**GNU Emacs Manual**

**GNU Emacs Manual**

Seventeenth Edition, Updated for Emacs Version 24.3.

Richard Stallman et al.

This is the Seventeenth edition of the GNU Emacs Manual,

updated for Emacs version 24.3.

Copyright © 1985–1987, 1993–2013 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under

the terms of the GNU Free Documentation License, Version 1.3 or any later

version published by the Free Software Foundation; with the Invariant Sections

being “The GNU Manifesto,” “Distribution” and “GNU GENERAL PUBLIC

LICENSE,” with the Front-Cover texts being “A GNU Manual,” and with the

Back-Cover Texts as in (a) below. A copy of the license is included in the

section entitled “GNU Free Documentation License.”

(a) The FSF’s Back-Cover Text is: “You have the freedom to copy and modify

this GNU manual. Buying copies from the FSF supports it in developing GNU

and promoting software freedom.”

Published by the Free Software Foundation

51 Franklin Street, Fifth Floor

Boston, MA 02110-1301 USA

ISBN 978-0-9831592-4-7

Cover art by Etienne Suvasa; cover design by Matt Lee.

Short Contents（短目录）

# Table of contents（目录表）

Preface（前言）

This manual documents the use and simple customization of the Emacs editor. Simple Emacs customizations do not require you to be a programmer, but if you are not interested in customizing, you can ignore the customization hints.

这本手册是为了更方便的定制Emacs编辑器，简单的定制Emacs不需要你是个专业程序员,如果你对定制不感兴趣,你可以忽略定制的提示.

This is primarily a reference manual, but can also be used as a primer. If you are new to Emacs, we recommend you start with the integrated, learn-by-doing tutorial, before reading the manual. To run the tutorial, start Emacs and type C-h t. The tutorial describes commands, tells you when to try them, and explains the results. The tutorial is available in several languages.

这本手册主要是为了作为参考之用, 但也可以作为初学者的入门教材. 当然在读这本手册之前,如果你是Emacs 新手, 我们建议你跟着在线的教程边学边用. 运行这个教程只需要在Emacs中敲击C-h t这个快捷键即可. 教程会描述很多命令,会告诉你什么时候用它们,而且会解释怎么用 和 使用的效果.

On first reading, just skim chapters 1 and 2, which describe the notational conventions of the manual and the general appearance of the Emacs display screen. Note which questions are answered in these chapters, so you can refer back later. After reading chapter 4, you should practice the commands shown there. The next few chapters describe fundamental techniques and concepts that are used constantly. You need to understand them thoroughly, so experiment with them until you are fluent.

第一次阅读这本手册时,你可以浏览第1章和第2章,这两章会描述一些手册的规定和Emacs显示的大概界面.注意,这些问题都会在章节中讲述,你可以先参考稍后再回来.如果你读完第4章,你应该可以练习一些在第四章出现的命令. 接下来的几个章节将会介绍基本的技术和一些常用的概念.你应该透彻的理解它们,然后不停的试验直到你能熟练运用为止.

Chapters 14 through 19 describe intermediate-level features that are useful for many kinds of editing. Chapter 20 and following chapters describe optional but useful features; read those chapters when you need them.

第14到19章描述了一些特性为多种类型编辑提供些帮助.第20章和下面的几个章节描述了一些可选的但是很有用的特性,你可以有选择的去阅读你感兴趣的.

Read the Common Problems chapter if Emacs does not seem to be working properly. It explains how to cope with several common problems (see Section 34.2 [Dealing with Emacs Trouble], page 436), as well as when and how to report Emacs bugs (see Section 34.3 [Bugs], page 440).

如果Emacs看起来不能正常工作,你可以去阅读常用问题章节(FAQ),那里不但解释了如何对付一些常用问题(请参考 段 33.2 , 410页) ,而且告诉你怎么去提交Emacs bugs(参考 段 33.3 ,414页).

To find the documentation of a particular command, look in the index. Keys (character commands) and command names have separate indexes. There is also a glossary, with a cross reference for each term.

要查询某个特定命令的文档,你可以查询索引,键(字符命令) 和 命令名称 都有单独的索引, 还有个词汇表,为每个条款进行交叉引用.

This manual is available as a printed book and also as an Info file. The Info file is for reading from Emacs itself, or with the Info program. Info is the principal format for documentation in the GNU system. The Info file and the printed book contain substantially the same text and are generated from the same source files, which are also distributed with GNU Emacs.

本手册可以作为印刷书籍  ,也可以作为信息文件. 信息文件使用了信息程序为了可以在线进行浏览, 这是在GNU系统中访问在线文档的主要手段.Emacs的信息文件和信息阅读器,包括GNU Emacs编辑器. 信息文件 和 印刷书中包含大致相同的文本 而且 来自是相同的源文件生成的, 这个也是GNU Emacs分配的.

GNU Emacs is a member of the Emacs editor family. There are many Emacs editors, all sharing common principles of organization. For information on the underlying philosophy of Emacs and the lessons learned from its development, see Emacs, the Extensible, Customizable Self-Documenting Display Editor, available from <ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-519A.pdf>..

GNU Emacs是Emacs编辑器的一个成员. 这里有很多Emacs编辑器,  都使用同一个组织原则, 对于Emacs的基本概念和它发展过程中一些经验教训信息,Emcs扩展 ,自定义文档显示 可以参考<ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-519A.pdf.>

This version of the manual is mainly intended for use with GNU Emacs installed on GNU and Unix systems. GNU Emacs can also be used on MS-DOS, Microsoft Windows, and Macintosh systems. The Info file version of this manual contains some more information about using Emacs on those systems. Those systems use different file name syntax; in addition MS-DOS does not support all GNU Emacs features. See Appendix G [Microsoft Windows], page 496, for information about using Emacs on Windows. See Appendix F [Mac OS / GNUstep], page 493, for information about using Emacs on Macintosh (and GNUstep) .

