的最后一个窗口等效于“隐藏”该文件。如果该缓冲区已改变但是没有将其写到磁盘中，那么编辑器可能会也可能不会让你关闭那个缓冲区打开的最后一个窗口。

当创建新窗口时，编辑器就会分割当前屏幕。对于大部分编辑器，创建的新窗口将显示当前正在编辑的文件的另一个视图。然后切换到想编辑的下一个文件的窗口，并指示编辑器在那里开始编辑该文件。每个编辑器都提供在窗口之间来回切换的 *vi* 和 *ex* 命令以及改变窗口大小、隐藏和重现窗口的功能。

在每个编辑器的章节，我们将展示简单的分割屏幕（编辑同样的两个文件）并介绍如何分割屏幕和在窗口之间移动。

GUI 接口

elvis、*vim* 和 *vile* 还提供能利用鼠标和位图化显示的图形用户接口（GUI）版本。除了支持 UNIX 下的 X Window 外，对于 MS-Windows 或其他窗口化系统的支持也可能是可用的。表 8-1 对不同克隆版本可用的 GUI 进行了总结。

注 4： 在克隆版本中，不需要通过分割屏幕来在文件之间进行复制和粘贴，只有最初的 *vi* 才在切换文件时放弃剪切缓冲区。

表 8-1 可用的 GUI

编辑器	X11	MS-Windows	OS/2	BeOS	Macintosh	Amiga
<i>elvis</i>	✓	✓	✓			
<i>vim</i>	✓	✓	✓	✓	✓	✓
<i>vile</i>	✓	✓	✓			

扩展的正则表达式

可用于 *vi* 搜索和替换正则表达式的元字符已在第六章“全局替换”中的“元字符在搜索模式中的使用”一节进行了描述。每个克隆版本把一些形式的扩展正则表达式或者作为选项提供，或者作为一直可用的功能来提供。通常这些正则表达式与 *egrep* 提供的正则表达式相同（或几乎相同）。遗憾的是，每个克隆版本的扩展方式与其他的克隆版本略有不同。

为了使你对扩展的正则表达式的功能有一定了解，我们要在 *nvi* 环境中列举它们。后面每个克隆版本的章节会对该编辑器的扩展语法进行介绍，这里不再对各个例子进行重复。

nvi 扩展正则表达式是由 POSIX 标准定义的扩展正则表达式 (ERE)。为了启动该功能，可从 *.nexrc* 文件或 *ex* 冒号提示符下使用 `set extended`。

除了在第六章介绍的标准元字符和在同一章的“POSIX 的方括号表达式”小节提到的 POSIX 括号表达式外，下面的元字符也是可用的：

- | 指示交替。例如，`a|b` 匹配 *a* 或 *b*。但是，该命令并不只限制于单个字符：`house|home` 匹配字符串 *house* 或 *home*。

(...)

用于分组，允许应用额外的正则表达式操作符。例如 `house|home` 可缩写（如果不是简化）为 `ho (use|me)`。* 操作符可应用到圆括号上：(`house|home`) * 匹配 *home*、*homeshome*、*househomehousehouse* 等等。

当设置 `extended` 时，使用圆括号分组后的文本产生的效果与在正规 `vi` 中 `\(...)` 分组后的文本相似，实际匹配的文本可以使用 `\1`、`\2` 等在替换命令的替换部分进行检索。在这种情况下，`\`（代表实际的左圆括号）。

- + 匹配前面正则表达式的一次或多次出现。这既可以是单个字符，也可以是圆括号中的一组字符。要注意 `*` 和 `+` 之间的区别，`*` 可允许不匹配任何内容，而这里的 `+` 必须至少是一个匹配。例如，`ho(use|me)*` 既匹配 `ho` 也匹配 `home` 和 `house`，但 `ho(use|me)+` 将不匹配 `ho`。
- ? 匹配前面正则表达式的零次或一次出现。这表示出现或者未出现“可选择”的文本。例如，`free?d` 将匹配 `fred` 或 `freed`，除此之外不匹配任何其他文本。

`{...}`

定义区间表达式。区间表达式描述了表示重复次数的计数数字，在下面的描述中，`n` 和 `m` 代表整型常数。

`{n}`

对前面正则表达式的 `n` 次重复进行匹配。例如，`(home|house){2}` 匹配 `homehome`、`homehouse`、`househome` 和 `househouse`，除此之外不匹配任何其他文本。

`{n,}`

对前面正则表达式的 `n` 次或更多次的重复进行匹配。可以把它认为“至少 `n` 次”。

`{n,m}`

对 `n` 到 `m` 次的重复进行匹配。由于该范围控制着在替换命令中有多少文本将被替换，因此它是很重要的（注 5）。

当没有设置 `extended` 时，`nvi` 使用 `\{` 和 `\}` 来提供同样的功能。

增强的标志

“Exuberant ctags” 程序被认为是比 UNIX 的 `ctags` 功能更强的 `ctags` 克隆版本。它

注 5： 虽然 `*`、`+` 和 `?` 操作符使用起来更方便，但是可以将其还原为 `{0,}`、`{1,}` 和 `{0,1}`。

生成了一种扩展的*tags*文件格式，该格式使得标志搜索和匹配变得更灵活和更容易处理。由于*vi*的许多克隆版本都支持它，因此我们首先对它进行介绍。

本章还介绍了标志栈：存储用：`:tag`或`^]`命令访问过的多个位置的功能。所有克隆版本都提供了标志栈。

Exuberant ctags

“Exuberant ctags”程序是由 Darren Hiebert 编写的，它的主页是 <http://home.hiwaay.net/~darren/ctags>。到编写这本书为止，该程序目前的版本是 2.0.3。下面关于该程序的功能列表摘自 *ctags* 发行版中的 *README* 文件。

- 它具有为所有类型的C和C++语言标记生成标志的能力，其中包括类名、宏定义、枚举名枚举值（枚举中的值）、函数（方法）定义、函数（方法）原型/声明、结构成员和类数据成员、结构名、类型定义、联合名和变量。
- 它支持 C 和 C++ 代码。
- 它具有很强的代码解析能力，很容易处理含有`#if`预处理程序的条件指令。
- 它也可以用来打印在源文件中发现的选定对象的可读列表。
- 它支持生成 GNU 的 *emacs-style* 标志文件 (*etags*)。
- 它可在 UNIX、QNX、MS-DOS、Windows 95/NT、OS/2 和 Amiga 上运行。一些预编译过的二进制代码可以从 Web 站点上获得。

Exuberant *ctags* 使用下一小节所介绍的格式生成 *tags* 文件。

新的标志格式

传统上，*tags*文件有三个用制表符分开的域：标志名（通常的标识符）、包含标志的源文件和标识符的位置标识。该标识或者是简单的行号，或者是由斜杠或问号包围的`nomagic`搜索模式。而且，*tags*文件通常是排过序的。

这是由UNIX的*ctags*程序生成的格式。实际上，*vi*的许多版本都允许在搜索模式域中存在任何命令（一个相当大的安全漏洞）。而且，由于未形成文献的实现巧合，如果行以分号和双引号（; "）结束，那么这两个字符后面的任何文本都将被忽略（正如双引号在*.exrc*文件中的作用一样，它标志着注释的开始）。

新格式向后兼容传统格式。前面的三个域是相同的：标志、文件名和搜索模式。*Exuberant ctags*只生成搜索模式，而不是任意的命令。扩展属性则放置在分隔符；" 的后面。各属性之间由制表符隔开，每个属性都由用冒号隔开的两个子域构成。第一个子域描述该属性的关键字，第二个子域是真实值。表 8-2 列出了所支持的关键字。

表 8-2 扩展的 *ctags* 的关键字

关键字	含义
kind	值是表示标志的词汇类型的单个字母，它可以是表示函数的 f、表示变量的 v 等等。由于默认的属性名是 kind，因此单个字母可以表示标志的类型（如 f 表示函数）
file	表示“静态的”、即文件的局部的标志。值应该是文件的名字 如果指定为一个空串（如 file:），应理解为与文件名域相同；加上这种特殊的情况一方面是出于简洁方面的考虑，另一方面是为了提供一种简单的方法来处理那些不在当前目录下的标志文件。文件名域的值总是相对于该 <i>tags</i> 文件本身所在的目录
function	表示局部标志，值是定义这些标志的文件名
struct	表示 struct 中的域，值是该结构的名字
enum	表示 enum 数据类型中的值，值是 enum 类型的名字
class	表示 C++ 成员函数和变量，值为该类的名字
scope	主要的用意是表示 C++ 类成员函数。它通常为 private，表示私有成员，或者省略以表示默认的公有成员，这样用户可以把标志搜索限制为只对公有成员
arity	表示函数。参数的数量

如果该域不含有冒号，就将其假定为 kind 类型。下面是一些例子：

```

ARRAYMAXED      awk.h      427;"d
AVG_CHAIN_MAX  array.c    38;"d      file:
array.c         array.c    1;"F

```

ARRAYMAXED是在`awk.h`中定义的一个C `#define`宏, AVG_CHAIN_MAX也是一个C宏, 但是它只是在`array.c`中使用。第三行有一些不同: 它是一个代表真实源文件的标志, 这由带有`-i F`选项的Exuberant *ctags*生成, 并且允许使用`:tag array.c`命令。更有用的是, 可以把光标放在某个文件名上, 然后使用`^】`命令就可以进入该文件。

在每个属性的值域部分, 反斜杠、制表符、回车键和换行符应该分别编码为`\\"`、`\t`、`\r`和`\n`。

扩展的`tags`文件可能有一些以`!_TAG_`开头的原始标志。这些标志通常会被排序到文件的前部, 它们对识别创建该文件的程序是很有用的。下面是Exuberant *ctags*生成的初始标志:

```

!_TAG_FILE_FORMAT      2          /extended format; ..../
!_TAG_FILE_SORTED     1          /0=unsorted, 1=sorted/
!_TAG_PROGRAM_AUTHOR   Darren Hiebert /darren@hiebert.com/
!_TAG_PROGRAM_NAME    Exuberant Ctags //
!_TAG_PROGRAM_URL     http://home.hiwaay.net/~darren/ctags /...
!_TAG_PROGRAM_VERSION  2.0.3     /with C++ support/

```

编辑器可能利用这些特殊标志来实现特殊功能。例如, *vim*发现`!_TAG_FILE_SORTED`标志后, 如果该文件确实已被排序, 它将会使用折半查找而不是线性查找对`tags`文件进行搜索。

如果使用`tags`文件, 我们建议安装Exuberant *ctags*。

标志栈

*ex*的`:tag`命令和*vi*的`^】`模式命令提供了一种根据`tags`文件所提供的信息搜索标识符的受限方法。每个克隆版本都通过维护一个标志位置栈而对该功能进行了扩展。每次调用*ex*的`:tag`命令, 或者使用*vi*的`^】`模式命令, 编辑器都会在搜

索指定的标志前保存当前位置。然后就可以使用 ^T 命令或者 ex 命令返回到已保存的位置。

下面列举了 Solaris vi 的标志栈和一个实例。每个克隆版本处理标志栈的方法将在每个编辑器的各自章节中进行介绍。

Solaris vi

令人惊讶的是，Solaris 2.6 版的 vi 支持标志栈。该功能完全没有在 Solaris 的 ex(1) 和 vi(1) 手册页中被文档化。出于完整性考虑，我们在表 8-3、表 8-4 和表 8-5 中对 Solaris vi 的标志栈进行了总结。Solaris vi 中的标志栈相当简单（注 6）。

表 8-3 Solaris vi 的标志命令

命令	功能
ta[g] [!] tagstring	编辑在 tags 文件中定义的包含 tagstring 的文件，如果当前缓冲区已修改而且没有保存，! 会强迫 vi 切换到新文件
po[p] [!]	弹出标志栈中的一个单元

表 8-4 Solaris vi 命令模式下的标志命令

命令	功能
^]	在 tags 文件中寻找光标下的标识符的位置，并移动到那个位置。如果设置标志入栈，当前位置将自动放入标志栈中。
^T	返回到标志栈中前面的位置，即弹出一个单元。

表 8-5 Solaris vi 中用于标志管理的选项

选项	功能
taglength, t1	控制要搜索的标志中有效字符的数目。默认值 0 表示所有的字符都是有效的

注 6：该信息是根据经验发现的。YMMV（你的里程数是会变的）。

表 8-5 Solaris vi 中用于标志管理的选项（续）

选项	功能
tags, tagpath	该值是要进行搜索标志的文件名列表，默认值是 "tags /usr/lib/tags"
tagstack	当设置为真 (true) 时，vi 会使每个位置进入标志栈中。使用 :set notagstack 来取消标志入栈

为了使你对标志栈的使用有个概念，我们列举了一个使用 Exuberant *ctags* 和 *vim* 的简单例子。

假如你正在编写一个使用 GNU 的 *getopt_long* 函数的程序，并需要了解该函数的更多信息。

GNU 的 *getopt* 由三个文件 *getopt.h*、*getopt.c* 和 *getoptl.c* 组成。

首先创建 *tags* 文件，然后在 *main.c* 中找到主程序后开始对它进行编辑。

```
S ctags *.[ch]
S ls
Makefile      getopt.c      getopt.h      getoptl.c      main.c      tags
S vim main.c
```

击键

结果

/getopt

```
/* option processing. ready, set, go! */
for (optopt = 0, old_optind = 1;
     (c = getopt_long(argc, argv, optlist,
                      optab, NULL)) != EOF;
     optopt = 0, old_optind = optind) {
    if (do_posix)
        optarg = TRUE;
```

编辑 *main.c* 并移动到调用 *getopt_long* 的位置。

^]

```

int
_getopt_long (argc, argv, options,
              long_options, opt_index)
    int argc;
    char *const *argv;
    const char *options;
    const struct option *long_options;
    int *opt_index;
{
    return _getopt_internal (argc, argv, options,
                           long_options,
                           opt_index, 0);
}
" getopt1.c " 189 lines, 4651 characters

```

对 *getopt_long* 进行标志搜索。*vim* 将移动到 *getopt1.c* 上，并把光标放置在 *getopt_long* 的定义上。

这将显示 *getopt_long* 是 *_getopt_internal* 的“打包”函数，将光标放置在 *_getopt_internal* 上并再次进行标志搜索。

击键

结果

8jf_ ^]

```

int
_getopt_internal (argc, argv, optstring,
                  longopts, longind, long_only)
    int argc;
    char *const *argv;
    const char *optstring;
    const struct option *longopts;
    int *longind;
    int long_only;
{
    optarg = NULL;
}
" getopt.c " 1000 lines, 28705 characters

```

现在已经移动到 *getopt.c* 中，如果要了解 *struct option* 更多的信息，可以把光标移动到 *option* 上再次进行标志搜索。

5jfo; ^]

```

one.) For long options that have a zero 'flag' field, 'getopt'
returns the contents of the 'val' field. */

struct option
{
    #if defined (__STDC__) && __STDC__
        const char *name;
    #else
        char *name;
    #endif
}
" getopt.h " 133 lines, 4691 characters

```

编辑器移动到 *getopt.h* 中的 *struct option* 定义处。现在可以仔细地查看解释它如何使用的注释了。

```
:tags
```

#	TO tag	FROM line	in file
1	getopt_long	205	main.c
2	_getopt_internal	75	getopt1.c
3	option	68	getopt.c

vim 中的 :tags 命令显示标志栈。

输入三次 ^T 将使你返回到 *main.c*, 这是开始标志搜索的位置。标志功能使你可以很容易地在编辑源文件时到处移动。

改进的功能

四种克隆版本都提供了使简单文本编辑起来更容易、更有效的功能。

编辑 ex 命令行

在输入 *ex* 模式命令时对它们进行编辑的功能，可能包括已保存的 *ex* 历史命令，还有把文件名和命令与选项之类的其他文本补充完整的功能。

没有行长度限制

编辑任意长度的行的功能，还有编辑包含任何 8 位字符的文件的功能。

无限撤消

成功撤消对某个文件进行的所有编辑的功能。

增量搜索

在输入搜索模式的同时搜索文本的功能。

左/右滚动

使较长的行延伸到屏幕边缘的后面而不是对其进行换行的功能。

可视模式

选择任意连续的文本块进行某些操作的功能。

模式标识

区别插入模式与命令模式的可见标识，以及当前行号和列号的标识。

命令行历史和完整化

csh、*tcsh*、*ksh* 和 *bash* 的用户已经知道，重新调用前面的命令、对它们做少量编辑后重新提交使这些 shell 更实用。

相对于 shell 用户，这种特性对编辑器用户同样有用。遗憾的是，UNIX 的 *vi* 没有提供任何保存和重新调用 *ex* 命令的功能。

每个克隆版本都对这种缺陷进行了补救。虽然每个编辑器为保存和重新调用历史命令提供了不同的方法，但是每个编辑器所采取的机制都是有效和有用的。

除了命令外，所有编辑器都可以提供某些类型的完整化。例如，你输入了某个文件名的开头，然后输入某个特殊字符（如制表符），编辑器就会把文件名补充完整。所有的编辑器都能把文件名补充完整，某些还可以把其他内容补充完整。细节将在每个编辑器的章节提供。

任意长度的行和二进制数据

这四种克隆版本都可以处理任意长度的行（注7），过去的 *vi* 版本通常把每行限制在 1000 个字符左右，更长的行将被截断。

这四种克隆版本也是完整 8-比特的，意思是它们可以编辑任何包含 8 位字符的文件，甚至还可以编辑二进制和/或可执行文件。在许多时候，这是非常有用的。你不必告诉每个编辑器文件是二进制的。

nvi

自动处理二进制数据，不需要特殊的命令行或 *ex* 选项。

elvis

在 UNIX 下，它不区别对待二进制文件与任何其他的文件。在其他的操作系统上，使用 *elvis.brf* 文件设置 *binary* 选项，以避免换行符的转换问题（*elvis.brf* 文件和 *hex* 显示模式将在第十章的“令人感兴趣的功能”节中进行介绍）。

注 7：高达 C 的 long 类型最大值 2 147 483 64。

vim

对行的长度没有限制。当没有设置 `binary` 时，*vim* 像 *nvi* 一样会自动处理二进制数据。但是，在编辑二进制文件时，需要使用 `-b` 命令行选项或者设置：`:set binary`。同时也将设置一些其他的 *vim* 选项，从而可以更容易地编辑二进制文件。

vile

自动处理二进制数据，不需要特殊的命令行或 *ex* 命令选项。

最后，还有一个难处理的细节。传统的 *vi* 总在最后追加换行符来写文件，在编辑二进制文件时，这可能会把一个字符添加到文件中，从而产生问题。默认情况下，*nvi* 和 *vim* 与 *vi* 是兼容的，将会添加那个新行。在 *vim* 中可以通过设置 `binary` 选项来避免发生这种情况。*elvis* 和 *vile* 从不追加额外的新行。

无限撤消

UNIX 的 *vi* 只允许撤消最后一次的修改，或把当前行恢复到对它进行任何修改之前的状态。所有的克隆版本都提供了“无限撤消”的功能，可以一直对修改进行撤消，直到撤消到该文件在任何修改之前的状态。

增量搜索

当使用增量搜索时，在你输入搜索模式的时候，编辑器就会进行匹配并在整个文件中移动光标。最后当输入 `RETURN` 键时，搜索结束（注 8）。如果以前从没有看过它，那么最初它是相当令人不安的，但是一段时间后就会习惯它了。

elvis 不支持增量搜索，*nvi* 和 *vim* 使用选项启动增量搜索，而 *vile* 使用两个特殊的 *vi* 模式命令。虽然 *vile* 可以在编译时就禁止增量搜索，但是默认情况下增量搜索是打开的。表 8-6 列出了每个编辑器提供的选项。

注 8： *emacs* 一直有增量搜索的功能。

表 8-6 增量搜索

编辑器	选项	命令	行为
<i>nvi</i>	searchincr		光标在输入的文件中移动，总是定位到所匹配文本的第一个字符上
<i>vim</i>	incserach		光标在输入的文件中移动， <i>vim</i> 高亮显示匹配前面输入的模式的文本
<i>vile</i>		$\wedge X \ S$, $\wedge X \ R$	光标在输入的文件中移动，总是定位到所匹配文本的第一个字符上。 $\wedge X \ S$ 增加了对文件进行向前搜索的功能，而 $\wedge X \ R$ 增加了向后搜索的功能

左右滚动

默认情况下，*vi* 和大部分的克隆版本都会使较长的行在屏幕边界处换行。因此，文件的单个逻辑行可能会占用屏幕上多个物理行的空间。

许多时候，如果把长行简单地隐藏在屏幕右边缘的后面可能会比对其进行换行要好。把光标移动到该行上，在向右移动行的同时屏幕也会跟着“滚动”，该功能在所有的克隆版本中都是可用的。通常，数字选项控制着滚动屏幕的尺寸，布尔选项控制着是换行还是隐藏在屏幕边缘的后面。*vile* 还有把整个屏幕向旁边移动的命令键。表 8-7 列出了每个编辑器中使用水平滚动的方法。

表 8-7 旁边滚动

编辑器	滚动数量	选项	行为
<i>nvi</i>	sidescroll = 16	leftright	默认情况下为关闭。当启动时，长行只是隐藏在屏幕边缘的后面。每次屏幕向左或向右滚动 16 个字符
<i>elvis</i>	sidescroll = 8	wrap	默认情况下为关闭。当启动时，长行只是隐藏在屏幕边缘的后面。每次屏幕向左或向右滚动 8 个字符

表 8-7 旁边滚动（续）

编辑器	滚动数量	选项	行为
<i>vim</i>	sidescroll = 0	wrap	默认情况下为关闭。当启动时，长行只是隐藏在屏幕边缘的后面。当 sidescroll 设置为 0 时，每次滚动会把光标放置在屏幕的中间；否则，屏幕将按预期数目的字符滚动
<i>vile</i>	sideways = 0	linewrap	默认情况下为关闭。当启动时，长行将换行。因此，默认设置为把长行隐藏在屏幕边缘的后面。长行的左右边界使用 < 和 > 进行标记。当 sideways 设置为 0 时，每个滚动会移动屏幕的三分之一；否则，屏幕将按预期数目的字符滚动
		horizscroll	默认情况下为启动。当启动时，光标沿着长行移动到屏幕的边缘时会替换整个屏幕。当没有启动时，只是当前行被替换；这在速度较慢的显示器上比较有用

vile 有两个额外的命令：`^X ^R` 和 `^X ^L`。这两个命令分别向右和向左滚动屏幕，而不改变光标在该行的当前位置。你不可能移动到光标位置而离开屏幕。

可视模式

通常，*vi* 中的操作作用于像行、单词、字符这样的文本单元或者作用于从当前光标位置到搜索命令所指定的位置之间的文本段。例如，`d / ^}` 会删除到下面以右大括号开始的行。*elvis*、*vim* 和 *vile* 都提供了明确选择操作要作用的文本区域的机制，尤其是使选择一个矩形文本块然后把操作应用到矩形内的所有文本成为可能。参考每个编辑器的对应章节来了解各种细节内容。

模式标识

vi 有两种模式：命令模式和插入模式。通常，不能通过查看屏幕来识别自己处于

哪种模式。而且，不使用 ^G 或 ex 的 := 命令就知道在文件中的位置通常也是很有用的。

`showmode` 和 `ruler` 这两个选项解决了这些问题。这四种克隆版本的选项名字和含义都一样，甚至 Solaris 的 `vi` 也有 `showmode` 选项。

表 8-8 列出了每个编辑器中的特殊功能。

表 8-8 位置和模式标识符

编辑器	使用 <code>ruler</code> 显示	使用 <code>showmode</code> 显示
<code>nvi</code>	行和列	插入、修改、替换和命令模式标识符
<code>elvis</code>	行和列	输入和命令模式标识符
<code>vim</code>	行和列	插入、替换和可视模式标识符
<code>vile</code>	行、列和文件百分比	插入、替换和覆盖模式标识符
<code>vi</code>	N/A	分别表示打开、输入、插入、追加、修改、替换、替换一个字符和替换模式的分隔模式标识符

`elvis` 和 `vim` 的 GUI 版本会根据当前模式改变鼠标形状。

编程辅助

`vi` 主要是作为程序员的编辑器而开发，它有许多功能方便那些编写 C 程序和 `troff` 文档的 UNIX 程序员的工作。（真正的程序员使用 `troff` 编写真正的文档。）一些克隆版本继承了这个传统，并且添加了许多功能，这使得它们对于那些“高级用户”更好用、更有效（注 9）。

其中有两个功能最值得讨论：

注 9：与所见即所得 (WYSIWYG) 的观点相比较，UNIX 是所要即所得的操作系统。（为此要感谢 Scott Lee。）

编辑编译加速

elvis、*vim* 和 *vile* 允许你很容易地调用 *make(1)*，从编译器中获取错误信息和自动移动到含有错误的行，然后可以修改错误并重新运行 *make*，所有这些都可以在编辑器内进行。

语法高亮显示

elvis、*vim* 和 *vile* 具有在不同类型的文件中高亮显示和/或改变不同语法单元颜色的功能。

编辑编译加速

编程通常由“编译 – 测试 – 调试”过程循环组成。修改源代码，编译新代码，然后对它进行测试和调试。在学习新语言时，语法错误尤为常见，因此在编译期间经常停止和重新启动（或挂起和再继续）编辑器是很麻烦的。

elvis、*vim* 和 *vile* 都提供了允许你在编译程序时不离开编辑器的功能。而且，它们还能获取编译器的输出，并使用它自动到达包含错误的每一行（注 10）。始终如一地使用这种功能可以节省时间和提高编程效率。

下面是使用 *elvis* 的一个例子。由于我们刚开始学习 C++，因此从约定性的第一个程序开始：

击键 结果

:w hello.c

```
#include <iostream.h>

void main()
{
    cout << "hello, world!\n";
}
```

输入程序时忘记了右引号，然后将其写到 *hello.c* 文件中。

注 10： 从 *emacs* 用户习惯的功能中继承来的另一个。

```
:make hello
```

```
hello.C:5:unterminated string or character
constant
hello.C:5:possible real start of unterminated
constant
make: *** [hello] Error 1
Hit <Enter> to continue _
```

输入执行 *make* 的 :*make* 命令，进而运行了 C++ 编译器（在该例子中是 g++）。

```
:errlist
```

```
#include <stream.h>

void main()
{
    cout << "hello, world!\n";
}
line 5: unterminated string or character
constant
```

:*errlist* 命令把光标移动到含有错误的行并在状态行上显示第一个编译错误信息。

你可以修复错误，重新保存文件，重新运行 :*make*，最终无错误地编译程序。

所有的编辑器都有相似的功能，它们都将补偿文件中的修改，准确地移动到下一个含有错误的行。更多的细节将在每个编辑器的章节提供。

语法高亮显示

elvis、*vim* 和 *vile* 都提供了某种方式的语法高亮显示，它们还提供了语法着色，在具有那种功能的显示器（如在 X11 或 Linux 控制台下）上改变文件中不同部分的颜色。参考每个编辑器的章节以获得更多的信息。

编辑器比较小结

大部分编辑器支持上面描述的绝大部分或所有功能。表 8-9 对每个编辑器所支持的功能进行了总结。当然，该表并不能提供全部的内容，各个细节将在每个编辑器的单独章节中提供。

表 8-9 功能总结表

功能	nvi	elvis	vim	vile
多窗口编辑	✓	✓	✓	✓
GUI		✓	✓	✓
扩展的正则表达式	✓	✓	✓	✓
增强的标志		✓	✓	✓
标志栈	✓	✓	✓	✓
任意长度的行	✓	✓	✓	✓
8 位数据	✓	✓	✓	✓
无限撤消	✓	✓	✓	✓
增量搜索	✓		✓	✓
左右滚动	✓	✓	✓	✓
模式标识符	✓	✓	✓	✓
可视模式		✓	✓	✓
编辑编译加速		✓	✓	✓
语法高亮显示		✓	✓	✓
多 OS 支持	✓	✓	✓	✓

后面内容预览

后面四章依次介绍 *nvi*、*elvis*、*vim* 和 *vile*，每章都有以下要点：

1. 编辑器的作者和产生原因。
2. 重要的命令行参数。
3. 联机帮助和其他文档。
4. 初始化—程序要读取的文件和环境变量以及读取的次序。
5. 多窗口编辑。
6. GUI 接口。

7. 扩展的正则表达式。
8. 改进的编辑功能（标志栈、无限撤消等）。
9. 编程辅助（编辑编译加速、语法高亮显示）。
10. 该编辑器令人感兴趣的特有功能。
11. 获取源代码的地方以及支持该编辑器的操作系统。

所有这些版本都用 *gzip* (GNU zip) 进行了压缩，如果你没有 *gzip*，可以从 <ftp://ftp.gnu.org/pub/gnu/gzip-1.2.4.tar> 上得到它。对解压缩非UNIX系统上 *gzip* 生成的 *tar* 文件，*elvis* *ftp* 站点上现有的 *untar.c* 是一个非常简单的程序。

由于这些编辑器中的每个都在继续发展，因此我们不可能对每个编辑器的功能进行彻底地介绍，有些功能都可能会马上过时。相反，我们只是选择了最重要的部分，并覆盖了那些最可能需要了解而又随着编辑器的发展最不可能改变的功能。如果需要知道如何使用编辑器的每个最新功能，那么应该把每个编辑器的联机文档作为本书的补充说明。

本章内容：

- 作者和历史
- 重要的命令行参数
- 联机帮助和其他的文档
- 初始化
- 多窗口编辑
- GUI 接口
- 扩展的正则表达式
- 改进的编辑功能
- 编程辅助
- 令人感兴趣的功能
- 源代码和支持的操作系统

第九章

nvi — 新 vi

nvi 是 “new vi” 的缩写，它最初诞生于伯克利加州大学 (UCB)、即著名的UNIX BSD 版本的发源地。本章将对其进行介绍。

作者和历史

最初的 *nvi* 是由 Bill Joy 于 20 世纪 70 年代后期在 UCB 开发的，那时他是一位计算机科学专业的研究生，现为 Sun Microsystem 公司的奠基人和副总裁。

Bill Joy 首先创建了 *ex*，他从 *ed* 编辑器的第六版开始并对其进行极大的完善。首次增强是开放模式，这是与 Chuck Haley 一起完成的。在 1976 到 1979 年期间，*ex* 发展为 *vi*。然后 Mark Horton 来到伯克利，对 *vi* 进行了许多工作，为它添加了宏和“其他功能”（注 1），从而使它可以在大部分终端和 UNIX 系统上运行。截止到 BSD 4.1 版（1981 年），该编辑器基本上具有了本书第一部分描述的所有功能。

注 1：出自 *nvi* 的参考手册。但并没有说明是哪些功能。

不管所有的变化如何，*vi* 的内核还是最初的 UNIX *ed* 编辑器。因此它不是可免费获得的代码。直到 20 世纪 90 年代早期，当 *vi* 在 BSD 4.4 版上运行时，BSD 的开发者们还是希望能获得以源代码形式免费发布的 *vi* 版本。

UCB 的 Keith Bostic 从 *elvis* 1.8（注 2）（这是个免费获得的 *vi* 克隆版本）着手，开始把它转变为“错误兼容的”*vi* 克隆版本。*nvi* 也遵循 POSIX 命令语言和公用标准（IEEE P1003.2），这是很有意义的。

虽然 Keith Bostic 不再属于 UCB，但是他仍然继续维护、改进和发布 *nvi*。本书出版时的 *nvi* 版本为 1.79。

由于 *nvi* 是 *vi* 的“官方”伯克利版本，因此它是很重要的。它是 4.4BSD-Lite II 的组成部分，是用于 NetBSD 和 FreeBSD 那样各种流行的 BSD 变体上的 *vi* 版本。

重要的命令行参数

在纯 BSD 环境中，*nvi* 可用 *ex*、*vi* 和 *view* 名字进行安装。通常它们都是到同一可执行文件的链接，并且 *nvi* 根据调用它的方式来决定相应的行为（UNIX 的 *vi* 也以这种方式工作）。*nvi* 允许在 *vi* 模式中使用 Q 命令切换到 *ex* 模式。除了初始化时设置 *readonly* 选项外，*view* 变体与 *vi* 相同。

nvi 有许多命令行参数，下面对其中最有用的一些参数进行介绍：

-c command

一启动就执行 *command*。虽然这是过去 *+command* 语法的 POSIX 版，但是 *nvi* 并没有限制于定位命令（原有的语法也可以被接受）。

-F 在开始编辑时不对整个文件进行复制。虽然这样做，运行速度可能会比较快，但是也为其他人在你编辑文件时修改该文件提供了可能性。

-R 以只读方式开始，并设置 *readonly* 选项。

注 2： 虽然最初只有很少或根本没有 *elvis* 代码留下来。

-r 恢复指定的文件，或者如果没有在命令行上列出文件，那么就列出所有可以恢复的文件。

-S 运行时设置 `secure` 选项，不允许访问外部程序（注 3）。

-s 进入批处理（脚本）模式。这适用于 `ex`，主要目的是运行编辑脚本。提示符和非错误信息都是无用的，这是过去“-”参数的 POSIX 版本，这两种版本 `nvi` 都支持。

-t tag

在指定的 `tag` 处开始编辑。

-w size

设置初始窗口的大小来排列行。

联机帮助和其他的文档

`nvi` 带有非常容易理解的可打印文档，尤其是它带有下列文档的 `troff` 源码、格式化的 ASCII 和格式化的 PostScript：

vi 参考手册

适用于 `nvi` 的参考手册，该手册描述了所有的 `nvi` 命令行选项、命令、选项和 `ex` 命令。

vi 手册页

适用于 `nvi` 的帮助页面。

vi 指南

该文档是对使用 `vi` 进行编辑的指导性介绍。

ex 参考手册

适用于 `ex` 的参考手册。该手册是 `ex` 最初的手册，它对于 `nvi` 中的功能有点过时。

注 3：与标有“安全”的任何内容一样，盲目信任它们通常是不恰当的。然而，Keith Bostic 却认为可以信任 `nvi` 的 `secure` 选项。

还包括记录 *nvi* 的某些内部构件并提供应该实现的功能列表的 ASCII 文件，以及可用于 *vi* 联机指南的文件。

真正嵌入 *nvi* 内部的联机帮助是极少的，主要由两个命令组成：`:exusage` 和 `:viusage`。这些命令为每个 *ex* 和 *vi* 命令提供了一行摘要。虽然这对于提醒用户它们如何工作是足够的，但是对于了解 *nvi* 的新的功能并不是非常好。

你可以把某个命令作为 `:exusage` 和 `:viusage` 命令的参数，这样 *nvi* 将只显示该命令的帮助信息。*nvi* 一共显示出两行内容：一行解释该命令功能，一行简要说明该命令的用法。

初始化

如果指定了 `-s` 或 “-” 选项，那么 *nvi* 将忽略所有的初始化。否则，*nvi* 将执行下面的步骤：

1. 读取并执行文件 “`/etc/vi.exrc`”，它必须属于 `root` 或用户。
2. 如果存在 `NEXINIT` 环境变量，那么执行它的值；否则，如果存在 `EXINIT` 就使用它。两者之中将使用一个，并忽略执行 `$HOME/.nexrc` 或 `$HOME/.exrc`。
3. 如果存在 `$HOME/.nexrc`，就会读取它并执行；否则，如果存在 `$HOME/.exrc`，就读取它并执行。只能使用其中的一个。
4. 如果设置了 `exrc` 选项，若存在 `./.nexrc` 或 `./.exrc` 就寻找它们并执行。只能使用其中的一个。

nvi 不执行除文件所有者之外任何人编写的任何文件。

nvi 文档建议用户把常用的初始化行为放到 `.exrc` 文件中（即 UNIX *vi* 的选项和命令），并在 *nvi* 专用初始化之前或之后让 `.nexrc` 文件执行 `:source .exrc`。

多窗口编辑

如果要在 *nvi* 中创建新窗口，可以使用 *ex* 编辑命令：Edit、Fg、Next、Previous、Tag 或 Visual 中的一个（这些命令都可以使用缩写形式）。如果光标在屏幕的上半部分，新窗口就会创建在屏幕的下半部分，反之亦然。然后可以使用 **CTRL-W** 切换到另一个窗口。

```
<preface id="VI6-CH-0">
<title>Preface </title>

<para>
Text editing is one of the most common uses of any computer system, and
<command>vi</command> is one of the most useful standard text editors on
your system.
With <command>vi</command> you can create new files, or edit any existing
UNIX text file.
</para>

ch00.sgm: unmodified: line 1
# Makefile for vi book
#
# Arnold Robbins

CHAPTERS = ch00_6.sgm ch00_5.sgm ch00.sgm ch01.sgm ch02.sgm ch03.sgm \
           ch04.sgm ch05.sgm ch06.sgm ch07.sgm ch08.sgm
APPENDICES = appa.sgm appb.sgm appc.sgm appd.sgm

POSTSCRIPT = ch00_6.ps ch00_5.ps ch00.ps ch01.ps ch02.ps ch03.ps \
             ch04.ps ch05.ps ch06.ps ch07.ps ch08.ps \
Makefile: unmodified: line 1
```

该例子展示了 *nvi* 对两个文件 *ch00.sgm* 和 *Makefile* 的编辑。分割屏幕是输入 *nvi ch00.sgm* 后输入 :Edit *Makefile* 的结果。每个窗口的最后一行起着状态行的作用，同时也是该窗口冒号命令执行的地方。状态行以相反的颜色进行高亮显示。

表 9-1 对窗口化的 *ex* 模式命令及其它们的功能进行了描述。

表 9-1 nvi 窗口管理命令

命令	功能
bg	隐藏当前窗口，该窗口可以使用 fg 和 Fg 命令进行回调
di [splay] b[uffers]	显示所有的缓冲区，包括命名的、未命名的和数字缓冲区
di [splay] s[creens]	显示所有不可见窗口中的文件名
Edit <i>filename</i>	在新窗口中编辑 <i>filename</i> 文件
Edit /tmp	创建编辑空缓冲区的新窗口，/tmp 被专门解释为创建新的临时文件
fg <i>filename</i>	把 <i>filename</i> 文件显示到当前窗口，先前的文件移动到后面
Fg <i>filename</i>	在新窗口中显示 <i>filename</i> ，分割当前窗口，而不是在所有打开的窗口中重新分配屏幕空间
Next	在新窗口中编辑参数列表中的下一个文件
Previous	在新窗口中编辑参数列表中的上一个文件（对应的移回上一文件的 previous 命令，存在于 nvi 中；它不在 UNIX vi 中）。
resize <i>±nrows</i>	按 <i>nrows</i> 的行数增加或减少当前窗口的大小
Tag <i>tagstring</i>	在新窗口中编辑含有 <i>tagstring</i> 的文件

【CTRL-W】命令从上到下在窗口之间循环，:q 和 ZZ 命令退出当前窗口。

你可以在多个窗口中打开同一个文件。虽然在一个窗口中所做的修改将会反映到其他窗口中，但是在 nvi 的插入模式下所做的修改，只有在通过输入【ESC】结束本次修改后才会显示在其他窗口中。只有当命令 nvi 离开打开文件的最后一个窗口，才会被提示保存修改。

GUI 接口

nvi 没有提供图形用户接口（GUI）版本。

扩展的正则表达式

扩展的正则表达式已在第八章中的“扩展的正则表达式”一节进行了介绍，这里我们只对 *nvi* 提供的元字符进行总结。*nvi* 也支持 POSIX 方括号表达式 ([[[:alnum:]]]) 等等。

你可以使用 :set extended 启动扩展的正则表达式匹配。

| 表示或。左边和右边不必是单个字符。

(...)