本手册的版本是给GNU和UNIX系统上GNU Emacs打算的, 当然GNU Emacs依然可以运行在VMS, MS-DOS(也可以成为 MS-DOG ^\_ ^), Microsoft Windows(微软系统) 和  Macintosh systems(苹果系统) 这些系统都各自有各自的文件名规则; 另外, VMS, MS-DOS 不支持GNU Emacs的所有特性, 想更多的了解Windows上使用GNU Emacs,请参考 附录 G [Microsoft  Windows], 467页,想更多的了解Macintosh上使用GNU Emacs,请参考附录 F[Mac OS], 462页.我们不会在这本手册中描述GNU Emacs在VMS中使用的方法.

Distribution（分发）

GNU Emacs is free software ; this means that everyone is free to use it and free to redistribute it under certain conditions. GNU Emacs is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of GNU Emacs that they might get from you. The precise conditions are found in the GNU General Public License that comes with Emacs and also appears in this manual1. See Appendix A [Copying], page 450.

1This manual is itself covered by the GNU Free Documentation License. This license is similar in spirit to the General Public License, but is more suitable for documentation. See Appendix B [GNU Free Documentation License], page 461.

One way to get a copy of GNU Emacs is from someone else who has it. You need not ask for our permission to do so, or tell any one else; just copy it. If you have access to the Internet, you can get the latest distribution version of GNU Emacs by anonymous FTP; see http://www.gnu.org/software/Emacs on our website for more information.

You may also receive GNU Emacs when you buy a computer. Computer manufacturers are free to distribute copies on the same terms that apply to everyone else. These terms require them to give you the full sources, including whatever changes they may have made, and to permit you to redistribute the GNU Emacs received from them under the usual terms of the General Public License. In other words, the program must be free for you when you get it, not just free for the manufacturer.

If you find GNU Emacs useful, please send a donation to the Free Software Foundation to support our work. Donations to the Free Software Foundation are tax deductible in the US. If you use GNU Emacs at your workplace, please suggest that the company make a donation. For more information on how you can help, see http://www.gnu.org/help/help.html.

We also sell hardcopy versions of this manual and An Introduction to Programming in Emacs Lisp, by Robert J. Chassell. You can visit our online store at http://shop.fsf.org/. The income from sales goes to support the foundation’s purpose: the development of new free software, and improvements to our existing programs including GNU Emacs.

If you need to contact the Free Software Foundation, see http://www.fsf.org/about/contact/, or write to