用于分组，允许使用额外的正则表达式操作符。

当设置 extended 时，使用圆括号分组后的文本产生的效果与在常规 *vi* 中 \(...\)\ 分组后的文本相似，实际匹配的文本可以使用 \1、\2 等在替换命令的替换部分进行检索。在这种情况下，\ (代表字面的左圆括号。

+ 匹配一个或多个前面的正则表达式。或者是单个字符，或者是圆括号中的一组字符。

? 匹配前面正则表达式的零次或一次出现。

(...)

定义区间表达式。区间表达式描述了重复次数的计数数字，在下面的描述中，*n* 和 *m* 代表整型常数。

{*n*}

对前面正则表达式的 *n* 次重复进行匹配。

{*n*,}

对前面正则表达式的 *n* 次或更多次的重复进行匹配。

{*n*,*m*}

对 *n* 到 *m* 次的重复进行匹配。

当没有设置 extended 时，*nvi* 使用 \{ 和 \} 来提供同样的功能。

当设置 `extended` 时，应该在上面元字符的前面添加反斜杠来匹配它们实际表示的文本。

改进的编辑功能

本节介绍了 *nvi* 使编辑简单文本更容易、更有效的功能。

命令行历史和完整化

nvi 对 *ex* 命令行进行保存，因此可以编辑它们以进行再次提交。

该功能由 `cedit` 选项控制。

当在冒号命令行上输入该字符串的第一个字符时，*nvi* 就会在新窗口中显示可以编辑的历史命令。当在任何指定的行上输入 `RETURN` 键时，*nvi* 就会执行该命令。对于该选项，使用 `ESC` 键是个好的选择（使用 `^V ^[` 输入该命令）。

由于 `RETURN` 键会真正地执行命令，因此在使用 `j` 或 `k` 键从一行移动到下一行时要小心。

除了能编辑命令行以外，还可以对文件名进行扩展。该功能由 `filec` 选项控制。

当在冒号命令行上输入该字符串的第一个字符时，*nvi* 会把光标前面被空白分割的单词看成尾部追加了一个 *，并进行 shell 风格的文件名扩展。同样，`ESC` 键对于该选项也是个好的选择（注 4）（使用 `^V ^[` 输入该命令）。当该字符与 `cedit` 选项相同时，只有将其作为冒号命令行上的第一个字符输入，才会执行命令行编辑。

可以非常容易地在 `.nexrc` 文件中设置这些选项：

注 4： 虽然 *nvi* 文档指出 `TAB` 键是另一个常用选择，但是我们不能使它正常工作。实际上，对于这两个选项使用 `ESC` 键都能很好地进行功能操作。

```
set cedit=^[
set filec=^[
```

标志栈

标志入栈已在第八章中的“标志栈”一节进行了介绍。*nvi*的标志栈在这四种克隆版本中是最简单的。表 9-2 和表 9-3 列出了它所使用的命令。

表 9-2 nvi 的标志命令

命令	功能
di [splay] t[ags]	显示标志栈
ta[g](!) tagstring	编辑在tags文件中定义的包含tagstring的文件。如果当前缓冲区已修改而且没有保存,!会强迫nvi切换到新文件
Ta[g](!) tagstring	除了在新窗口中编辑文件外,与:tag一样
tagp[op](!) tagloc	弹出给定的标志,或者如果没有提供tagloc,就弹出最近使用的标志。该位置可能是关心的标志文件名或指示栈中位置的数字
tagt[op](!)	弹出栈中最早的标志,清除该过程中的标志栈

表 9-3 nvi 命令模式下的标志命令

命令	功能
^]	在tags文件中寻找光标处标识符的位置并移动到该位置。当前位置将自动压入标志栈
^T	返回到标志栈中的上一个位置,即弹出一个单元

你可以把tags选项设置为*nvi*在哪寻找标志的文件名列表,这样便提供了一种简化的搜索路径机制。默认值为“tags /var/db/libc.tags /sys/kern/tags”,这在BSD 4.4版本的系统上将首先查找当前目录,随后查找C程序库和操作系统源代码文件使用的标记文件。

taglength选项控制着标志串中有意义字符的数目。默认值0表示使用所有的字符。

nvi 的行为与 *vi* 相似，它使用从光标位置开始的光标后面的“单词”，如果光标位于 *main* 中的 *i* 上，那么 *nvi* 将搜索标识符 *in*，而不是 *main*。

nvi 遵循传统的 *tag* 文件格式。但该格式非常有限，尤其是它没有编程语言作用域的概念，这个概念允许在不同上下文中使用相同的标识符来表示不同的含义。该问题在 C++ 中更为严重，因为 C++ 明确允许函数名重载，即不同的函数可以使用相同的名字。

nvi 通过使用一种完全不同的机制：*cscope* 程序来限制 *tag* 文件，*cscope* 是专有的但是相对于贝尔实验室软件工具箱中的程序来说又是比较廉价的软件。*cscope* 读取 C 源文件并创建描述该程序的数据库。*nvi* 提供了查询该数据库和处理查询结果的命令。由于 *cscope* 不是通用的，因此我们不在这里讨论它的使用。*nvi* 文档提供了 *nvi* 命令的所有细节。

虽然 Exuberant *ctags* 生成的扩展 *tag* 文件格式并不在 *nvi* 1.79 中产生任何错误，但 *nvi* 并没有使用这种格式。

无限撤消

在 *vi* 中，点（.）命令实现“再做一次”的功能；它重复你最后执行的编辑操作，可能是删除、插入或替换。

nvi 把点命令推广为完全的“撤消”命令，即使最后一次命令是表示“撤消”的 u，也可以使用它进行撤消。

因此，如果要进行一系列的“撤消”命令，可以首先输入 u，然后每输入一个 .，*nvi* 就会继续撤消修改，逐步使文件接近它的最初状态。

最后将到达文件的初始状态。在这时输入 . 将会产生响铃（或屏幕闪烁）现象。现在可以通过输入 u 来开始“撤消上面的撤消操作”，然后使用 . 重新应用接下来的修改。

nvi 不允许为 u 或 . 命令指定执行的次数。

任意长度的行和二进制数据

nvi 可以编辑含有任意长度的行和任意行数的文件。

nvi 自动处理二进制数据，不需要任何特殊的命令行选项或 *ex* 选项。可以用 ^X 后面跟一两个十六进制数字来把任何 8 位字符输入到文件中。

增量搜索

正如在第八章中的“增量搜索”一节提到的，可以使用 :set searchincr 启动 *nvi* 中的增量搜索。

光标在输入的文件中移动，并且总是定位到所匹配文本的第一个字符上。

左右滚动

正如在第八章中的“左右滚动”一节提到的，可以使用 :set leftright 启动 *nvi* 中的左右滚动。在从左向右滚动时，sidescroll 的值控制着 *nvi* 移动屏幕的字符数目。

编程辅助

nvi 没有提供特殊的编程辅助功能。

令人感兴趣的功能

虽然 *nvi* 是最小的克隆版本，没有大量追加功能，但是它还是有一些值得一提的重要功能。

国际化支持

nvi 中的大部分信息和警告消息都可以通过使用称为“消息目录”的工具转

化为不同的语言。*nvi* 使用 *nvi* 发布的“*catalog/README*”文件所描述的简明机制对该工具进行了改进。它提供的信息种类有Dutch、English、French、German、Russian、Spanish 和 Swedish。

任意的缓冲区名字

过去，*vi* 缓冲区的名字被限制为字母表中的 26 个字符。而 *nvi* 允许使用任何字符作为缓冲区的名字。

对 /tmp 的特殊解释

对于任何需要文件名参数的 *ex* 命令，如果使用特殊名字 /tmp，那么 *nvi* 将使用惟一的临时文件名替换它。

源代码和支持的操作系统

可以从 <http://www.bostic.com/vi> 上获得 *nvi*，可以在其中下载 *nvi* 的当前版本，或把通知用户 *nvi* 新版本和 / 或新功能的邮箱添加到邮递列表中。

nvi 的源代码是免费发布的，许可期限描述在发布的“*LICENSE*”文件里，它们允许以源代码和二进制格式发布。

nvi 构建并运行在 UNIX 下，它也可以构建在 LynxOS 2.4.0 和更新版本下运行。虽然它还可能构建和运行在其他 POSIX 兼容系统上，但是文档中并没有特别列出已知的操作系统。

nvi 的编译是简单的，可以通过 *ftp* 获得该程序，对它进行解压缩后运行 *configure* 程序，然后再运行 *make* 即可。

```
$ gzip -d < nvi.tar.gz | tar -xvpf -
...
$ cd nvi-1.79; ./configure
...
$ make
...
```

配置和建立 *nvi* 应该不会遇到什么问题。使用 *make install* 安装 *nvi*。

如果需要报告 *nvi* 中的漏洞或问题，可以与 Keith Bostic 联系，邮箱是 *bostic@bostic.com*。



第十章

elvis

本章内容：

- 作者和历史
- 重要的命令行参数
- 联机帮助和其他的文档
- 初始化
- 多窗口编辑
- GUI 接口
- 扩展的正则表达式
- 改进的编辑功能
- 编程辅助
- 令人感兴趣的功能
- elvis 的未来
- 源代码和支持的操作系统

elvis 是由 Steve Kirkendall 编写和维护的，其早期版本是 *nvi* 的基础。本章就是使用 *elvis* 编辑的。

作者和历史

在我们感激 Steve Kirkendall 的帮助的同时，我们也用他自己的话来讲述这段历史：

我是在使用名为 *stevie* 的早期克隆版本崩溃后开始编写 *elvis* 1.0 的，这次崩溃使我丢失了几小时的工作成果，并彻底地击垮了我对该程序的信心。*stevie* 将编辑缓冲区放在 RAM 中，这在 Minix 系统中并不实际。所以，我就开始编写自己的克隆版本，它将自己的编辑缓冲区放到一个文件中，这样即使我的编辑器崩溃了，也可以在那个文件中找到所编辑的文本。

elvis 2.x 版与 1.x 版几乎完全不同。我之所以编写这个新版本，是因为我的第一个版本从现有的 *vi* 和 Minix 上继承了太多的缺陷。其中最大的变化是

新版本支持多编辑缓冲区和多窗口，而这两个功能没有一个能够很容易地从版本 1.x 中加以改进而得到，我同时想去掉对行长度的限制，并有以 HTML 格式编写的联机帮助。

关于“elvis”这个名字，Steve 说他选择它的部分原因是由于想看看有多少人会问他为什么选择这个名字（注 1）！名字中包括字母“vi”是大多 vi 克隆版本的共同特征。

重要的命令行参数

elvis 并不安装成 *vi*，当然也可以那样执行。如果将 *elvis* 作为 *ex* 调用，那么就可以将其作为一个行编辑器运行，它允许使用 Q 命令从 *vi* 模式切换到 *ex* 模式。

elvis 有很多命令行选项，最有用的如下所示：

- a 把命令行中指定的每个文件加载到一个单独的窗口。
- r 在系统崩溃后进行恢复。
- R 以只读方式开始编辑每个文件。
- i 在输入模式而不是命令模式下进行编辑，这种方式对于初学者比较容易。
- s 设置整个会话的 *safer* 选项，而不仅仅是执行 *.exrc* 文件。虽然这增加了一定的安全性，但是也不能盲目相信它。在 *elvis* 2.1 中，这个选项被改名为 -S，（依据 POSIX 标准）-s 用来提供 *ex* 编辑处理。

-f *filename*

使用 *filename* 来寻找会话文件，不使用默认文件名。会话文件将在下面讲述。

-G *gui*

使用给定的接口，默认是 *termcap* 接口。其他的选项包括 *x11*、*win32*、*curses*、*open* 和 *quit* 等。不是所有的接口都可以编译到 *elvis* 版本中。

注 1： 在最近的 8 年中，我只接到 4 次询问！ A.R.

-c *command*

在启动时运行 *command*。这是过去的 +*command* 语法的 POSIX 版（原有语法也可以使用）。

-t *tag*

在指定的 *tag* 处开始编辑。

-v 输出更详细的状态信息，用来诊断初始化文件的问题。

-? 显示可以使用的选项的摘要。

联机帮助和其他的文档

elvis 对这一点非常重视。联机帮助很全面，并且全部用 HTML 格式编写。这使得读者可以使用自己喜欢的 Web 浏览器阅读。*elvis* 还有一个 HTML 显示方式（后面将会介绍），这样在 *elvis* 中就可以很方便地阅读在线帮助。

当阅读 HTML 文件时，可以使用标志命令 (^] 和 ^T) 到达不同的主题然后再返回，这使得浏览帮助文件非常容易。

当然，*elvis* 也出现在 UNIX *man page* 命令中。

初始化

这部分介绍 *elvis* 的会话文件并详细说明了它在初始化中所采取的步骤。

会话文件

elvis 计划最终符合 COSE（通用开放系统环境）标准，这要求程序保存它们的状态，并在将来能够返回到已保存的状态。

为了实现这项功能，*elvis* 将全部状态都保存到会话文件中。通常 *elvis* 会在启动时

创建该会话文件，在退出时将其删去。但是如果 *elvis* 系统崩溃了，留下的会话文件可以用来恢复所编辑的文件。

初始化步骤

elvis 按以下步骤进行初始化。值得注意的是，用于自定义 *elvis* 的许多选项都从编辑器选项转移到初始化文件中。

1. 初始化所有的硬编码选项。
2. 在已编译的接口中选择一个到 *elvis* 中，*elvis* 会在那些已编译并能工作的接口中选择一个“最好的”。例如，虽然认为 X11 接口比 *termcap* 接口好，但如果当前 X Windows 不能运行，就有可能不能使用 X11。
选定的接口能够为特殊的初始化选项处理命令行。
3. 如果会话文件不存在，则创建会话文件，否则，读取它（为恢复准备）。
4. 使用 ELVISPAT H 环境变量初始化 *elvispath* 选项，否则赋给它默认值。虽然 “`~/.elvislib:/usr/local/lib/elvis`” 是常用值，但是实际值将取决于配置和创建 *elvis* 的方式。
5. 在 ELVISPAT H 路径下查找名为 *elvis.ini* 的 *ex* 脚本并执行它。*elvis.ini* 文件默认完成如下操作：
 - 根据当前的操作系统选择一个两拼一表。两拼一是用来定义系统扩展的 ASCII 字符集和如何输入扩展的 ASCII 字符的一种方法。
 - 根据编辑器的名字（比如 *ex* 或 *vi* 模式）来设置选项。
 - 处理与系统相关的特别部分，比如为 X11 设置颜色和向界面中增加菜单。
 - 选择一个初始化文件名，或者是 UNIX 下的 *.exrc*，或者是非 UNIX 环境下的 *elvis.rc*。并把该文件称为 *f*。
 - 如果存在 EXINIT 环境变量，运行它的值；否则，运行 `:source ~/f`。这里 *f* 是上一步选定的文件名。
 - 如果已经设定 *exrc* 选项，那么在当前目录下对 *f* 运行命令 `:safer`。

- 对于X11，如果还没有设定普通、粗体和斜体等字体格式，就设定它们。
6. 如果读之前、读之后和写之前、写之后命令文件存在，就加载它们，同时也加载 *elvis.msg* 文件。所有这些文件都会在本章的后面进行介绍。
 7. 加载并显示命令行中指定的第一个文件。
 8. 如果给出了 -a 选项，加载并显示剩下的文件，每个文件都在它自己的窗口中。

多窗口编辑

可以使用 *ex :split* 命令在 *elvis* 中创建一个新的窗口，然后可以使用常规的 *ex* 命令，比如 *:e filename* 或 *:n* 来编辑一个新文件。这是最简单的方法。其他更简洁的方法将在后面介绍。你可以使用 **[CTRL-W]**、**[CTRL-W]** 在窗口间来回切换。

```
<preface id="VI6-CH-0">
<title>Preface</title>

<para>
Text editing is one of the most common uses of any computer system, and
<command>vi</command> is one of the most useful standard text editors on
your system.
With <command>vi</command> you can create new files, or edit any
existing UNIX text file.

# Makefile for vi book
#
# Arnold Robbins

CHAPTERS=ch00_6.sgm ch00_5.sgm ch00.sgm ch01.sgm ch02.sgm ch03.sgm \
         ch04.sgm ch05.sgm ch06.sgm ch07.sgm ch08.sgm
APPENDICES=appa.sgm appb.sgm appc.sgm appd.sgm

POSTSCRIPT=ch00_6.ps ch00_5.ps ch00.ps ch01.ps ch02.ps ch03.ps \
           ch04.ps ch05.ps ch06.ps ch07.ps ch08.ps \
           appa.ps appb.ps appc.ps appd.ps
```

分割出来的窗口是键入 *elvis ch00.sgm* 命令后输入 *:split Makefile* 的结果。

和 *nvi* 一样，*elvis* 给出每个窗口的状态行。*elvis* 的独特之处在于它使用一个高亮显示的下划线而不是相反的颜色来表示状态行。*ex* 冒号命令在各个窗口状态行中运行。

表 10-1 描述了窗口化的 *ex* 模式命令及它们的功能。

表 10-1 elvis 窗口管理命令

命令	功能
sp[lit] [file]	创建新窗口，如果存在文件 <i>file</i> 就将其载入，否则新窗口显示当前文件
new	创建一个新的空缓冲区，然后创建一个新的窗口来显示该缓冲区
sne[w]	
sn[ext] [file...]	创建新窗口，显示参数列表中下一个文件。不影响当前的文件
sN[ext]	创建新窗口，显示参数列表中前一个文件。不影响当前的文件
sre[wind] [!]	创建新窗口，显示参数列表中第一个文件。相对于 :next 命令，将“当前”文件重新设定为第一个文件。不影响当前的文件
sl[ast]	创建新窗口，显示参数列表中最后一个文件。不影响当前的文件
sta[g] [!] tag	创建新窗口来显示发现 <i>tag</i> 的文件
sa[ll]	为文件创建新窗口，这些文件都在参数列表中提到，但都没有创建窗口
wi[dow] [target]	如果没有 <i>target</i> 选项，则列出所有的窗口。 <i>target</i> 的可选值在表 10-2 中讲述
close	关闭当前窗口，窗口正在显示的缓冲区保持为不激活。如果修改了缓冲区内容，其他 <i>elvis</i> 退出命令将阻止你退出，除非显式地保存或放弃缓冲区
wquit	将缓冲区写回文件，并关闭窗口。无论文件是否修改，它都将被保存
qall	对每个窗口发布 :q 命令。不影响没有窗口的缓冲区

表 10-2 描述了窗口化的 *ex* 参数和它们的意义。

表 10-2 elvis 窗口命令参数

参数	意义
+	切换到下一个窗口，类似 <code>^W K</code>
++	切换到下一个窗口，换行类似 <code>^W ^W</code>
-	切换到前一个窗口，类似 <code>^W J</code>
--	切换到前一个窗口，换行
<i>num</i>	切换到 windowid = <i>num</i> 的窗口
<i>buffer-name</i>	切换到正在编辑的命名缓冲区的窗口

elvis 提供很多在窗口之间切换的 *vi* 模式命令。在表 10-3 中对它们进行总结。

表 10-3 elvis 的 vi 命令模式下的窗口命令

命令	功能
<code>^W C</code>	隐藏缓冲区并关闭窗口。这和 :close 命令相同
<code>^W D</code>	在“通常模式”和缓冲区的常用显示模式之间切换显示模式。这是一个针对窗口的选项。显示模式在“显示模式”一节中有讲述
<code>^W J</code>	移动到下一个窗口
<code>^W K</code>	移动到上一个窗口
<code>^W N</code>	创建一个新窗口，并创建一个在该窗口显示的新缓冲区。它和 :snew 命令相似
<code>^W Q</code>	保存缓冲区并关闭窗口，和 <code>zz</code> 相同
<code>^W S</code>	分割当前的窗口，相当于 :split 命令
<code>^W S</code>	切换 <code>wrap</code> 选项。这个选项控制的是较长的行换行还是整个屏幕向右滚动。这是一个针对窗口的选项。这个选项在本章后面的“左右滚动”部分有讲述
<code>^W]</code>	创建一个新的窗口，接下来查找光标底下的标志。它和 :stag 命令相似
<code>[count] ^W ^W</code>	移动到下一个或到第 <i>count</i> 个窗口

表 10-3 elvis 的 vi 命令模式下的窗口命令（续）

命令	功能
<code>^W +</code>	增加当前窗口的大小（只对 <i>termcap</i> 接口有效）
<code>^W -</code>	减少当前窗口的大小（只对 <i>termcap</i> 接口有效）
<code>^W \</code>	使当前窗口尽可能地大（只对 <i>termcap</i> 接口有效）

GUI 接口

本节的屏幕插图和讲解是由 Steve Kirkendall 提供的。

elvis 的 X11 接口支持滚动条和鼠标，并允许选择所用字体。在 *elvis* 创建窗口后，就无法更改字体了。字体必须全部是等宽字体，通常是一个 Courier 或一个“固定”字体的变体。

elvis 2.0 的 X11 接口支持多种字体和颜色，以及闪烁的光标，光标通过改变形状来指示编辑模式（插入还是命令），接口还支持滚动条和鼠标操作。可以用鼠标来选择文件、在应用程序间进行剪切和粘贴并完成标志的查找。

elvis 2.1 增加了配置工具栏、对话窗口、一个状态栏和 -client 标志。它在单色的 X 终端上也能很好地工作。

由于 *elvis* 2.1 比 2.0 显著改进了 X11 的接口，并且因为它在本书出版的时候发布，所以本章剩余部分的内容是适用于 *elvis* 2.1 的。许多特点、很多命令行选项和经由 X 资源来配置 *elvis* 的功能都是 2.0 版本所没有实现的。

基本的窗口

基本的 *elvis* 窗口如图 10-1 所示。

elvis 提供一个单独的文本查找弹出式对话框，如图 10-2 所示。

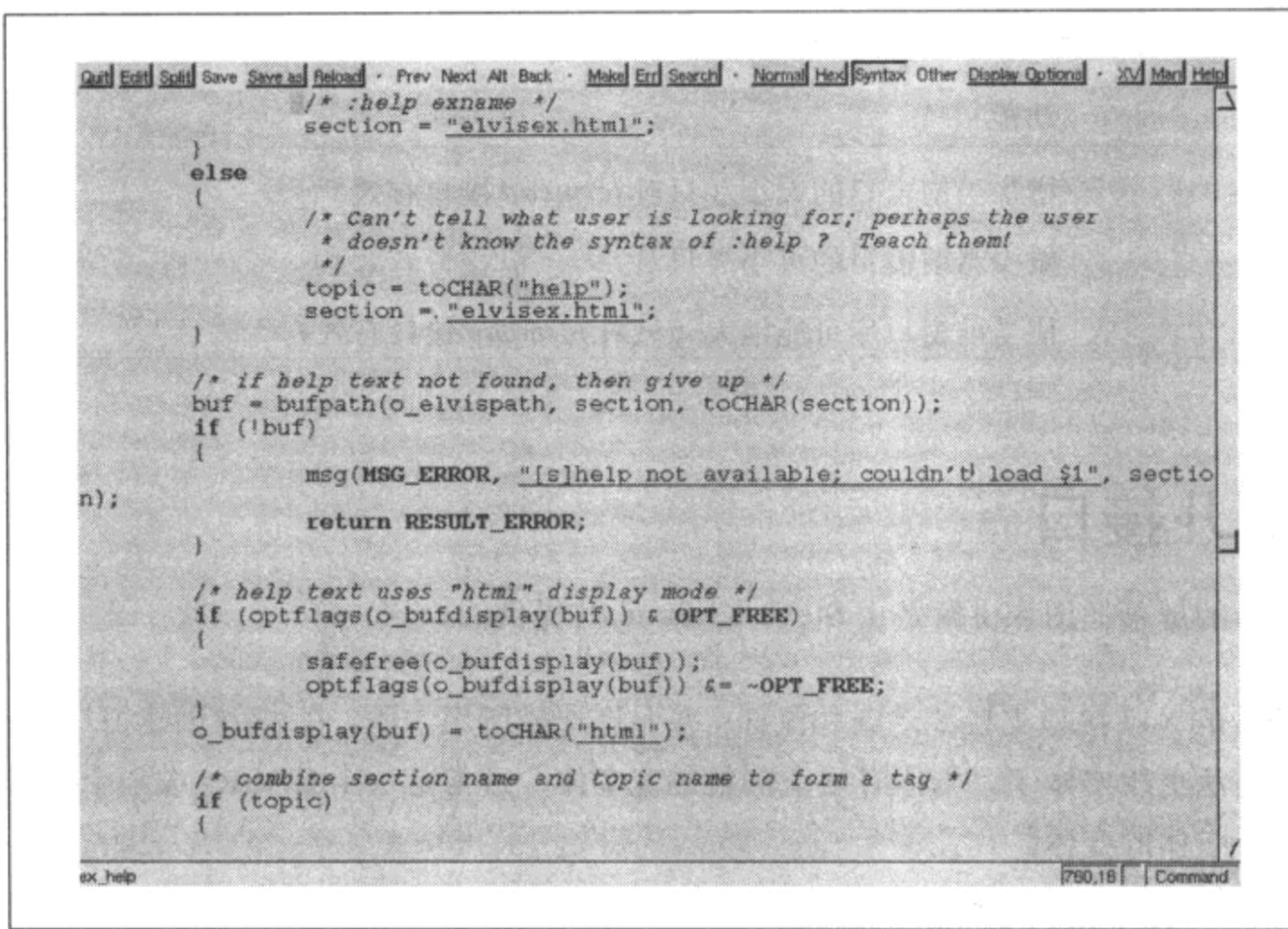


图 10-1 elvis 的 GUI 窗口

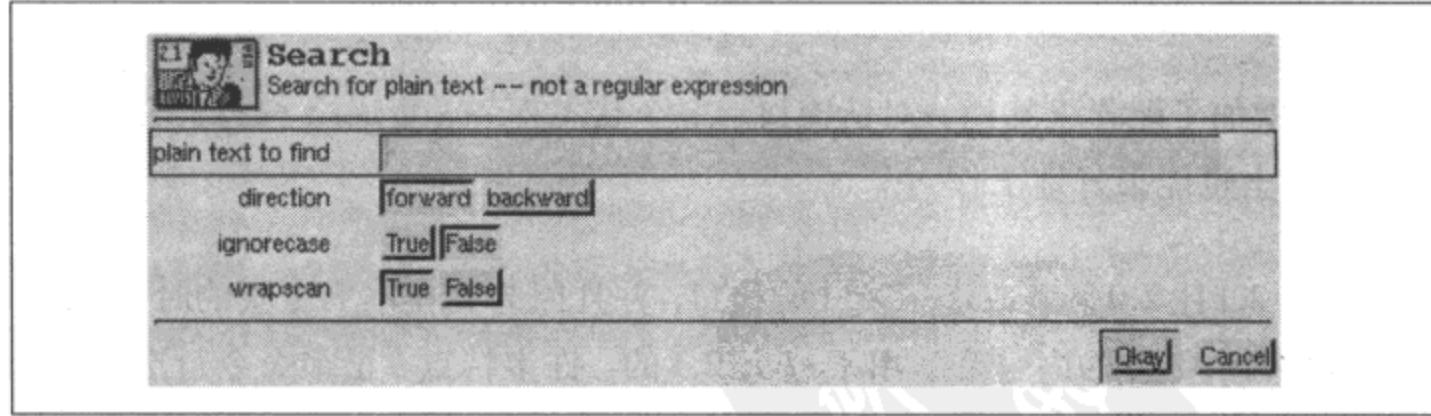


图 10-2 elvis 的查找对话框

它的外观类似于 Motif，但是 *elvis* 实际上并没有使用 Motif 库。

命令行选项可以让你选择 *elvis* 所用的四种不同的字体，常规、斜体、粗体和“控制”字体，“控制”字体用于工具栏文本和按钮标志。也可以指定前景和背景的颜色、起始窗口的几何形状和 *elvis* 是否该开始图标化。

新的`-client`选项能使`elvis`查找一个正在运行`elvis`进程，并向它发出消息，请求它编辑在命令行中命名的文件。这样允许用户在`elvis`当前编辑的文件和新文件之间共享复制的文本和其他的信息。

除了工具栏外，还有一个状态栏，用来显示状态信息和所有可得到的关于工具栏按钮的信息。

鼠标功能

鼠标的功能是试图在`xterm(1)`和编辑器之间建立平衡，为了正确做到这一点，`elvis`将单击和拖动操作区别开了。

拖动鼠标可以选定文本，按住键1（通常是左键）拖动选定字符；按住键2拖动（通常是中键）选定的是一个矩形区域；按住键3拖动（通常是右键）选定整行。这些操作分别对应于`elvis`的`v`、`^V`、`V`命令（这些命令稍后在本章中介绍）。当拖到一端放开键后，选中的文件将立即复制到一个X11剪切缓冲区中，因此，可以将它粘贴到诸如`xterm`之类的另一个应用中。这时文件仍保持选中，因此可以对其应用另一个运算符命令。

单击键1取消全部选中区域，并将光标移动到点击的字符处，单击键3移动光标，但不取消选中区域；可以通过这项功能来增大选中部分。

单击键2从X11剪切缓冲区（类似于`xterm`）中“粘贴”文件。如果输入一个`ex`命令行，则文本将粘贴到命令行中，就好像已经输入它了。如果在一个可视命令模式或输入模式下，文件将粘贴到编辑缓冲区中。无论在窗口的什么位置点击，在进行粘贴时，`elvis`通常将文本插入到文件中光标的位置处。

双击键1模拟`^】`按键，使`elvis`在点击的字上完成标志查找工作。如果`elvis`碰巧是用HTML文档显示的，标志查找会跟踪超文本链接，因此可以在任何带有下划线的正文上双击来查看描述正文的主题。双击键3模拟`^T`按键，将使你退回到最近查找的标志处。

工具栏

X11 接口支持用户可以配置的工具栏。在默认状态下，如果 `~/.exrc` 文件没有 `set notoolbar` 命令，那么工具栏是有效的。

默认工具栏已经定义了一些按钮，可以使用`:gui` 命令重新组合工具栏。

这种命令有很多。而且可以重新组合工具栏来符合自己的要求，去掉一个或全部已有的按钮、增加新按钮、控制按钮或按钮组之间的空间的大小。这里有一个简单的例子：

```
:gui Make:make
:gui Make " Rebuild the program
:gui Quit:q
:gui Quit?!modified
```

这组命令添加两个新的按钮。第一行添加的按钮是“Make”，当按下它时就会执行`:make` 命令（`:make` 命令稍后会在本章讲述）。第二行添加“Make”按钮的描述文本，当按下该按钮时，文本将在状态行显示。在这种情况下，“并不引出注释，它只是`:gui` 命令的一个操作符。

第二个按钮是“Quit”，由第三行创建，用于退出程序。第四行改变了它的操作。如果条件`(!modified)` 为真，按钮命令将正常执行。但是如果条件假，按钮将忽略鼠标的任何点击操作。并且它将显示为“平面的”而不是有通常的 3D 的外观。因此，如果当前的文件修改了，就不能使用“Quit”按钮退出。

你可以创建弹出式对话框，它在按下工具栏按钮的时候出现。对话框可以设置预定义的变量（选项）的值。这些变量可以在与按钮相关联的 `ex` 命令上得到测试。一共有 26 个预定义的变量，命名为 `a~z`，都是为此类用户“程序”而预留的。下面的例子和一个名叫“Split”的新按钮的对话框相关联：

```
:gui Split"Create a new window, showing a given file
:gui Split;"file to load:" (file) f = filename
:gui Split:split (f)
```

第一个命令和“Split”按钮的描述文本相关联。第二个命令创建一个弹出式对话框：它的提示是 *File to load:*，并设置 *filename* 选项。*(file)* 表示可以输入任意的字符串。但 **TAB** 键可以用来完整化文件名。*f=filename* 将 *filename* 的值复制到 *f* 中。最后，第三行命令实际上以 *f* 的值运行 :split 命令，*f* 是由用户提供的新文件名。

工具栏的功能很灵活，查看联机帮助可以获得全部详细资料。

选项

有很多选项可以控制 X11 接口，通常可以在 *.exrc* 文件中设置它们。有很多选项和缩写用来设置不同的字体，激活并配置工具栏、状态栏、滚动条和光标。在使用 **^W ^W** 命令进行窗口切换时，其他的选项控制光标的动作；在 *elvis* 退出时，它们决定光标是否返回到原来的 *xterm*。

联机文档描述了所有关于 X11 的 *ex* 选项，我们这里介绍一些比较重要的。

autoiconify

通常，当 **^W ^W** 命令切换到一个图标化的窗口时，该窗口取消图标化。当 *autoiconify* 为真，*elvis* 就会将旧窗口图标化，从而使打开的 *elvis* 窗口数保持不变。

blinktime

该值是在 1 和 10 之间的数，用来表明光标应该在十分之几秒后变得不可见。值取 0 取消闪烁。

firstx, firsty, stagger

firstx 和 *firsty* 控制 *elvis* 创建的第一个窗口的位置。如果没有设定，*-geometry* 选项或窗口管理器将控制它的位置。如果 *stagger* 设定一个非 0 的值，每个新建的窗口的像素将放在当前窗口的右下方。设为 0 则由窗口管理器负责安排窗口。

outlinemono

当设定它并用单色 X 显示时，*elvis* 在字符周围提供一个白色的边框。这使得

文本更加易读。其值的范围是 0 到 3，0 代表没有边框，3 表示边框最多。默认值是 2。这个选项对彩色的 X 显示无效。

stopshell

它存放了可运行一个交互式 shell 的命令，用来执行 *ex* 的 :shell 和 :stop 命令及可视命令 ^Z。默认是 xterm &，它在另一个窗口启动一个交互式终端模拟器。

xscrollbar

值 left 和 right 将滚动条放到指定的窗口一边，不指定则不能设置滚动条，默认值是 right。

elvis 2.1 增加了通过 X 资源进行配置的功能。这些资源的值可以被命令行标志覆盖。或者在初始化脚本中使用 :set 或 :color 命令进行显式配置。*elvis* 的资源如表 10-4 所列。

表 10-4 *elvis* 的 X 资源

类资源（类名字为小写）	类型	默认值
Elvis.Toolbar	Boolean	true
Elvis.Statusbar	Boolean	true
Elvis.Font	Font	fixed
Elvis.Geometry	Geometry	80x34
Elvis.Foreground	Color	black
Elvis.Background	Color	gray90
Elvis.MultiClickTimeout	Timeout	3
Elvis.Control.Font	Font	variable
Elvis.Cursor.Foreground	Color	red
Elvis.Cursor.Selected	Color	red
Elvis.Cursor.BlinkTime	Timeout	3
Elvis.Tool.Foreground	Color	black
Elvis.Tool.Background	Color	gray75
Elvis.Scrollbar.Foreground	Color	gray75

表 10-4 elvis 的 X 资源（续）

类资源（类名字为小写）	类型	默认值
Elvis.Scrollbar.Background	Color	gray60
Elvis.Scrollbar.Width	Number	11
Elvis.Scrollbar.Repeat	Timeout	4
Elvis.Scrollbar.Position	Edge	right

“Timeout” 类型给出时间值，以十分之一秒计算。“Edge” 类型给出一个滚动条的位置，值为 left、right、none 其中之一。

举例来说，如果 X 资源数据库包括行 elvis.font:10x20，默认的文件的字体值是 10x20。如果没有设置 normalfont 选项，那么就采用这个值。

扩展的正则表达式

扩展的正则表达式在第八章中的“扩展的正则表达式”一节已经进行了介绍。在 elvis 中可得到其他的元字符有：

\+ 匹配一个或多个前面的正则表达式。

\? 匹配零个或一个前面的正则表达式。

\@ 匹配光标下的单词。

\= 表明当文件相匹配时将光标放在什么位置。例如，hel\=lo 会将光标放到 hello 的下一次出现的第二个 l 处。

\{...\}

表示一个区间表达式，例如 x\{1, 3\} 用来匹配 x、xx、xxx。

POSIX 方括号表达式（字符类等）是不能使用的（注 2），也不能使用 | 字符进行选择或使用圆括号进行分组。

注 2： 虽然它们存在于 elvis 2.0 中，但是它们不能工作。不过在 elvis 2.1 中已经可以使用了。

改进的编辑功能

本节介绍了 *elvis* 可使简单的文本编辑更容易、更有效的功能。

命令行历史和完整化

你在 *ex* 命令行输入的所有内容都保存在一个名为“Elvis ex history”的缓冲区中。它和其他的 *elvis* 缓冲区一样可以访问，但以窗口方式观察时，则不能直接使用它。

为了访问历史记录，可以在终端上使用箭头键去显示以前的命令并编辑它们。使用↑和↓键翻看列表，使用←和→键在命令行中移动。你可以通过输入来插入字符，也可以使用退格键来删除它们。就像在常规 *vi* 缓冲区中编辑一样，退格键可以删除字符。但是在击键的时候行并不更新，所以要十分小心。

将文件写入了 Elvis ex history 缓冲区后，**TAB** 键可以用做文件名扩展。将前面的单词认为是文件名的一部分。接下来 *elvis* 查询所有匹配的文件，如果有多个匹配，它就会填充尽可能多的字符，并发出嘟嘟声；否则，如果匹配文件没有另外的字符来表示，*elvis* 会列出所有匹配的文件名，并重新显示命令行。如果有唯一的匹配，*elvis* 将完整化这个名字并追加一个制表符。如果没有匹配，*elvis* 就简单地插入一个制表符。

在 tab 的前面输入 ^V，就可以得到一个真正的制表符。也可以通过设置 Elvis ex history 缓冲区的 inputtab 选项为 tab 来完全禁止文件名的完整化。可以通过以下命令：

```
:{elvis ex history}set inputtab=tab
```

标志栈

标志入栈在第八章中的“标志栈”一节中已经介绍过了。在 *elvis* 中，标志入栈非常简单。如表 10-5 和表 10-6 所示。

表 10-5 elvis 的标志命令

命令	功能
ta[g] [!] [tagstring]	编辑含有 <i>tagstring</i> 的文件, <i>tagstring</i> 在 <i>tags</i> 文件中有定义。如果当前的缓冲区已经更改但没有保存, ! 将强迫 <i>elvis</i> 切换到一个新文件中
stac[k]	显示当前的标志栈
po[p] [!]	从栈中弹出一个光标的位置, 将光标恢复到它的前一个位置

表 10-6 elvis 的命令模式下的标志命令

命令	功能
^]	在 <i>tags</i> 文件中查找光标下的标识符的位置, 并移动到该位置。当前位置自动压入标志栈中
^T	返回到标志栈中的上一个位置, 即弹出一个元素

和传统的 *vi* 不同, 当键入 ^] 时, *elvis* 查找包含光标的整个单词, 而不仅仅是光标位置以后的部分单词。

在 HTML 模式 (在“显示模式”一节讲述) 下, 除了:tag 希望给出的不是标志的名字而是 URL 以外, 所有的命令都同样工作。URL 不依赖于一个 *tags* 文件, 所以将在 HTML 模式下忽略 *tags* 文件。*elvis* 2.0 并不支持任何网络协议 (注 3), 所以, 它的 URL 只包含一个文件名和 / 或 HTML 格式的 #label。

有几个:set 选项影响 *elvis* 如何对标志进行操作, 如表 10-7 所示。

表 10-7 elvis 的标志管理选项

命令	功能
taglength, tl	控制在标志中要查找的重要的字符的个数。默认值 0 表明所有的字符都是重要的

注 3: 在 *elvis* 2.1 中可以支持网络协议。详细信息请参考“elvis 功能”一节。

表 10-7 elvis 的标志管理选项（续）

命令	功能
tags, tagpath	这个值是一个目录和 / 或文件名列表，在该列表中查找 tags 文件。 <i>elvis</i> 在每一个是目录的条目中查找一个名叫 tags 的文件。为了允许在目录中出现空格，因此在列表中的条目是冒号隔开的（在 DOS/Windows 上是分号）。默认值是“tags”，在当前目录下查找名为 tags 的文件。这可以通过设置 TAGPATH 环境变量而将其覆盖
tagstack	当设为真时， <i>elvis</i> 把每个位置压入标志栈，使用 :set notagstack 来禁止标志入栈

elvis 2.1 版本（在从这次编写开始起的第二次测试中）支持前面讲述的增强的 tags 文件格式。*elvis* 有自己的 ctags 版本。*elvis* 2.1 中有前面讲述的增强格式。下面是一个它专门创建的 !_TAG_ 行的例子：

```
!_TAG_FILE_FORMAT      2      /supported features/
!_TAG_FILE_SORTED     1      /0=unsorted, 1=sorted/
!_TAG_PROGRAM_AUTHOR   Steve Kirkendall      /kirkenda@cs.pdx.edu/
!_TAG_PROGRAM_NAME    Elvis Ctags      //
!_TAG_PROGRAM_URL     ftp://ftp.cs.pdx.edu/pub/elvis/README.html  //
!_TAG_PROGRAM_VERSION 2.1      //
```

最后，在 *elvis* 中每一个窗口都有它自己的标志栈。

无限撤消

在使用 *elvis* 撤消和恢复到多次的修改之前，必须首先将 undolevels 选项设置成 *elvis* 允许的次数。负值禁止任何撤消动作（这并不是很有用）。*elvis* 文档警告说每次撤消在会话文件（该文件描述了用户编写的会话）中大约需要 6K 字节，这将会很快占用大量的存储空间。建议不要将 undolevels 设置为大于 100，“最好远低于此数”。

一旦将 undolevels 设成了非零值，就可以像平常一样输入文本。随后的每一个

u命令就撤消一次操作。为了恢复操作（撤消撤消操作），可以使用（更适合记忆的）**[CTRL-R]**命令。

在*elvis*中，undolevels的默认值是0，这和UNIX中的*vi*相似。该选项适用于每个正在编辑的缓冲区。如何为每个编辑的文件设置该参数请参见“初始化步骤”一节。

一旦设定了参数undolevels，无论u命令还是^R命令都可以按照这个设定的次数进行撤消或恢复。

任意长度的行和二进制数据

*elvis*可以编辑含有任意长度的行，或是任意行数的文件。

在UNIX下，*elvis*处理二进制文件和处理其他文件没有什么区别。在其他系统中，使用*elvis.brf*文件设置binary选项，这样就避免了换行符转换问题。你可以通过输入^X后面跟两个十六进制数来输入8位文本。使用hex显示模式来编辑二进制文件是很好的方法（*elvis.brf*文件和hex模式在“令人感兴趣的功能”一节有详述）。

左右滚动

正如在第八章中“增量搜索”一节提到的，可以在*elvis*中使用:set nowrap命令设置左右滚动条。sidescroll值在左右滚动的时候控制*elvis*屏幕所移动的字符数。**^W S**命令触发这个选项的值。

可视模式

通过使用表10-8中所示的命令，*elvis*允许你选择区域，一次一个字符、一行或者一个矩形。

表 10-8 elvis 块模式命令字符

命令	功能
v	启动区域选择，一次选中一字符模式
V	启动区域选择，一次选中一行模式
^V	启动区域选择，一次选中一矩形模式

当选中文本时，*elvis* 将高亮显示它（使用相反的颜色）。为了选中文本，只须使用通常的键操作。下面的屏幕显示了一个选中的矩形区域：

```
The 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the book into the late 1990's.
In particular, besides the &ldquo;original&rdquo; version of
<command>vi</command> that comes as a standard part of every UNIX
system, there are now a number of freely available &ldquo;clones&rdquo;;
or work-alike editors.
```

elvis 对选中的文件区域只允许进行很少的几种操作。有些操作只对整行有效，即使你选中的区域并不包括整行（见表 10-9）。

表 10-9 elvis 块模式操作

命令	功能
c, d, y	修改、删除或复制文本。只有 d 真正对矩形有效
<, >, !	向左、向右移动文本，过滤文本。这些操作对包含已标记区域的整行有效

在使用 d 命令删除了这个区域后，屏幕显示如下：

```
the 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the 90's.
In particulo;original&rdquo;version of
<command>vi as a standard part of every
system, there are n available &ldquo;clones&rdquo;;
or work-alike editors.
```

编程辅助

这部分讲述 *elvis* 的编程辅助功能。

编辑 – 编译加速

elvis 提供在对程序操作时更容易留在编辑器中的命令。你可以重新编译单个文件，也可以重新构建整个程序，并可以在编辑器中一次处理一个编译错误。*elvis* 命令总结如表 10-10 所示。

表 10-10 *elvis* 程序开发命令

命令	选项	功能
cc[!] [args]	ccprg	运行 C 编译器，它在重新编辑独立文件时有用
mak[e][!] [args]	makeprg	可以重新编译任何需要重新编译的内容（通常通过 make(1)）
er[rlist][!] [file]		移动到下一个错误位置

`:cc` 命令可以重新编译一个独立的源文件。可以在冒号命令行运行它。例如，如果正在编辑文件 `hello.c`，则输入 `:cc`，*elvis* 就会对其进行编译。

如果对 `:cc` 命令提供追加的参数，那么这些参数将传递到 C 编译器上。在这种情况下，需要提供所有的参数，包括文件名。

`:cc` 命令通过执行 `ccprg` 选项的文本来工作。默认值是 "`cc ($1?$1:$2)`"。*elvis* 将 `$2` 设置为当前的源文件名，将 `$1` 设置为 `:cc` 命令中给定的参数。如果出现给定的参数，`ccprg` 就将采用给定的值。否则，它就会仅仅将当前文件名传递给系统 `cc` 命令（当然，可以根据需要来改变 `ccprg`）。

类似地，`:make` 命令用来重新编译需要重新编译的一切内容。它通过执行 `makeprg` 选项的内容来实现。`makeprg` 的默认值是 "`make $1`"。因此，可以键入 `:make hello`，即仅仅对 `hello` 程序进行重新编译，或只输入 `:make` 重新编译每个程序。

elvis 捕获编译或 *make* 的输出，并查找类似文件名或行号的内容。当 *elvis* 发现了可能的选择时，*elvis* 就会将之当成输出，并移动到第一个发生错误的位置。`:errlist` 命令依次在每一个连续的错误的位置移动。当移动到每个错误的位置时，*elvis* 在状态行依次显示每个错误的信息文本。

如果对`:errlist` 提供了一个 *filename* 参数，则 *elvis* 将从该文件中载入大量错误信息，并移动到第一个错误的位置。

vi 模式命令 *（星号）是和`:errlist` 等价的。当处理许多错误时，这个命令使用起来很方便。

最后，*elvis* 的一个真正优点是它补偿了文件中的变化。在你增加或删除行时，*elvis* 保持跟踪状态，这样在要处理下一个错误时可以停在正确行，这并不一定和编译器错误信息显示的行号绝对一致。

语法高亮显示

为了使 *elvis* 做到语法高亮显示，可使用`:display syntax` 命令。它是个针对单个窗口的命令（其他的 *elvis* 显示模式将在“显示模式”节讲述）。*elvis* 一共可用六种不同的字体来显示文本：常规，粗体，斜体，下划线，加粗和固定字体（它们可以缩写为一个单个字母），这种语法显示模式使用下面的选项把字体和各种不同的语法部分联系起来。

- `commentfont`: 该字体（常规、斜体等）用来表示编程语言的注释。
- `functionfont`: 该字体用来表示函数名。
- `keywordfont`: 该字体用来表示编程语言的关键字。
- `preffont`: 该字体用来表示 C 或 C++ 预处理指示。
- `stringfont`: 该字体用来表示字符串常量（比如在 Awk 中的 "Don't panic!"）。
- `variablefont`: 该字体用来表示变量、域等。

- otherfont: 该字体用于不符合其他字体但又不能以普通字体显示的情况下。(例如, 用 C 的 `typedef` 关键字定义的类型名)。

各种语言的注释、函数、关键字等的说明都存储在 `elvis.syn` 文件中。这个文件中已有很多详细的说明文稿。例如, 下面有一段 Awk 语法的说明:

```
# Awk. This is actually for Thompson Automation's AWK compiler, which is
# somewhat beefier than the standard AWK interpreter.
language tawk awk
extension .awk
keyword BEGIN BEGINFILE END ENDFILE INIT break continue do else for function
keyword global if in local next return while
comment #
function (
string "
regexp /
useregexp (,~
other allcaps
```

格式基本上是自我解释的, 这种格式在 `elvis` 的联机文档上得到了充分的说明。

`elvis` 将字体和文件语法的不同部分联系起来的理由是, `elvis` 能够把文件按照在屏幕上显示的那样打印出来 (参见 “显示模式” 中有关 :lpr 命令的讨论)。

除了可以为各种成份指定字体, 还可以通过命令 :color 将每种字体 (常规, 斜体等) 和一种颜色关联起来。

在非位图的显示设备上, 比如 Linux 控制台, 所有的字体都将映射成控制台驱动程序使用的字体。这样就很难区分常规字体和斜体, 但是, 在某些显示设备上(比如 Linux 控制台), 可以改变不同字体的颜色。如果使用装有 `elvis` 的 Linux 系统, 用它编辑一个适当的 C 源文件, 接下来调用下面的命令:

```
:display syntax
:color normal white
:color bold yellow
:color emphasized green
:color italic cyan
:color fixed red
```

屏幕上将出现黄色加亮的C关键字、蓝色的注释、绿色的预处理指示、红色的字符和字符串常量。遗憾的是，我们不能在打印时重现此效果。

在*elvis*中，语法颜色是对单个窗口有效的。你可以在一个窗口中为斜体字改变颜色，但这不会影响其他窗口中的斜体字的颜色。甚至在两个窗口正在显示同一个文件时也是如此。

语法着色使得编写程序变得更加生动有趣。但是在选择颜色的时候必须小心。

令人感兴趣的功能

*elvis*有一系列令人感兴趣的功能。

国际化支持

像*nvi*一样，*elvis*也有将消息翻译成不同语言的方法。通过*elvispath*找到*elvis.msg*文件，并把它装入名为Elvis messages的缓冲区内。

消息有“terse message（短消息）：long messages（长消息）”这种格式。在打印一条消息之前，*elvis*先检查短格式消息，如果有一个对应的长格式消息，那就使用该长格式，否则就用短格式的消息。

显示模式

这也许是*elvis*最令人们感兴趣的功能了。对于特定类型的文件，*elvis*在屏幕上安排它们的格式，给出一个几乎是WYSIWYG所见即所得的效果。*elvis*同样可以使用类似的格式将缓冲区打印到几种打印设备上。下面有专门介绍显示模式的小节。

处理文件前、后的操作

*elvis*可以载入四个文件（如果它们存在），这使得在读写文件之前或之后，可以自定义它的动作。下面有专门介绍这个功能的小节。

打开模式

*elvis*是惟一的一种真正实现*vi*的打开模式的克隆版本（将打开方式想像成和

vi 相似，只不过它只有一行的窗口。打开模式的“优点”是它可以在没有光标移动功能的终端上使用)。

安全

:safer 命令可以设定 safer 选项来执行非主目录下的 .exrc 文件或其他不可信任的文件。一旦设置好了 safer，“某些命令被禁用，文件中的通配符扩展也变得无效，某些选项也将锁定（包括 safer 选项自己）。” *elvis* 文档并不是如此具体。不要盲目相信 *elvis* 为你提供了彻底的安全。

内置计算器

elvis 使用一个内置的计算器（有时指的是文档中称为求值程序的工具）扩展了 *ex* 命令语言。它能够识别 C 表达式语法，并在 :if、:calc 和 :eval 命令中常常使用。具体细节和在 *elvis* 发布版本初始化文件请参见联机帮助。

宏调试器 (2.1)

elvis 2.1 带有一个 *vi* 宏的调试器（:map 命令）。这在编写复杂的输入或命令映射时很有用。

显示模式

elvis 有几种显示模式。根据文件的类型，*elvis* 产生该文件的一种格式化版本，达到一种 WYSIWYG 所见即所得的效果。显示模式在表 10-11 中列出。

表 10-11 *elvis* 显示模式

命令	显示方式
normal	没有格式，以在文件中存在的方式进行显示文本
syntax	和 normal 模式相似，但是启动语法着色
hex	一个交互式十六进制转储，是旧式的大型机十六进制转储。它比较适合编写二进制文件
html	一个简单的网页格式化程序。标志命令可以用来跟踪链接并返回
man	简单的 man page（手册页）模式。类似 nroff -man 的输出

:normal 命令会将显示模式从一种格式化的视图切换成 normal 视图。使用 :display mode 可以切换回来。^W d 命令作为简便操作，可以为窗口触发一个显示模式。

在所有可用的模式中，html 和 man 在性质上最具有 WYSIWYG 特性。联机文档明确地定义了 *elvis* 所识别的全部两种标记语言的子集。

elvis 使用 html 模式来显示它的联机帮助，帮助是使用 HTML 格式编写的，并且其中有很多交叉引用的链接。

下面的例子显示了 *elvis* 编辑 HTML 帮助文件。屏幕是分开的。两个窗口显示的是同一个缓冲区，下面的窗口在使用 html 显示模式，但上面的窗口使用的是 normal 显示模式。

```
<html><head>
<title>Elvis 2.0 Sessions</title>
</head><body>

<h1>10. SESSIONS, INITIALIZATION, AND RECOVERY</h1>
```

This section of the manual describes the life-cycle of an edit session. We begin with the definition of an [edit session](#SESSION) and what that means to elvis. This is followed by sections discussing [initialization](#INIT) and [recovery after a crash](#RECOVER).

10.0 SESSIONS, INITIALIZATION, AND RECOVERY

This section of the manual describes the life-cycle of an edit session. We begin with the definition of an **edit session** and what that means to elvis. This is followed by sections discussing **initialization** and **recovery after a crash**.

10.1 Sessions

man 显示模式也很值得关注，因为通常必须对 man page 进行排版和打印，以表示进行了一次布局工作。下面从联机帮助中引用的部分看起来非常合理：

Troff 始终没有设计为交互式编辑，尽管我已尽到了努力，但在 man 模式下进行编辑仍然是一个不好的体验。我建议，做修改时习惯使用 normal 模式，在预览这些修改的效果时使用 man 模式。可以使用 ^W d 命令在两种模式间进行切换。

作为一个值得注意的追加功能，html 和 man 模式都可以和 :color 命令一起工作。:color 命令在“语法高亮显示”一节有详细介绍。语法高亮显示在 man 模式下非常合适。例如，在 Linux 控制台的默认设置下，只有粗体文本 (.B) 与普通文本有区别。但是借助于语法变色，粗体和斜体 (.I) 文本就可以分辨了。模式命令总结在表 10-12 中。

表 10-12 elvis 显示模式命令

命令	功能
di [splay] [[mode] [lang]]	将显示模式改为 mode。使用 lang 来表示语法模式
no [rmal]	和 :display normal 相似，但是比它易于输入

bufdisplay 选项和每个窗口都有联系，它必须设置为 elvis 支持的显示模式中的一种。标准 elvis.arf 文件（参见下一小节）将查看缓冲区文件名的扩展，并试图设成一种比 normal 模式更有吸引力的模式。

最后，elvis 能够将它的 WYSIWYG 格式打印缓冲区的内容。:lpr 命令可以格式化一行（或整个缓冲区，默认值）的打印范围。可以将其打印到一个文件或一个命令管道中。默认情况下，elvis 将它打印到一个执行系统假脱机打印命令的管道上。

:lpr 命令由几个选项控制，如表 10-13 所示。

表 10-13 elvis 的打印管理选项

命令	功能
lpctype, lp	打印机类型
lpconvert, lpcvt	如果设置，将 Latin-8 扩展 ASCII 转换成 PC-8 扩展 ASCII
lpcrlf, lpc	打印机需要 CR-LF 来结束每一行
lpout, lpo	要打印的文件或命令
lpcolumns, lpcols	打印机的宽度
lpwrap, lpw	模拟换行
lplines, lprows	打印机的页长度
lpformfeed, lpff	在最后一页后，进纸
lppaper, lpp	纸张的大小（信纸、A4 等）。这只对 PostScript 打印机有意义

大多数选项都是自我解释的。*elvis* 支持几种打印机类型，如表 10-14 所示。

表 10-14 lpctype 选项值

命令	打印机类型
ps	PostScript，每张纸一逻辑页
ps2	PostScip，每张纸两逻辑页
epson	大多数点阵打印机，不支持图形字符
pana	松下点阵打印机
ibm	具有 IBM 图形字符的点阵打印机
hp	Hewlett-Packard 打印机和大多数非 PostScript 激光打印机
cr	行打印机，利用回车完成输入
bs	通过退格键完成输入，这个设置和传统 UNIX 的 <i>nroff</i> 很相似
dumb	通常的 ASCII，没有字体控制

如果你有一个 PostScript 打印机，一定要使用值为 ps 或 ps2 的 lpctype。使用后者节约纸张，这在打印草案的时候特别有用。

处理文件前、后的操作

在读写文件时, *elvis* 给你在四个时间点上控制它的动作的能力: 在读文件之前和之后, 或在写文件之前和之后。*elvis* 通过在每个时间点上分别执行四个 *ex* 脚本的内容来实现这种操作。这些脚本在 *elvispath* 选项列出的目录中进行查找。

elvis.brf

这个文件是在读一个文件之前执行。默认版本检查文件的扩展, 并试着去确定文件是否是二进制的。如果它是二进制的, 就启动 *binary* 选项, 以防止 *elvis* 将换行符 (它可能是文件中实际的 CR-LF 对) 转换为内部的换行符。

elvis.arf

这个文件是在读一个文件之后执行, 默认版本检查文件的扩展并启动语法高亮显示。

elvis.bwf

这个文件是在写一个文件之前执行, 特别是在用缓冲区内容完全替换原始文件之前。默认版本将原始文件复制到一个带有 *.bak* 扩展的文件中。为了实现这个功能, 必须设置 *backup* 选项。

elvis.awf

这个文件是在写一个文件之后执行。尽管在这里可以将钩子 (hook) 加入到源代码控制系统中, 但是, 它没有一个默认文件。

使用命令文件来控制这些动作是很有效的, 它可以使你很容易地将 *elvis* 的动作调节成满足你所需要的。在其他的编辑器中这些功能大多被固定为了代码。

elvis 的未来

在编写这本书的时候, *elvis* 2.1 正在二级测试的后期, 在本书出版时, 它也许已经发布了。Steve Kirendall 提供了在 *elvis* 2.1 中的可能变化和新功能的列表:

- 在 Windows 95 和 Windows/NT 下, 有一个图形版本的 *elvis*。这是在 2.0 版本中包括的文本模式版本的补充。

- 增加了文本模式的 OS/2 版本。
- 在 X Window 下，有一个新的状态栏和一个可配置工具栏。这个工具栏可以调用可配置的对话窗口。同时，许多 X 特性在标准的 X 资源数据库中有默认值。新的命令行标志包括 -mono, -fork 和 -client。
- DOS 版提供鼠标支持，这类似于 X Window。
- *elvis* 2.1 版本提供增强的标志格式，在第八章中“Exuberant ctags”一节有详细的论述。

elvis 2.1 借助标志进行了一些创造性工作。当读入过量的标志时，它将试图去猜测哪一个是你要找的，并显示出最可能的一个。如果拒绝了这一个（通过再次输入 ^] 或 :tag），*elvis* 就会显示下一个最有可能匹配的，直到找到为止。它还将注明拒绝或接受的标志的属性，并使用它来提高后续猜测性启发式搜索。

:tag 命令的语法已经进行了扩展，允许根据特定功能而不仅仅是根据标志的名字进行查找。它的功能很强大，但是在讲述则太复杂了 [在 Steve Kirendall 的邮件信息中]。在手册 [联机帮助] 中有完整的一章讲述标志的用法。

同时有一个:browse 命令可以一次查找所有可能匹配的标志，并根据它们建立一个 HTML 表。根据这个表，可以通过超文本链接查找任意的匹配标志。

最后，*elvis* 2.1 有一个新的 tagprg 选项，如果设置了这个选项，它将放弃内置的查找算法，并通过运行一个外部程序来完成查找。

- 如果正在使用 syntax 显示模式，可视的 % 命令已经扩展成能够识别 #if, #else 和 #endif 指令。
- 增加了一个新的 tex 显示模式。虽然它是不可编程的，但是它多少还是有用的。
- ^W d 命令在 2.1 版本中比 2.0 版中更加小巧灵活。如果情况适当，该命令将会在 syntax 模式和其他任意的显示模式 (html、man、tex) 中选定一种。这使得编辑网页更加方便了。

- *elvis* 可以通过 HTTP 或 FTP 获取文件。也可以通过 FTP 进行写操作。只要在 *elvis* 希望输入文件名的位置处简单地给出一个 URL 就行。为了访问在一个 FTP 站点上你自己的账户（不是匿名账户），URL 的目录名位置必须以 ~/ 开头，这样 *elvis* 会读取 `~/.netrc` 文件以寻找正确的名字和密码。`html` 显示模式充分利用了这些功能（网络功能在 Windows 和 OS/2 下也有效）。
- 为了符合 POSIX 标准，对命令行标志进行了改动。`-s` 原来是设置 `safer` 标志，用来实现特别的安全，但是现在这个标志让 *elvis* 从标准输入中读取并执行文件（这和 *nvi* 相匹配。A.R）。现在使用大写的 `-S` 命令来设置 `safer`。
- 增加了一个新标志 `-o filename`，因此可以将启动的消息重定向到一个文件中，而不是标准的输出/错误输出。这对 Windows 95 或 Windows NT 的用户是非常重要的，因为 Windows 放弃向标准的输出/错误输出写任何文件。这几乎将不可能诊断 Winelvis 配置出现的问题。借助 `-o filename`，可以将诊断信息输入到一个文件并在以后进行查看。
- 增加了一个新命令 `:alias`，用来定义 `ex` 宏。它的目的是模拟 `csh alias` 命令。
- *elvis* 2.0 错误地实现了 POSIX 命名字符类（使用正则表达式）。*elvis* 2.1 修正了它。例如，可以通过 `/\\<[[:alpha:]_][[:alnum:]_]*` 来查找一个 C 标识符。

源代码和支持的操作系统

elvis 的正式 WWW 地址是 `ftp://ftp.cs.pdx.edu/pub/elvis/README.html`，在那里可以下载发布资料。或直接使用 `ftp` 从 `ftp://ftp.cs.pdx.edu/pub/elvis/elvis-2.0.tgz` 获得它。

elvis 的源代码是免费发布的。在发布的 `COPYING` 文件中描述了许可条款。并且允许以源代码和二进制形式发布。*elvis* 2.1 将会在 `perl` 的 Artistic 许可证条款中发布。

elvis 可在 UNIX、MS-DOS、Windows 95 或 Windows NT 上运行。在编写这一部分时，OS/2 的版本正在开发，但是并没有结合到源代码中（只能看到前面的部分）。

编译 *elvis* 是非常容易的。通过 *ftp* 或经过 Web 浏览器得到它的发布版本。对它进行解压（注 4）后运行 *configure* 程序，然后再运行 *make* 即可。

```
$ gzip -d < elvis-2.0.tgz | tar -xvpf -
...
$ cd elvis-2.0; ./configure
...
$ make
...
```

elvis 配置和构建应该不会遇到什么问题。使用 *make install* 安装它。

注意： 在 *elvis* 2.0 中，若在 Linux 系统中使用 GCC，则应该在不进行优化的情况下重新编译 *lp.c* 文件。否则，至少在我们的经验中，当使用 :lpr 命令格式化和打印编辑缓冲区的内容时，*elvis* 将会进行信息转储。

如果需要报告 *elvis* 中的漏洞或问题，可以与 Steve Kirkendall 联系，其邮箱是 *kirkenda@cs.pdx.edu*。

注 4：对于在非 UNIX 系统上解压缩由 *gzip* 压缩的 *tar* 文件，从 *elvis* *ftp* 站点获得的 *untar.c* 是非常方便、简单的程序。

本章内容：

- 作者和历史
- 重要的命令行参数
- 联机帮助和其他的文档
- 初始化
- 多窗口编辑
- GUI 接口
- 扩展的正则表达式
- 改进的编辑功能
- 编程辅助
- 令人感兴趣的功能
- 源代码和支持的操作系统

第十一章

vim — 改进的 vi

Vim 代表 “Vi Improved”，它由 Bram Moolenaar 编写，并由他继续进行维护。现在 *vim* 也许是使用最广泛的 *vi* 克隆版本，有一个专门关注它的 Internet 域 (*vim.org*)。本书的更新使用了各式各样的 *vim* 版本，后期工作大多数是用 5.0 版完成的。当完成本书的更新时，5.1 版也问世了，该发布版本主要是用于修补漏洞。

作者和历史

本节是根据 *vim* 作者 Bram Moolenaar 提供的资料改编的，我们非常感谢他。

作者买了一台 Amiga 计算机后就开始编写 *vim*。由于来自 UNIX 学术界，他从使用名为 “stevie” 的类 *vi* 编辑器开始，但是这个编辑器很不完善。幸运的是它带有源代码，这是编写 *vim* 的基础。起初是为了使该编辑器与 *vi* 更好地兼容和修补一些漏洞。后来该编辑器变得非常有用，于是就在 Fred Fish 磁盘 591 上发布了 *vim* 1.14 版（用于 Amiga 的免费软件收集品）。

一些人开始使用该编辑器并喜欢上它，然后开始帮助开发它。最初编写了UNIX版本，接下来编写了对MS-DOS和其他操作系统的版本。*vim*成了应用最广泛的*vi*克隆版本之一。更多的功能被逐渐加上：多级撤消、多窗口等。虽然有些功能是*vim*独有的，但很多是受其他*vi*克隆版本的启发而产生的。它的目标一直是为用户提供最好的功能。

今天，*vim*无论在何处都是功能最全的*vi*风格的编辑器之一。其联机帮助也很全面（下面有更详细的讲述）。

*vim*的一个独特的功能就是它能够从右到左输入，这对像Hebrew和Farsi的语言是很有用的。这也表明了*vim*的多功能性。在5.0版中与*vi*兼容性也得到了改进，而且也进一步调整了性能。成为一个稳定、专业软件开发者可以依赖的编辑器是*vim*的另一个设计目标。*vim*极少崩溃，一旦发生了还可以恢复所做的更改。

*vim*的开发还在继续。*vim*6.0版计划包含支持折叠（能够隐藏部分文本，如函数体）。致力于增加功能和把*vim*移植到更多平台的人员在不断增加，而且移动到不同计算机系统的质量也在不断地提高。MS-Windows版将支持对话框和文件选择器，这将使难以学习的*vi*命令向广大用户开放。

重要的命令行参数

*vim*根据调用的方式来决定工作方式。如果将*vim*作为*ex*调用，那么就可以将其作为一个行编辑器运行。同时，它还允许使用Q命令从*vi*模式切换到*ex*模式。如果将*vim*作为*view*调用，它将在*vi*模式下开始运行，但会将每个文件标记为只读方式。

当*vim*作为*gvim*或*gview*调用时，在X Window或其他合适的图形界面下*vim*将启动GUI版本。如果在每个名字前加上前导r，*vim*将进入“受限”模式，这种情况下的一些功能将是无效的。

*vim*有很多命令行选项，下面介绍了最有用的一些选项：

-c command

在启动时执行 *command*。虽然这是过去的 +*command* 语法的 POSIX 版，但是 *vim* 不局限于命令的位置（原有语法也可接受）。你可以给出多达 10 个的 -c 命令。

-R 以只读方式开始，并设置 readonly 选项。

-r 恢复指定的文件，如果没有在命令行上列出文件，那么就列出所有可以恢复的文件。

-s 进入批处理（脚本）模式，这仅用于 *ex*，目的是运行编辑脚本，这是 POSIX 版过去的 “-” 参数。

-b 以二进制模式开始，它设定了几个使编辑二进制文件成为可能的选项。

-f 用于 GUI 版本，在前台运行。它必须由调用 *vim* 的程序使用，并一直等到它结束，如邮件处理程序。

-g 如果编译进了该功能，将以 *vim* 的 GUI 版本开始。

-o [N]

如果设定了将打开 *N* 个窗口，否则为每个文件参数打开一个窗口。

-i viminfo

读取指定的而不是默认的 *viminfo* 文件进行初始化。

-n 不产生交换文件。虽然不可以进行恢复，但它在诸如软盘类的低速介质上编辑文件时非常有用。

-q filename

将 *filename* 看成“快速修补”文件。该文件应该含有有一个错误信息列表，*vim* 将使用这个列表在程序中定位每个错误。“快速修补”模式在“编辑 – 编译加速”一节讲述。

-u vimrc

读取指定的 *vimrc* 文件进行初始化，跳过所有其他的正常初始化步骤。

-U *gvimrc*

读取指定的 *gvimrc* 文件进行 GUI 初始化，跳过所有其他的正常 GUI 初始化步骤。

-Z 进入受限模式（与在名字前有一个前导 *r* 相同）。当启动它时不能启动 shell 命令，也不能挂起编辑器。

-i、-n、-u、-U 选项将在后面详细地介绍。*vim* 还有更多的选项，感兴趣的读者可以参考联机文档以获取细节内容。

联机帮助和其他的文档

vim 有广泛而全面的联机帮助。该帮助由 50 多个 ASCII 文本文件组成，总计大约 25 500 行文本。

在线帮助实质上是超文本的，你可以使用标志命令 ^] 和 ^T 沿着某个引用前进和返回到先前的位置。如果有彩色显示器，则使用带有语法着色的帮助尤其高效。

超文本格式是 *vim* 独有的。但 *doc* 目录下含有 *makefile* 和 *awk* 脚本，它们可以把文件转换成 HTML，以便用 Web 浏览器仔细查看 (*html* 显示模式在 *elvis* 中能很好地工作)。起始点是 *help.html*，来自联机帮助的起始点 *help.txt*。

当然，联机帮助也包括 *vim* 的 UNIX 手册页。

可以使用 :help 命令启动帮助系统，它将分割窗口。在没有参数的情况下，*vim* 显示 *help.txt* 文件。如果对 :help 设定了参数，*vim* 就会尽可能地查找关于该主题的帮助。在我们的实践中，它能够很好地工作（该功能看起来好像建立在已应用于帮助文件文本的标志机制之上）。

初始化

本节讲述 *vim* 的初始化步骤，其中包括 *vim* 的 GUI 版本采用的步骤。

所有 vim 调用的初始化

vim 按以下步骤进行初始化：

1. 在 SHELL 和 TERM 环境变量中分别设定 shell 和 term 选项。在 MS-DOS 和 Win32 中，如果没有设定 SHELL，就使用 COMSPEC 设定它。
2. 如果提供了 -u 选项，则执行给定的文件，并跳过其余基于初始化的启动文件。-s 选项对 *ex* 模式有同样的功能，只有 -u 选项被解释。使用 -u NONE 会使 *vim* 跳过所有的进一步初始化步骤。
3. 执行系统范围的 *vimrc* 文件。在编译 *vim* 时设置确切的路径，通常值为 */usr/local/share/vim/vimrc*。
4. 依次寻找下面的项，执行首次找到的项中的指令：
 - 环境变量 VIMINIT。
 - 用户 *vimrc* 文件，在 UNIX（或 Linux）下是 *\$HOME/.vimrc*。在非 UNIX 系统中这个位置会有所不同。如果不存在 *.vimrc*, *vim* 就会查找 *_vimrc*。在非 UNIX 系统下，这个过程将反过来。
 - 环境变量 EXINIT。
 - 用户 *.exrc* 文件，*\$HOME/.exrc*。在非 UNIX 系统中，将使用 *_exrc*。然而在这种情况下，*vim* 仅仅查找其中的一个，但不是两个。
5. 如果设置了 exrc 选项，*vim* 就在当前目录下依次寻找下面四个文件中第一个，找到后将忽略后面的文件。
 - *.vimrc*
 - *_vimrc*
 - *.exrc*
 - *_exrc*

在 MS-DOS 和 Win32 系统中，*_xxxrc* 文件在 *.xxxrc* 之前进行查找。

6. 如果还没有设置上述文件, `shellpipe` 和 `shellredir` 选项就根据 `shell` 选项的值进行初始化。`shellredir` 选项在“编辑 – 编译加速”一节讲述。
7. 如果在命令行给出了 `-n` 选项, 则将 `updatecount` 设为 0 (该选项控制着交换文件更新的频率。频率越高, 交换文件就与所有的更改同步越好, 但是这可能会降低系统性能。0 表示从不更新)。
8. 如果提供了 `-b` 选项, 就为编辑二进制文件设置适当的选项。
9. 进行 GUI 初始化, 参见下一节。
10. 如果设置了 `viminfo`, 将读取这里指定的文件。
11. 如果提供了 `-q` 选项, 就读取命名快速修补文件, 快速修补文件在“编辑 – 编译加速”一节介绍。
12. 打开并填充所有的窗口, 类似于对每个窗口执行 `-o` 选项。如果提供了 `-q` 选项, 将光标移动到第一个错误处。
13. 如果使用了 `-t` 选项, 就跳到由它指定的标志位置。执行用 `-c` 给出的全部命令。