Free Software Foundation

51 Franklin Street, Fifth Floor

Boston, MA 02110-1301

US A

**Acknowledgments**

Contributors to GNU Emacs include Jari Aalto, Per Abrahamsen, Tomas Abrahamsson, Jay K. Adams, Alon Albert, Michael Albinus, Nagy Andras, Benjamin Andresen, Ralf Angeli, Dmitry Antipov, Joe Arceneaux, Emil Astr¨om, Miles Bader, David Bakhash, Juanma Barranquero, Eli Barzilay, Thomas Baumann, Steven L. Baur, Jay Belanger, Alexander L. Belikoff, Thomas Bellman, Scott Bender, Boaz Ben-Zvi, Sergey Berezin, Karl Berry, Anna M. Bigatti, Ray Blaak, Martin Blais, Jim Blandy, Johan Bockg˚ard, Jan B¨ocker, Joel Boehland, Lennart Borgman, Per Bothner, Terrence Brannon, Frank Bresz, Peter Breton, Emmanuel Briot, Kevin Broadey, Vincent Broman, Michael Brouwer, David M. Brown, Stefan Bruda, Georges Brun-Cottan, Joe Buehler, Scott Byer, W lodek Bzyl, Bill Carpenter, Per Cederqvist, Hans Chalupsky, Chris Chase, Bob Chassell, Andrew Choi, Chong Yidong, Sacha Chua, Stewart Clamen, James Clark, Mike Clarkson, Glynn Clements, Andrew Cohen, Daniel Colascione, Edward O’Connor, Christoph Conrad, Ludovic Court`es, Andrew Csillag, Toby Cubitt, Baoqiu Cui, Doug Cutting, Mathias Dahl, Julien Danjou, Satyaki Das, Vivek Dasmohapatra, Dan Davison, Michael DeCorte, Gary Delp, Nachum Dershowitz, Dave Detlefs, Matthieu Devin, Christophe de Dinechin, Eri Ding, Jan Dj¨arv, Lawrence R. Dodd, Carsten Dominik, Scott Draves, Benjamin Drieu, Viktor Dukhovni, Jacques Duthen, Dmitry Dzhus, John Eaton, Rolf Ebert, Carl Edman, David Edmondson, Paul Eggert, Stephen Eglen, Christian Egli, Torbj¨orn Einarsson, Tsugutomo Enami, David Engster, Hans Henrik Eriksen, Michael Ernst, Ata Etemadi, Frederick Farnbach, Oscar Figueiredo, Fred Fish, Steve Fisk, Karl Fogel, Gary Foster, Eric S. Fraga, Romain Francoise, Noah Friedman, Andreas Fuchs, Shigeru Fukaya, Hallvard Furuseth, Keith Gabryelski, Peter S. Galbraith, Kevin Gallagher, Fabi´an E. Gallina, Kevin Gallo, Juan Le´on Lahoz Garc´ıa, Howard Gayle, Daniel German, Stephen Gildea, Julien Gilles, David Gillespie, Bob Glickstein, Deepak Goel, David De La Harpe Golden, Boris Goldowsky, David Goodger, Chris Gray, Kevin Greiner, Michelangelo Grigni, Odd Gripenstam, Kai Großjohann, Michael Gschwind, Bastien Guerry, Henry Guillaume, Doug Gwyn, Bruno Haible, Ken’ichi Handa, Lars Hansen, Chris Hanson, Jesper Harder, Alexandru Harsanyi, K. Shane Hartman, John Heidemann, Jon K. Hellan, Magnus Henoch, Markus Heritsch, Dirk Herrmann, Karl Heuer, Manabu Higashida, Konrad Hinsen, Anders Holst, Jeffrey C. Honig, Tassilo Horn, Kurt Hornik, Tom Houlder, Joakim Hove, Denis Howe, Lars Ingebrigtsen, Andrew Innes, Seiichiro Inoue, Philip Jackson, Martyn Jago, Pavel Janik, Paul Jarc, Ulf Jasper, Thorsten Jolitz, Michael K. Johnson, Kyle Jones, Terry Jones, Simon Josefsson, Alexandre Julliard, Arne Jørgensen, Tomoji Kagatani, Brewster Kahle, Tokuya Kameshima, Lute Kamstra, Ivan Kanis, David Kastrup, David Kaufman, Henry Kautz, Taichi Kawabata, Taro Kawagishi, Howard Kaye, Michael Kifer, Richard King, Peter Kleiweg, Karel Kl´ıˇc, Shuhei Kobayashi, Pavel Kobyakov, Larry K. Kolodney, David M. Koppelman, Koseki Yoshinori, Robert Krawitz, Sebastian Kremer, Ryszard Kubiak, Igor Kuzmin, David K˚agedal, Daniel LaLiberte, Karl Landstrom, Mario Lang, Aaron Larson, James R. Larus, Vinicius Jose Latorre, Werner Lemberg, Frederic Lepied, Peter Liljenberg, Christian Limpach, Lars Lindberg, Chris Lindblad, Anders Lindgren, Thomas Link, Juri Linkov, Francis Litterio, Sergey Litvinov, Emilio C. Lopes, Martin Lorentzon, Dave Love, Eric Ludlam, K´aroly L˝orentey, Sascha L¨udecke, Greg McGary, Roland McGrath, Michael McNamara, Alan Mackenzie, Christopher J. Madsen, Neil M. Mager, Ken Manheimer, Bill Mann, Brian Marick, Simon Marshall, Bengt Martensson, Charlie Martin, Yukihiro Matsumoto, Tomohiro Matsuyama, David Maus, Thomas May, Will Mengarini, David Megginson, Stefan Merten, Ben A. Mesander, Wayne Mesard, Brad Miller, Lawrence Mitchell, Richard Mlynarik, Gerd Moellmann, Stefan Monnier, Keith Moore, Jan Moringen, Morioka Tomohiko, Glenn Morris, Don Morrison, Diane Murray, Riccardo Murri, Sen Nagata, Erik Naggum, Gergely Nagy, Nobuyoshi Nakada, Thomas Neumann, Mike Newton, Thien-Thi Nguyen, Jurgen Nickelsen, Dan Nicolaescu, Hrvoje Niksic, Jeff Norden, Andrew Norman, Kentaro Ohkouchi, Christian Ohler, Kenichi Okada, Alexandre Oliva, Bob Olson, Michael Olson, Takaaki Ota, Pieter E. J. Pareit, Ross Patterson, David Pearson, Juan Pechiar, Jeff Peck, Damon Anton Permezel, Tom Perrine, William M. Perry, Per Persson, Jens Petersen, Daniel Pfeiffer, Justus Piater, Richard L. Pieri, Fred Pierresteguy, Fran¸cois Pinard, Daniel Pittman, Christian Plaunt, Alexander Pohoyda, David Ponce, Francesco A. Potorti, Michael D. Prange, Mukesh Prasad, Ken Raeburn, Marko Rahamaa, Ashwin Ram, Eric S. Raymond, Paul Reilly, Edward M. Reingold, David Reitter, Alex Rezinsky, Rob Riepel, Lara Rios, Adrian Robert, Nick Roberts, Roland B. Roberts, John Robinson, Denis B. Roegel, Danny Roozendaal, Sebastian Rose, William Rosenblatt, Markus Rost, Guillermo J. Rozas, Martin Rudalics, Ivar Rummelhoff, Jason Rumney, Wolfgang Rupprecht, Benjamin Rutt, Kevin Ryde, James B. Salem, Masahiko Sato, Timo Savola, Jorgen Schaefer, Holger Schauer, William Schelter, Ralph Schleicher, Gregor Schmid, Michael Schmidt, Ronald S. Schnell, Philippe Schnoebelen, Jan Schormann, Alex Schroeder, Stefan Schoef, Rainer Schoepf, Raymond Scholz, Eric Schulte, Andreas Schwab, Randal Schwartz, Oliver Seidel, Manuel Serrano, Paul Sexton, Hovav Shacham, Stanislav Shalunov, Marc Shapiro, Richard Sharman, Olin Shivers, Tibor Simko, Espen Skoglund, Rick Sladkey, Lynn Slater, Chris Smith, David Smith, Paul D. Smith, Wilson Snyder, William Sommerfeld, Simon South, Andre Spiegel, Michael Staats, Thomas Steffen, Ulf Stegemann, Reiner Steib, Sam Steingold, Ake Stenhoff, Peter Stephenson, Ken Stevens, Andy Stewart, Jonathan Stigelman, Martin Stjernholm, Kim F. Storm, Steve Strassmann, Christopher Suckling, Olaf Sylvester, Naoto Takahashi, Steven Tamm, Luc Teirlinck, Jean-Philippe Theberge, Jens T. Berger Thielemann, Spencer Thomas, Jim Thompson, Toru Tomabechi, David O’Toole, Markus Triska, Tom Tromey, Enami Tsugutomo, Eli Tziperman, Daiki Ueno, Masanobu Umeda, Rajesh Vaidheeswarran, Neil W. Van Dyke, Didier Verna, Joakim Verona, Ulrik Vieth, Geoffrey Voelker, Johan Vromans, Inge Wallin, John Paul Wallington, Colin Walters, Barry Warsaw, Christoph Wedler, Ilja Weis, Zhang Weize, Morten Welinder, Joseph Brian Wells, Rodney Whitby, John Wiegley, Sascha Wilde, Ed Wilkinson, Mike Williams, Roland Winkler, Bill Wohler, Steven A. Wood, Dale R. Worley, Francis J. Wright, Felix S. T. Wu, Tom Wurgler, Yamamoto Mitsuharu, Katsumi Yamaoka, Masatake Yamato, Jonathan Yavner, Ryan Yeske, Ilya Zakharevich, Milan Zamazal, Victor Zandy, Eli Zaretskii, Jamie Zawinski,

Andrew Zhilin, Shenghuo Zhu, Piotr Zielinski, Ian T. Zimmermann, Reto Zimmermann, Neal Ziring, Teodor Zlatanov, and Detlev Zundel.

Introduction（简介）

**You are reading about GNU Emacs, the GNU incarnation of the advanced, selfdocumenting, customizable, extensible editor Emacs. (The ‘G’ in ‘GNU’ is not silent.)**

**We call Emacs advanced because it can do much more than simple insertion and deletion of text. It can control subprocesses, indent programs automatically, show multiple files at once, and more. Emacs editing commands operate in terms of characters, words, lines, sentences, paragraphs, and pages, as well as expressions and comments in various programming languages. Self-documenting means that at any time you can use special commands, known as help commands, to find out what your options are, or to find out what any command does, or to find all the commands that pertain to a given topic. See Chapter 7 [Help], page 37.**

**Customizable means that you can easily alter the behavior of Emacs commands in simple ways. For instance, if you use a programming language in which comments start with ‘<\*\*’and end with ‘\*\*>’, you can tell the Emacs comment manipulation commands to use those strings (see Section 23.5 [Comments], page 243). To take another example, you can rebind the basic cursor motion commands (up, down, left and right) to any keys on the keyboard that you find comfortable. See Chapter 33 [Customization], page 404.**

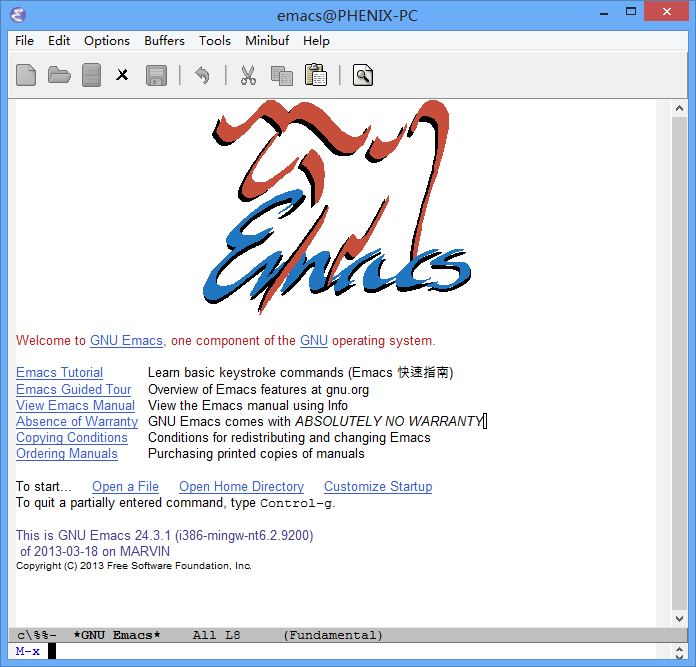
**Extensible means that you can go beyond simple customization and create entirely new commands. New commands are simply programs written in the Lisp language, which are run by Eames’s own Lisp interpreter. Existing commands can even be redefined in the middle of an editing session, without having to restart Emacs. Most of the editing commands in Emacs are written in Lisp; the few exceptions could have been written in Lisp but use C instead for efficiency. Writing an extension is programming, but non-programmers can use it afterwards. See Section “Preface” in An Introduction to Programming in Emacs Lisp, if you want to learn Emacs Lisp programming.**

# ****The Organization of the Screen（屏幕布局）****

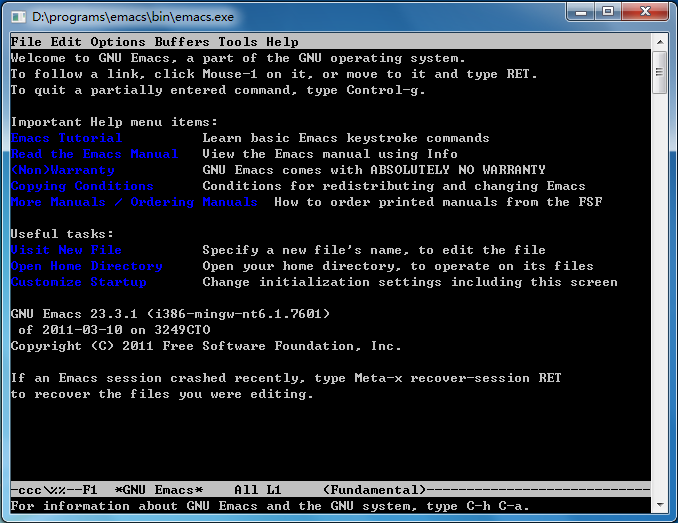
On a graphical display, such as on GNU/Linux using the X Window System, Emacs occupies a “graphical window”. On a text terminal, Emacs occupies the entire terminal screen. We will use the term frame to mean a graphical window or terminal screen occupied by Emacs. Emacs behaves very similarly on both kinds of frames. It normally starts out with just one frame, but you can create additional frames if you wish (see Chapter 18 [Frames], page 157).

图形界面下，如使用X Window系统的GNU/Linux，Emacs占用一个“图形窗口（graphical window）”。文本终端下，Emacs占据整个终端屏幕。我们使用术语窗格（frame）指代Emacs占用的图形窗口或终端屏幕。Emacs在两种窗格（frame）中都表现的非常相似。正常情况下 Emacs只启用一个frame，但如你所愿，你也可以创建多个额外的窗格（frames）（参见 Chapter 18 [Frames], page 157）。

*译者注：Emacs中的专有名词，如Frame，window，minibuffer等，只做一次翻译，在后面使用中还是使用英文，可理解。可参考下图：*



**图1-1** Emacs graphical window



**图1-2** Emacs terminal screen

Each frame consists of several distinct regions. At the top of the frame is a menu bar, which allows you to access commands via a series of menus. On a graphical display, directly below the menu bar is a tool bar, a row of icons that perform editing commands if you click on them. At the very bottom of the frame is an echo area, where informative messages are displayed and where you enter information when Emacs asks for it.