上述是一些初始化步骤。与其他领域一样, *vim* 的额外功能也提供了额外的灵活性和定制功能。

就像 *nvi* 一样, 可以在 `.exrc` 文件中进行正常的初始化操作 (也就是用于 UNIX *vi* 和 / 或其克隆版本的选项和命令), 并使 `.vimrc` 文件在 *vim* 进行特定的初始化之前或之后执行 `:source .exrc`。

viminfo 文件很像 *elvis* 中的会话文件, 用于保存两次登录之间编辑会话的大部分状态。*viminfo* 文件存放下述内容:

- 命令行历史
- 搜索串历史
- 寄存器的内容

- 文件标记，它指向文件中的位置
- 上次查找 / 替换模式（用于 n 和 &）

vim 在启动时读取这个文件，并在退出时，将文件的当前状态与文件的当前内容合并后，重新写入该文件。

GUI 的初始化

如果运行 *vim* 的 GUI 版，为了在后台运行，*vim* 通常会派生出一个新进程，以便可以继续给父 shell 发送命令。-f 选项禁止此项功能。

如果提供了-U 选项，*vim* 会执行给定的文件并跳过基于初始化的其余 GUI 启动文件。使用 -U NONE 选项，*vim* 将跳过所有的进一步初始化过程。

如果不使用 -U 选项，*vim* 首先读取系统级 *gvimrc* 配置文件（通常是 /usr/local/share/vim/gvimrc），然后读取用户 *gvimrc* 文件 (\$HOME/.gvimrc)。

可以使用这些文件配置 GUI，尤其是可以在此处创建自己的菜单。

多窗口编辑

有大量用于控制窗口的 *vi* 模式命令，以及很多与大部分 *vi* 模式命令相对应的 *ex* 命令。

正如在 *elvis* 中所做的那样，:split 命令将创建新窗口，然后可以使用 *ex* 命令 :e filename 在新窗口中编辑新文件。类似于 *elvis*，[CTRL-W] [CTRL-W] 可以在窗口之间来回切换。

```
<preface id="VI6-CH-0">
<title>Preface </title>

<para>
Text editing is one of the most common uses of any computer system, and
```

<command>vi</command> is one of the most useful standard text editors on your system.
With <command>vi</command> you can create new files, or edit any existing UNIX text file.

```
ch00.sgm
# Makefile for vi book
#
# Arnold Robbins

CHAPTERS = ch00_6.sgm ch00_5.sgm ch00.sgm ch01.sgm ch02.sgm ch03.sgm \
           ch04.sgm ch05.sgm ch06.sgm ch07.sgm ch08.sgm
APPENDICES = appa.sgm appb.sgm appc.sgm appd.sgm

POSTSCRIPT = ch00_6.ps ch00_5.ps ch00.ps ch01.ps ch02.ps ch03.ps \
             ch04.ps ch05.ps ch06.ps ch07.ps ch08.ps \
```

Makefile

分割窗口是输入 vim ch00.sgm 后输入 :split Makefile 的结果。

与 *nvi* 和 *elvis* 不同, *vim* 的所有窗口都共享屏幕的底行来执行 *ex* 命令。但是, 如果文件已经被修改, 则每个文件的状态行将显示 [+]。选项控制状态行是反向显示还是高亮显示, 以及底部窗口是否有状态栏。默认情况下, 当有多个窗口时, 底部窗口没有状态栏。

表 11-1 描述了最重要的 *vim* 窗口管理命令。

表 11-1 *vim* 的窗口管理命令

命令	功能
[N]sp[lit] [position] [file]	将当前窗口分成两个。N 是新窗口的高度, position 指示光标在文件中的位置。如果给定了 file, 就在新窗口中编辑它
[N]new[position] [file]	创建一个新窗口, 编辑一个空缓冲区。如果给定了文件, 就用它代替空缓冲区。N 和 position 与在 :split 中的意义相同

表 11-1 vim 的窗口管理命令（续）

命令	功能
[N]sv[iew][position][file] q[uit] [!]	除了为缓冲区设置 readonly 选项外，与 :split 一样退出当前窗口（如果在最后窗口中给出就完全退出）。如果它是已修改缓冲区的最后一个窗口，该命令就会失败，除非给出！，但这种情况就会丢失所有的修改。当设置 hidden 选项时，即使使用！也不能释放缓冲区
clo[se] [!]	关闭当前窗口。设置 hidden 选项将只隐藏未保存的缓冲区，如果没有设置该选项，则该命令就会失败。如果提供了结尾！，就会强行关闭窗口，即便它是最后一个窗口且缓冲区已被修改
hid[e]	如果当前窗口不是屏幕上的最后一个，就关闭它。如果它是缓冲区上最后打开的窗口，则缓冲区将被隐藏起来
on[ly] [!]	使该窗口成为屏幕上的惟一窗口。不关闭其他已更改的窗口，除非设置了 hidden 或使用了！。在任何情况下，都不会丢失更改。其他的缓冲区可能被隐藏起来，但并没有放弃它们
res[ize][± n]	按 n 增加或减少当前窗口的高度
res[ize][n]	如果提供了 n，就把当前窗口的高度设置为 n；否则，在不隐藏其他窗口的情况下，将它设置得尽可能大
qa[l]l] [!]	退出 vim。即使一些缓冲区已被修改但还没有被保存，！也会强迫退出
wqa[l]l] [!]	保存所有已修改的缓冲并退出。！强迫写入 readonly 缓冲区。只要有缓冲区没有对其写入，vim 就不会退出
xa[l]l] [!]	
wa[l]l] [!]	写入所有带文件名的已修改缓冲区。！强迫写入 readonly 缓冲区
[N]sn[ext]	分割窗口并移动到参数列表中的下一个文件，如果给出了数字 N，就移动到第 N 个文件
sta[g][tagname]	分割窗口，并在新窗口中运行 :tag 命令

有很多用于管理参数列表和打开缓冲区列表的命令。例如, :all 命令为每个命令行参数创建窗口。详见 *vim* 联机帮助。上表只列出了最有用的命令。

由于 *vim* 拥有大多数 *ex* 命令, 因此它也拥有大部分的 *vi* 模式命令, 如表 11-2 所示。与大多数 *vi* 命令一样, 可以用一个计数给很多窗口命令加上前缀。

表 11-2 vi 模式中的 vim 窗口命令

命令	功能
<code>^W s</code>	与没有 <i>file</i> 参数的 :split 命令相同。 <code>^W ^S</code> 可能不适用于所有的终端
<code>^W S</code>	
<code>^W ^S</code>	
<code>^W n</code>	与没有参数 <i>file</i> 的 :new 命令相同
<code>^W ^N</code>	
<code>^W ^</code>	运行 :split #, 分割窗口并编辑替换文件。如给出数字, 则编辑第 <i>n</i> 个缓冲区
<code>^W ^ ^</code>	
<code>^W q</code>	与 :quit 命令相同。 <code>^W ^Q</code> 不一定适用于所有的终端
<code>^W ^Q</code>	
<code>^W c</code>	与 :close 命令相同
<code>^W o</code>	类似于 :only 命令
<code>^W ^O</code>	
<code>^W <DOWN></code>	将光标切换到当前窗口之下的第 <i>n</i> 个窗口。 <i>n</i> 作为前缀参数提供
<code>^W ^J</code>	
<code>^W j</code>	
<code>^W <UP></code>	将光标切换到当前窗口之上的第 <i>n</i> 个窗口。 <i>n</i> 作为前缀参数提供
<code>^W ^K</code>	
<code>^W k</code>	
<code>^W w</code>	如果提供计数, 则切换到第 <i>n</i> 个窗口; 否则切换到当前窗口的下一个窗口。如果位于底端窗口, 就切换到顶端窗口
<code>^W ^W</code>	
<code>^W ^W</code>	如果提供计数, 则切换到第 <i>n</i> 个窗口; 否则切换到当前窗口的上一个窗口。如果位于顶端窗口, 就切换到底端窗口
<code>^W t</code>	将光标切换到顶端窗口
<code>^W ^T</code>	

表 11-2 vi 模式中的 vim 窗口命令（续）

命令	功能
<code>^W b</code>	将光标切换到底端窗口
<code>^W ^B</code>	
<code>^W p</code>	切换到最近访问（前一个）的窗口
<code>^W ^P</code>	
<code>^W r</code>	向下循环移动所有窗口，光标停留在同一窗口内
<code>^W ^R</code>	
<code>^W R</code>	向上循环移动所有窗口，光标停留在同一窗口内
<code>^W x</code>	如没有给出计数，则将当前窗口和下一个窗口交换。如果没有下一个窗口，则和前一个窗口交换。如果给出了计数，将当前窗口和第 <i>n</i> 个窗口交换（开始窗口为 1）。光标切换到另一个窗口
<code>^W =</code>	使所有窗口具有相同的高度
<code>^W -</code>	减少当前窗口的高度，前面计数表明减少量
<code>^W +</code>	增加当前窗口的高度，前面计数表明增加量
<code>^W _</code>	将当前窗口设成前面计数给定的值的大小，与带有绝对计数作参数的 :resize 相似。如没有计数，窗口将变得尽可能大
<code>^W ^_</code>	
<code>zN [RETURN]</code>	将当前窗口的高度设成 <i>N</i>
<code>^W]</code>	分割当前窗口。在上面的新窗口中，用光标上的标识符作为标志并跳到该处。前面计数表示新窗口的大小
<code>^W ^]</code>	
<code>^W f</code>	分割当前窗口，在新窗口里编辑名字在光标下的文件。该命令进行了相当复杂的文件查找，详见 :help ^W_f
<code>^W ^F</code>	
<code>^W i</code>	打开并进入新窗口。将光标移动到与光标下关键字相匹配的第一行上。
<code>^W ^I</code>	从文件开始处进行查找并忽略类似注释的行，如果给出前面计数，就跳到第 <i>n</i> 个匹配行，不忽略注释
<code>^W d</code>	打开一个新窗口，光标出现在包含光标下关键字的第一个宏定义行，从文件开始处进行查找。如果给出前面计数，将跳到相应的匹配行
<code>^W ^D</code>	

其他说明：由于 `^C` 通常是一个中断符，它结束取消命令，因此 `^W ^C` 不关闭当前窗口。如果启动了鼠标支持并且正在使用 *vim* 的 GUI 版，那么就可以使用鼠标

拖住窗口的状态行来重新设定窗口的大小。最后，*vim*有许多控制各种命令行为的选项，尤其是调试hidden、splitbelow、equalalways、winheight、cmdheight选项。详细内容请参考联机帮助。

GUI 接口

本节的屏幕图像和解释是由 Bram Moolenaar 提供的，我们对他表示感谢。

vim 的 GUI 通过 Athena 和 Motif 接口可用于 UNIX、Windows 95 和 Windows NT 及 BeOS 上。用于 Amiga、VMS 和 Macintosh 的 GUI 版正在开发。屏幕图像如图 11-1 所示。

GUI 版本的主要优点是可以使用全部的颜色，不会出现很多终端仿真程序所存在的颜色配置问题。图片显示的是 Motif 版本。你不能在单色图片上看到在文本中用来高亮显示条目的各种颜色。例如注释是蓝色的，字符串是深红色的。

GUI 窗口在顶部有一个菜单，每个窗口的右边有个滚动条，底部有个水平滚动的滚动条。滚动条不仅方便了浏览文件，而且它们也指示出在文件中的当前位置。

光标在底行的 /free 之后，/free 是正在输入的搜索命令。光标是加亮绿色并不断闪烁，这使它在有颜色的文本中很容易找到。使用 guicursor 选项配置颜色和闪烁。在插入模式下，光标变成竖条状。在替换模式下，它变成一半高度的光标。这样就很容易确定当前的模式。

顶部窗口包含一个 shell 脚本，中间窗口包含一个 *makefile*，下部窗口包含一个 C 程序。当打开文件时，它们都会自动高亮显示。这是 *vim* 版支持的大约 70 种语法中的 3 种（参考“语法高亮显示”一节）。

在文本中 *to* 的所有出现都使用黄色的背景高亮显示，这是 hlsearch 选项的作用。它显示的是上一次使用的搜索模式的匹配。当在源代码中查找何处使用了某个变量时，该功能非常有用。它可以很容易地找到所有的匹配，并使用 n 命令将跳到下一个匹配处。

The screenshot shows the vim GUI window displaying the source code of the Vim configuration script. The window has a menu bar with File, Edit, Window, IDE, Syntax, Words, L, and Help. The main area contains the following code:

```
#!/bin/sh

# Guess values for system-dependent variables and create Makefiles.
# Generated automatically using autoconf version 2.12
# Copyright (C) 1992, 93, 94, 95, 96 Free Software Foundation, Inc.
#
# This configure script is free software; the Free Software Foundation
# gives unlimited permission to copy, distribute and modify it.

# Defaults:
ac_help=
ac_default_prefix=/usr/local
# Any additions from configure.in:
ac_help="$ac_help
--enable-perlinterp      Include Perl interpreter."
ac_help="$ac_help
configure
ALL_GUI_PRO = gui.pro gui_motif.pro gui_athena.pro gui_x11.pro gui_w32.pro gui_a
### our grand parent directory should know who we are...
### only used for "make tar"
VIMVERSION = `eval "basename \`cd ..;/ pwd\``

### Command to create dependencies based on #include "..."
### prototype headers are ignored due to -DPROTO, system
### headers #include <...> are ignored if we use the -MM option, as
### e.g. provided by gcc-cpp.
### Include USE_GUI to get dependency on gui.h
CPP_DEPEND = $(CC) -M$(CPP_MM) $(DEPEND_CFLAGS)

# flags for cproto
#     __inline and __attribute__ are not recognized by cproto
#     maybe the "/usr/bin/cc -E" has to be adjusted for some systems

Makefile
/* vi:set ts=8 sts=4 sw=4:
 *
 * VIM - Vi IMproved    by Bram Moolenaar
 *
 * Do ":help uganda" in Vim to read copying and usage conditions.
 * Do ":help credits" in Vim to see a list of people who contributed.
 */
#define EXTERN
#include "vim.h"

#endif SPAWN
#include <spawn.h>          /* special MSDOS swapping library */
#endif

static void mainerr ARGS((int, char u *));
main.c
/free
```

图 11-1 vim 的 GUI 窗口

顶部窗口中的反白显示*free*是正在命令行输入的搜索模式的当前匹配项，这是insearch选项的作用。每次为模式输入一个字符，都将调整匹配。必要时，为了显示下一个匹配，文件可以进行滚动。这样可以直接看到搜索命令的下一个匹配位置，并且调整模式直到到达了相应的位置。这在模式中寻找输入错误时非常有用。

在这张图中看不到文件浏览器和对话框，它们会在从菜单中选择命令的时候用到。这是*vim* 5.2 版的一项新功能（注 1）。例如，File/Save as 菜单将弹出一个文件浏览器，在浏览器中可以选择要写入的文件的名字。如果文件已经存在，将弹出一个对话框，询问是否覆盖原文件。这比使用！跳过错误消息要好得多。缺点是不得不使用鼠标，并在键盘和鼠标之间不断地换手。如果不想这样，则不要使用菜单。可以通过删除guioptions 选项中的m标记来禁用菜单（并为其他的文件腾出了空间）。

vim 的一项很好的特性是几乎一切的功能都可以进行配置，同时也包括菜单。如果你不喜欢系统提供的菜单，则可以定义自己的菜单。这与定义映射很相似。例如，为了增加一个 IDE/Make -n 菜单，可以执行:make -n 命令：

```
:amenu IDE.Make-n :make -n<CR>
```

为了使菜单名字上有点和空格，可以在它前面加上反斜杠。要得到同一菜单条目，就要在 -n 前面加一空格：

```
amenu IDE.Make\ -n :make -n<CR>
```

在参数中不需要反斜杠，它只出现在菜单名字中。所有这些内容和功能构成了一个良好的 GUI 环境，同时所有好的原有 vi 命令仍然可以像在终端版本中那样使用。

联机帮助详细地介绍了所有的 GUI 选项和如何创建定制的菜单。

注 1： 在本书出版时 5.2 版本正在测试发布。A.R.

扩展的正则表达式

在所有的克隆版本中，*vim* 提供了最丰富的正则表达式的匹配功能集。下面的大部分描述文本都摘自 *vim* 文档：

\| 表示或， house \| home。

\+ 匹配一个或多个前面的正则表达式。

\= 匹配零个或一个前面的正则表达式。

\{n, m}

尽可能多地匹配 *n* 到 *m* 个前面的正则表达式，*n* 和 *m* 是 0 到 32 000 之间的数。

vim 要求只有左大括号前面有反斜杠，而不要求右大括号。

\{n}

匹配 *n* 个前面的正则表达式。

\{n, }

尽可能多地匹配至少 *n* 个前面的正则表达式。

\{ , m}

尽可能多地匹配 0 到 *m* 个前面的正则表达式。

\{ }

尽可能多地匹配零个或多个前面的正则表达式（与 * 相同）。

\{ -n, m}

尽可能少地匹配 *n* 到 *m* 个前面的正则表达式。

\{ -n}

匹配 *n* 个前面的正则表达式。

\{ -n, }

尽可能少地匹配至少 *n* 个前面的正则表达式。

\{ - , m}

尽可能少地匹配 *n* 到 *m* 个前面的正则表达式。

- \i 匹配任何由 `isident` 选项定义的标识符字符。
- \I 与 \i 相似，但不包括数字。
- \k 匹配任何由 `iskeyword` 选项定义的关键字字符。
- \K 和 \k 相似，但不包括数字。
- \f 匹配任何由 `isfname` 选项定义的文件名字符。
- \F 和 \f 相似，但不包括数字。
- \p 匹配任何由 `isprint` 选项定义的可打印字符。
- \P 和 \p 相似，但不包括数字。
- \s 匹配任何空白字符（指空格和制表符）。
- \S 匹配任何非空格和制表符。
- \b 退格。
- \e 转义。
- \r 回车。
- \t 制表符。
- \n 保留以备将来使用。实际上，它将用来进行多行模式匹配。详见 *vim* 文档。
- \~ 匹配最近给定的替换（即代替）字符串。
- \{\dots\}

提供用于 *、\+、和 \= 的分组，同时在替换命令的替换部分使匹配的子文本可用 (\1、\2 等)。

- \1 匹配 \(\) 和 \) 中的第一个子表达式所匹配的相同字符串。例如，\([a-z]\)\.\1 匹配 *ata*、*ehe*、*tot* 等等。 \2、\3 等可以用来代表第二个、第三个等子表达式。

`isident`、`iskeyword`、`isfname` 和 `isprint` 选项定义了在标识符、关键字和文件名中出现的字符及那些可打印的字符。使用这些选项可以使正则表达式匹配更加灵活。

改进的编辑功能

本节介绍了 *vim* 可使简单的文本编辑更容易、更有效的功能。

命令行历史和完整化

vim 在它的扩展命令语言中保留用户的 *ex* 命令、搜索字符串和表达式的历史记录。它们是三个单独的历史记录。历史记录的大小由 `history` 选项控制，默认值是 20。虽然 *vim* 只维护一个命令就需要许多步骤，但你可以在 `.vimrc` 文件中增加它的值。

如果要访问历史记录，可以在冒号命令行上使用 ↑ 光标键。这将回到已保存的命令（最近的优先）。↓ 键将向前移动。可以使用 ← 和 → 键在命令行上移动。默认情况下，输入的文本将插入到命令行中。可以使用键盘上的 **[INS]**（插入）键关闭该模式，在这种情况下，输入的文本将替换命令行中的原有内容。**[BACKSPACE]** 键用来删除字符。

可以使用 **[SHIFT]** 或 **[CTRL]** 键结合 ← 和 → 键，从而使光标向左或向右一次一个单词地移动。但这或许不能在所有的键盘上都适用。可以使用 ^B 或 **[HOME]** 键将光标移动到命令行的开始处，也可以使用 ^E 或 **[END]** 键将光标移动到命令行的结尾处。控制键功能总是有效。

可以改变 **[ESC]** 键的行为。如果 *vim* 在 *vi* 兼容模式下，**[ESC]** 键的行为与 **[RETURN]** 键相似，都用来执行命令。当关闭 *vi* 兼容模式后，**[ESC]** 将退出命令行而不执行任何操作。

vim 还提供了在 *ex* 命令行上的完整化功能。当要使 *vim* 进行完整化操作时，`wildchar` 选项包含了输入的字符。默认值是制表符，可以对下面的内容进行完整化：

命令名字

可用于命令行的开始。

标志值

在输入:`tag`之后

文件名

当输入带有文件名参数的命令时。在文件名完整化期间，如果有多个文件名匹配一个模式，那么为了选出`vim`真正要用的，`suffixes`选项的值将在它们中间设置优先级（具体细节参见：`:help suffixes`）。

选项值

在输入:`:set`命令时，它有两项功能：当输入选项本身的名字时，按[TAB]键就会把选项名补充完整。接下来可以输入=符号并再次按[TAB]键，`vim`就会填写变量的当前值。

除了对[TAB]键进行扩展外，许多其他的控制键也提供了追加功能。表11-3描述了这些命令以及它们的功能。

表 11-3 vim 的命令行完整化命令

命令	功能
<code>^D</code>	列出与模式匹配的名字。对应的文件名、目录将高亮显示
<code>wildchar</code> 的值	(默认值是制表符)完成一次匹配、插入已有的文件，如果有多个匹配将插入第一个。按[TAB]会在其他匹配中连续循环
<code>^N</code>	如果还有匹配，到多重 <code>wildchar</code> 匹配的下一个；否则，调用最近的历史行
<code>^P</code>	如果还有匹配，到多重 <code>wildchar</code> 匹配的上一个；否则，调用较老的历史行
<code>^A</code>	插入所有与模式相匹配的名字
<code>^L</code>	如果有确切匹配，就插入它；否则，就扩展成多重匹配的最长公共前缀

完整化功能很广泛，详细内容请参考`:help cmdline`。除了命令行完整化外，`vim`还提供插入模式完整化。

当输入文本，尤其是在程序中输入文本时，经常会出现相同的单词。*vim* 提供向前或向后查找匹配半完成单词的命令。例如，如果你正在输入本文并且已输入了 *ex*，使用 ^P 命令可以把它补充为 *example*。这是减少输入字符数量和避免拼写错误的好方法。

完整化不仅对输入文本中的单词起作用，也可以从更远的位置获取单词。表 11-4 列出了相关命令的概述。

表 11-4 vim 插入模式的完整化命令

命令	功能
^N	从当前缓冲区中把单词补充完整，向前查找（帮助记忆：next）
^P	从当前缓冲区中把单词补充完整，向后查找（帮助记忆：previous）
^X ^K	从字典中把单词补充完整
^X ^I	从头文件中把单词补充完整
^X ^D	从头文件中把宏（定义过的单词）补充完整
^X ^]	从标志文件中把单词补充完整
^X ^F	把文件名补充完整
^X ^L	从当前缓冲区中把一行补充完整

详细信息参见：`:help ins-completion`

标志栈

标志栈已在第八章中的“标志栈”一节进行了介绍。*vim* 提供了最丰富的工具集与标志一起工作。除了标志入栈的功能外，如果有多个可匹配的标志，还可以在其中进行选择。也可以在一条命令中完成选择标志和分割窗口的操作。参考表 11-5 中 *vim* 的标志命令列表。

表 11-5 vim 的标志命令

命令	功能
ta[g][!] [tagstring]	编辑在tags文件中定义的包含tagstring的文件。如果当前缓冲区已修改但没有保存, !将强迫把vim切换到新文件上。这个文件是否可写取决于autowrite选项的设置
[count]ta[g][!]	跳到标志栈中第count个较新的入口
[count]po[p][!]	将光标位置弹出栈, 把光标恢复到它以前的位置。如果给出了count, 就跳到第count个原来的人口
tags	显示标志栈的内容
ts[elect][!] [tagstring]	使用标志文件中的信息列出与tagstring相匹配的标志。如果没有指定tagstring, 将使用标志栈中上次使用的标志名
sts[elect][!] [tagstring]	除了为选中的标志分割窗口外, 其余与:tselect相似
[count]tn[ext][!]	跳到第count个下一个相匹配的标志(默认值为1)
[count]tp[revious][!]	跳到第count个上一个相匹配的标志(默认值为1)
[count]tN[ext][!]	
[count]tr[ewind][!]	跳到第一个相匹配的标志。如果指定了count, 就跳到第count个相匹配的标志
tl[ast][!]	跳到最后一个相匹配的标志

通常, vim 显示已跳过的匹配标志和所有匹配的数量:

```
tag 1 of >3
```

它使用大于号表示还没有发现所有的匹配。可以使用:tnext 和:tlast 来进行更多的匹配。如果由于其他的消息而不能显示该消息, 可使用:0tn 命令查看它。

:tags 命令的输出如下所示。当前位置标有一个大于号(>):

```
# TO tag      FROM line in file
1 1 main          1  harddisk2: text/vim/test
```

```
> 2 2 FuncA          58 -current-
3 1 FuncC          357 harddisk2: text/vim/src/amiga.c
```

:tselect命令允许在多于一个的匹配标志中进行选择，“优先级”(pri域)指示匹配的性质(全局与静态，确定条件与不相关条件等等)，这将在*vim*文档中有更详尽的介绍。

```
nr pri kind tag          file ~
1 F   f   mch_delay      os_amiga.c
      mch_delay(msc, ignoreinput)
> 2 F   f   mch_delay      os_msdos.c
      mch_delay(msc, ignoreinput)
3 F   f   mch_delay      os_unix.c
      mch_delay(msc, ignoreinput)
Enter nr of choice (<CR> to abort):
```

:tag和:tselect命令可以带有以/开头的参数。这种情况下，将该参数看成一个正则表达式。*vim*会查找所有与给定正则表达式相匹配的标志(注2)。例如，:tag /normal将查找NORMAL宏、normal_cmd函数等等。使用:tselect /normal并输入相应的标志的序号。

表11-6对*vi*命令模式的命令进行了描述。与其他的编辑器一样，除了可以使用键盘以外，如果*vim*版本支持鼠标，则还可以用鼠标。

表11-6 vim命令模式的标志命令

命令	功能
^]	在tags文件中查找光标处标识符的位置，并移动到该位置。当前位置自动被压入标志栈
g <LeftMouse>	
CTRL-<LeftMouse>	
^T	返回标志栈中的上一位置，即弹出一个元素。前面的计数指定了弹出元素的个数

表11-7对影响标志搜索的*vim*选项进行了描述。

注2：在5.1版以前，*vim*根据有或是没有特殊字符来把:tag或:tselect参数当成正则表达式处理。/的使用消除了这个过程的二义性。

表 11-7 用于标志管理的 vim 选项

命令	功能
taglength, tl tags	控制要查找的标志中有效的字符个数。默认值0表示所有字符都有效 该值是用于搜索标志的文件名列表。特殊情况下，如果文件名以 ./ 开头，就使用当前文件的路径名的目录部分代替点，以便使用不同目录下的 tags 文件。默认值是 “./tags, tags”
tagrelative	当设为真（默认）且使用另一个目录的 tags 文件时，认为该 tags 文件中的文件名是相对于该 tags 文件所在的目录的

vim 5.1 版随着 Exuberant *ctags* 程序的 2.0.3 版一起发布。在编写本节的时候，使用的是 Exuberant *ctags* 程序的最新版本。

虽然 *vim* 可使用 *emacs* 样式的 *etags* 文件，但是这仅仅保持了向下兼容；该格式并没有记录在 *vim* 文档中，也不提倡使用 *etags* 文件。

最后，类似于 *elvis*，*vim* 也查找包含光标的整个单词，而不是从光标位置向前的部分单词。

无限撤消

在 *vim* 中，通过 *undolevels* 选项可以对多级修改进行撤消和恢复。该选项是一个数字，表明 *vim* 允许的“撤消”次数。负值将禁止任何撤消操作（这很少使用）。

当 *undolevels* 被设置为非零值时，可以像通常一样输入文本。随后的每个 *u* 命令将撤消一次修改。可以使用（相当好记的）**CTRL-R** 命令进行恢复（撤消撤消操作）。

vim 与 *elvis* 不同，它在启动时用于 *undolevels* 的默认值是 1000，对于任意给定的编辑会话，这几乎就是无限了。而且，该选项是全局的，不是针对每个缓冲区的。

一旦设定了 *undolevels*，*u* 命令或 **^R** 命令就只能进行给定次数的撤消或恢复操作。

vim 实际上以两种不同的方式实现撤消和恢复操作。当 cpoptions (兼容性选项) 含有字母 u 时, u 命令像在 *vi* 中一样工作, ^R 重复上一个操作 (和 *nvi* 中的 . 命令相似)。当 cpoptions 选项中没有 u 时, u 撤消一次操作, ^R 恢复一次操作。这使用起来很容易, 但与 *vi* 不兼容。

任意长度的行和二进制数据

vim 对行数或行的长度没有限制。在编辑二进制文件时, 或者使用 -b 命令行选项或者使用 :set binary 命令。这样就设置了 *vim* 的其他几个选项, 从而使二进制文件编辑起来更加容易。为了输入 8 位文本, 可在输入 ^V 命令后输入三个十进制数。

增量搜索

正如第八章中的“增量搜索”一节所提到的, 在 *vim* 中可以使用 :set incsearch 来启动增量搜索。

光标随着输入在文件中移动, *vim* 高亮显示与当前输入相匹配的文本。

你可能带 hlsearch 选项来使用这个功能, 它将会高亮显示最近搜索模式的所有匹配。该选项在程序源代码中查找特定变量或函数的所有应用时尤其有用。

左右滚动

正如第八章中的“左右移动”一节所提到的那样, 在 *vim* 中使用 :set nowrap 来启动左右滚动。sidescroll 的值控制着 *vim* 从左向右滚动时屏幕移动的字符数目。当 sidescroll 设为零时, 每次滚动将会把光标放置到屏幕的中间; 否则, 屏幕将按预期的字符数目滚动。

vim 也有几个向两边滚动窗口的命令, 如表 11-8 所示。

表 11-8 vim 向两边滚动的命令

命令	功能
z1	将窗口左滚
zh	将窗口右滚
zs	将窗口滚动使光标在屏幕的左边（开始）
ze	将窗口滚动使光标在屏幕的右边（结束）

可视模式

vim 允许你使用表 11-9 中所描述的命令选择区域，一次一个字符、一行或一个矩形。

表 11-9 vim 的块模式命令字符

命令	功能
v	启动区域选择，一次选中一个字符模式
V	启动区域选择，一次选中一行模式
^V	启动区域选择，一次选中一个矩形模式

vim 会高亮显示（使用反白）正在选择的文本。只使用通常的移动键就可以进行选择。如果设置了 `showmode`, *vim* 就会指示该模式为可视字符、可视行、可视块中的一种。如果 *vim* 正在 *xterm* 中运行，那么也可以使用鼠标来选择文本（详细信息参考:`:help mouse-using`)。这种处理也可以在 GUI 版本中使用。下面的屏幕显示了一个矩形区域：

```
The 6th edition of <citetitle>Learning the vi Editor</citetitle>
brings the book into the late 1990's.
In particular, besides the &ldquo;original&rdquo; version of
<command>vi</command> that comes as a standard part of every UNIX
system, there are now a number of freely available &ldquo;clones&rdquo;
or work-alike editors.
```

在使用 `~` 操作符之后，屏幕将如下所示：

The 6th edition of <citetitle>Learning the vi Editor</citetitle> brings the BOOK INTO THE LATE 1990's. In particular, besides the "original" version of <command>vi</COMMAND> THAT COMES as a standard part of every UNIX system, there are now a number of freely available "clones" or work-alike editors.

vim 允许对选中的文本进行多种操作。即使已选定的区域不包括完整的行，有些操作也会作用于整行。

vim 有很多专门处理逐渐增多的“已清除”区域的命令，并且允许对高亮显示的文本使用几乎所有的 *vi* 模式命令，以及一些专门用于可视模式的命令。

在定义要操作的区域时，许多命令可以很容易地把单词、句子或 C/C++ 代码块作为单独对象进行处理。表 11-10 对这些命令进行了描述。这些命令可以使用自身来扩大使用区域或与运算符结合使用。例如，*daB* 删除大括号所包含的文本块，包括大括号在内。

表 11-10 vim 的块模式对象选择命令

命令	选择
aw	单词（有空格）
iw	内部单词（没有空格）
aW	大写单词（WORD）（有空格）
iw	内部大写单词（没有空格）
as	句子（有空格）
is	内部句子（没有空格）
ap	段落（有空格）
ip	内部段落（没有空格）
ab	(...) 块（包括圆括号）
ib	内部(...) 块（不包括圆括号）
aB	(...) 块（包括大括号）
iB	内部(...) 块（不包括大括号）

术语“word”和“WORD”与w和W移动命令有相同的含义。

vim 允许对高亮显示的文本进行多种操作。表 11-11 对可用的操作符进行了总结。

表 11-11 vim 的块模式操作符

命令	操作
~	改变选中文本的大小写类型
o, O	移动到高亮显示文本的另一端。o从高亮显示区域的开始移到末尾，反之亦然。在块模式下，O移动到在当前行中文本的另一端。可以从新位置开始继续扫描其他区域
<, >, !	左移或右移文本，过滤文本。它对包含已标记区域的整行进行操作，以后如果对块操作，则只能移动块
=	通过equalprg选项命名的程序过滤文本(通常是简单的文本格式化程序，如fmt)。它对包含已标记区域的整行进行操作
gq	格式化包含已标记区域的行，使之不再设为textwidth的值。它对包含已标记区域的整行进行操作
:	启动作用于高亮显示的行的ex命令。它对包含已标记区域的整行进行操作
c, d, y	修改、删除或复制文本。虽然c命令只是在块的第一行输入文本，但这些命令都可以对矩形文本进行操作
c, r, s	修改高亮显示的文本
C, S, R	如果使用[CTRL-V]，将删除矩形并在第一行进入插入模式。否则，就会替换整行
x	删除高亮显示的文本
X, Y	删除或复制包含高亮显示区域的整行
D	删除到行尾。当使用[CTRL-V]时，高亮显示的块和到每行末尾的其余文本都将被删除。如果不使用[CTRL-V]，将删除整行
J	合并已高亮显示的行，它对包含已标记区域的整行进行操作
U	生成大写字母，该命令只对显示模式有效
u	生成小写字母，该命令只对显示模式有效
^]	在标志搜索中把高亮显示的文本作为标志进行查找

编程辅助

无论对于编辑－编译－调试过程循环还是对于语法高亮显示, *vim*都有很强的功能。

编辑－编译加速

vim 的这项功能是根据用于 Amiga 的 Manx Azte C 编译器的“快速修补”模式而产生的。实际上, *vim* 文档也把该功能称为“快速修补”模式。该功能非常灵活, 允许你根据编程环境对它们进行处理 (见表 11-12)。

表 11-12 vim 的程序开发命令

命令	功能
mak[e] [arguments]	根据下面描述的一系列选项的设置运行 <i>make</i> , 然后移动到第一个错误处
cf[file] [!] [errorfile]	读取错误文件并跳到第一个错误处。如果提供了 <i>errorfile</i> , 使用它来存放错误并把 <i>errorfile</i> 选项设置为该文件。如果当前缓冲区已被修改但没有保存, ! 强迫 <i>vim</i> 移动到另一个缓冲区
cl[ist] [!] [count]cn[ext] [!]	列出含有文件名的错误。如果给出!, 则列出所有的错误 显示后面第 <i>count</i> 个包含文件名的错误。如果根本不存在该文件名, 则移到后面第 <i>count</i> 个错误处
[count]cN[ext] [!] [count]cp[revious] [!]	显示前面第 <i>count</i> 个包含文件名的错误。如果根本不存在该文件名, 移到前面第 <i>count</i> 个错误处
clast[!] [n]	如果给出了 <i>n</i> , 则显示错误 <i>n</i> ; 否则, 显示最后一个错误
crewind[!] [n]	如果给出了 <i>n</i> , 则显示错误 <i>n</i> ; 否则, 显示第一个错误
cc[!] [n]	如果给出了 <i>n</i> , 则显示错误 <i>n</i> ; 否则, 重新显示当前错误
cq[uit]	带错误代码退出, 使编译器不再编译同一个文件。这主要用于 Amiga 编译器

与 *elvis* 一样, 当移动到错误处时, *vim* 也在文件中提供修改的补偿功能, 以便移动到下一个错误时停在正确的行上。

表 11-13 列出了 *vim* 用来控制 :make 命令的选项。

表 11-13 vim 的程序开发选项

选项	值	功能
shell	/bin/sh	用来执行重构用户程序命令的 shell
makeprg	make	真正处理所有重新编译的程序
shellpipe	2>&1 tee	如果使 shell 将编译的标准输出和标准错误保存到错误文件中，则必须也要应用该选项
makeef	/tmp/vim##.err	包含编译器输出的文件的名字。## 使 vim 创建唯一的文件名
errorformat	%f:%l:\%m	对编译器中错误消息的外观描述。实例值用于 GCC (CNU 的 C 编译器)

在执行 :make 时，*vim* 通过组合上面描述的各种条目构造一个命令。你提供的任何参数都在合适的位置传递给 *make*，然后将该命令人回显在屏幕上。例如，如果输入 :make -k，那么可能会看到如下内容：

```
:!make -k 2>&1 | tee /tmp/vim34215.err
...
```

通过使用 *tee*(1) 程序，可以把 *make* 和编译器的输出保存到错误文件中 (*/tmp/vim34215.err*)，同时也将其发送到标准输出设备，在本例中就是你的显示器。

当 *make* 结束时，*vim* 读取错误文件，并到达第一个错误处。为了寻找文件名和行号，*vim* 使用 errorformat 选项的值分析错误文件的内容（该变量的格式在 :help errorformat 中有详细描述）。接下来可以使用 :cc 命令来查看错误消息，并使用 :cnext 命令移动到下一个错误处。

语法高亮显示

在 *vim* 中，语法高亮显示主要靠颜色的不同。为了启动语法高亮显示，把 syntax on 放到 .vimrc 文件中，这将使 *vim* 读取 syntax.vim 文件，该文件定义了默认的高亮显示颜色，然后设定适合于每种语言的高亮显示的相关内容。

*vim*有非常强大的用于定义语法高亮显示的子语言。在*vim* 5.1版中，*syntax.txt*帮助文件对它的描述超过了1500行。因此，我们不想在这里给出所有的细节。但是，下面的样本文件应该使你对*vim*有些概念。该例子包含了用于Awk的语法文件的几部分：

```
* Vim syntax file
" Language: awk, nawk, gawk, mawk
" Maintainer: Antonio Colombo <antonio.colombo@jrc.org>
" Last change: 1997 November 29

" Remove any old syntax stuff hanging around
syn clear

" A bunch of useful Awk keywords
syn keyword awkStatement      break continue delete exit
...
syn keyword awkFunction       atan2 close cos exp int log rand sin \e
                             sqrt srand
...
syn keyword awkConditional    if else
syn keyword awkRepeat         while for do
...
syn keyword awkPatterns       BEGIN END
syn keyword awkVariables      ARGC ARGV FILENAME FNR FS NF NR
...
" Octal format character.
syn match  awkSpecialCharacter contained "\\\\[0-7]\\{1,3\\}"
" Hex format character.
syn match  awkSpecialCharacter contained "\\x[0-9A-Fa-f]\\+"
...
syn match  awkFieldVars       "\\$[0-9]\\+"
...
syn match  awkCharClass       contained "\\[:[^:\\]]*:\\]"
syn match  awkRegExp          contained "/\\^\"ms=s+1
syn match  awkRegExp          contained '\\$/\"me=e-1
syn match  awkRegExp          contained "[?.*{}\\]+"
...
" Numbers, allowing signs (both -, and +)
" Integer number.
```

```

syn match awkNumber           "[+-]\-\<[0-9]\+\>"           " Floating point number.
syn match awkFloat            "[+-]\=\<[0-9]\+\.\[0-9]\+\>"      ...
...
syn match awkComment          "#.*" contains=awkTodo

if !exists("did_awk_syntax_inits")
let did_awk_syntax_inits = 1
" The default methods for highlighting. Can be overridden later
hi link awkConditional       Conditional
hi link awkFunction          Function
hi link awkRepeat             Repeat
hi link awkStatement          Statement
...
hi link awkNumber             Number
hi link awkFloat              Float
...
...
hi link awkComment            Comment
...
endif

let b:current_syntax = "awk"

```

上面的文件使用 syntax keyword 给某些关键字的类命名（例如真实的 Awk 关键字和内置函数），使用 syntax match 给正则表达式所匹配的某些对象类型（例如数字）命名。然后 hi link 语句把已命名的对象类与预先定义的高亮显示标准联系起来。

Syntax.vim 文件预先定义了标准的规范，其中的几行如下所示：

```

hi Comment    term=bold ctermfg=Cyan guifg=#80a0ff
hi Constant   term=underline ctermfg=Magenta guifg=#ffa0a0
hi Special    term=bold ctermfg=LightRed guifg=Orange
hi Identifier term=underline ctermfg=DarkCyan guifg=#40ffff
...