每个frame包含几个截然不同的区域。Frame顶部是一个菜单栏（menu bar），你可以通过一系列的菜单来执行一些命令。在图形显示下，处于菜单栏下面的是工具栏（tool bar），这里有一行图标，通过点击他们可以执行编辑命令。在每个frame的底部时回显区（echo area），这里会显示一些通知消息，你可以在这里输入一些Emacs需要的交互性信息。

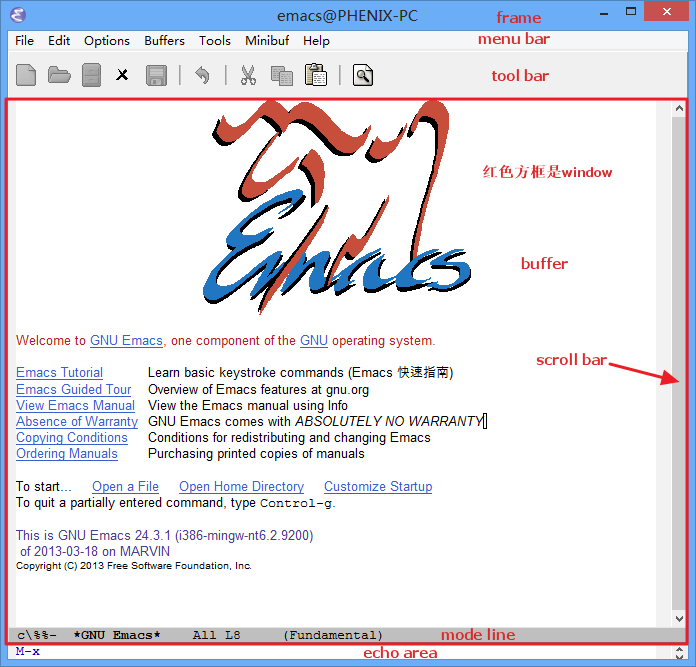
The main area of the frame, below the tool bar (if one exists) and above the echo area, is called the window. Henceforth in this manual, we will use the word “window” in this sense. Graphical display systems commonly use the word “window” with a different meaning; but, as stated above, we refer to those “graphical windows” as “frames”.

Tool bar（如果存在的话）之下，echo area之上的是Frame的主要区域，称作窗口（window）。图形显示系统中经常使用的窗口是另外一个意思，但是，正如上面所言，我们把这些“图形窗口（graphical windows）”称作“窗格（frame）”。

An Emacs window is where the buffer—the text you are editing—is displayed. On a graphical display, the window possesses a scroll bar on one side, which can be used to scroll through the buffer. The last line of the window is a mode line. This displays various information about what is going on in the buffer, such as whether there are unsaved changes, the editing modes that are in use, the current line number, and so forth.

Emacs window是一个缓冲区（buffer），其中显示你正在编辑的文本。在图形显示中，window一侧有一个滚动条，它可以用来滚动整个缓冲区。Window的最后一行是模式行（mode line）。这里的信息显示buffer所处的状态，例如是否有未保存的更改，正在使用的编辑模式（modes），当前的行号等等。

译者注：参考下图

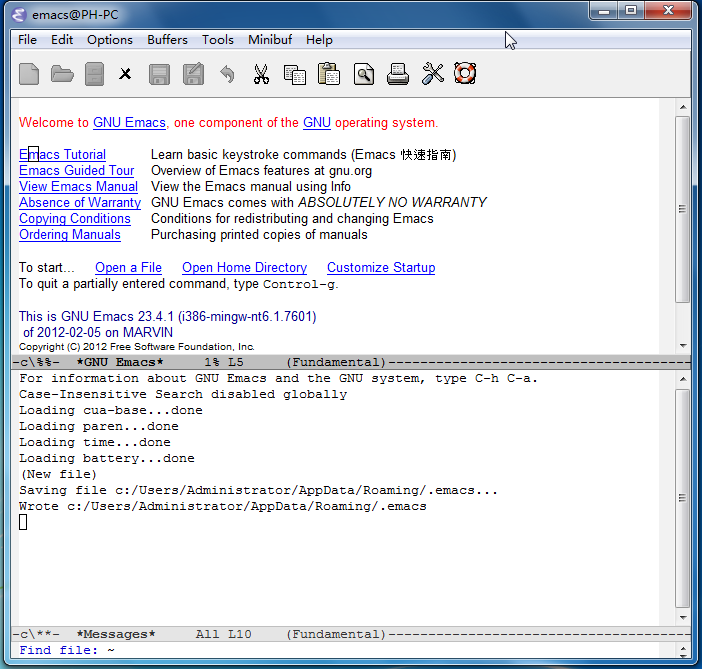


**图1-3** Emacs layout

When you start Emacs, there is normally only one window in the frame. However, you can subdivide this window horizontally or vertically to create multiple windows, each of which can independently display a buffer (see Chapter 17 [Windows], page 151).

启动Emacs后，一般frame中只有一个window。然而，可以通过横向或者纵向细分当前window来创建多个windows，每个都可以单独显示一个buffer(参见Chapter 17 [Windows], page 151)。

译者注：参考下图

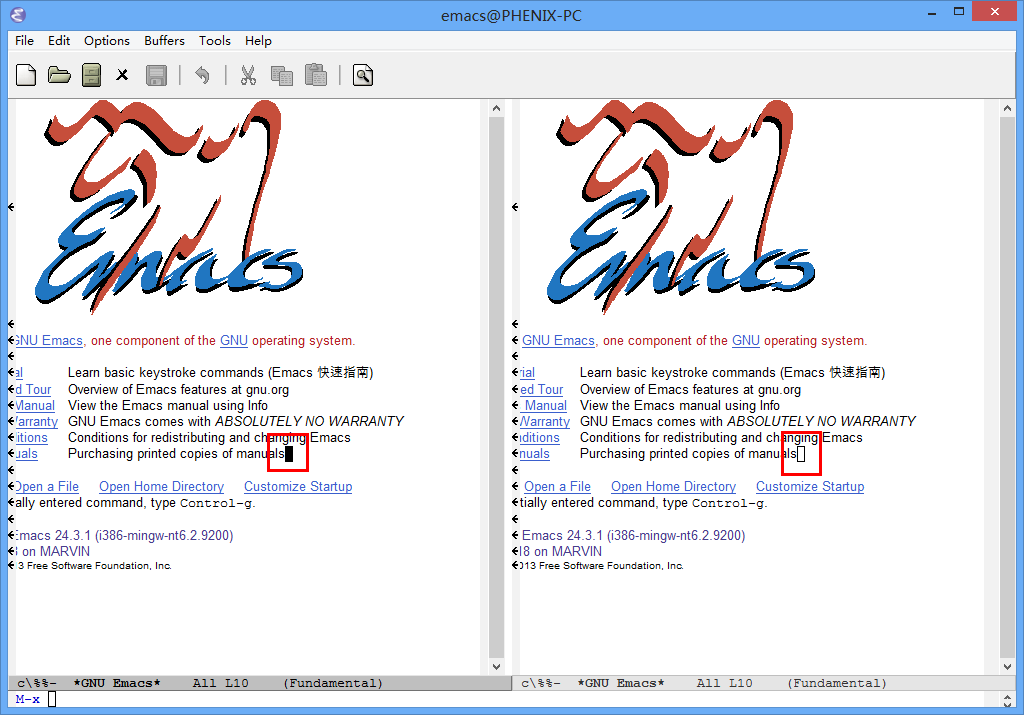


**图1-4** Emacs multiple windows

At any time, one window is the selected window. On a graphical display, the selected window shows a more prominent cursor (usually solid and blinking); other windows show a less prominent cursor (usually a hollow box). On a text terminal, there is only one cursor, which is shown in the selected window. The buffer displayed in the selected window is called the current buffer, and it is where editing happens. Most Emacs commands implicitly apply to the current buffer; the text displayed in unselected windows is mostly visible for reference. If you use multiple frames on a graphical display, selecting a particular frame selects a window in that frame.

任何时候，只能选中一个window。在图形显示中，选中的window显示一个更加突出的光标（cursor）（通常是正在闪烁的一个方块）；其他的windows显示一个不太显眼的cursor（通常是个空心方块）。选定window中显示的buffer，被称作当前缓冲区（current buffer），也就是正在编辑的区域。大多数的命令默认用于current buffer；未选中window中显示的文本仅供参考。如果图形显示中使用多frames，选择一个特定的frame同时也选中其中的一个window。

译者注：参考下图



**图1-5** Emacs cursor

## Point（位点）

The cursor in the selected window shows the location where most editing commands take effect, which is called point 1 . Many Emacs commands move point to different places in the buffer; for example, you can place point by clicking mouse button 1 (normally the left button) at the desired location.

选中window中cursor显示的位置大多数编辑命令会生效，该处称之为位点（point）1。许多Emacs命令将point到buffer中不同的位置；比如，通过点击鼠标按钮把point移动到需要的位置。

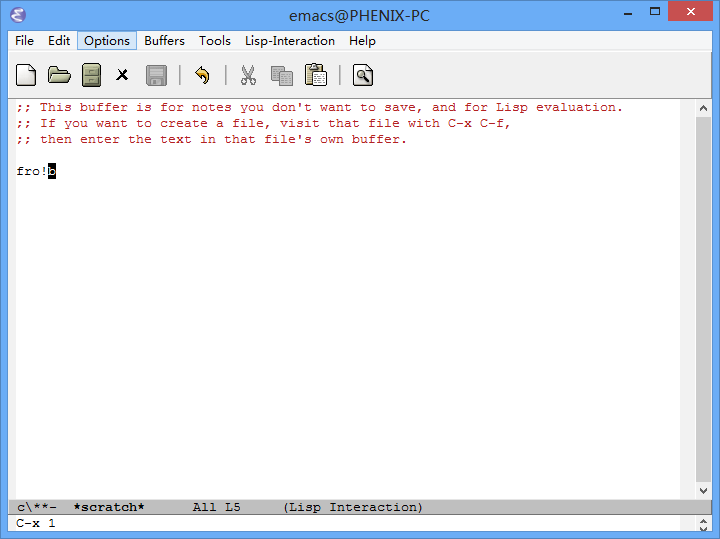
1 The term “point” comes from the character ‘.’, which was the command in TECO (the language in which the original Emacs was written) for accessing the editing position.

1术语“point”来自于字符“.”,它是TECO（编写Emacs原来所用的语言）语言中访问编辑位置的命令。

By default, the cursor in the selected window is drawn as a solid block and appears to be on a character, but you should think of point as between two characters; it is situated before the character under the cursor. For example, if your text looks like ‘frob’ with the cursor over the ‘b’, then point is between the ‘o’ and the ‘b’. If you insert the character ‘!’ at that position, the result is ‘fro!b’, with point between the ‘!’ and the ‘b’. Thus, the cursor remains over the ‘b’, as before.

默认情况下，在选定的window中cursor是一个实心方块，它显示在一个字符之上， 但此时应该认为point是在两个字符之间；point位于cursor下的字符前。例如像“frob”的文本，如果cursor在‘b’上面，point就是在‘o’和‘b’中间。如果在该位置插入字符‘！’，结果是‘fro!b’，这时point在‘!’ 与 ‘b’之间。因此，cursor仍像刚才一样，停留在‘b’之上。

译者注：参考下图



**图1-6** Emacs cursor and point

If you are editing several files in Emacs, each in its own buffer, each buffer has its own value of point. A buffer that is not currently displayed remembers its value of point if you later display it again. Furthermore, if a buffer is displayed in multiple windows, each of those windows has its own value of point.

如果在Emacs中同事编辑多个文件，每个文件都有各自的buffer，每个buffer都有各自的point值。当前未显示的buffer会记住自己的point值。此外，如果一个buffer显示在多个windows中，每个windows都有各自的point值。

See Section 11.20 [Cursor Display], page 83, for options that control how Emacs displays the cursor.

参见Section 11.20 [Cursor Display], page 83，来了解Emacs控制cursor显示的选项。

## The Echo Area（回显区）

The line at the very bottom of the frame is the echo area. It is used to display small amounts of text for various purposes.

Frame最下面的一行是回显区。它用来显示用于各种目的的少量文本。

The echo area is so-named because one of the things it is used for is echoing, which means displaying the characters of a multi-character command as you type. Single-character commands are not echoed. Multi-character commands (see Section 2.2 [Keys], page 11) are echoed if you pause for more than a second in the middle of a command. Emacs then echoes all the characters of the command so far, to prompt you for the rest. Once echoing has started, the rest of the command echoes immediately as you type it. This behavior is designed to give confident users fast response, while giving hesitant users maximum feedback.