```

第一个参数定义了类，其余的则定义了与终端类型相对应的高亮显示类型。term 用于普通终端，cterm 用于彩色终端（在本例中即 ForeGround 颜色），gui 用于 vim 的 GUI 接口。

在 *vim* 中，语法颜色是全局性的。改变 `comment` 的颜色就会改变所有窗口中所有注释的颜色而无论你正在编辑哪种编程语言。

由于语法描述使用属性连接，因此可以对特定语言进行修改。例如，为了改变用于 Awk 的注释的颜色，可以定义 `awkComment` 的属性，如下所示：

```
hi awkComment guifg=Green
```

vim 为不同的语言提供了大量的语法描述。虽然 Awk 的着色有点过于鲜艳（很多红色和品红色），但用于区分文本的颜色还是相当友好的，UNIX邮箱文件的颜色方案也不错，HTML 模式也很吸引人。

令人感兴趣的功能

vim 是一个非常有特色的编辑器。我们在这里没有进行详细描述，只是选择了几个最重要且特有的功能进行讨论：

自动文件类型检测

vim 会注意文本文件的行的结束方式。它把变量 `fileformat` 设置为 `dos` (`CR-LF`)、`unix` (`LF`) 或 `mac` (`CR`) 中的一个来指示文件的当前模式。默认情况下，*vim* 会以同样的格式回写文件，但是如果改变了 `fileformat` 的值，*vim* 就会使用约定的格式。这是一种易于在 Linux (或 UNIX) 文件和 MS-DOS 文件之间进行转换的方法，并使在 UNIX 或 Linux 环境下编辑 DOS 文件变得容易（相比而言，其他的克隆版本都在每行末尾显示 `^M`）。

vim 是“慈善软件”

授权条款在本章的后面描述，它们是相当自由的。然而，作者鼓励喜欢 *vim* 的用户对乌干达的儿童中心捐款。

重要的 C 编程扩展

vim 有一整套用来处理 C 或 C++ 程序的功能。

“自动命令”功能

vim 定义了很多事件，如读取文件之前或之后、进入或离开窗口等等。对于每个事件，都可以建立一个“自动命令”，即当该事件发生时执行命令。

vim 是慈善软件

Bram Moolenaar 对 *vim* 采取了与通常共享软件或免费软件的作者不同的做法。如果你使用 *vim* 并喜欢它，Moolenaar 先生将要求你捐款帮助在乌干达的孤儿，我们为他的努力喝彩。

Moolenaar 先生在 Kibaale 的 Kibaale 儿童中心 (KCC) 当了一年的志愿者，Kibaale 是乌干达南部的一个小镇，靠近坦桑尼亚。KCC 的工作是为该地方的孩子提供食品、医疗和教育。这个地方受爱滋病的危害超过世界上任何一个地方。因为爱滋病的高发率，很多孩子都成了孤儿。

为了继续支持 KCC，Moolenaar 先生计划发起一个基金会并组织募捐。你可以在 *vim* 发行版本的 *uganda.txt* 文件中发现一个相当长的说明，其中包括邮送捐款指南。你也可以查看 <http://www.vim.org/iccif/>。

C 和 C++ 编程功能

在 *vi* 最主要的继承版本中，*vim* 是第一个而且是最重要的程序员编辑器，尤其是 C 程序员的编辑器。而且 C++ 编程人员也可以使用它。*vim* 有很多方便 C 程序员工作的功能。我们在这里介绍一些最重要的功能。

智能缩进

vi 的所有版本都有 autoindent 选项。当设置它时，当前行就会自动缩进与它相邻行同样的大小。这对于想对代码进行缩进的 C 程序员、对于任何需要通过缩进而在文本中指示某种类型的结构的人都是很便捷的。

vim 增强了这项功能，它有另外两个选项，`smartindent` 和 `cindent`。选项 `cindent` 在二者中更为重要，也是本节的主题。表 11-14 列出了 *vim* 的缩进和格式化选项。

表 11-14 *vim* 的缩进和格式化选项

选项	功能
<code>autoindent</code>	简单缩进。使用前一行的缩进格式
<code>smartindent</code>	除了识别一些 C 语法外，与 <code>autoindent</code> 相似。不如 <code>cindent</code> 好
<code>cindent</code>	启动用于 C 程序的自动缩进，并且相当灵活。C 格式受本表中其他选项的影响
<code>cinkeys</code>	输入触发缩进选项的键
<code>cinoptions</code>	允许修改已经指定的缩进样式
<code>cinwords</code>	表示在下面的行中进行额外缩进的关键字
<code>formatoptions</code>	由很多控制不同动作的单字母标志组成，特别是输入注释时，如何安排对它们进行格式化
<code>comments</code>	描述用于不同类型的注释的不同格式化选项。包含所有那些像在 C 中一样以分隔符开始并以分隔符结束的选项，以及那些像在 <i>makefile</i> 或 shell 程序中以单个符号开始并到行尾结束的选项

在正确设置后，*vim* 就会在输入时自动调整 C 程序的缩进。例如，在输入 `if` 后，*vim* 会对下一行自动缩进。如果 `if` 块包含在大括号中，那么在输入右大括号时，*vim* 就会自动将它缩回一个制表位，与它所属的 `if` 对齐。还有一个例子，其设置如下所示，在输入与 `case` 相匹配的冒号后，*vim* 就会把包含 `case` 的行向左移动一个制表符的距离，与所属的 `switch` 对齐。

在我们看来，下面的 `.vimrc` 会产生令人非常满意的格式化 C 代码。

```
set nocp incsearch
set cinoptions=:0,p0,t0
set cinwords=if,else,while,do,for,switch,case
set formatoptions=tcqr
set cindent
syntax on
source ~/.exrc
```

`nocp` 选项关闭了严格的 *vi* 兼容能力。`incsearch` 选项启动了增量搜索功能。对 `cinoptions`、`cinwords` 和 `formatoptions` 的设置不同于默认值，结果将产生一种非常严格的“K & R”C 格式化样式。最后，启动语法着色，接着从用户的 `.exrc` 文件中读取其余的 *vi* 选项。

我们建议你在启动 *vim* 时按照上面所示的那样设置选项，然后花些时间进行 C 或 C++ 程序设计。即便是使用该功能五分钟，也比我们在印刷纸上展示的静态例子给用户留下印象深刻。

头文件搜索

通常，在设计大型 C 程序时，了解定义某个特殊类型的名字、函数、变量或宏的位置是很有用的。虽然标志功能可以帮助实现这一点，但进行标志搜索会真正移动到找到的位置，这也许超出了你的需要。

vim 有许多在当前文件或所包含的头文件中搜索某个关键字的其他出现的命令。我们在这里对它们进行了总结。

vi 和 *ex* 命令分为四类：第一类显示特定对象的第一次出现（在状态行），第二类显示特定对象的所有出现，第三类跳到第一次出现所在的位置，第四类打开一个新窗口并跳到第一次出现所在的位置。通过这四种命令，可以查找关键字（通常是光标下的标识符）和光标下标识符的宏定义。

这些命令使用智能语法功能（`comment` 变量已在前面描述过）来忽略注释中所搜索标识符的出现。使用前面的计数，它们就会找到第 `count` 个出现。如果没有给出其他说明，标识符搜索就从文件的开始处进行。

参考表 11-15 中的 *vim* 标识符搜索命令列表。

表 11-15 vim 的标识符搜索命令

命令	功能
[i	显示含有光标下的关键字的第一个行
]i	显示含有光标下的关键字的第一个行, 但从文件的当前位置开始查找。这个命令在给定计数后就会更加有效
[I	显示所有含有光标下的关键字的行, 显示文件名和行数
]I	显示所有含有光标下的关键字的行, 但从文件的当前位置开始查找
[^I	跳到光标下的关键字首次出现的位置 (注意 ^I 指 TAB 键)
] ^I	跳到光标下的关键字首次出现的位置, 但从当前位置开始查找
^W i	打开一个窗口, 显示光标下的标识符首次 (或第 <i>count</i> 次) 出现的位置
^W ^I	
[d	显示用于光标下的标识符的首次宏定义
]d	显示用于光标下的标识符的首次宏定义, 但从当前位置开始查找
[D	显示用于光标下的标识符的所有宏定义, 显示文件名和行数
]D	显示用于光标下的标识符的所有宏定义, 但从当前位置开始查找
[^ D	跳到用于光标下的标识符的首次宏义处
] ^D	跳到用于光标下的标识符的首次宏定义处, 但从当前位置开始查找
^W d	打开一个窗口, 显示用于光标下的标识符的首次 (或第 <i>count</i> 次) 宏定义的位置
^W ^D	

define 和 include 两个选项描述了定义宏和包含源文件的源代码行。虽然它们适合于 C 的默认值, 但是可以修改它们以适合自己的编程语言 (例如, define 的值: ^\(\# \s* define \! [a-z]* \s* const \s* [a-z]* \)) 可以用来查找 C++ 命名常量的定义)。

也可以使用 ex 命令实现同样的功能, 如表 11-16 所示。

表 11-16 ex 模式下的 vim 标识符搜索命令

命令	功能
[range]is[earch] [!] [count] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 [i 和]i 相似。默认值是整个文件。如果给出了!, 将强行查找注释。如没有给出/, 则查找单词; 如果有, 则查找正则表达式
[range]il[ist] [!] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 [I 和]I 相似。默认值是整个文件
[range]ij[ump] [!] [count] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 [^I 和] ^I 相似。默认值是整个文件
[range]isp[lit] [!] [count] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 ^W i 和 ^W ^I 相似。默认值是整个文件
[range]ds[earch] [!] [count] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 [d 和]d 相似。默认值是整个文件
[range]dl[ist] [!] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 [D 和]D 相似。默认值是整个文件
[range]dj[ump] [!] [count] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 [^D 和] ^D 相似。默认值是整个文件
[range]dsp[lit] [!] [count] [/]pattern[/]	除了在 <i>range</i> 行内查找外, 与 ^W d 和 ^W ^D 相似。默认值是整个文件
che[ckpath] [!]	列出所有不能找到的头文件。如果有!, 则列出所有的头文件

path 选项用来查找没有绝对路径名的头文件, 其默认值为 ., /usr/include, ., 它会在编辑文件所在的目录中、/usr/include 中或在当前目录中搜索。

编程时的光标移动命令

许多增强的和新的光标移动命令, 使得查找搜索匹配结构的配对项以及应该匹配却没有匹配的结构变得更为容易, 例如, 没有相对应的 #endif 的 #if 语句。如果不指定默认值为 1, 大多数这类命令的前面都可以带有计数。

扩展的匹配命令列表见表 11-17。

表 11-17 vim 扩展的匹配命令

命令	功能
%	扩展匹配 C 的 /* 和 */ 注释以及预处理条件语句: #if、#ifdef、#ifndef、#elif、#else 和#endif。
[(移动到第 <i>count</i> 个前面不匹配的 (
[)	移动到第 <i>count</i> 个后面不匹配的)
[{	移动到第 <i>count</i> 个前面不匹配的 {
[}	移动到第 <i>count</i> 个后面不匹配的 }
[#	移动到第 <i>count</i> 个前面不匹配的 #if 或 #else
] #	移动到第 <i>count</i> 个后面不匹配的 #else 或#endif
[* , [/	移动到第 <i>count</i> 个前面不匹配 C 注释的开始符 /*
] * ,] /	移动到第 <i>count</i> 个后面不匹配 C 注释的结束符 */

自动命令

vim 允许指定在特定事件发生时应该执行的操作。该功能将提供很大的灵活性和控制能力。但是大多数时候，权力越大责任也越大；*vim* 文档警告用户对自动命令功能的使用要小心谨慎，以免意外破坏文件。

这种功能复杂且详细。在本节中，我们对它的常见功能进行概述，并给出一个例子。

自动命令被命名为:`:autocmd`。通用语法为:

```
:au event filepat command
```

event 是使用该命令的事件类型，如读一个文件之前和之后 (`FileReadPre` 和 `FileReadPost`)、写文件之前和之后 (`FileWritePre` 和 `FileWritePost`)、以及

进入或离开窗口 (WinEnter 和 WinLeave)。还有很多已定义的事件，在事件名字中不区分大小写。

filepat 是 *vim* 应用于文件名的 shell 风格的通配符模式。如果匹配它们，那么自动命令就会应用于该文件。

command 是任何一个 *ex* 模式命令。*vim* 有专门获取文件名不同部分的语法，如文件的扩展或没有扩展的名字。虽然它们可用在任意 *ex* 命令中，但是结合自动命令的使用效果会更好。

作用于同一文件和文件模式的多个自动命令会把命令添加到列表上。通过把!添加在:*autocmd* 命令的后面，可以把自动命令从事件和文件模式的特殊组合中删除。

一个很好的例子就是编辑使用 *gzip* 程序压缩的文件。该文件在开始编辑时会自动解压，并且在写回文件时重新进行压缩（为了可读性将第四行分开了）：

```
:autocmd! BufReadPre,FileReadPre      *.gz set bin
:autocmd! BufReadPost,FileReadPost     *.gz '[,'!]!gunzip
:autocmd  BufReadPost,FileReadPost     *.gz set nobin
:autocmd  BufReadPost,FileReadPost     *.gz \
execute ':doautocmd BufReadPost " . expand("%:r")'

:autocmd! BufWritePost,FileWritePost   *.gz !mv <afile> <afile>:r
:autocmd  BufWritePost,FileWritePost   *.gz !gzip <afile>:r

:autocmd! FileAppendPre              *.gz !gunzip <afile>
:autocmd  FileAppendPre              *.gz !mv <afile>:r <afile>

:autocmd! FileAppendPost             *.gz !mv <afile> <afile>:r
:autocmd  FileAppendPost             *.gz !gzip <afile>:r
```

前四个命令是用来读取压缩后的文件。本组命令中的前两个使用! 删除所有以前为压缩文件 (*.gz) 定义的自动命令。压缩的文件以二进制文件形式读入缓冲区，因此第一个命令启动 bin (binary 的缩写) 选项。

vim 为刚读取文本的第一行和最后一行设置 ‘[和 ‘] 标记。第二个命令使用它们对刚读取到缓冲区中的文件进行解压缩。

接下来的两行取消了 *binary* 选项的设置，然后使用任何能应用于非压缩版文件的自动命令（例如语法高亮显示）。%:r 是没有扩展的当前文件名。

再接下来两行用来写压缩后的文件。本组中的第一个命令首先删除所有以前为压缩文件 (*.gz) 的这些事件定义的自动命令。该命令调用 shell 对文件进行重命名，以使它们不再有扩展名 .gz，然后运行 *gzip* 压缩文件。<afile>:r 是没有扩展的文件名（<afile>:r 的使用仅限于自动命令）。*vim* 将没有压缩的缓冲区写到带有 .gz 扩展的文件中，因此需要对它进行重命名。

该组命令中的第二行运行 *gzip* 来压缩文件。*gzip* 自动重命名文件，并加上 .gz 扩展名。

最后四行处理对已压缩文件追加文本的情况，其中的前两行在文件添加内容之前对该文件解压缩并进行重命名。

最后，后面的两行在写好文件后对它重新进行压缩，目的是避免没有对文件进行压缩。

本节只接触了自动命令的一小部分。例如，可以对自动命令分组，以便于能够整体地执行或删除它们。所有在“语法高亮显示”一节描述的语法着色命令都被放置到 highlight 组中，然后在读取适当的文件时自动命令就会执行它们。

举例来说，如果不想一直让 *.vimrc* 文件执行用于智能 C 缩进的 set cindent，则可以使用自动命令使它只用于 C 源代码，如下所示：

```
autocmd BufReadPre, FileReadPre *.[ch]y set cindent
```

源代码和支持的操作系统

vim 有自己的 Internet 域，最好从它的主页（地址是 <http://www.vim.org/>）开始。

vim 的 FAQ (常见问题解答) 位于 <http://www.vim.org/faq/>, 并且在 <http://www.vim.org/mail.html> 的开始还有几个与 *vim* 相关的邮件列表。

有许多 *vim* 主发布站点的镜像 *ftp* 站点, 它们都和 *ftp.country.vim.org* 一样均可以访问。使用表 11-18 列出的两个字母来代替国家。更多的细节, 包括其他的镜像站点, 可以通过网页上的链接和从 *ftp://ftp.nl.vim.org/pub/vim/MIRRORS* 文件中获得。其他站点都是 *ftp.nl.vim.org* 的镜像, 当通过 *ftp* 获取文件时, 应尽可能访问最近的一个。

表 11-18 *vim* 发布站点的国家代码

代码	国家
au	澳大利亚
ca	加拿大
gr	希腊
hu	匈牙利
jp	日本
kr	韩国
nl	荷兰
pl	波兰
us	美国

vim 的源代码是免费发布的。虽然允许以二进制和源代码形式发行, 但是如果你修改了 *vim* 并重新发布, 则必须让维护者获知相应的修改, 以便在后面的版本中包含它。*vim* 也是“慈善软件”, 这在本章前面已经讨论过了。

vim 已移植到下列系统上:

- Amiga (这是诞生 *vim* 的系统)。
- Acorn Archimedes。完成的最后一次操作移植是 2.0 版, 新的移植正在进行, 它将包含在 5.2 版中。
- BeOS。截止到 *vim* 5.1 版, Intel 和非 Intel 的 CPU 都支持。

- MS-DOS。
- Apple Macintosh。最初到 Macintosh 的移植支持 3.0 版，5.x 版的移植仍标识为处于 Alpha 阶段。
- Atari 微型机上的 MiNT。
- OS/2。
- UNIX。实际上任何 UNIX 的变体都可以工作，*vim* 使用 GNU Autoconf 进行配置。
- VMS。
- Windows 95 和 Windows NT。控制台版和 GUI 版都可以得到，在 Windows 3.1 中使用 32 位的 DOS 版。

联机帮助对 *vim* 到每个操作系统的移植特性进行了描述。

编译 *vim* 很容易。通过 *ftp* 获得它的发布版本，对它进行解压缩后运行 *configure* 程序，然后运行 *make*：

```
$ gzip -d < vim-5.1.tar.gz | tar -xvpf -
...
$ cd vim-5.1; ./configure
...
$ make
...
```

应该正确地配置和创建 *vim*，然后运行 *make install* 安装它。

如果你要报告 *vim* 中的漏洞和问题，请通过邮箱 *Bram@vim.org* 与 Bram Moolenaar 联系。

第十二章

vile — 类 Emacs 的 vi

本章内容：

- 作者和历史
- 重要的命令行参数
- 联机帮助和其他的文档
- 初始化
- 多窗口编辑
- GUI 接口
- 扩展的正则表达式
- 改进的编辑功能
- 编程辅助
- 令人感兴趣的功能
- 源代码和支持的操作系统

vile 代表 “vi Like Emacs”，它起初是 MicroEMACS 3.9 版的副本，经修改后才具有了 *vi* 的特性。目前 *vile* 有三个维护者：Paul Fox、Tom Dickey 和 Kevin Buettner。目前版本是 8.0，实际上与 7.4 版相同，只是修补了一些漏洞。本章就是用 *vile* 编写的。

作者和历史

Paul Fox 是这样描述 *vile* 的早期历史的：

vile 的设计目标与其他的克隆版本一直存在着差别。虽然 *vile* 从来没有真正想成为“克隆”版本，但是很多人发现它很像克隆版本。我开始编写它的原因是 1990 年我想在多个窗口中编辑多个文件，10 年来我一直使用 *vi*，当 Micro-EMACS 的代码在我的新闻阅读器出现时，由于在工作时有很多时间可以自己支配，于是我就从以各种方式修改现有的键映射表开始，然后全力解决“喂！‘插入’模式在哪里？”的问题。我逐步进行删改，最后终于在

1991年或1992年发布了*vile*（发布后不久，主要版本号都跟随发行的年号：7.3版是在97年发行的第三版）。

但是我的目标一直就是保持键盘使用的方便（而不是显示的视觉问题），并按照自己的意愿尽可能地为我使用的命令保留键盘使用的方便。*vile*有相当引人注目的*ex*模式，它能很好地工作——只是看起来有点怪怪的，并且有很多命令超出了现有分析程序的范围。出于同样的原因，*vile*也不能对现有的.*exrc*文件进行完全地解析，因为我认为那并不是很重要——它只能解析简单的.*exrc*文件，较复杂的就需要一些技巧。但是当你专注于*vile*的内置命令/宏语言时，你将很快忘记曾经关注过.*exrc*文件。

1992年12月Tom Dickey开始编写*vile*，起初只是创建补丁文件，后来开发了很多重要的功能和扩展，例如行编号、名字完整化和激活缓冲区列表窗口。Tom说：“对于我的设计目标，将许多功能组合起来要比实现很多功能更重要。”

1994年2月，Kevin Buettner开始编写*vile*。起初他完成了对X11版(*xvile*)漏洞的修补，接下来进行了许多改进，如滚动条。渐渐就发展到对Motif、OpenLook和Athena的窗口小部件设置的支持。令人惊奇的是，由于Athena的窗口小部件并没有作为“被广泛使用的无漏洞版本”，kevin竟然编写了使用原始Xt工具箱的版本。该版本的完成为Athena版提供了超强的功能。Kevin对*vile*最初支持GNU Autoconf做出了贡献。

目前，*vile*的维护由“委员会”进行，Tom Dickey是主要维护人，Paul负责管理邮件列表。

就近期来说，未来的工作将集中在完善Perl的集成上，并增强主模式的概念（下面将进行论述）。

重要的命令行参数

尽管*vile*不希望作为*vi*或*ex*调用，但它仍可以作为*view*调用。在这种情况下，*vile*以只读方式处理每个文件。与其他的克隆版本不同，*vile*没有行编辑器模式。

下面是一些重要的 *vile* 命令行参数：

-? *vile* 显示简短的用法摘要后退出。

-g *N*

vile 将从指定的行号处开始编辑第一个文件，也可以用 +*N* 的形式给出。

-s *pattern*

vile 将首先在第一个文件中搜索给定的模式。这也可以用 +/*pattern* 的形式给出。

-t *tag*

在指定的标志处开始编辑。-T 选项与之等价，也可以在 X11 选项分析接受 -t 时使用该选项。

-h 使用帮助文件时调用 *vile*。

-R 以“只读”模式调用 *vile*，在该模式下禁止写操作（如果把 *vile* 作为 *view* 调用或者在启动文件中设置了 readonly 模式，也会出现相同的情况）。

-V 在“视图”模式下调用 *vile*。在该模式下禁止对任何缓冲区进行修改。

@*cmdfile*

vile 将把指定的文件作为它的启动文件运行，并绕开任何正常的启动文件（如 *.vilerc*）或环境变量（如 VILEINIT）。

联机帮助和其他的文档

vile 目前有专用（很大）的 ASCII 文本文件 *vile.hlp*。`:help`（可缩写成 `:h`）命令将在新窗口中打开该文件，然后可以使用标准的 *vi* 搜索技巧搜索个别主题的信息了。由于它是平面 ASCII 文件，所以很容易打印和阅读。

除了帮助文件外，*vile* 还有许多用来显示编辑器功能和状态信息的内置命令。一些最常用的命令如下：

`:show-commands`

创建新窗口以显示所有 *vile* 命令的完整列表，并带有每个命令的简短描述。

该信息放在自己的缓冲区中, 这些缓冲区与其他的*vile*缓冲区一样。尤其是, 可以很容易地把这些信息写到文件中供以后打印。

:apropos

列出名字中包含给定子串的所有命令。这比在帮助文件中只根据某个特定主题随机查找信息要容易得多。

:describe-key

使用某键或某键序列提示用户, 然后显示该命令的描述信息。例如, `x` 键将完成 `delete-next-character` 的功能。

:describe-function

使用函数名提示用户, 然后显示这个函数的描述信息。例如, `delete-next-character` 函数删除当前光标位置右边给定数目的字符。

`:apropos`、`:describe-function` 和 `:describe-key` 命令都会给出描述性信息和所有其他的同义词(为了方便, 一个函数可能有多个名字)、所有与它绑定的其他键(由于很多键序列可以绑定到同一个函数)以及该命令是“移动处理”还是“运算符”。`:describe-function next-line` 的输出是这些命令的一个很好的例子:

```
"next-line"          ^J      j      # -B
or    "down-arrow"
or    "down-line"
or    "forward-line"
(motion: move down CNT lines)
```

这里显示了所有的四个名字和三个绑定键(序列 `# -B` 是向上箭头在*vile*的独立终端上的表示——帮助文件中有这些名字的完整列表)。

`VILE_STARTUP_PATH` 环境变量可以设置为用于帮助文件的由冒号分割的搜索路径(注 1)。`VILE_STARTUP_PATH` 环境变量可用来覆盖帮助文件的名字(通常是 `vile.hlp`)。

注 1: 虽然帮助文件表示在搜索启动文件时也可使用该路径, 但是 7.4 版本的源代码并不支持该功能。它实际上是 `:source` 命令使用的搜索路径。在 8.0 版本中, 对此进行了修补——启动文件和 `:source` 命令使用相同的机制。

在线搜索帮助、内置命令与键描述和命令完整化的结合使帮助功能很容易使用。

初始化

vile 和 *xvile* 进行下面的初始化：

1. (只对 *xvile*) 如果为环境变量 `XVILE_MENU` 提供了值, 就用它作为菜单描述文件的名字。否则就使用 `.vilemenu`。该文件的目的是为 X11 接口设置默认菜单。然后还可以在其他的启动文件中增加或忽略其中的一些菜单。
2. 如果存在命令行上指定的 `@cmdfile` 文件就运行它。绕开所有其他的初始化步骤, 否则这些初始化步骤都要执行。
3. 如果存在 `VILEINIT` 环境变量, 执行它的值, 否则就查找初始化文件。
4. 如果 `VILE_STARTUP_FILE` 环境变量存在, 则使用它作为启动文件的名字。如果它不存在, 则在 UNIX 上使用 `.vilerc`, 在其他系统上使用 `vile.rc`。
5. 首先在当前目录下查找启动文件, 然后再在用户主目录下查找。使用最先找到的。

至于 *nvi* 和 *vim*, 可以将共同的初始化操作放到 `.exrc` 文件中 (即用于 UNIX *vi* 或其他克隆版本的选项和命令), 并在 *vile* 进行特定初始化之前或之后让 `.vilerc` 文件执行: `:source .exrc`。

多窗口编辑

vile 与其他的克隆版本有点不同。它是作为 Micro-Emacs 的一个版本, 后来才修改为具有 *vi* 特征 (“手感”) 的编辑器。

emacs 版本一直具有的功能之一就是处理多窗口和多文件, 就这一点而言, *vile* 是第一个提供多窗口和编辑缓冲区的类 *vi* 编辑器。

如同在 *elvis* 和 *vim* 中一样, :split 命令将创建新窗口 (注 2)。接下来可以使用 *ex* 命令 :e *filename* 在新窗口中编辑新文件。但从这以后操作就不同了, 尤其是在窗口之间进行切换的 *vi* 命令模式键更是截然不同。

```
<preface id="VI6-CH-0">
<title>Preface </title>

<para>
Text editing is one of the most common uses of any computer system, and
<command>vi</command> is one of the most useful standard text editors>
With <command>vi</command> you can create new files, or edit any exist>
file.
</para>
ch00.sgm
# Makefile for vi book
# Arnold Robbins

CHAPTERS = ch00_6.sgm ch00_5.sgm ch00.sgm ch01.sgm ch02.sgm ch03.sgm \
           ch04.sgm ch05.sgm ch06.sgm ch07.sgm ch08.sgm
APPENDICES = appa.sgm appb.sgm appc.sgm appd.sgm

POSTSCRIPT = ch00_6.ps ch00_5.ps ch00.ps ch01.ps ch02.ps ch03.ps \
             ch04.ps ch05.ps ch06.ps ch07.ps ch08.ps \
==== Makefile =[modified] ===== top ===
```

分割屏幕是输入 *vile ch00.sgm* 后再输入 :split 和 :e Makefile 的结果。

与 *vim* 一样, 所有的窗口都共享底行来执行 *ex* 命令。每个窗口有自己的状态行, 用等号填充状态行来表明当前窗口。在插入模式下, 状态行也在第二列中显示 I, 当文件已经修改但没有写入时, 文件名后将附有 [modified]。

vile 在命令与键序列绑定方面也与 *emacs* 相似。表 12-1 列出了命令和与它们对应的键序列。有些情况下, 两组键序列完成同样的操作, 如 delete-other-windows 命令。

注 2: 该操作表示 *vile* 允许用户编写命令, 该命令的真实名字是 split-current-window。

表 12-1 vile 的窗口管理命令

命令	键序列	功能
delete-other-windows	^O , ^X 1	删除除当前窗口之外的所有窗口
delete-window	^K , $\text{^X } 0$	如果当前窗口不是最后一个窗口, 就删除它
edit-file, E, e	^X e	将指定 (或在光标下的, 如 ^X e) 文件和现有缓冲区放入窗口
find-file		
grow-window	V	按 <i>count</i> 行增加当前窗口的大小
move-next-window-down	$\text{^A } \text{^E}$	将下一个窗口向下 (或缓冲区向上) 移动 <i>count</i> 行
move-next-window-up	$\text{^A } \text{^Y}$	将下一个窗口向上 (或缓冲区向下) 移动 <i>count</i> 行
move-window-left	$\text{^X } \text{^L}$	把窗口向左滚动 <i>count</i> 列, 如果没有指定 <i>count</i> , 则滚动半个屏幕
move-window-right	$\text{^X } \text{^R}$	把窗口向右滚动 <i>count</i> 列, 如果没有指定 <i>count</i> , 则滚动半个屏幕
next-window	$\text{^X } \text{o}$	切换到下一个窗口
position-window	z <i>where</i>	按 <i>where</i> 指定的光标进行重新组织: 中心 (., M, m) 顶部 ([RETURN], H, t) 或底部 (-, L, b)
previous-window	$\text{^X } \text{o}$	切换到前一个窗口
resize-window		将当前窗口改变为 <i>count</i> 行, <i>count</i> 作为前缀参数提供
restore-window		返回到用 save-window 保存的窗口
save-window		对某个窗口做标记, 以便以后使用 restore-window 返回到该窗口
scroll-next-window-down	$\text{^A } \text{^D}$	将下一个窗口向下移动 <i>count</i> 个半屏幕, <i>count</i> 作为前缀参数提供
scroll-next-window-up	$\text{^A } \text{^U}$	将下一个窗口向上移动 <i>count</i> 个半屏幕, <i>count</i> 作为前缀参数提供
shrink-window	V	把当前窗口减小 <i>count</i> 行, <i>count</i> 作为前缀参数提供

表 12-1 vile 的窗口管理命令（续）

命令	键序列	功能
split-current-window	$\wedge X 2$	将窗口分成两半， <i>count</i> 是 1 或 2，选择当前窗口。 <i>count</i> 作为前缀参数提供
view-file		把指定文件或现有缓冲区放入窗口，将其标记为“只读”的
historical-buffer	-	显示最初的 9 个缓冲区的列表。将数字移动到指定缓冲区，__ 则移动到最近编辑的文件
toggle-buffer-list	*	向上/向下弹出一个显示所有 <i>vile</i> 缓冲区的窗口

GUI 接口

该部分的屏幕图像和解释是由 Kevin Buettner、Paul Fox 和 Tom Dickey 提供的，我们感谢他们。

有很多用于 *vile* 的 X11 接口，每个接口都使用基于 Xt 库的不同工具箱；有不使用工具箱的普通“无工具箱”版本，该版本只提供通用滚动条和用于形状管理的公告栏小部件；还有很多使用 Motif、Athena 或 OpenLook 工具箱的版本。在这些版本中，由于一些 *vile* 的作者最经常使用“无工具箱”版，因此该版本可能使用得最为广泛。但是 Motif 和 Athena 版本有更多的功能，如菜单支持。

幸运的是，每个版本的基本接口都是相同的。它们都提供了惟一的顶级窗口，并且可以分成两个或多个长方块。这些长方块可以轮流显示一个缓冲区或多个缓冲区，或是两者混合的多个视图。在 *vile* 用语中，尽管这些长方块称为“窗口”，但为了避免混淆，我们在下面的讨论中仍继续把它们称为“长方块”。

构建 xvile

如果要构建 *xvile*，就必须选择要使用的工具箱版本。这项操作可以在使用 *configure* 命令配置 *vile* 时进行。相关的选项有：

--with-screen=value

指定终端驱动。默认值是 `tcap`, 代表 *termcap/terminfo* 驱动。其他值包括 `ncurses` (*terminfo* 的一种特例)、`X11`、`OpenLook`、`Motif`、`Athena`、`Xaw`、`Xaw3d`、`neXtaw` 和 `ansi`。

--with-scr=value

和 --with-screen 选项一样。

--with-x

使用 X Window 系统, 这是“无工具箱”版本。

--with-Xaw3d

链接 `Xaw` 3D 库。

--with-neXtaw

链接连接 `neXt Athena` 库。

--with-Xaw-scrollbars

使用 `Xaw` 滚动条而不是 `vile` 通用滚动条。

基本的外观和功能

图 12-1 显示了 *xvile* 的 Motif 接口, 它与 `Athena` 接口相似。

图 12-1 显示了三个长方块:

1. *vile* 的手册页, 它显示了下划线和粗体字的应用。
2. 一个来自 *tin* 的 `misc.c` 缓冲区, 它显示了语法高亮显示 (还有表示预处理声明的下划线和表示引用字符的粗体字)。
3. 三行的当前 (由较黑的状态行标识) 长方块, 命名为 [Completions], 表示文件名完整化。该长方块与一个小缓冲区 (冒号命令行) 相关: 第一行显示 *Completions prefixed by /tmp/m:*, 小缓冲区显示 *Find file: m*。长方块的剩余部分包含着匹配的实际文件名。第一行 [Completions] 和内容随着用户补充文件名而变化 (按 `TAB` 键让 *vile* 显示减少的选项组)。