Echo area这样命名是因为它的用途之一就是回显（echoing），也就是说显示键入的多字符命令的字符。单字符命令不会被回显。多字符命令在输入过程中（参见Section 2.2 [Keys], page 11）暂停超过一秒会回显。Emacs之后回显该命令目前输入的字符，来提示剩余部分。一旦回显开始，你键入的剩余部分会立即回显。设计该行为是为了给自信的用户快速的回应，同时给犹豫不决的用户最大程度的反馈。

The echo area is also used to display an error message when a command cannot do its job. Error messages may be accompanied by beeping or by flashing the screen.

Echo area常用来显示命令不能正常工作时的错误消息。错误消息可能会伴随着蜂鸣声或屏幕闪烁。

Some commands display informative messages in the echo area to tell you what the command has done, or to provide you with some specific information. These informative messages, unlike error messages, are not accompanied with a beep or flash. For example, C-x = (hold down CTRL and type x, then let go of CTRL and type =) displays a message describing the character at point, its position in the buffer, and its current column in the window. Commands that take a long time often display messages ending in ‘...’ while they are working (sometimes also indicating how much progress has been made, as a percentage), and add ‘done’ when they are finished.

一些命令在echo area显示通知消息来告诉你执行了什么命令，或者向你提供一些特殊的信息。这些不同于错误消息的通知消息，不会伴随蜂鸣或闪屏。例如，C-x =显示一个消息，用来描述point处字符在buffer中的位置，和在window中的当前列。需要很长时间的命令在执行的时候，往往显示以‘...’结尾的消息，（有时候也以百分比的形式标明执行进度），完成以后添加“done”。

Informative echo area messages are saved in a special buffer named \*Messages\*. (We have not explained buffers yet; see Chapter 16 [Buffers], page 142, for more information about them.) If you miss a message that appeared briefly on the screen, you can switch to the \*Messages\* buffer to see it again. The \*Messages\* buffer is limited to a certain number of lines, specified by the variable message-log-max. (We have not explained variables either; see Section 33.2 [Variables], page 412, for more information about them.) Beyond this limit, one line is deleted from the beginning whenever a new message line is added at the end.

通知类的会回显区消息保存在一个名为\*Messages\*的特殊buffer中。（我们还没有解释buffer；参见Chapter 16 [Buffers], page 142，了解关于它们的更多知识。）如果你错过了一个短暂显示在屏幕上的消息，你可以切换到\*Messages\* buffer来再看一次。\*Messages\* buffer可以通过变量（variables）message-log-max来限制显示的行数。（我们还没有解释变量，参见Section 33.2 [Variables], page 412）超出此限制，末尾添加一行新消息行就会在开始删除一行。

See Section 11.23 [Display Custom], page 85, for options that control how Emacs uses the echo area.

参见Section 11.23 [Display Custom], page 85，了解Emacs使用echo area的控制选项。

The echo area is also used to display the minibuffer, a special window where you can input arguments to commands, such as the name of a file to be edited. When the minibuffer is in use, the text displayed in the echo area begins with a prompt string, and the active cursor appears within the minibuffer, which is temporarily considered the selected window. You can always get out of the minibuffer by typing C-g. See Chapter 5 [Minibuffer], page 26.

Echo area也会用来显示迷你缓冲区（minibuffer），minibuffer是一个特殊的window，用来输入命令的参数，比如一个将要编辑的文件的名字。当使用minibuffer时，echo area显示的文本以一个提示字符串开始，活动的cursor显示在minibuffer中，此时minibuffer临时被认为是选中的window。通过键入C-g，你可以退出minibuffer。参见Chapter 5 [Minibuffer], page 26。

## The Mode Line（模式行）

At the bottom of each window is a mode line, which describes what is going on in the current buffer. When there is only one window, the mode line appears right above the echo area; it is the next-to-last line in the frame. On a graphical display, the mode line is drawn with a 3D box appearance. Emacs also usually draws the mode line of the selected window with a different color than that of unselected windows, in order to make it stand out.

每个windo的最下面是模式行（mode line），它当前buffer所处状态。如果只有一个window，mode line正好显示在echo area的上面；它是frame的倒数第二行。在图形显示中，mode line被绘制为一个3D方形外观。Emacs通常也会将选中的window的mode line绘制一个不同的颜色，用来和那些未选中的windows进行区分，使得它突出显示。

The text displayed in the mode line has the following format:

Mode line中用下面的格式显示文本：

cs:ch-fr buf pos line (major minor)

On a text terminal, this text is followed by a series of dashes extending to the right edge of the window. These dashes are omitted on a graphical display.

在文本终端，这些文本后面是一系列的虚线直到window的右侧边缘。图形显示中将忽略这些虚线。

The cs string and the colon character after it describe the character set and newline convention used for the current buffer. Normally, Emacs automatically handles these settings for you, but it is sometimes useful to have this information.

Cs字符串和冒号描述了当前buffer使用的字符集和换行符约定。通常，Emacs自动处理这些设置。但是有这个信息有时候也是有用的。

cs describes the character set of the text in the buffer (see Section 19.6 [Coding Systems], page 178). If it is a dash (‘-’), that indicates no special character set handling (with the possible exception of end-of-line conventions, described in the next paragraph). ‘=’ means no conversion whatsoever, and is usually used for files containing non-textual data. Other characters represent various coding systems—for example, ‘1’ represents ISO Latin-1.

Cs描述了buffer中文本的字符集。（参见Section 19.6 [Coding Systems], page 178）。如果它是一个虚线(‘-’)，就标明没有处理特殊的字符集（下一段中描述的行约定可能是一个例外）。‘=’表示没有任何转换，通过用语含有费文本数据的文件。其他的字符代表不同的编码系统，例如，‘1’ 代表 ISO Latin-1。

On a text terminal, cs is preceded by two additional characters that describe the coding systems for keyboard input and terminal output. Furthermore, if you are using an input method, cs is preceded by a string that identifies the input method (see Section 19.4 [Input Methods], page 175).