图 12-1 vile 的 GUI 窗口

图 12-2 也显示了三个窗口块：

1. [help]长方块, 显然它显示了编辑器最重要的功能(如何不修改文件退出)。
2. [Buffer List]长方块指示[Help]是#(前面的)缓冲区。由于只有“可见”缓冲区才显示在[Buffer List]的副本中, 因此%(当前的)缓冲区没有显示在列表上。对*命令使用参数就会显示不可见缓冲区。缓冲区1和缓冲区2是charset.c和misc.c。由于已经加载它们, 因此它们的大小(8931和54 866)将显示在[Buffer List]中。由于缓冲区3、4、5(color.c, config.c, 和curses.c)还没有加载, 因此第一列显示u, 其大小显示为0。

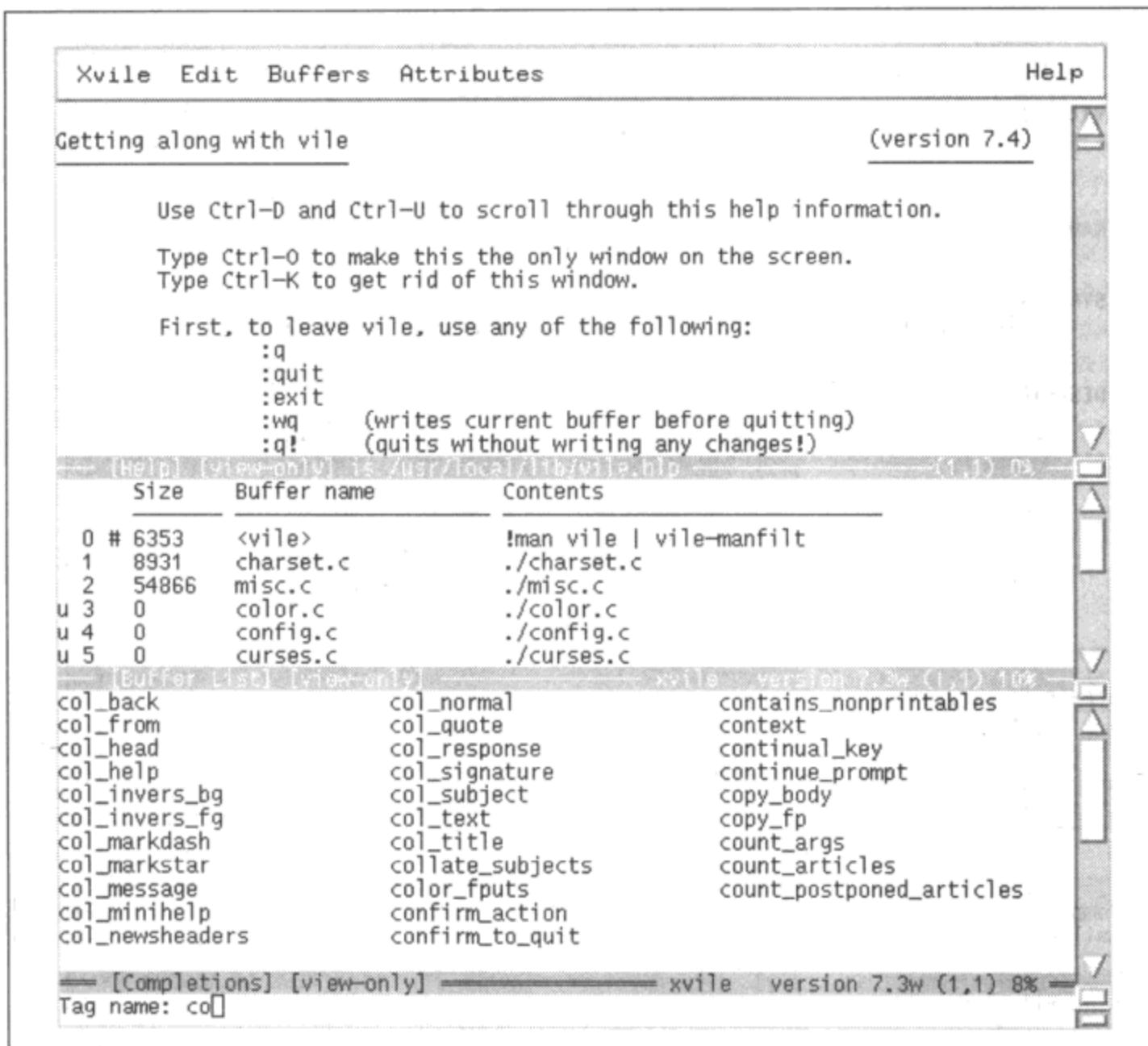


图 12-2 vile 中的缓冲区和 completion

3. [Completions] 缓冲区是激活的。这次它显示的是和 *co* 部分匹配的标志完整化。但 *Completions prefixed* 消息并没有显示出来，因为缓冲区已经滚动到下面了，这是按下 **TAB** 键的另一种结果：尽管此时窗口很小，*vile* 通过循环滚动操作使得所有的选择都能显示出来。（当光标在小缓冲区中时，**v/v** 命令对 [Completion] 缓冲区不起任何作用；[Completion] 缓冲区会按大小自动排列。）

滚动条

每个长方块的右边都有一个滚动条，它以习惯方式在缓冲区中移动。但是要注意，

习惯方式随着工具箱的不同而变化。在Athena和“无工具箱”版本中，鼠标的中键可用来四处拖动“姆指”或可见指示符。鼠标的左右键分别用来在缓冲区中向下或向上移动。移动的数量依赖于鼠标光标在滚动条上的位置。将鼠标放到滚动条顶部附近时，文本按行滚动；当鼠标放置在底部时，文本将按整个长方块滚动。

Motif和OpenLook滚动条也许比较常见。鼠标最左边的键完成一切操作。点击小箭头将会向上或向下移动一行。为了快速移动可拖动滚动条指示符，通过在指示符的上面或下面点击可实现整个长方块地上下滚动。OpenLook滚动条还提供了快速移动到缓冲区顶部或底部的追加机制。

在每个版本中，滚动条的上面或下面都有一个小手柄，可用来调节两个相邻长方块的大小。在xvile的“无工具箱”版本中，长方块的尺寸调整柄与两个相邻窗口块的模式行混合在一起。在其他版本中，尺寸调整柄更容易识别。但在每种情况下，当鼠标的光标放在尺寸调整柄上时，就会变成深颜色的竖直双箭头。点击并拖动手柄就能改变窗口大小。

通过按下控制键并在滚动条上点击鼠标左键，就可以将长方块一分为二。这样就拥有了某特定缓冲区的两个视图。如果希望使用别的缓冲区替代其中一个视图，可以使用其他的vile命令。通过按住控制键同时在滚动条上点击鼠标中键，可以删除一个长方块。有时在创建了很多长方块后，也许发现想在整个窗口中只显示一个长方块。为了完成该操作，可按住控制键同时在滚动条上点击右键，所有其他的长方块都将移走，整个xvile窗口只剩下在其上点击的那个长方块。

设定光标位置和鼠标运动

在某个长方块的文本区域内点击鼠标的左键可以设置光标。这样不仅设定了光标的位置，而且也确定了将要编辑的长方块。如果只要设定长方块而不移动光标的原来位置，可以在编辑的文本下面的模式行上点击鼠标。

类似于4j，鼠标点击也被视为一种移动操作。为了删除5行，可以键入删除当前行和下面四行d4j。也可以使用鼠标点击来做到这一点。将光标放到想开始删除的位置并按下d，然后在缓冲区中结束删除的位置点击鼠标。点击鼠标是真正的移动操作，它也可以和其他的操作符一起使用。

选择

通过按下鼠标左键并拖动鼠标可以进行选择。释放鼠标按钮将复制选择选中的文本并可进行粘贴(如果需要)。通过按住控制键同时按住鼠标左键拖动鼠标可以强行选择一个矩形区域。如果拖出了当前的窗口，文本将向相应的方向滚动。如果可能，则可容纳比窗口还大的选择区域。随着使用时间的增加，滚动的速度也要增加，从而能够快速地选中大区域的文本。

在单词或单行上双击或三击，就可以选中它们。

选择区域还可以通过单击鼠标右键进行扩展，和通过左键操作一样，可通过按住右键拖动所选择的区域来对它进行调整或滚动。选择可以扩展到为同一个缓冲区打开的任意窗口内，就像在选择开始的窗口一样。也就是说，如果你有一个缓冲区的两个视图(在两个不同的长方块中)，一个存放缓冲区的头，另一个存放缓冲区的尾。通过在显示缓冲区开始部分的长方块的起始处点击左键，然后在显示缓冲区的结束部分的长方块的结束处点击右键，就可以选中整个缓冲区。而且，还可以通过按住控制键的同时使用鼠标右键对选中的区域进行矩形扩展。

使用鼠标的中键来粘贴选中的区域。在默认情况下，它将粘贴到最后的文本光标所在的位置。如果在点击鼠标中键的同时按住 shift 键，就会粘贴到鼠标光标所在的位置。

在其中的一个模式行上双击鼠标就可删除选中的区域(如果它属于 *xvile*)。

剪贴板

数据可经过 PRIMARY 选择在许多的 X 应用之间进行交换。选择的设定和操作如上所述。

其他的应用软件，包括最著名的 OpenLook 应用软件，也使用 CLIPBOARD 选择在应用程序中交换数据。在许多 Sun 的键盘上，按下 **COPY** 键会将选中的文本移动到剪贴板中，按下 **PASTE** 键就会进行粘贴。如果发现不能将在 *xvile* 中选中的文本粘贴到其他应用中或者反过来，那么可能是由于这些应用软件使用的是

CLIPBOARD 选择而不是 PRIMARY 选择。(在老的应用软件中使用的其他机制涉及到剪切缓冲区串的使用。)

xvile 提供了两个操作剪切板的命令, 它们是 copy-to-clipboard 和 paste-from-clipboard。当执行 copy-to-clipboard 时, 当前选中的内容将复制到指定的剪贴板清除寄存器 (在寄存器列表中通过; 表示)。当应用程序需要剪贴板的选择内容时, *xvile* 将这个清除寄存器中的内容交给它。paste-from-clipboard 命令从 CLIPBOARD 选择的当前拥有者那里请求剪贴板数据。

Sun 系统的用户可能想将下面的键绑定存放在 .vilerc 文件中, 以方便在他们的键盘上使用 COPY 和 PASTE 键:

```
Bind-key copy-to-clipboard  #-^  
Bind-key paste-from-clipboard  #-*
```

本章后面将会对键绑定进行详细地描述。

资源

xvile 有很多用来控制外观和操作的资源。如果想正确地显示斜体或倾斜字, 那么字体选择尤其重要。*vile* 的文档有完全的资源列表和一组简单的 .Xdefault 人口。

添加菜单

Motif 和 Athena 版本支持菜单。用户定义的菜单项来自当前或主目录中的 .vilemenu 文件。

xvile 有三种类型的菜单项:

- 内置菜单项, 即专门用于菜单系统, 如重新读取 .vilerc 文件或创建 *xvile* 的新副本。
- 直接调用内置命令的菜单项 (如显示 [Buffer List])。

- 调用任意命令串的菜单项（如运行像搜索命令之类的交互式宏）。

对后面两个进行区分是因为作者希望 *vile* 在执行命令之前能检查命令的有效性。

本章中的工具条包括一个有注释的样本文件 *.vilemenu*。

扩展的正则表达式

扩展的正则表达式已经在第八章中的“扩展的正则表达式”一节进行了介绍。虽然 *vile* 有必要提供与 *nvi* 的 extended 选项同样的功能，但是它们的语法有一些不同，*vile* 主要依赖于额外的反斜杠转义字符：

\| 表示或， house \| home

\+ 匹配前面正则表达式的一个或多个实例。

\? 匹配前面正则表达式的零个或多个实例。

\(...\)

提供用于 *、\+、和\?的分组，同时在替换命令的替换部分使匹配的子文本可用 (\1、\2 等)。

\s \S

分别匹配空白和非空白字符。

\w \W

分别匹配“组成单词”的字符（字母数字和下划线 ‘_’）和非组成单词的字符。例如，\w\+ 将匹配 C/C++ 标识符和关键字（注 3）。

\d \D

分别匹配数字和非数字字符。

\p \P

分别匹配打印字符和非打印字符。认为空白是可以打印的。

注 3：同样也匹配以前导数字开始的标识符，通常这不是个大问题。

vile 允许转义序列 \b, \f, \r, \t 和 \n 出现在替换命令的替换部分。它们分别表示退格、格式输入、回车、制表符和新建行。而且，在 *vile* 文档中还有：

要注意，*vile* 仿照 *perl* 的 \u\L\l\E 的处理而不是 *vi* 的处理。给出 :s/\(abc\)/\u\L\l\E /，*vi* 将会使用 *abc* 进行替换，而 *vile* 和 *perl* 将用 *Abc* 替换。这在处理大写单词时非常有用。

Example .vilemenu 文件

```
# lines beginning with 'C' define a menu heading
C:Xvile
# the following four entries are Buttons invoking menu
# system builtins
B:New:new_xvile
B>Edit .vilerc:edit_rc
B:Parse .vilerc:parse_rc
B>Edit .vilemenu:edit_mrc
B:Quit:quit
#
C:Editing
B:Search Forward...:cmd search-forward
B:Search Backward...:cmd search-reverse
# lines beginning with S are separators
S
B:Manual for...:29
S
# where the command to be executed is given as a number, like the
# two above and the three below, the menu system will translate
# this to an invocation of the command execute-macro-<number>.
B:Indent Level...:31
B:Window Title...:35
B:Font...:36
#
C:Buffers
# run a command name (in this case "toggle-buffer") by simply
# naming it
B:Toggle Show:toggle-buffer
# one line starting with 'L' is allowed, at the end of a menu --
# it causes a buffer list menu to be created.
L:list_buff
```

```
#  
C:Fonts  
B:5x8:setv $font 5x8  
B:7x14:setv $font 7x14  
B:8x13:setv $font 8x13  
B:8x16:setv $font 8x16  
B:9x15:setv $font 9x15  
B:10x20:setv $font 10x20  
B:12x24:setv $font 12x24  
#  
C:Attributes  
B:C/C++:30  
B:Pascal:32  
#  
C:Help:help  
B:About:version  
S  
B:General:help  
B:Bindings:describe-bindings  
B:Motions:describe-motions  
B:Operators:describe-operators  
S  
# prefixing a command with "cmd" will force it to be run as from  
# the ':' line, so that it can prompt for input correctly.  
B:Apropos...:cmd apropos  
B:Apropos...:apropos set  
B:On Function...:cmd describe-function  
B:On Key...:describe-key &gt;  
S  
B:Settings:setall  
B:Variables:show-variables  
B:Registers:show-registers
```

改进的编辑功能

本节描述了 *vile* 使简单的文本编辑更容易、更有效的功能。

命令行历史和完整化

vile 将所有的 *ex* 命令存放在名为 [history] 的缓冲区中。history 选项控制着这

项功能，它的默认值为真。关闭它将会取消历史记录功能并删除 [history] 缓冲区。命令 `show-history` 将分割窗口并在新窗口中显示 [history] 缓冲区。

从 *vile* 7.4 版开始，冒号命令行就是个真正的小缓冲区。可以使用它从 [history] 缓冲区中取回行并进行编辑。

可以使用 `↑` 和 `↓` 键在历史记录中后退或前进，使用 `←` 和 `→` 键在行中移动。`BACKSPACE` 键可用来删除字符。输入的其他字符将插入到光标的当前位置。

可以输入 `mini-edit` 字符（默认是 `^G`）将小缓冲区固定到 *vi* 模式。在进行该操作时，*vile* 将使用 `mini-hilite` 选项指定的机制来加亮小缓冲区。默认值是 `reverse`，用来反转图像。在 *vi* 模式下，可以使用 *vi* 风格的命令进行定位。在 8.0 版本中，还可以使用 `i`、`I`、`a`、和 `A` 等 *vi* 命令。

另一个令人注意的功能是，*vile* 将使用历史记录显示前面与正输入的命令相对应的数据。例如，在输入 `:set` 和一个空格后，*vile* 将使用 `Global value:` 来提示用户。此时，可以使用 `↑` 键来查看前面已经设定的全局变量，以及是否想改变它们中的一个。

ex 命令行提供了各种各样的完整化方法，在输入命令名字时，可以随时按 `TAB` 键。*vile* 将尽可能地填充出命令名字的剩余部分。如果你再次点击 `TAB` 键，*vile* 将会在新窗口中显示所有可能的补充内容。

完整化适用于内置的或用户定制的 *vile* 命令、标志、文件名、模式（在本章后面讲述）、变量和终止符（退格、暂停等待之类的字符设置，它们都来自于 `stty` 配置）。

另一方面，也导致了一个值得注意的现象。在 *vi* 风格的编辑器中，虽然命令可能有很长的名字，但是由于要进行缩写，因此它们最初的几个字符是惟一的。在 *emacs* 风格的编辑器中，虽然命令名字的最初几个字母并不惟一，但是命令完整化依然能减少输入，让用户从繁琐的输入中解放出来。

标志栈

标志入栈已经在第八章的“标志栈”一节进行了描述。在 *vile* 中，标志入栈是可用的，并且非常简单。它与其他的克隆版本有些不同，*vi* 模式命令中的许多都可以用于搜索标志和弹出标志栈。表 12-2 列出了 *vile* 的标志栈命令。

表 12-2 *vile* 的标志命令

命令	功能
ta[g] [!] [tagstring]	编辑 <i>tags</i> 文件中定义的含有 <i>tagstring</i> 的文件，即使当前缓冲区已更改但未保存，! 也会强制 <i>vile</i> 切换到新文件中
pop[!]	从栈中弹出一个光标的位置，将光标恢复到它的前一个位置处
next-tag	在 <i>tags</i> 文件中继续查找更多的匹配
show-tagstack	创建新窗口显示标志栈。显示随着标志进栈或出栈而不断变化

表 12-3 描述了 *vi* 模式命令。

表 12-3 *vile* 命令模式的标志命令

命令	功能
^]	在 <i>tags</i> 文件中查找光标下的标识符的位置，然后移动到该位置。当前位置自动压入标志栈中
^T	返回到标志栈中的前一个位置，即弹出一个元素
^X ^]	
^A ^]	和:next-tag 命令相同

与在其他编辑器中一样，一些选项控制着 *vile* 管理与标志相关命令的方式，如表 12-4 所示。

表 12-4 vile 的标志管理选项

选项	功能
taglength	控制要搜索的标志中有意义的字符数量。默认值0表明所有字符都有意义
tagignorecase	使标志搜索忽略大小写， 默认情况下该选项为假（false）
tagrelative	在使用另一个目录下的 <i>tags</i> 文件时，认为该 <i>tags</i> 文件中的文件名是相对于该 <i>tags</i> 所在目录的
tags	可设置为用空格分隔的 <i>tags</i> 文件列表，在搜索标志时使用。默认情况下 <i>vile</i> 把所有的 <i>tags</i> 文件都加载到不同的隐藏缓冲区中，如果用户需要编辑，可以对它们进行编辑。可以将环境变量和 shell 通配符放入 <i>tags</i> 中
tagword	利用光标后的整个单词进行标志查找，而不是只使用从当前光标位置开始的部分单词。默认情况下该选项关闭，这样能使 <i>vile</i> 和 <i>vi</i> 兼容

无限撤消

vile 和其他编辑器在原理上相似，但在操作上却不同。和 *elvis* 和 *vim* 一样，*vile* 也可以设置撤消限制，但和 *nvi* 调整一样，它使用 . 命令进行下一撤消或恢复操作。各自的 *vi* 模式命令都实现了连续的撤消和恢复操作。

vile 使用 *undolimit* 选项控制它要存储的修改次数。默认值是 10，说明最多可以撤消最近的 10 次修改。设成零就允许真正的“无限撤消”，但这样会耗费大量的存储空间。

首先使用 *u* 或 *^X u* 命令启动撤消操作。接下来每个连续的 . 命令就会撤消一次。和 *vi* 一样，两个 *u* 命令触发改变的状态；然而每个 *^X u* 命令会进行另一个撤消操作。

^X r 命令进行恢复操作。在第一个 *^X r* 命令后输入 . 命令就会进行连续的恢复操作。可以给出 *^X u* 和 *^X r* 命令的次数，在这种情况下，*vile* 将按要求的次数完成撤消或恢复操作。

任意长度的行和二进制数据

vile 可以编辑含有任意长度的行和任意行数的文件。

vile 自动处理二进制数据，不需要专门的命令行和选项。为了输入八进制文本，要在输入 ^V 后输入一个 x 和两个十六进制数字，或一个 0 和三个十进制数，或三个十进制数。

增量搜索

正如在第八章的“增量搜索”一节所提到的那样，在 *vile* 中可以使用 ^X S 和 ^X R 命令来实现增量搜索。而不必设定一个选项来启动增量搜索。

光标随着你的输入在文件中移动，并且总是停在匹配文本的第一个字符处。^X S 在文件中向前递增搜索，而 ^X R 将向后递增搜索。

你可能希望将这些命令添加到 *.vilerc* 文件中，使熟悉的 / 和 ? 查找命令能进行增量搜索。

```
bind-key incremental-search /
bind-key reverse-incremental-search ?
```

另一个令人注意的是“可视匹配”功能，这个功能将高亮显示所有匹配表达式的实例。对应到 *.vilerc* 文件为：

```
set visual-matches reverse
```

该命令指示 *vile* 对可视匹配使用相反的图像。由于高亮显示有时可能会在视觉上造成注意力分散，= 命令将关闭所有的当前高亮显示的操作，直到输入新的查找模式。

左右滚动

正如第八章中的“左右移动”一节所提到的那样，可以使用 :set nolinewrap 来

启动 *vile* 中的左右滚动。与其他的编辑器不同，左右滚动在 *vile* 是默认的。较长的行在左右边界标有 < 和 >。 *sideways* 的值控制 *vile* 从左向右滚动时屏幕所移动的字符数目。如果 *sideways* 设成 0，每次将移动屏幕的三分之一。否则屏幕将按照指定的字符数滚动。

可视模式

vile 在突出所要操作的文本的方法上与 *elvis* 与 *vim* 不同，它使用“引用移动操作”命令 q。

可以在区域的开始处输入 q，使用任何其他的 vi 移动操作到达该区域另一端，然后用另一个 q 结束引用移动操作。*vile* 将高亮显示已标记的区域。

q 命令的参数决定着高亮显示的方式。1q（同 q）进行正常高亮显示，2q 进行逐行高亮显示，3q 进行矩形高亮显示。

通常，伴随引用移动操作的是 d 或 y 那样的操作符。因此，d3qjjwq 删除由操作所指定的矩形区域。当不带操作符使用时，区域保持高亮显示，可看成最近使用了 ^S。因此，d ^S 将删除高亮显示的区域。

此外，也可以通过使用标志来标识矩形区域（注 4）。即使用标志指示特定的字符（当它与 ' 一起指示时）或特定的行（当它与 ' 一起指示时）。另外，对一个标志（假设用 mb 设定过），使用 'b 代替' b 指示标志可改变正在进行的操作——d' b 将删除一组行，而 d' b 将删除两个部分行以及在两行之间的行。使用 ' 形式的标记引用会比' 形式的标记引用指定更“准确”的区域。

vile 增加了第三种形式的标记引用。 \ 命令是查找标记的另一种方法。对其自身而言，它的行为与 ' 一样，把光标移动到设定标记的字符处。但当带有操作符时，其处理方式就大不相同了。标记引用变成了“矩形”，以至于 d\b 操作将删除对角由光标和带有 b 标记的字符所标识的矩形区域字符。

注 4：感谢 Paul Fox 的讲解。

击键

ma

The 6th edition of <citetitle>Learning the vi Editor</citetitle> brings the book into the late 1990s. In particular, besides the “original” version of <command>vi</command> that comes as a standard part of every UNIX system, there are now a number of freely available “clones” or work-alike editors.

结果

在单词 “book” 的 *b* 字母处设置 a 标记。

击键

3jfr

The 6th edition of <citetitle>Learning the vi Editor</citetitle> brings the book into the late 1990s. In particular, besides the “original” version of <command>vi</command> that comes as a standard part of every UNIX system, there are now a number of freely available “clones” or work-alike editors.

结果

把光标移动到单词 “number” 的 *r* 处来标记对角。

击键

^A ~\a

The 6th edition of <citetitle>Learning the vi Editor</citetitle> brings the BOOK INTO The late 1990s. In particuLAR, BESIDES the “original” version of <command>vi</COMMAND> that comes as a standard part of every UNIX system, there are nOW A NUMBER of freely available “clones” or work-alike editors.

结果

转换以标记 a 为边界的矩形区域内字符的格式。

定义任意区域并对它们进行操作的命令总结如表 12-5 所示。

表 12-5 vi 的块模式操作

命令	操作
q	启动或结束引用操作
^A r	打开一个矩形区域

表 12-5 vile 的块模式操作（续）

命令	操作
>	将文本向右移动，当区域是矩形时，与 ^A r 相同
<	将文本向左移动，当区域是矩形时，与 d 相同
y	复制整个区域。vile 记住它是一个矩形区域
c	更改区域。对于非矩形区域，删除所有两端点之间的所有文本并进入插入模式。对于矩形区域，提示要填充这些行的文本
^A u	将区域中的文本全部改成大写
^A l	将区域中的文本全部改成小写
^A ~	对区域内的所有字母字符转换其大小写类型
^A [SPACE]	使用空格填充该区域
p, P	粘贴文本，如果原来的文本是矩形，vile 就进行矩形粘贴
^A p, ^A P	强行把以前复制的文本当成矩形区域进行粘贴。最长的复制行的宽度就是该矩形的宽度

编程辅助

本部分对 *vile* 的编程辅助功能进行了讨论。

编辑－编译加速

vile 使用两个简单的 *vi* 模式命令管理程序开发，如表 12-6 所示。

表 12-6 vile 程序开发的 vi 模式命令

命令	功能
^X !command[RETURN]	运行 command，把输出保存到名为 [output] 的缓冲区中
^X ^X	寻找下一处错误。vile 分析输出结果并移动到每个连续的错误位置处

vile 能够识别由 GNU 的 *make* 生成的 *Entering directory XXX* 及 *Leaving directory XXX* 信息，允许它寻找正确的文件，即便它位于不同的目录下也是如此。

在缓冲区 [Error Expressions] 中使用正则表达式可以对错误信息进行分析。*vile* 自动创建该缓冲区，然后在使用 ^X ^X 时使用该缓冲区。可以根据需要对它增加表达式，而且它还有一个扩展语法，该语法允许用户指定文件名、行数、列等在错误信息中出现的位置。虽然联机文档上提供了详细内容，但是由于它“不需要那些项”就能很好地工作，因此用户也许不需要进行任何改动。

vile 的错误发现程序也关注文件的变化，记录处理每项错误时的添加和删除信息。

错误发现程序应用于最近的由读取 shell 命令所创建的缓冲区。例如，^X !command 创建名为 [Output] 的缓冲区，:e !command 创建名为 [!command] 的缓冲区。错误发现程序能够很好地设定它们。

使用 :error-buffer 命令可以指定错误发现程序使用任意缓冲区（不只限于 shell 命令的输出），这样就允许对以前的编译器或 egrep 的运行结果应用错误发现程序。

语法高亮显示

vile 依靠外部程序的帮助来提供语法着色功能。实际上，可以使用三种程序：一种用于 C 程序，一种用于 Pascal 程序，另一种用于 UNIX 的 man page。*vile* 文档提供了用于 .vilerc 文件中的样本宏：

```
30 store-macro
    write-message "[Attaching C/C++ attributes...]"
    set-variable %savcol $curcol
    set-variable %savline $curline
    set-variable %modified $modified
    goto-beginning-of-file
    filter-til end-of-file "vile-c-filt"
    goto-beginning-of-file
    attribute-cntl_a-sequences-til end-of-file
    ~if &not %modified
```

```

        unmark-buffer
~endif
%savline goto-line
%savcol goto-column
write-message "[Attaching C/C++ attributes...done ]"
~endm
bind-key execute-macro-30 ^X-q

```

这是在 C 源代码上运行 *vile-c-filt* 的结果。该程序进而依赖于 *\$HOME/.vile.keywords* 的内容，它决定了提供给不同文本的属性。（*B* 代表粗体，*U* 代表下划线，*I* 代表斜体，*C* 代表 16 中不同颜色中的一种。）下面是 Kevin Buettner 的版本：

```

Comments:C2
Literal:U
Cpp:CB
if:B
else:B
for:B
return:B
while:B
switch:B
case:B
do:B
goto:B
break:B

```

语法着色工作在 *vile* 7.4 和 8.0 版本的 X11 接口上进行。让它在 Linux 控制台上工作有点复杂，这依赖于编译时使用的屏幕处理接口。

ncurses 库

使用 *--with-screen=nucurses* 来配置 *vile* 并重新创建语法高亮显示，然后就能在该范围以外操作。

termcap 库

这是 *vile* 的默认配置方法。使用该版本要求用户有用于 Linux 控制台的正确的 */etc/termcap* 入口。下面的 *termcap* 入口可以正常工作（注 5）：

注 5： 该入口是 Kevin Buettner 中支持的。注意，Linux 的发布是不同的。这里是在 Redhat Linux 4.2 下测试的。你不需要修改你的 */etc/termcap/* 文件。

```
console@linux:con80x25:dump:\n
:do=\^J;co#80:li#25:c1=\E[H\E[J:sf=\ED:sb=\EM:\n
:le=\^H:bs:am:cm=\E[%i%d;%dH:nd=\E[C:up=\E[A:\n
:ce=\E[K:cd=\E[J:so=\E[7m:se=\E[27m:us=\E[4m:ue=\E[24m:\n
:md=\E[1m:mr=\E[7m:mb=\E[5m:me=\E[m:is=\E[1;25r\E[25;1H:\n
:ll=\E[1;25r\E[25;1H:al=\E[L:dc=\E[P:dl=\E[M:\n
:it#8:ku=\E[A:kd=\E[B:kr=\E[C:kl=\E[D:kb=\^H:ti=\E[r\E[H:\n
:ho=\E[H:kP=\E[5~:kN=\E[6~:kH=\E[4~:kh=\E[1~:kD=\E[3~:kI=\E[2~:\n
:k1=\E[[A:k2=\E[[B:k3=\E[[C:k4=\E[[D:k5=\E[[E:k6=\E[[7~:\n
:k7=\E[18~:k8=\E[19~:k9=\E[20~:k0=\E[21~:K1=\E[1~:K2=\E[5~:\n
:K4=\E[4~:K5=\E[6~:\n
:pt:sr=\EM:vt#3:xn:km:bl=\^G:vi=\E[?25l:ve=\E[?25h:vs=\E[?25h:\n
:sc=\E7:rc=\E8:cs=\E[%i%d;%dr:\n
:r1=\Ec:r2=\Ec:r3=\Ec:\n
:vb=\E[?5h\E[?5l:\n
:ut:\n
:Co#8:\n
:AF=\E[%a+c\036%dm:\n
:AB=\E[%a+c\050%dm:
```

一方面，由于语法高亮显示是由外部程序完成的，因此对不同语言编写多种加亮效果是有可能的。另一方面，由于这种功能是很基本的，因此不是非编程人员所能处理的。联机帮助描述了加亮过滤器的工作原理。

ftp://ftp.clark.net/pub/dickey/vile/utilities 目录下含有许多用户发布的用于对 *makefile* 文件、*L^ATEX* 输入、Perl、HTML 和 *troff* 进行着色的过滤器。甚至还含有一个根据 RCS 文件中行的存在时间对它们进行着色的宏！

令人感兴趣的功能

vile 有很多令人注意的功能，它们也是本节的主题。

vile 的编辑模型

vile 的编辑模型与 *vi* 的有些不同，它是根据 *emacs* 的概念提供键重新绑定和更动态的命令行。

主模式

vile 支持编辑“模式”。有很多选项组可使 *vile* 方便地编辑不同类型的文件。

过程语言

vile 的过程语言允许用户定义使编辑器更加具有可编程性、更加灵活的函数和宏。

各种各样的小功能

很多小功可以使日常编辑更加容易。

vile 的编辑模型

在 *vi* 和其他的克隆版本中，编辑功能都被“固化”在编辑器中。命令字符与它们所完成的功能之间的联系都内置在代码中。例如，*x* 键删除一个字符，*i* 键进入插入模式。不使用非常手段，就不能切换两键的功能（即使实际上是可以切换的）。

vile 的编辑模型则不同，该模型是通过 MicroEMACS 源于 *emacs* 的。编辑器已经定义、命名了函数，每个函数只有单一的编辑任务，如 `delete-next-character` 或 `delete-previous-character`。然后很多函数都与按键绑定在一起，例如把 `delete-next-character` 与 *x* 绑定。

可以用 `:bind-key` 命令非常容易地改变绑定。至于参数，则要你给出函数名，然后给出函数要绑定的键序列。可以将下面的命令加入 `.vilerc` 文件中：

```
bind-key incremental-search /  
bind-key reverse-incremental-search ?
```

这些命令将 `/` 和 `?` 查找命令更改为进行增量搜索。

除了预定义函数以外，*vile* 还有一个简单的编程语言，该语言允许用户编写过程，可以将执行某个过程的命令与一个按键序列绑定。GNU *emacs* 使用 Lisp 的变体作为它的语言，它的功能非常强大。*vile* 有一种比较简单、但用途不是很广泛的语言。

而且，与在 *emacs* 中一样，*vile* 命令行也是具有交互性的。许多命令为它们操作数提供了默认值，如果这个默认值不合适，则可以编辑它，或通过按下 `[RETURN]`

键进行选择。当用户输入诸如更改或删除之类的 *vi* 模式编辑命令时，则将在状态行上看到操作的反馈信息。

Paul 早期所指的“令人惊奇的”的 *ex* 模式可在 :s (替换) 命令的行为中很好地反映出来。它对命令的每一个部分都进行提示：搜索模式、替换文本和所有的标志。

举例来说，假如用户想把文件中 *perl* 的所有实例都改为 *awk*。在其他的编辑器中，可以简单地输入 :1,\$s/perl/awk/g [RETURN]，这些命令都将显示在命令行上。下面的屏幕设置描述了当用户输入时，可以在 *vile* 冒号命令行上看到的信息。

击键 结果

:1,\$s

:1,\$s

替换命令的第一部分。

/

substitute pattern: _

vile 提示搜索的模式。此处将把先前的任何模式提示出来，以进行重用。

perl/

replacement string: _

在下一个 / 定界符处，*vile* 将提示输入替换文本。任何以前的输入文本都将提示出来，以进行重用。

awk/

(g)lobally, ([1-9])th occurrence on line,

(c)onfirm, and/or (p)rint result: _

在最后一个定界符处，*vile* 提示输入可选标志。输入所需的标志，然后按 [RETURN] 键。

为了可读，最后一个提示换行了。*vile* 在一行中打印该信息。

对所有相应的 *ex* 命令，*vile* 都按照这种方式进行。例如，读命令 (:r) 将提示输入读入的最后一个文件的名字。要再次读入该文件，只需按 [RETURN] 键即可。

最后，*vile* 的 *ex* 命令分析器比其他编辑器的功能要弱。例如，不能使用搜索模式指定行范围 (:/now/, /forever/s/perl/awk/g)，并且，没有实现移动命令 (m)。实际上，这些没有实现的功能对你并没有太大影响。

主模式

主模式就是编辑某一特定类的文件时应用选项设置的集合。这些选项大都是针对单个缓冲区的，如制表符设置。主模式概念最早是在 *vile* 7.2 中引入的，在 7.4 版和 8.0 版中得到了更充分的扩展。

vile 有一个预定义主模式 `cmode`，用于编辑 C 和 C++ 程序。在 `cmode` 下，可以使用 % 来匹配 C 的预处理条件 (`#if`、`#else` 和 `#endif`)。*vile* 会根据大括号 ({ 和 }) 的位置自动缩进源代码。并且它还会智能地安排 C 注释。`tabstop` 和 `shiftwidth` 选项也是根据某个通用主模式设置的。

使用主模式，还可以将同样的功能应用到用其他语言编写的程序中。下面由 Tom Dikey 提供的例子定义了用于编辑 Bourne shell 脚本的新主模式 `shmode`（它对所有 Bourne 风格的 shell 都适用，如 `ksh`、`bash` 或 `zsh`）。

```
define-mode sh
set shsuf "\.sh$"
set shpre "#!\\s*/.*sh\\>$"
define-submode sh comment-prefix "^\\s*/[:#]"
define-submode sh comments "^.\\s*/\\?[:#]\\s+/\\?\\s*$"
define-submode sh fence-if   "^\\s*\\<if\\>"
define-submode sh fence-elif  "^\\s*\\<elif\\>"
define-submode sh fence-else  "^\\s*\\<else\\>"
define-submode sh fence-fi   "^\\s*\\<fi\\>"
```

变量 `shsuf` (shell 的后缀) 描述了指示文件为 shell 脚本的文件名后缀。变量 `shpre` (shell 的导言) 描述了指示该文件含有 shell 代码的文件首行。`define-submode` 命令则添加选项，这些选项只能应用于相应的主模式已被设置的缓冲区。这个例子设置了智能注释格式和用于匹配 shell 程序的智能 % 命令。

过程语言

vile 的过程语言几乎未对 MicroEMACS 的过程语言进行改动。注释以分号或双引号开始，环境变量名（编辑器选项）以 \$ 开头，用户变量名以 % 开始。很多内置函数用于完成比较和测试条件，它们的名字均以 & 开头。流控制命令和一些其他

命令以 ~ 开头。@ 加字符串提示用户输入，并返回用户的答案。下面来自于 *macros.doc* 文件的例子可以让读者了解一下该语言的风格：

```

~if &sequal %curplace "timespace vortex"
    insert-string "First, rematerialize\n"
~endif
~if &sequal %planet "earth"      ;If we have landed on earth...
    ~if &sequal %time "late 20th century" ;and we are then
        write-message "Contact U.N.I.T."
    ~else
        insert-string "Investigate the situation....\n"
        insert-string "(SAY 'stay here Sara')\n"
    ~endif
~elseif &sequal %planet "luna"   ;If we have landed on our neighbor...
    write-message "Keep the door closed"
~else
    setv %conditions @"Atmosphere conditions outside? "
    ~if &sequal %conditions "safe"
        insert-string &cat "Go outside....." "\n"
        insert-string "lock the door\n"
    ~else
        insert-string "Dematerialize..try somewhere else"
        newline
    ~endif
~endif

```

可以将这些过程存入到已编号的宏中，或将它们的名字和一些键绑定起来。上面的过程在使用 Tardis *vile* 的版本时非常有用。

下面来自 Paul Fox 的例子执行 *grep*，在所有的 C 源代码文件中搜索光标处的单词。然后将结果放入以该单词命名的缓冲区中，并设置好一切，以便内置错误发现程序 (^X ^X) 能够将这个输出作为它的行列表进行访问。最后，该宏与 ^A g 进行绑定。~force 命令允许后面的命令失败但不产生错误信息：

```

14 store-macro
    set-variable %grepfor $identifier
    edit-file &cat "!egrep -n " &cat %grepfor " *.[ch]"
    ~force rename-buffer %grepfor
    error-buffer $cbufname
~endm
bind-key execute-macro-14 ^A-g

```

最后，可以将 `read-hook` 和 `write-hook` 变量分别设置为在读文件之后和写文件之前要运行的过程的名字。这样可以执行与 *elvis* 中提前或置后操作文件和 *vim* 中的自动命令相似的操作。

该语言的功能很强，包括流程控制和比较功能，以及可访问 *vile* 大量内部状态的各种变量。在 *vile* 发行版的 *macros.doc* 文件中详细描述了该语言。

各式各样的小功能

有必要提及一下其他的几个较小功能：

使用管道输入到 *vile* 中

如果 *vile* 是管道命令行上的最后一个命令，那么它将为用户创建名为 [Standard Input] 的缓冲区，并对它进行编辑。这也许是“终结所有的分页程序的一个分页程序”。

编辑 DOS 文件

当设为真时，`dos` 选项将使 *vile* 在读文件时去掉文件中行尾的回车符，在写文件时将它们重新写回。这使得在 UNIX 或 Linux 系统中很容易编辑 DOS 文件。

文本重新排版

`^A f` 命令对文本重新排版，在选定的文本中完成单词换行。它能区别 C 和 shell 注释（以 * 或 # 开头的行）及引用的 email（以 > 开头）。这和 UNIX 的 *fmt* 命令有点相似，但执行速度要比 *fmt* 快得多。

格式化信息行

变量 `modeline-format` 是 *vile* 控制模式行格式的字符串。模式行在每个窗口的底部，描述缓冲区的状态，如缓冲区的名字、当前的主模式、修改状态、插入还是命令模式等等。

该字符串包含 `printf` (3) 格式的百分号序列。例如，`%b` 代表缓冲区的名字，`%m` 代表主模式，如果已设置 `ruler`，则 `%l` 代表行数。字符串中不是格式设定符的字符都将逐字符输出。

vile 还有许多其他功能。*vi* 式的“手感”使它便于移动。*vile* 编程能力提供了很大的灵活性，对于初学者，它的交互特性和默认值的使用也许要比传统的*vi* 更友好。

源代码和支持的操作系统

vile 的正式 WWW 地址是 <http://www.clark.net/pub/dickey/vile/vile.html>, *ftp* 地址是 <ftp://ftp.clark.net/pub/dickey/vile/vile.tar.gz>。文件 *vile.tar.gz* 一直是当前版本的符号链接。

vile 是使用 ANSI C 编写的，它可以在 UNIX、VMS（包括 VAX C 和 DEC C）、MS-DOS、Win32 控制台和 Win32 GUI 以及 OS/2 上运行。

编译 *elvis* 是非常容易的。通过 *ftp* 或经过 Web 主页获得它的发布版本，对其解压缩后运行 *configure* 程序，接下来运行 *make*:

```
$ gzip -d <vile.tar.tgz | tar -xvpf-
...
$ cd vile-8.0; ./configure
...
$ make
```

配置和构建 *vile* 应该不会遇到什么问题。运行 *make install* 安装它。

注意：如果希望使用某个 Linux 控制台，并且希望能够语法着色，那么你可以在运行 *configure* 时加上下面的参数：*--with-screen=ncurses*

如果需要报告 *vile* 中的漏洞或问题，可以发邮件到 *vile-bugs@foxharp.boston.ma.us*。这是报告漏洞的最好方法。也可以通过 *dickey@clark.net* 邮箱直接与 Tom Dickey 联系。

第三部分

附录

第三部分为 *vi* 用户提供了感兴趣的参考材料，该部分包括一些附录：

- 附录一，快速参考
- 附录二，*ex* 命令
- 附录三，设置选项
- 附录四，问题列表
- 附录五，*vi* 和 Internet





附录一

快速参考

该附录列出了各项 *vi* 命令和 *ex* 命令的功能。

表 A-1 移动命令

命令	功能
字符	
h、j、k、l	左、下、上、右 (\leftarrow 、 \downarrow 、 \uparrow 、 \rightarrow)
文本	
w、W、b、B	按单词向前向后移动
e、E	移动到单词的结尾
)、(移动到下句、前句的开始
}、{	移动到下段、前段的开始
]、[[移动到下节、前节的开始
行	
[RETURN]	下行的第一个非空格字符
0、\$	当前行的最前、最后位置
^	当前行的第一个非空格字符
+、-	下行、前行的第一个非空格字符
n!	当前行的第 n 列
H	屏幕的顶行

表 A-1 移动命令（续）

命令	功能
M	屏幕中间的行
L	屏幕的底行
nH	顶行后的第 n 行
hL	底行前的第 n 行
滚动	
[CTRL-F]、[CTRL-B]	向前、向后滚动一屏
[CTRL-D]、[CTRL-U]	向下、向上滚动半屏
[CTRL-E]、[CTRL-Y]	窗口底部、顶部的多显示一行
z [RETURN]	将带有光标的行重定位到屏幕顶部
z .	将带有光标的行重定位到屏幕中部
z -	将带有光标的行重定位到屏幕底部
[CTRL-L]	重新刷新屏幕（不滚动）
搜索	
/pattern	向前搜索 pattern
?pattern	向后搜索 pattern
n、N	以相同、相反方向重复上一搜索
/、?	向前、向后重复上一搜索
fx	在当前行中向前搜索字符 x
Fx	在当前行中向后搜索字符 x
tx	在当前行中向前搜索到 x 前面的字符
Tx	在当前行中向前搜后到 x 后面的字符
;	重复上一当前行搜索
,	以反方向重复上一当前行搜索
行号	
[CTRL-G]	显示当前行号
nG	移动到行号为 n 的行
G	移动到文件的最后一行
:n	移动到文件中的第 n 行
标识位置	
mx	把当前位置标识为 x
`x	将光标移动到 x 标识处
``	返回到上一标识或上下文
'x	移动到包含 x 标识的行的开始位置
''	返回到包含上一标识的行的开始位置

表 A-2 编辑命令

命令	功能
输入	
i、a	在光标前、后插入文本
I、A	在行的开始前、结尾后插入文本
o、O	在光标的下面、上面插入新行并等待输入文本
修改	
r	替换字符
cw	修改单词
cc	修改当前行
cmotion	修改光标和 <i>motion</i> 目标之间的文本
C	修改到行尾
R	在字符上输入（覆盖）
s	替换：删除字符并插入新文本
S	替换：删除当前行并插入新文本
删除、移动	
x	删除光标处的字符
X	删除光标前面的字符
dw	删除单词
dd	删除当前行
dmotion	删除光标和 <i>motion</i> 目标之间的文本
D	删除到行尾
p、P	在光标后、前粘贴已删除的文本
"np	把 <i>n</i> 删除缓冲区中的文本粘贴到光标后（支持前 9 次删除操作）
复制	
yw	复制（拷贝）单词
yy	复制当前行
"ayy	复制当前行到命名缓冲区 a (a-z) 中，大写的名字后追加文本
ymotion	复制光标和 <i>motion</i> 目标之间的文本
p、P	在光标后、前粘贴已复制的文本
"aP	把缓冲区 a 中的文本粘贴到光标的前面 (a-z)
其他命令	
.	重复上一编辑命令
u、U	取消上次编辑操作，恢复当前行
J	合并两行

表 A-2 编辑命令（续）

命令	功能
<i>ex</i> 编辑命令	
:d	删除行
:m	移动行
:co 或 :t	复制行
:., \$d	从当前行一直删除到文件尾
:30,60m0	把第 30 行到第 60 行移动到文件顶部
:., /pattern/co\$	将从当前行到包含 <i>pattern</i> 的行中间的所有内容复制到文件尾

表 A-3 是出命令

命令	功能
zz	保存修改后的文本并退出该文件
:x	保存修改后的文本并退出该文件
:wq	保存文本并退出该文件
:w	写（保存）文本
:w!	写（保存）文本，忽略保护
:30,60w <i>newfile</i>	把第 30 行到第 60 行写到 <i>newfile</i> 文件中
:30,60w>> <i>file</i>	把第 30 行到第 60 行追加到 <i>file</i> 文件中
:w %. <i>new</i>	把当前缓冲区名为 <i>file</i> 的文件改写为 <i>file.new</i>
:q	退出文件
:q!	退出文件，忽略保护
Q	退出 vi 并调用 ex
:e <i>file2</i>	不离开 vi 编辑 <i>file2</i> 文件
:r <i>newfile</i>	将 <i>newfile</i> 文件的内容读取到当前文件中
:n	编辑下个文件
:e!	将当前文件返回到上次写（保存）时的版本
:e #	编辑可替换的文件
:vi	从 ex 中调用 vi 编辑器
:	从 vi 编辑器中调用一个 ex 命令
%	当前的文件名（替换到 ex 命令行中）
#	可替换的文件名（替换到 ex 命令行中）

表 A-4 Solaris vi 命令模式下的标志命令

命令	功能
^]	在 <i>tags</i> 文件中寻找光标下的标识符的位置，并移动到那个位置。如果标志入栈被设置，当前位置将被自动放入标志栈中
^T	返回到标志栈中前面的位置，即弹出一个单元

表 A-5 命令行参数

命令	功能
vi <i>file</i>	在 <i>file</i> 文件上调用 vi 编辑器
vi <i>file1 file2</i>	按文件序列调用 vi 编辑器
view <i>file</i>	以只读方式在 <i>file</i> 文件上调用 vi 编辑器
vi -R <i>file</i>	以只读方式在 <i>file</i> 文件上调用 vi 编辑器
vi -r <i>file</i>	在系统瘫痪后恢复文件和最近的编辑
vi -t <i>tag</i>	寻找标志并在它的定义处开始编辑
vi -w <i>n</i>	把窗口大小设置为 <i>n</i> ；在慢连接时很有用
vi + <i>file</i>	打开 <i>file</i> 并跳转到最后一行
vi + <i>n</i> <i>file</i>	打开 <i>file</i> 并直接跳转到第 <i>n</i> 行
vi -c <i>command file</i>	打开 <i>file</i> ，执行命令，该命令通常为搜索命令或行号（POSIX）
vi +/ <i>pattern file</i>	打开 <i>file</i> 直接到 <i>pattern</i> 处
ex <i>file</i>	在 <i>file</i> 上调用 ex 编辑器
ex - <i>file < script</i>	在 <i>file</i> 上调用 ex 编辑器，从 <i>script</i> 中执行命令；不显示提供消息的信息和提示
ex -s <i>file < script</i>	在 <i>file</i> 上调用 ex 编辑器，从 <i>script</i> 中执行命令；不显示提供消息的信息和提示（POSIX）

表 A-6 其他的 ex 命令

命令	功能
缩写注	
:map <i>x sequence</i>	把按键 <i>x</i> 定义为命令 <i>sequence</i> , <i>x</i> 可以是多个字符
:map! <i>x sequence</i>	把 <i>x</i> 定义为插入模式下的命令 <i>sequence</i>
:unmap <i>x</i>	取消映射 <i>x</i>
:unmap! <i>x</i>	取消插入模式下的映射 <i>x</i>
:ab <i>abbr phrase</i>	把 <i>phrase</i> 缩写为 <i>abbr</i> ; 当在输入模式下输入 <i>abbr</i> 时, 它就被扩展为完整的单词或短语
:unab <i>abbr</i>	取消 <i>abbr</i> 缩写
定制环境 ^注	
:set <i>option</i>	激活 <i>option</i>
:set <i>option=value</i>	为 <i>option</i> 赋值
:set <i>nooption</i>	停用 <i>option</i>
:set	显示用户设置的选项
:set all	显示当前所有的选项设置, 包括默认的和用户设置的
:set <i>option?</i>	显示 <i>option</i> 的值
访问 UNIX	
:sh	调用 shell
^D	从 shell 返回到编辑器
:! <i>command</i>	给出 UNIX 命令
:n, m! <i>command</i>	用 UNIX 的 <i>command</i> 过滤第 <i>n</i> 至第 <i>m</i> 行的内容
:r ! <i>command</i>	把 UNIX <i>command</i> 的输出读取到当前文件中

注：在 *.exrc* 文件中，省略在 *ex* 命令开始处的冒号。

附录二

ex 命令

本附录描述了 *ex* 命令的语法，然后列出了按字母顺序排列的命令表。

命令语法

为了从 *vi* 中输入 *ex* 命令，可使用这种格式：

```
:[address] command[options]
```

address 是表示 *command* 目标的行号或行的范围。如果没有指定行号，(通常) 当前行就是该命令的目标。

地址符号

在 *ex* 命令语法中，可以用表 B-1 中列出的任何格式指定 *address*。

表B-1 ex 的地址语法

地址	范围
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, 当前行重设为第 x 行
0	文件的顶部
.	当前行
n	绝对行数 n
\$	最后一行
%	所有的行; 等同于 1, \$
x-n	x 前面的 n 行
x+n	x 后面的 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
' '	前一标记
/pat/ 或?pat?	向前或向后跳转到包含 pat 匹配文本的行

选项符号

在 ex 命令语法中, 选项可能是下列任何一个:

! 指示不同格式的命令, 覆盖正常的行为。

count

该命令将被重复的次数。count 不能位于命令之前, 因为 ex 命令前面的数字将作为行地址。d3 表示删除从当前行开始的 3 行; 3d 则表示删除第三行。

file

命令对象的文件名。% 代表当前文件; # 代表上一文件。

按字母顺序排列的命令表

在本节中, *ex* 命令的全名作为关键字列出。每个关键字的右边或下边是语法, 其中使用了该命令最短的缩写。语法下面是简要的描述。

abbrev

`ab [string text]`

定义 *string* 输入时将转换为 *text*。如果 *string* 和 *text* 都没有被指定, 那么列出所有的缩写。

append

`[address] a[!]`

text

在指定的 *address* 处添加 *text*, 如果没有指定地址, 则在当前地址处添加。加上! 可转换在输入中将使用的 autoindent 设置。也就是说, 如果启动了 autoindent, ! 就会禁用它。

args

`ar`

显示参数列表中的成员 (命令行上的文件名), 并在方括号 ([]) 中显示当前参数。

change

`[address] c[!]`

text

用 *text* 替换指定的行。加上! 可在 *text* 输入过程中转换 autoindent 的设置。

copy

`[address] co destination`

把 *address* 中包括的行复制到指定的 *destination* 地址处。命令 t (“to”的缩写) 是 copy 的同义词。

delete

[*address*] d[*buffer*]

删除 *address* 中包括的行。如果 *buffer* 被指定，把这些文本保存或追加到命名的缓冲区中。缓冲区名字是小写的 a-z 字母。大写字母会把文本追加到该缓冲区中。

edit

e[!] [+*n*] [*filename*]

开始对 *filename* 文件进行编辑。如果没有给出 *filename*，在当前文件的副本上进行。加上!后，即使当前文件从上次修改以来没有保存过，也可以对新文件进行编辑。使用 +*n* 参数，可在第 *n* 行上开始编辑。或者 *n* 可能是个模式，格式为 /*pattern*。

file

f [*filename*]

把当前的文件名修改为 *filename*，将使其作为“没有编辑过”的文件。如果没有指定文件名，就显示文件的当前状态。

global

[*address*] g[!]/*pattern*/[*commands*]

在所有包含 *pattern* 的行上执行 *commands*，如果指定了地址 *address*，那么就是该范围内的所有行。如果没有指定 *commands*，就显示所有符合条件的行。加上!可在所有不包含 *pattern* 的行上执行 *commands*。

insert

[*address*] i[!]

text

.

在指定地址之前的行上插入 *text*，如果没有指定地址就在当前位置。加上!可在输入 *text* 期间改变 autoindent 的设置。

join

[*address*] j[!][*count*]

把文本放置在一行中指定的范围内，并且空白在点（.）后被调整为两个空

格字符，在)后被调整为零个空格字符，其他情况调整为一个空格字符。加上!可阻止空白调整。

k [address] k char

使用*char*（单个的小写字母）标识指定的地址，以后可以使用‘x’返回到该行。*k*等同于*mark*。

list [address] l [count]

显示指定的行，并把制表符显示为^I，行尾显示为\$。

map

map char commands

把指定的命令序列定义为可视模式下名为*char*的宏，*char*通常是代表键盘上一个功能键的#n序列，或者是一个或多个字符。

mark

[address] ma char

使用*char*（单一的小写字母）标识指定的行，稍后使用‘x’返回到该行。

move

[address] m destination

由*address*指定的行移动到*destination*地址。

next

n[!] [[+n]] filelist

编辑命令行参数列表中的下一个文件。使用*args*可列出这些文件。如果提供了*filelist*，就使用*filelist*代替当前的参数列表并开始编辑第一个文件。使用*+n*参数，可在第*n*行开始编辑。或者*n*可能是个模式，其格式为*/pattern*。

number

[address] nu [count]

显示*address*指定的每行，并在前面加上缓冲区行号。可使用#作为*number*的另一个缩写。

open

[address] o [/pattern/]

在 *address* 指定的行或与 *pattern* 匹配的行处进入 *open* 模式 (*vi*)。可使用 Q 退出开放模式。

preserve

pre

在系统将要瘫痪时保存当前的编辑缓冲区。

print

[*address*] p [*count*]

显示 *address* 指定的行，P 是另一个缩写。

put

[*address*] pu [*char*]

从 *char* 指定的命名缓冲区中把前面删除或复制的行恢复到 *address* 指定的行，如果没有指定 *char*，就恢复最后一次删除或复制的文本。

quit

q[!]

中止当前的编辑会话。使用! 可忽略上次保存以来所做的修改。如果该编辑会话包括参数列表中仍没有被访问过的文件，那么就输入 q! 或输入两次 q 退出。

read

[*address*] r *filename*

把 *filename* 文件中的文本复制到 *address* 指定的行的后面。如果没有指定 *filename*，就使用当前的文件名。

read

[*address*] r ! *command*

读取 *command* 的输出到 *address* 指定的行后的文本。

recover

rec [*filename*]

从系统保存区域中恢复 *filename* 文件。

rewind

rew[!]

重绕参数列表并开始编辑列表中的第一个文件。即使当前文件在上次修改后没有被保存，加上!也要重绕。

set se parameter parameter2

使用每个 *parameter* 给选项赋值，如果没有提供 *parameter*，就显示所有已改变默认值的选项。对开关选项来说，每个 *parameter* 可以用“*option*”或“*nooption*”这样的短语来表达。其他的选项可使用语法“*option=value*”进行赋值。使用 *set option?* 将显示 *option* 的值。

shell

sh

创建新 shell，shell 结束后继续编辑。

source

so filename

从 *filename* 文件中读取 *ex* 命令并执行。

substitute

[address] s [/pattern/repl/] [options]

使用 *repl* 替换指定行中的每个 *pattern* 实例。如果没有指定 *pattern* 和 *repl*，就重复最后一次替换操作。*g* 选项可替换该行中的 *pattern* 的所有实例。*c* 选项会在每次修改前提示确认信息（需要指出该命令名不能在 Solaris 2.6 的 *vi* 中工作）。请参考第六章“全局替换”以了解更多细节内容。

t [address] t destination

把 *address* 中包含的行复制到指定的 *destination* 地址处。*t* 等价于 *copy*。

tag [address] ta tag

将焦点从编辑切换到 *tag*。

unabbreviate

una word

从缩写列表中删除 *word*。

undo

u

恢复最后一次编辑命令所做的修改。

unmap

unm *char*

从宏列表中删除 *char*。

v [address] v/pattern/ [commands]

在所有不包含 *pattern* 的行上执行 *commands*。如果没有指定 *commands*, 就显示所有这样的行。v 等价于 g!。

version

ve

显示编辑器的当前版本号和编辑器上次被修改的日期。每个副本都显示正确的信息。

visual

[address] vi [*type*] [*count*]

在 *address* 指定的行处进入可视模式。使用 Q 退出。*type* 可以是 - 、^ 或 . 中的一个（参考 z 命令）。*count* 指定了初始的窗口大小。

visual

vi [+n] [*filename*]

在可视模式下开始编辑 *filename* 文件。

write

[address] w[!] [[>>] *filename*]

把 *address* 指定的行写到 *filename* 文件中，如果没有指定 *address*, 就写缓冲区中的全部文本。如果还没有指定 *filename*, 就把缓冲区中的内容保存到当前文件中。如果使用了 >> *filename*, 就把内容写到指定的 *filename* 文件的结尾。加上!可强迫编辑器覆盖 *filename* 文件中的任何现有内容。

write

[address] w !*command*

把 *address* 指定的行写入 *command*。

wq**wq [!]**

写并退出该文件。该文件总是可写的。

xit x

如果从上次保存后已修改了缓冲区就写文件，然后退出。

yank**[address] y [char] [count]**把 *address* 指定的行放到 *char* 指定的命名缓冲区中，如果没有指定 *char* 就放到公共缓冲区中。**z [address] z [type] [count]**将 *address* 指定的行的文本显示在窗口顶部。*type* 可以是下面的任意一个：

- + 把指定的行放置在窗口顶部（默认）。
- 把指定的行放置在窗口底部。
- . 把指定的行放置在窗口中央。
- ^ 显示前一窗口。
- = 把指定的行放置在窗口中央并使该行作为当前行。

count 指定要显示的行数。**! [address] !command**在 shell 中执行 *command*。如果指定了 *address*，就把 *address* 包含的行作为 *command* 的标准输入，并用输出和错误输出替换这些行（这被称为用 *command* 过滤文本）。**= [address] =**显示 *address* 所表示的行的行号，默认值是最后一行的行号。**< >****[address] < [count]**

或

[address] > [count]

按指定方向移动 *address* 指定的行。在移动行时只有开头的空格和制表符才被添加或删除。*shiftwidth* 选项控制着被转移的列数。重复 < 或 > 会增加转移次数，例如，:>>> 的转移次数是:> 的 3 倍。

address

address

显示 *address* 指定的行。

RETURN

RETURN

显示文件中的下一行。

& [*address*] & [*options*] [*count*]

重复前面的替换命令。

~ [*address*] ~ [*count*]

使用最近的 s (替换) 命令中的替换模式代替上次使用的正则表达式 (即使来自搜索也不来自 s 命令)。参考第六章中的“更多的替换技巧”一节可了解更多细节。



附录三

设置选项

本附录对有关 Solaris 2.6 *vi*、*nvi* 1.79、*elvis* 2.0、*vim* 5.1 和 *vile* 8.0 的重要 set 命令选项进行了描述。

Solaris 2.6 vi 的选项

表 C-1 对重要的 set 命令选项进行了简要的描述。在第一列中，选项按字母的顺序列出；如果选项可被缩写，就把缩写显示在圆括号内。第二列显示了 *vi* 在没有调用明确的 set 命令（手工或在 .exrc 文件中）时所使用的默认设置。最后一列描述了该选项的功能及其在什么情况下需要设置。

表 C-1 Solaris 2.6 vi 的选项

选项	默认设置	描述
autoindent (ai)	noai	在插入模式下，对每行按与上行或下行的同样标准进行缩进，与 shiftwidth 选项结合使用
autoprint (ap)	ap	在每个编辑命令后显示所做的修改 (对全局替换来说，显示最后一个替换)

表 C-1 Solaris 2.6 vi 的选项（续）

选项	默认设置	描述
autowrite (aw)	noaw	在用:n 打开另一个文件或用:! 给出 UNIX 命令前如果文件被修改就自动写（保存）它
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）
directory (dir)	/tmp	指定 ex/vi 存储缓冲区文件的目录（该目录必须是可写的）
edcompatible	noedcompatible	记下最近的替换命令（全局的、确认的）所使用的标记，以便在下次替换命令中使用它们。不管什么名字，没有实际的 ed 版本能真正地这样工作
errorbells (eb)	errorbells	当发生错误时响铃
exrc (ex)	noexrc	允许执行位于用户主目录之外的.exrc 文件
hardtabs (ht)	8	为终端部件的制表符指定范围
ignorecase (ic)	noic	在搜索期间忽略大小写
lisp	nolisp	以正确的 lisp 格式插入缩进, ()、{}、[[和]] 都被修改为具有 lisp 的含义
list	nolist	把制表符显示为 ^I，用 \$ 标识行尾（使用 list 分辨尾部的字符是 tab 还是空格）
magic	magic	通配符字符 .(点)、*(星号) 和 [](方括号) 在模式中有特殊含义
mesg	mesg	在 vi 中进行编辑时允许系统信息显示在终端上
novice	nonovice	要求使用长的 ex 命令名，如 copy 或 read
number (nu)	nonu	在编辑会话期间在屏幕的左边显示行号

表 C-1 Solaris 2.6 vi 的选项 (续)

选项	默认设置	描述
open	open	允许从 <i>ex</i> 中进入 <i>open</i> 或 <i>visual</i> 模式，虽然 Solaris 2.6 vi 中没有该选项，但是传统上它在 vi 中，所以可能会在当前所使用的 UNIX 版本的 vi 中
optimize (opt)	noopt	显示多行时取消行尾的回车，显示带有引导空白（空格或制表符）的行时加速到哑终端的输出
paragraphs (para)	IPLPPPQP LIpplpipbp	为移动使用 { 或 } 定义段标志，默认值中的字符对是表示段开始的 <i>troff</i> 宏的名字
prompt	prompt	在给出 vi 的 Q 命令时显示 ex 命令提示符 (:)
readonly (ro)	noro	如果你不在写命令后使用 ! (与 w、zz 或 autowrite 一起工作)，任何对文件的写（保存）都会失败
redraw (re)		进行编辑时刷新屏幕（即插入模式立即覆盖现有字符和被删除的行）。默认情况下取决于行速和终端类型。noredraw 被用于慢速的哑终端上：被删除的行显示为 @，所插入的文本在按下 [ESC] 之前显示为对现有文本的覆盖
remap	remap	允许嵌套的映射序列
report	5	如果做了会影响不少于特定数量的行的编辑，那么就在状态行上显示信息。例如，6dd 会报告 “6 lines deleted”的信息
scroll	{1/2 窗口}	用 ^D 和 ^U 命令滚动的行数
sections (sect)	SHNHH HU	为使用 [[和]] 移动定义节标识。默认值中的字符对是表示节开始的 <i>troff</i> 宏的名字

表 C-1 Solaris 2.6 vi 的选项 (续)

选项	默认设置	描述
shell (sh)	/bin/sh	shell 转义 (:!) 和 shell 命令 (:sh) 所使用的 shell 路径。默认值从 shell 环境中获得，它随着系统的不同而变化
shiftwidth (sw)	8	定义在使用 autoindent 选项时反向 (^D) 制表符中的空格数目，也使用于 <> 和 >> 命令
showmatch (sm)	nosm	在 vi 中，当输入) 或 } 时，光标会暂时移动到相匹配的 (或 { (如果没有相匹配的，就发出表示错误信息的铃声)。对编程非常有用
showmode	noshowmode	在插入模式下，在提示行上显示指示当前正使用的输入类型的信息。例如，“OPEN MODE”或“APPEND MODE”
slowopen (slow)		在输入期间禁止显示。默认情况下取决于线路速度和终端类型
tabstop (ts)	8	指定 TAB 在编辑会话中缩进的字符数目 (打印机仍使用 8 字符的系统制表符)
taglength (tl)	0	为标志指定有意义的字符数目。默认值 (0) 表示所有的字符都是有意义的
tags	tags /usr/lib/tags	指定包含标志的文件的路径名 (请参考 UNIX 的 ctags 命令，在默认情况下，vi 在当前目录和 /usr/lib/tags 中搜索标志文件)
tagstack	tagstack	允许标志位置进入标志栈
term		设置终端类型
terse	noterse	显示较短的错误信息
timeout (to)	timeout	键盘映射在 1 秒钟后超时 ^注

表 C-1 Solaris 2.6 vi 的选项（续）

选项	默认设置	描述
ttytype		设置终端类型，这只是 term 的另一个名字
warn	warn	显示 “No write since last change” 警告信息
window (w)		在屏幕上显示文件中特定数目的行。默认情况下取决于线路速度和终端类型
wrapscan (ws)	ws	搜索在文件的两端绕回
wrapmargin (wm)	0	定义右边距的空白。如果大于 0，就自动输入回车进行断行
writeany (wa)	nowa	允许保存到任何文件

注：当要映射几个键（例如，:map zzz 3dw）时，你可能想使用 notimeout，否则就需要在 1 秒钟内输入 zzz。如果有插入模式下的光标键映射（例如，:map! ^[OB ^[ja），则应该使用 timeout；否则 vi 将在你输入了另一个键后才会对 [ESC] 做出反应。

nvi 1.79 的选项

共有 78 个选项可影响 nvi 1.79 的行为，表 C-2 对其中最重要的一些进行了总结。大部分在表 C-1 中描述过的选项在这里没有重复。

表 C-2 nvi 1.79 的设置选项

选项	默认设置	描述
backup		描述所使用的备份文件名的字符串。在写入新数据之前把文件的当前内容保存到该文件中。例如，“N%.bak”可使 nvi 包含文件尾部的版本号；版本号将一直增加
cdpath	环境变量 CDPATH 或当前目录	:cd 命令所使用的搜索路径

表 C-2 nvi 1.79 的设置选项（续）

选项	默认设置	描述
cedit		当该串的第一个字符被输入到冒号命令行时， <i>nvi</i> 就打开一个新窗口显示可以编辑的历史命令。在任何指定的行输入 RETURN 就会执行该行。 ESC 对该选项来说是个很好的选择（使用 ^v ^[输入它）
comment	nocomment	如果第一个非空行以 /*、 // 或 # 开始， <i>nvi</i> 就在显示该文件之前跳过注释文本。这可避免过长的显示
directory (dir)	环境变量 TMPDIR 或 /tmp	该目录存放 <i>nvi</i> 的临时文件
extended	noextended	使用 <i>egrep</i> 类型的扩展正则表达式进行搜索
filec		当该串的第一个字符被输入到冒号命令行时， <i>nvi</i> 就把光标前面空格隔开的单词作为有 * 扩展对待，并进行 shell 格式的文件名扩展。 ESC 对该选项来说是个很好的选择（使用 ^v ^[输入它）。当该字符与 cedit 选项的值相同时，只有该字符作为冒号命令行上的第一个字符输入时才进行命令行编辑
iclower	noiclower	只要搜索模式不包含大写字母，所有的正则表达式搜索都忽略大小写
leftright	noleftright	长行在屏幕上从左向右滚动，而不是换行
lock	lock	<i>nvi</i> 试着对文件加排它锁，对那些不能被加锁的文件创建只读会话
octal	nooctal	用八进制代替十六进制显示未知的字符
path		冒号隔开的目录列表， <i>nvi</i> 将在这里寻找要编辑的文件

表 C-2 nvi 1.79 的设置选项（续）

选项	默认设置	描述
readdir	/var/tmp/vi.recover	存储恢复文件的目录
ruler	noruler	显示光标所在的行和列
searchincr	nosearchincr	进行增量搜索
secure	nosecure	禁止使用文本过滤（:r!、:w!）对外部程序的访问，使 vi 模式下的! 和 ^Z 命令以及 ex 模式下的!、shell、stop 和 suspend 命令不可用。一旦设置就不能被修改
shellmeta	~{[*?\${'`'}\`}	这些字符中的任何一个出现在 ex 命令中的文件名参数中时，该参数要由 shell 选项指定的程序进行扩展
showmode (smd)	noshowmode	在状态行中显示指示当前模式的字符串。如果文件已被修改会显示一个 *
sidescroll	16	当 leftright 为真时，屏幕向左或向右移动的列数
taglength (tl)	0	为标志指定有效字符的个数，默认值（0）表示所有字符都是有效的
tags (tag)	tags /var/db/libc.tags /sys/kern/tags	可能的标志文件列表
tildeop	notildeop	~ 命令要进行相关的移动，不只是在前面计数
wraplen (wl)	0	除了指定行在距离右边界多少字符时被分开外，它与 wrapmargin 选项相同。wrapmargin 的值会覆盖 wraplen 的值

elvis 2.0 的选项

共有 144 个选项可影响 elvis 2.0 的行为，表 C-3 对其中最重要的一些进行了总结。大部分选项在表 C-1 中描述过，这里没有重复。

表 C-3 elvis 2.0 的设置选项

选项	默认设置	描述
autoiconify (aic)	noautoiconify	当调用新窗口时，最小化以前的窗口
backup (bk)	nobackup	在把当前文件写到磁盘之前备份该文件 (<i>xxx.bak</i>)
binary (bin)		缓冲区的数据不是文本，该选项被自动设置
boldfont (xfb)		粗体字体的名称
bufdisplay (bd)	normal	缓冲区的默认显示模式 (hex、html、man、normal 或 sysntax)
ccprg (cp)	cc (\$1?\${1:\$2})	调用:cc 的 shell 命令
commentfont (cfont)		注释所使用的字体名称
directory (dir)		存储临时文件的目录，默认值取决于当前使用的系统
display (mode)	normal	当前显示模式的名称，由:display 命令设置
elvispath (epath)		搜索配置文件的目录列表，默认值取决于当前使用的系统
focusnew (fn)	focusnew	迫使键盘焦点跳转到新窗口中
functionfont (ffont)		函数名所使用的字体名称
gdefault (gd)	nogdefault	使替换命令改变所有的实例
home (home)	\$HOME	文件名中 ~ 代表的主目录
italicfont (xfi)		斜体字体的名称
keywordfont (kfont)		保留单词所使用的字体名称
lpcolumns (lpcols)	80	打印页的宽度；用于:lpr
lpCrLf (lpc)	nolpcrlf	打印机需要表示文件中新行的 CR-LF；用于:lpr
lpFormFeed (lpff)	nolpformfeed	在最后一页之后发送格式信息；用于:lpr

表 C-3 elvis 2.0 的设置选项 (续)

选项	默认设置	描述
lplines (lprows)	60	打印页的长度; 用于:lpr
lppaper (lpp)	letter	用于 PostScript 打印机的纸张大小 (信纸、a4、…); 用于:lpr
lpout (lpo)		打印机文件或过滤器, 用于:lpr。典型值可能是!lpr。默认情况下取决于当前使用的系统
lptype (lpt)	dumb	打印机类型, 用于:lpr。该值应该是ps、ps2、epson、pana、ibm、hp、cr、bs 或 dumb 中的一个
lpwrap (lpw)	lpwrap	模拟自动换行; 用于:lpr
makeprg (mp)	make \$1	表示:make 的 shell 命令
normalfont (xfn)		正常字体的名称
otherfont (ofont)		其他符号所使用的字体
preffont (pfont)		预编译命令所使用的字体
ruler (ru)	noruler	显示光标所在的行和列
safer (trapunsafe)	nosafer	用于安全方面; 可以使用:safer 命令设置它, 不能直接对它进行设置
showmarkups (smu)	noshowmarkups	用于 man 和 html 方式, 只显示光标位置处的组成
sidescroll (ss)	0	向一旁滚动的数量。0 等同于 vi, 进行自动换行
stringfont (sfont)		字符串所使用的字体
taglength (tl)	0	为标志指定有效的字符数目, 默认值(0) 表示所有字符都有效
tags (tagpath)	tags	可能的标志文件列表
tagstack (tsk)	tagstack	在标志栈中记下标志搜索的位置
undolevels (ul)	0	可回退命令的数目, 0 等同于 vi。可以把该值设置得大一些

表 C-3 elvis 2.0 的设置选项（续）

选项	默认设置	描述
variablefont (vfont)		变量所使用的字体
warpback (wb)	nowarpback	一退出就把指针移回到调用 <i>elvis</i> 的 <i>xterm</i>
warpto (wt)	don't	^{^W} ^{^W} 如何强制指针移动: don't 表示不移动, scrollbar 把指针移动到滚动条, origin 把指针移动到左上角, corners 把指针移动到距离当前光标位置最近和最近的角。这会强迫 X 显示到全景, 以确保窗口全部在屏幕上

vim 5.1 的选项

vim 5.1 共有 170 个选项可影响它的行为, 表 C-4 对其中最重要的一些进行了总结。大部分选项在表 C-1 中描述过, 这里没有重复。

下表中的总结很简单, 更多有关各个选项的信息可在 *vim* 的联机帮助中找到。

表 C-4 vim 5.1 的设置选项

选项	默认设置	描述
background (bg)	dark 或 light	<i>vim</i> 尝试使用那些适合于特殊终端的背景和前景颜色
backspace (bs)	0	控制是否可以在换行符/或插入的开始位置进行回退。值为: 0 表示与 <i>vi</i> 兼容, 1 表示可在换行符上回退, 2 表示可在插入的开始位置回退, 3 表示两者都允许
backup (bk)	nobackup	在覆盖文件之前进行备份, 在成功地写文件之后抛弃它。要在写文件时进行备份, 可使用 writebackup 选项

表 C-4 vim 5.1 的设置选项（续）

选项	默认设置	描述
backupdir (bdir)	., ~/tmp/, ~/	备份文件所使用的目录列表，用逗号隔开。备份文件将被创建在可用列表的第一个目录下。如果为空，就不能创建备份文件。名称.（点号）表示与被编辑的文件是同一目录
backupext (bex)	~	该字符串被追加在文件名的后面作为备份文件的名字
binary (bin)	nobinary	改变其他的一些选项使二进制文件编辑起来更容易。这些选项原有的值被记下并在 bin 转换回来后重新使用。每个缓冲区都有它自己对已保存选项值的设置。在编辑二进制文件前应该设置该选项。也可以使用 -b 命令行参数进行设置
cindent (cin)	nocindent	启动自动智能的 C 程序缩进
cinkeys (cink)	0{,0},:,0#,!^F, o,O,e	在插入模式下被输入时可使当前行重新缩进的键的列表。只有 cindent 被设置时才起作用
cinoptions (cino)		影响 cindent 在 C 程序中缩进行的方式。详细信息请参考联机帮助
cinwords (cinw)	if, else, while, do, for, switch	当 smartindent 或 cindent 被设置时，这些关键字在下一行进行额外的缩进。对 cindent 来说，这只在适当的位置进行（在{...}内）
comments (com)		用逗号隔开的可以作为注释行开始的字符串列表。详细信息请参考联机帮助
compatible (cp)	cp, 当发现 .vimrc 文件时为 nocp	使 vim 的行为在更多方面更像 vi，这里没有详细描述。默认情况下是已设置的，以避免发生意外。拥有 .vimrc 文件可取消 vi 兼容；通常这是需要的结果
cpoptions (cpo)	aABceFs	一列单个字符标识，每个都指示在不同的方面 vim 将是否确切地与 vi 相似。当为空时，使用 vim 的默认设置。详细信息请参考联机帮助

表 C-4 vim 5.1 的设置选项（续）

选项	默认设置	描述
define (def)	<code>^#\s*define</code>	描述宏定义的搜索模式。默认值表示 C 程序。对 C++ 来说，应使用 <code>^\\(#\\s*define\\ [a-z]*\\s*const\\s*[a-z]*\\)</code> 。当使用 :set 命令时，需要加倍使用反斜杠
directory (dir)	<code>., ~tmp, /tmp</code>	用于交换文件的目录名列表，用逗号隔开。交换文件将被创建在第一个可用的目录中。如果为空，就不使用交换文件，也就不可能恢复。名字 . (点号) 表示把交换文件放在被编辑文件所在目录下。建议在列表中首先使用 .，目的是在两次编辑同一文件时产生警告
equalprg (ep)		= 命令使用的外部程序名。当该选项为空时，使用内部格式化函数
errorfile (ef)	errors.err	用于快速修改模式的错误文件名。当使用 -q 命令行参数时，errorfile 被设置为下一个参数
errorformat (efm)	(太长，因而无法显示)	对用于错误文件中行的格式的 scanf 风格的描述
expandtab (et)	noexpandtab	在输入制表符时，把它扩展为适当数目的空格
fileformat (ff)	unix	在读 / 写当前缓冲区时对行结束方式的描述。可能的值为 dos (CR-LF)、unix (LF)、和 mac (CR)。vim 通常自动对此进行设置
fileformats (ffs)	dos, unix	列出 vim 在读取时将试着对文件使用的行结束方式。多个名字可在读取文件时启动对行尾的自动检测
formatoptions (fo)	vim 默认为: tcq, vi 默认为: vt	一系列描述自动格式化如何进行的字母。详细信息参考联机帮助
gdefault (gd)	nogdefault	使替换命令改变所有的实例
guifont (gfn)		在启动 vim 的 GUI 版本时可使用的字体列表，用逗号隔开

表 C-4 vim 5.1 的设置选项（续）

选项	默认设置	描述
hidden (hid)	nohidden	当缓冲区被从窗口卸载时，对它进行隐藏而不是抛弃
hlsearch (hls)	nohlsearch	高亮显示最近的搜索模式的所有匹配
history (hi)	<i>vim</i> 默认为：20, <i>vi</i> 默认为：0	控制多少 <i>ex</i> 命令、搜索串和表达式存储在历史命令中
icon	noicon	<i>vim</i> 试着改变与正在运行的窗口相关联的图标名。iconstring 选项优先于该选项
iconstring		用于窗口的图标名的字符串值
include (inc)	^# \s*include	指定用于发现 include 命令的搜索模式。默认值是用于 C 程序的
incsearch (is)	noincsearch	启动增量搜索
isfname (isf)	@,48-57,/.,-,_,+,,,\$,:,~	可以包含在文件和路径名中的字符列表。非 UNIX 系统有不同的默认值。@ 字符代表字母表上的任何字符。它也被用在下面的其他 isxxxx 选项中
isident (isi)	@,48-57,_,192-255	可以包含在标识符中的字符列表。非 UNIX 系统可能会有不同的默认值
iskeyword (isk)	@,48-57,_,192-255	可以包含在关键字中的字符列表。非 UNIX 系统可能会有不同的默认值。关键字与许多命令一起用于搜索和识别中，如 w、[i 等
isprint (isp)	@,161-255	可以直接显示在屏幕上的字符列表。非 UNIX 系统可能会有不同的默认值
makeef (mef)	/tmp/vim##.err	用于 :make 命令的错误文件名。非 UNIX 系统可能会有不同的默认值。用数字取代 ## 可创建惟一的名字
makeprg (mp)	make	用于 :make 命令的程序。值中的 % 和 # 都会被扩展

表 C-4 vim 5.1 的设置选项（续）

选项	默认设置	描述
mouse		在 <i>vim</i> 的非 GUI 版本中激活鼠标，这可用于 MS-DOS、Win32 和 <i>xterm</i> 。详细信息请参考联机帮助
mousehide (mh)	nomousehide	在输入期间隐藏鼠标箭头。当移动鼠标时重新显示箭头
paste	nopaste	修改大量的选项，目的是在用鼠标把文本粘贴到 <i>vim</i> 窗口中时不损坏被粘贴的文本。关闭它就使这些选项恢复为它们先前的值。详细信息请参考联机帮助
ruler (ru)	noruler	显示光标位置的行号和列号
secure	nosecure	取消启动文件中某些类型的命令。如果对 <i>.vimrc</i> 和 <i>.exrc</i> 文件没有所有权，该选项就会被自动激活
shellpipe (sp)		用于捕获 <i>:make</i> 的输出到文件中的 shell 串。默认值取决于 shell
shellredir (srr)		用于捕获过滤器的输出到临时文件中的 shell 串。默认值取决于 shell
showmode (smd)	<i>vim</i> 默认为: smd, <i>vi</i> 默认为: nosmd	把表示插入、替换和可视模式的信息显示在状态行上
sidescroll (ss)	0	水平滚动的列数。值为 0 就把光标放在屏幕的中间
smartcase (scs)	nosmartcase	如果搜索模式含有大写字符，就忽略 ignorecase 选项
suffixes	*.bak, ~, .o, .h, .info, .swp	在文件名完整化期间，当有多个文件相匹配时，该变量的值就在它们之间设置一个优先权，目的是选出 <i>vim</i> 真正要使用的那个文件
taglength (tl)	0	定义标志有效字符的数目。默认值 (0) 表示所有字符都有效

表 C-4 vim 5.1 的设置选项（续）

选项	默认设置	描述
tagrelative (tr)	vim 默认为: tr, vi 默认为: notr	另一目录下的 tags 文件中的文件名被认为是相对于该 tags 文件所在的目录的
tags (tag)	./tags, tags	用于:tag 命令的文件名，即加上冒号并把整个串放在信息列表中，中间用空格或逗号隔开。前面的 ./ 被当前文件的完整路径替代
tildeop (top)	notildeop	使 ~ 命令的行为类似于运算符
undolevels (ul)	1000	可被撤消的最大修改数目。值为 0 表示与 vi 兼容：一级回退，u 可撤消它自己。非 UNIX 系统可能会有不同的默认值
viminfo (vi)		启动时读取 viminfo 文件并在退出时写它。取值很复杂；它控制着 vim 将存储到文件中的不同类型的信息。详细信息请参考联机帮助
writebackup (wb)	writebackup	在覆盖文件之前进行备份。除非 backup 选项也被设置，否则成功写文件后就删除该备份

vile 8.0 的选项

vile 8.0 共有 92 个选项可影响它的行为，表 C-5 对其中最重要的一些进行了总结。大部分选项在表 C-1 中描述过，这里不再重复。

表 C-5 vile 8.0 的设置选项

选项	默认设置	描述
alt-tabpos	noatp	控制光标是否可位于表示 TAB 字符的空白的左端或右端
animated	animated	当搜索缓冲区中的内容改变时自动对它们进行更新

表 C-5 vile 8.0 的设置选项（续）

选项	默认设置	描述
autobuffer (ab)	autobuffer	使用“最近所使用”类型的缓冲；这些缓冲区按使用的顺序进行排序。否则，缓冲区就会保持它们被编辑的顺序
autosave (as)	noautosave	自动保存文件。在每 autosavecnt 个字符被输入后就保存文件
autosavecnt (ascnt)	256	指定自动保存之前所输入的字符数目
backspacelimit (bl)	backspacelimit	如果被禁止使用，那么在插入模式下就可以把指针回退到插入开始的地方
backup-style	off	控制在写文件时创建备份文件的方式。可能的值为：off 表示不备份，.bak 表示 DOS 类型的备份，tilde 表示 UNIX 下 emacs 格式的 <i>hello.c~</i> 备份
bcolor		在支持它的系统上设置背景颜色
check-modtime	nocheck-modtime	如果文件自上次读取或保存后已被修改，就会发出“ <i>file newer than buffer</i> ”的警告信息并提示用户确认
cmode	off	用于 C 代码的内置主模式
comment-prefix	<code>^\s*\(\s*[#*>]\)\+\n</code>	描述在对注释进行重新格式化时需要保持的行的前面部分。默认值对 <i>Makefile</i> 、shell 和 C 注释以及电子邮件都有效
comments	<code>^\s*/\ ?\(\s*[#*>]\)\+\n/\ +/\ ?\s*\$</code>	定义注释段分隔符的正则表达式，其目的是为了在重新格式化时保留注释内的各段
dirc	nodirc	在为使文件名完整而对目录进行扫描时， <i>vile</i> 要检查每个名字。这样可以在提示符中区分目录名和文件名
dos	nodos	在读取文件时从 CR-LF 对中去掉 CR 并在写文件时把它们加上。用于非现有文件的新缓冲区继承操作系统的行格式，而不管 dos 为何值

表 C-5 vile 8.0 的设置选项（续）

选项	默认设置	描述
fcolor		在支持它的系统上设置前景颜色
fence-begin	/*	用于简单、非嵌套语块边界的首部和尾部的正则表达式，如 C 注释
fence-end	*/	
fence-if	^\s*#\s*if	用来分辨面向行的、嵌套语块边界的开始、结束，“else if” 和 “else”的正则表达式，例如 C 预编译器的控制行
fence-elif	^\s*#\s*elif\>	
fence-else	^\s*#\s*else\>	
fence-fi	^\s*#\s*endif\>	
fence-paris	{ } () []	每对字符都表示一组应该与 % 相匹配的“语块边界”
glob	!echo %s	控制对文件名提示中的通配符字符（如 * 和 ?）的处理方式。值为 off 禁止扩展，为 on 使用可处理常用 shell 通配符和 ~ 符号的内部规则。用于 UNIX 的默认值可保证与当前使用的 shell 兼容
history (hi)	history	把来自冒号命令行的命令记录到 [History] 缓冲区中
horizscroll (hs)	horizscroll	向长行尾部移动的同时向一边移动整个屏幕。如果没被设置，就只移动当前行
linewrap (lw)	nolinewrap	把长的逻辑行转换为屏幕上的多行
maplonger	nomaplonger	映射功能尽量匹配最长的映射序列，而不是最短的
meta-insert-bindings (mib)	nomib	控制 8-bit 字符在输入期间的行为。通常，键绑定只可在命令模式下进行：在插入模式下，所有字符都是自我插入。如果该模式启动并且与函数绑定的元字符（即第八位为 1 的字符）被输入，那么那个函数绑定将被承认并可在插入模式内执行。任何没有被绑定的元字符都将保持自我插入
mini-edit	^G	切换到小型缓冲区编辑模式的字符

表 C-5 vile 8.0 的设置选项（续）

选项	默认设置	描述
mini-hilite (mh)	reverse	当用户切换到小型缓冲区编辑模式中时，指定所使用的高亮属性
popup-choices (pc)	delayed	在进行完整化时控制对弹出帮助窗口的使用。值为 off 表示没有窗口，为 immediate 表示立即弹出，为 delayed 表示等待第二个 TAB 键
preamble (pre)		描述将用于设置相应主模式的文件名首行的正则表达式
resolve-links	noresolve-links	如果被设置，vile会在一些路径为符号链接的情况下彻底解析文件名。这有助于你避免通过不同的路径名对同一物理文件无意地进行多次编辑
ruler	noruler	在状态行中显示当前的行和列，以及光标之前行占当前缓冲区中的行的百分比
showmode (smo)	noshowmode	在模式行上显示表示插入和替换模式的符号
sideways	0	指示用于横向滚动位移量的值，它控制着屏幕按多少字符向左或向右滚动。值为 0 每次就按屏幕的三分之一滚动
suffixed (suf)		描述将用于设置对应的主模式的文件名的正则表达式。被用作主模式功能的部分，而不是独立使用
tabinsert (ti)	tabinsert	允许把物理的 tab 字符输入到缓冲区中。如果被关闭 (notabinsert)，vile 将不能把 TAB 输入到缓冲区中，作为替代它将总是输入对应数目的空格
tagignorecase (tc)	notagignorecase	使标志搜索忽略大小写
taglength (tl)	0	为标志指定有效字符的数目，默认值 (0) 表示所有字符都有效。这并不影响用光标获得的标志，它们一直被准确地匹配（这不同于其他的编辑器）

表 C-5 vile 8.0 的设置选项（续）

选项	默认设置	描述
tagrelative (tr)	tagrelative	当使用另一个目录下的 <i>tags</i> 文件时，在那个 <i>tags</i> 文件中的文件名被认为是相对于该 <i>tags</i> 文件所在目录的
tags	<i>tags</i>	空格隔开的文件列表，从这些文件中查找标志引用
tagword (tw)	notagword	使用光标处的整个单词进行标志搜索，而不是使用从当前光标位置开始的子单词
undolimit (ul)	10	限制可被撤消的修改数量，0表示“没有限制”
unprintable-as-octal (uo)	ununprintable-as-octal	显示第 8 位为 1 的不可打印字符。否则使用十六进制。第 8 位为 0 的不可打印字符总是以控制字符的形式显示
visual-matches	none	控制搜索模式的所有匹配实例的高亮显示。可能的值为：none 表示不加亮，或者 underline、bold 和 reverse 分别表示各自的高亮显示类型。颜色也可以被用在支持它的系统上
xterm-mouse	noxterm-mouse	允许在 <i>xterm</i> 的内部使用鼠标。详细信息请参考联机帮助

附录四

问题列表

本附录对该书所提供的问题列表进行了综合。为了参考方便，我们在下面列出了全部列表。

打开文件所遇到的问题

- ✓ 调用 vi 时，出现[打开模式]提示信息。

可能没有正确地标识终端类型，立即键入 :q 退出编辑会话，检查环境变量 \$TERM。它应该被设置为当前终端的名字。或者让系统管理员提供适当的终端类型设置。

- ✓ 看到下列信息中的一个：

Visual needs addressable cursor or upline capability
Bad termcap entry
Termcap entry too long
terminal:Unknown terminal type
Block device required
Not a typewriter

终端类型或者是没有定义，或者 *terminfo* 或 *termcap* 条目有问题。输入 :q

退出，检查 \$TERM 环境变量，或者要求系统管理员为环境选择一个终端类型。

- ✓ 认为文件存在时出现 [new file] 提示信息。

你可能在一个错误的目录下。输入 :q 退出，然后检查一下你是否在那个文件所在的目录下（在 UNIX 命令提示符下输入 pwd）。如果所在的目录是正确的，检查该目录下的文件列表（用 ls）看该文件是否以一个略有不同的名字存在。

- ✓ 调用 vi，但却得到了冒号提示符（表示当前处于 ex 行编辑模式）。

可能是因为在 vi 出现在屏幕上之前输入中断信号。可以在 ex 提示符 (:) 后键入 vi 进入 vi。

- ✓ 出现下列提示信息中的一个：

```
[Read only]
File is read only
Permission denied
```

“Read only” 意味着你只能浏览该文件，而不能保存所做的修改。这种情况可能是因为以“浏览模式”（用 view 或 vi -R）调用 vi，或者没有对该文件进行写操作的权限，请参考下面的“保存文件时所遇到的问题”一节。

- ✓ 出现下列提示信息中的一个：

```
Bad file number
Block special file
Character special file
Directory
Executable
Non-ascii file
file non-ASCII
```

当前要编辑的文件不是正规的文本文件，输入 :q! 退出，然后检查当前要编辑的文件，也许要使用 file 命令。

- ✓ 若输入 :q，由于上述原因中的任何一个，就会出现下面的提示信息：

```
No write since last change (:quit! overrides).
```

你已经修改了该文件，但是没有对它进行保存。输入:`:q!`退出`vi`，在该会话中所做的修改将不会保存在该文件中。

保存文件所遇到的问题

- ✓ 试着写文件时得到了下列提示信息中的一个：

```
File exists
File file exists - use w!
[Existing file]
File is read only
```

输入`:w! file`将覆盖现有文件，或者输入`:w newfile`把编辑后的版本保存在新文件里。

- ✓ 想写文件时没有写该文件的权限，得到“*Permission denied*”的提示信息。

用`:w newfile`把缓冲区写到一个新文件中。如果对该目录有写的权限，就可以使用`mv`命令以新文件的拷贝取代它的原始版本；否则，可以输入`:w pathname/file`把缓冲区写到具有写权限的目录下（如主目录或`/tmp`）。

- ✓ 写文件时得到文件系统满的提示信息。

输入`:!rm junkfile`删除那些不需要的（大）文件以释放一些空间（使用感叹号开始`ex`命令可以访问 UNIX）。

或者输入`:!df`看其他的文件系统是否还有一些空间。若有，选择那个文件系统中的一个目录，用`:w pathname`把文件写到那个目录（`df`是 UNIX 用来检查磁盘剩余空间的命令）。

- ✓ 系统使你进入开放模式并说明文件系统满。

带有`vi`的临时文件的磁盘已满，输入`:!ls /tmp`看是否可以删除一些文件以获取一些磁盘空间。若有，创建一个临时的 UNIX shell 来删除文件或调用其他的 UNIX 命令（注 1）。通过输入`:sh`可以创建一个 shell；输入`[CTRL-D]`

注 1：`vi`可能把它的临时文件保存在`/usr/tmp`、`/var/tmp`或当前目录中；用户可能需要查找以确定所用空间的确切位置。

或 exit 将终止该 shell 的执行并返回 vi 编辑环境（在大部分 UNIX 系统上，使用作业控制 shell 时，可以通过简单地输入 **CTRL-Z** 来挂起 vi 并返回到 UNIX 命令提示符下；输入 fg 即可返回到 vi）。释放了一些空间后，可以用 :w! 来写文件。

- ✓ 写文件时得到了磁盘空间分配额已到的提示信息。

试着使用 ex 命令 :pre (:preserve 的缩写) 来强迫系统保存缓冲区内容。如果不行的话，删除一些文件。使用 :sh (或 **CTRL-Z**，如果使用作业控制系统的话) 离开 vi 并删除文件。删除完成后使用 **CTRL-D** (或 fg) 返回 vi，然后使用 :w! 写文件。

进入可视模式所遇到的问题

- ✓ 在 vi 中进行编辑时，意外地进入了 ex 编辑器。

vi 命令模式下的 Q 激活 ex。在 ex 编辑器中的任何时候，使用命令 vi 可返回到 vi 编辑器。

使用 vi 命令所遇到的问题

- ✓ 输入命令时，文本在屏幕上跳跃并且不按照它应有的方式进行工作。

确保没有在想输入 J 命令时输入了 J。

你可能按下了 **CAPS** 键，而没有注意到它。由于 vi 是大小写敏感的，也就是说，大写命令 (I、A、J 等) 与小写命令 (i、a、j) 不同，因此，所有命令都没有被解释为小写，而被解释为大写。再次按下 **CAPS** 键返回小写状态，并按下 **ESC** 键以确保处于命令模式，然后输入 U 恢复上次的行修改或者输入 u 撤消上次命令。你可能还必须要做些另外的编辑以完全恢复文件的错乱部分。

删除操作所遇到的问题

- ✓ 删错了文本而想恢复它。

有几种方法可以恢复错删的文本。如果你刚刚删除了一些东西并且意识到要恢复它，那么只需输入 `u` 就能取消上次命令（如，`dd`）。这只在没有给出任何进一步的命令时起作用，因为 `u` 只能取消最近的一次命令。相反，`U` 将能把该行恢复到它的原来状态；这种方法只在对该行进行任何修改起作用之前有效。

然而，由于 `vi` 把最近的 9 次删除操作保存在 9 个已编号的删除缓冲区中，因此仍可以使用 `p` 命令恢复最近的删除操作。例如，如果已知倒数第三次删除操作是想恢复的那次，那么输入：

```
“3p”
```

把 3 号缓冲区的内容输出到光标下面的那行上。这只对已删除的行起作用，单词或部分行都不能被保存在缓冲区中。如果你想恢复已删除的单词或部分行，并且 `u` 不起作用，那么就只能使用 `p` 命令了。这会恢复你最后一次所删除的东西。

附录五

vi 和 Internet

毫无疑问，*vi* 是友好的，尤其
是对它的朋友而言。

至少从 1980 年以来，人们已牢固地把 *vi* 作为“标准的” UNIX 全屏编辑器铭记在 UNIX 文化中。

vi 有助于加强 UNIX, UNIX 进而为今天的 Internet 奠定了基础。因此，在 Internet 上有不少专注于 *vi* 的 Web 站点。本附录描述了 *vi* 高手可使用的一些 *vi* 资源。

从哪里开始：毫无疑问，没有比把万维网站点公布在印刷的书中更易过时的行为
了。我们试着公布那些我们认为将具有较长使用期限的 URL。

elvis 文档的“提示”小节中已列出了许多引人关注的与 *vi* 相关的 Web 站点（这
是我们开始的地方），同时 USENET *comp.editors* 新闻组也是一个浏览的好地方。

vi 的 Web 站点

有两个与 *vi* 相关的 Web 主要站点：Thomer M. Gil 的 *vi* 爱好者主页和 Sven Guckes 的 *Vi* 专栏。每个站点都含有大量到其他受人关注的 *vi* 相关主题的链接。

vi 爱好者主页

vi 爱好者主页的 Web 地址为 <http://www.cs.vu.nl/~tmgil/vi.html>，该站点包含下列内容：

- 所有知名 *vi* 克隆版本的列表，以及到其源代码或二进制版本的链接
- 到其他站点的链接，其中包括 Sven Guckes 的 *Vi* 专栏
- 大量不同级别到 *vi* 文档、手册、帮助和指南的链接
- 用于书写 HTML 文档和解决汉诺塔问题的 *vi* 宏以及用于其他宏的 *ftp* 站点
- 各式各样的 *vi* 链接：诗歌、有关 *vi* “真正历史”的故事、*vi* 与 *emacs* 的讨论，以及 *vi* 咖啡杯（请参看下面的内容）

这里还有很多其他内容，它仅仅是一个伟大的起点。

Vi 专栏

vi 专栏的 Web 地址为 <http://www.math.fu-berlin.de/~guckes/vi>，该站点包含下列内容：

- 不同 *vi* 克隆版本的选项和功能的详细比较
- 不同 *vi* 版本的屏幕
- 各种 *vi* 克隆版本的列表以及带有各种克隆版本的联系信息（名字、地址和 URL）的列表
- 到一些 FAQ（常间问题）的链接
- 一些关于 *vi* 的可爱的引用文字，例如本章开始处的那个
- 其他的链接，包括到 *vi* 咖啡杯的一个链接

vi 爱好者主页把该站点作为“全世界惟一一个比你正在浏览的站点好的站点”。该站点确实非常值得浏览。

VI Powered!

我们发现的更有趣的东西之一是 *VI Powered* 标志（图 E-1）。这是个小小的 GIF 文件，可以把它加到个人 Web 主页上来标识你用 *vi* 创建了它。

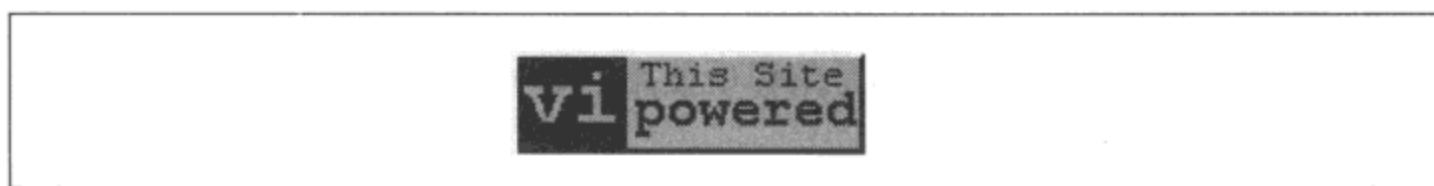


图 E-1 VI Powered!

最初使用 *VI Powered* 标志的主页是 <http://www.abast.es/~avelle/vi.html>，该主页用西班牙语编写。英文主页是 <http://www.darryl.com/vi.html>，介绍添加该图标主页是 <http://www.darryl.com/addlogo.html>。添加该图标的过程由一些简单的步骤组成：

1. 下载图标。在 Web 浏览器中输入 <http://www.darryl.com/vipower.gif>，然后把它保存为一个文件。
2. 在 Web 页面的合适位置添加下列代码：

```
<A HREF="http://www.darryl.com/vi.html">
<IMG SRC="vipower.gif">
</A>
```

这就把该图标放到了页面上并把它制作成一个超级链接，当该链接被选择时就会到达 *VI Powered* 的主页。还可以为那些使用非图形浏览器的用户在 标记中添加 ALT="This Web Page is vi Powered" 属性。

3. 在 Web 页面的 <HEAD> 部分添加下列代码：

```
<META name="editor" content="/usr/bin/vi">
```

就像真正的程序员将利用 *troff* 避开 WYSIWYG 字处理程序一样，真正的 Web 站点管理员也利用 *vi* 避开异样的 HTML 编辑工具。你可以自豪地使用 *VI Powered* 图标显示这一事实。

另外还可以在 <http://www.vim.org/pics.html> 上发现其他的图标（如“vi制造”、“vi设计”等）。这些也可能会比 VI Powered 图标更符合你的爱好。

用于 Java 爱好者的 vi

本小节有关你所喝的 java，而不是编程所使用的 Java（注 1）。

我们假设的真实程序员，当在使用 vi 书写她的 C++ 代码、*troff* 文档和 Web 页面时偶尔会无意间想起咖啡杯。现在她可以从带有 vi 命令参考的杯中喝咖啡了。

咖啡杯的 URL 为 <http://www.vireference.com/vimug.htm>。这些杯分为 4 卷，每个杯上都印有简明的 vi 命令总结。该 Web 站点上有订购和销售信息；可以与一个或更多的朋友分用这 4 卷。

在线的 vi 指南

这两个主页上有大量到 vi 文档的链接，然而专用注解是来自 Walter Zintz 所著《Unix World》杂志的九部分在线指南。Web 地址为 <http://www.wcmh.com/uworld/archives/95/tutorial/009/009.html>（从 vi 的这两个主页上链接过去可能会比较方便）。该指南包含下列内容：

- 编辑器基本原理
- 行模式地址
- g（全局）命令
- 替换命令
- 编辑环境（set 命令、标志和 EXINIT 与 .exrc）
- 地址和行
- 替换命令 r 和 R

注 1： 虽然它是适合的，Java 语言来自 Sun Microsystems，vi 最初的作者 Bill Joy 是该公司
的奠基人和副总裁。

- 自动缩进
- 宏

同时可用的还有在线测验，可以使用它检测自己对指南中材料的掌握情况。或者可以直接使用该测验来检验本书的内容。

让你的朋友惊叹！

从长远看，最有用的事情也许是在 *alf.uib.no* *ftp* 文档中搜集与 *vi* 相关的信息。原始文档位于 *ftp://alf.uib.no/pub/vi* 上。虽然我们很少能成功地访问该站点，但是可以到 *ftp://ftp.uu.net/pub/text-processing/vi* 上找到该文档的镜像。那个目录下的 *INDEX* 文件对文档中的内容进行了描述，并列出了那些可能在地理上离你更近的镜像。

不过，这些文件的最后一次更新是在 1995 年 5 月。幸好，*vi* 的基本原理没有改变，文档中的信息和宏仍是有用的。该文档包含 4 个子目录：

docs

关于 *vi* 的文档，也有一些 *comp.editors* 的信件。

macros

vi 的宏。

comp.editors

邮寄给 *comp.editors* 的各种材料。

programs

用于各种平台的 *vi* 副本的源代码（和其他编辑器）。从这里下载这些东西要当心，许多已经过期了。

docs 和 *macros* 是最令人关注的。*docs* 目录中有大量的论文和参考书目，包括初学者指南、bug 的解释、快速参考和许多“如何”类型的短小论文（如，在 *vi* 中如何只对句子的首字母进行大写）。这里甚至还有一首关于 *vi* 的歌曲！

macros 目录下有 50 多个分别处理不同事情的文件，这里我们只列举了其中的 3 个（以.Z 结尾的文件使用 UNIX 的 *compress* 程序进行的压缩，可以用 *uncompress* 或 *gunzip* 进行解压缩）。

evi.tar.Z

emacs 的“仿真程序”，其目的是把 *vi* 变成无模式编辑器（总是处于插入模式，用控制键执行命令）。它通过取代 EXINIT 环境变量的 shell 脚本实现。

hanoi.Z

这也许是 *vi* 最著名的不寻常用法；一组解决汉诺塔编程问题的宏。该程序只能显示移动，不能真正地写磁盘。出于兴趣，我们在后面也列举了该算法。

turing.tar.Z

该程序使用 *vi* 来实现真实的旋转机器！观看它执行程序相当令人惊讶。

这里有许多更有趣的宏，其中包括 Perl 和 RCS 模式以及 Word Star 仿真程序。

多体验，少填充

vi 与 *emacs*，在没有确认 UNIX 团体中最长久的连续争议之前，我们不能把 *vi* 作为 UNIX 文化中的一部分进行讨论（注 2）。

对哪个更好的讨论已经出现在 *comp.editors*（和其他新闻组）上许多年了。在上面描述的 *ftp* 文档中有一些有关这些讨论的摘要，和指示 Web 页面上较新版本的线索。

注 2：还不错，虽然它实际上是一场派别之间的斗争，但是我们正试着变好（另一场派别斗争是 BSD 与 System V，已被 POSIX 解决了，虽然 System V 赢了，但是 BSD 也得到了具有特殊意义的特许权）。

vi 版的汉诺塔

```

" From: gregm@otc.otca.oz.au (Greg McFarlane)
" Newsgroups: comp.sources.d,alt.sources,comp.editors
" Subject: VI SOLVES HANOI
" Date: 19 Feb 91 01:32:14 GMT
"
"
" Submitted-by: gregm@otc.otca.oz.au
" Archive-name: hanoi.vi.macros/part01
"
"
" Everyone seems to be writing stupid Tower of Hanoi programs.
" Well, here is the stupidest of them all: the hanoi solving
" vi macros.
"
"
" Save this article, unshar it, and run uudecode on
" hanoi.vi.macros.uu. This will give you the macro file
" hanoi.vi.macros.
"
" Then run vi (with no file: just type "vi") and type:
" :so hanoi.vi.macros
" g
" and watch it go.
"
"
" The default height of the tower is 7 but can be easily changed
" by editing the macro file.
"
"
" The disks aren't actually shown in this version, only numbers
" representing each disk, but I believe it is possible to write
" some macros to show the disks moving about as well. Any takers?
"
"
" (For maze solving macros, see alt.sources or comp.editors)
"
"
" Greg
"
"
" ----- REAL FILE STARTS HERE -----
set remap
set noterse
set wrapscan
" to set the height of the tower, change the digit in the following
" two lines to the height you want (select from 1 to 9)
map t 7
map! t 7
map I. 1G/t^MX/^0^M$P1GJ$An$BGC0e$X0E0F$X/T^M@f^M@h^MSA1GJ@f01$Xn$PU

```

— 续 —

```
map g iL
map I KMYNOQNOSkRTV
map J /^0[^t]*$^M
map X x
map P p
map U L
map A "fyl
map B "hyl
map C "fp
map e "fy21
map E "hp
map F "hy21
map K 1Go^[
map M dG
map N yy
map O p
map q t11D
map Y o0123456789Z^[0q
map Q 0iT^[
map R $rn
map S $r$
map T ko0^M0^M^M^[
map V Go/^[
```

一些支持 *vi* 的较好论述有：

- *vi* 可用于每种 UNIX 系统。如果你正在安装系统或从一个系统移到另一个系统，无论如何你都必须使用 *vi*。
- 通常你可以把手指保持在键盘上的主行上，这对触摸打字员来说是很受欢迎的。
- 命令是一个（或有时是两个）常见字符；它们比 *emacs* 需要的所有控制符和通配符更容易输入。
- *vi* 通常比 *emacs* 小，对资源有更小的敏感度。启动时间更是快多了，有时达到快 10 倍。
- 鉴于 *vi* 克隆版本已添加了像增量搜索、多窗口和缓冲区、GUI 接口、语法突出和智能缩进以及通过扩展语言获得的可编程性之类的功能，即使两个编辑器之间的功能差距没有完全消失的话，也已变得非常微小了。

为了完整性，另外两条也应该提到。首先，确实有两个流行的 *emacs* 版本：最初的 GNU *emacs* 和起源于 GNU *emacs* 较早版本的 *xemacs*。两个版本各有优势和不足以及它们各自的热爱者（注 3）。

其次，GNU *emacs* 一直带有 *vi* 仿真程序包，直到最近，它们也不是非常好用。这已经改变了，“viper 模式”被认为是一个极好的 *vi* 仿真程序。对于那些对此有兴趣的人来说，它可以起到学习 *emacs* 的桥梁作用。

总之，要记着你是使用编辑器的最终决策者，应该使用那些可使自己具有最高效率的工具。对许多任务来说，*vi* 和它的克隆版本都是极好的工具。

vi 引证

最后，这里有一些其他关于 *vi* 的引证，谦虚的 Bram Moolenaar, *vim* 的作者：

论题：*vi* 是完美的。

论据：在罗马数字中 VI 是 6。小于 6 的自然数能被 6 整除的是 1、2 和 3， $1 + 2 + 3 = 6$ ，因此 6 是完美的数字，从而 *vi* 也是完美的。

—— Arthur Tateishi

来自 Nathan T. Oelger 的反应：

这样的话，上面的证明把 VIM 放在哪里了？罗马数字中的 VIM 可能是： $(1000 - (5+1))=994$ ，这碰巧等于 $2*496+2$ 。496 是被 1、2、4、8、16、31、62、124、和 248 可整除的，且 $1+2+4+8+16+31+62+124+248=496$ 。因此，496 是个完美的数字。从而，*vim* 比 *vi* 要完美两倍，外加一对特别少量的糖果。

也就是说，*vim* 比完美还要好。

这个引证好像是真正的 *vi* 爱好者对它的概括。

注 3：毫无疑问，这些人有着讨厌 *vi* 的共同观点！

对我来说`vi`是禅，使用`vi`就是使用禅。每个命令都是心印，这对用户来说是深奥的，对未入门的人来说是无法了解的。每次使用它你都能发现真理。

—— Satish Reddy



有关此电子图书的说明

本人由于一些便利条件，可以帮您提供各种中文电子图书资料，且质量均为清晰的 PDF 图片格式，质量要高于网上大量传播的一些超星 PDG 的图书。方便阅读和携带。只要图书不是太新，文学、法律、计算机、人文、经济、医学、工业、学术等方面的图书，我都可以帮您找到电子版本。所以，当你想要看什么图书时，可以联系我。我的 QQ 是：85013855，大家可以在 QQ 上联系我。

此 PDF 文件为本人亲自制作，请各位爱书之人尊重个人劳动，敬请您不要修改此 PDF 文件。因为这些图书都是有版权的，请各位怜惜电子图书资源，不要随意传播，否则，这些资源更难以得到。