在文本终端，cs前面两个额外的字符用来描述键盘输入和终端输出所使用的编码系统。此外，如果你正在使用一个输入法，cs前面还有一个用来致命输入法的字符串（参见Section 19.4 [Input Methods], page 175）。

The character after cs is usually a colon. If a different string is displayed, that indicates a nontrivial end-of-line convention for encoding a file. Usually, lines of text are separated by newline characters in a file, but two other conventions are sometimes used. The MSDOS convention uses a “carriage-return” character followed by a “linefeed” character; when editing such files, the colon changes to either a backslash (‘\’) or ‘(DOS)’, depending on the operating system. Another convention, employed by older Macintosh systems, uses a “carriage-return” character instead of a newline; when editing such files, the colon changes to either a forward slash (‘/’) or ‘(Mac)’. On some systems, Emacs displays ‘(Unix)’ instead of the colon for files that use newline as the line separator.

Cs之后的字符通常是一个冒号。如果显示一个不同的字符串，表示编码文件使用了一个非平凡的行结束约定。通常，文件中的文本行被换行符分割，但有时也使用其他两个公约。MSDOS约定使用”回车-换行“符；当编辑此类文件时，冒号变为反斜杠(‘\’) 或 ‘(DOS)’，这取决于操作系统。另一个约定，旧的Macintosh系统采用的，使用回车字符来代替换行符；当编辑此类文件时，冒号变为斜杠(‘/’) 或 ‘(Mac)’。在一些系统中，对于使用换行符作为行分隔符的文件，Emacs显示‘(Unix)’而不是冒号。

The next element on the mode line is the string indicated by ch. This shows two dashes (‘--’) if the buffer displayed in the window has the same contents as the corresponding file on the disk; i.e., if the buffer is “unmodified”. If the buffer is modified, it shows two stars (‘\*\*’). For a read-only buffer, it shows ‘%\*’ if the buffer is modified, and ‘%%’ otherwise. The character after ch is normally a dash (‘-’). However, if the default-directory for the current buffer is on a remote machine, ‘@’ is displayed instead (see Section 15.1 [File Names], page 118).

Mode line的下一个元素是ch代表的字符串。如果显示在window中的buffer与对应的磁盘文件有相同内容的话就显示两个虚线(‘--’)；也就是说，buffer没有被修改。如果buffer被修改，它会显示两颗星(‘\*\*’)。对于一个只读buffer，如果被修改，它显示‘%\*’，否则是‘%%’。Ch之后的字符通常是一个虚线(‘-’)。然而，如果当前buffer的默认目录在远程机器上，‘@’就会取而代之。

fr gives the selected frame name (see Chapter 18 [Frames], page 157). It appears only on text terminals. The initial frame’s name is ‘F1’.

Fr给出了选定的frame名字（参见Chapter 18 [Frames], page 157）。它通常只出现在文本终端上。初始化的frame的名字是‘F1’。

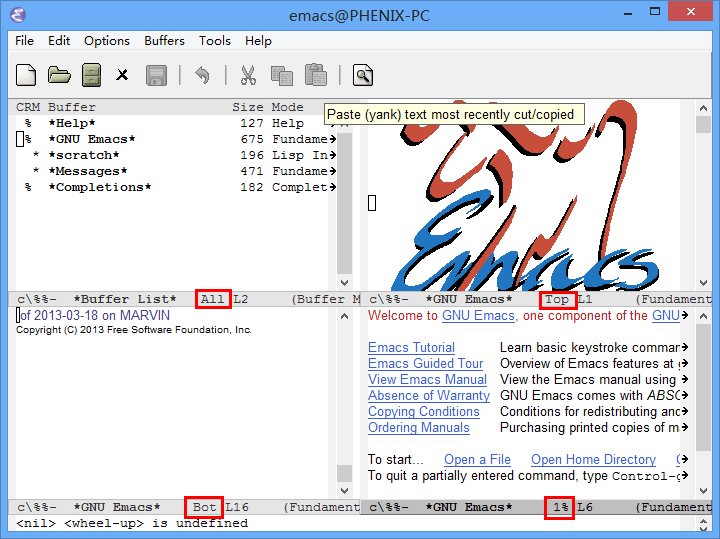
buf is the name of the buffer displayed in the window. Usually, this is the same as the name of a file you are editing. See Chapter 16 [Buffers], page 142.

Buff是window中显示的buffer的名字。通常与正在编辑的文件的名字相同。参见Chapter 16 [Buffers], page 142。

pos tells you whether there is additional text above the top of the window, or below the bottom. If your buffer is small and all of it is visible in the window, pos is ‘All’. Otherwise, it is ‘Top’ if you are looking at the beginning of the buffer, ‘Bot’ if you are looking at the end of the buffer, or ‘nn%’, where nn is the percentage of the buffer above the top of the window. With Size Indication mode, you can display the size of the buffer as well. See Section 11.18 [Optional Mode Line], page 81.

Pos说明是否有其他文本位于window的顶部之上或者底部之下。如果你的buffer比较小，所有都显示在window中，pos就是‘All’。否则，如果你在查看buffer开始它显示Top，buffer结尾显示Bot，或者是‘nn%’，其中nn是window顶部上方在buffer中的百分比。使用Size Indication mode，你同样可以显示buffer的大小。参见Section 11.18 [Optional Mode Line], page 81。

译者注：参考下图



**图1-7** Emcsa modeline pos

line is the character ‘L’ followed by the line number at point. (You can display the current column number too, by turning on Column Number mode. See Section 11.18 [Optional Mode Line], page 81).

Line是字符‘L’ 后跟point所在行号。（你可以通过启用Column Number mode显示当前列号。参见Section 11.18 [Optional Mode Line], page 81）。

major is the name of the major mode used in the buffer. A major mode is a principal editing mode for the buffer, such as Text mode, Lisp mode, C mode, and so forth. See Section 20.1 [Major Modes], page 193. Some major modes display additional information after the major mode name. For example, Compilation buffers and Shell buffers display the status of the subprocess.

Major是buffer中使用的主模式（major mode）。Major mode是一个buffer正在使用的编辑模式，比如 Text mode，Lisp mode，C mode等等。参见ection 20.1 [Major Modes], page 193。一些major mode在主模式名后会显示额外的信息。例如，编译缓冲区和shell缓冲区会显示子进程的状态。

minor is a list of some of the enabled minor modes, which are optional editing modes that provide additional features on top of the major mode. See Section 20.2 [Minor Modes], page 194.

Minor是一个已启用的辅助模式（minor mode）的列表，辅助模式是可选的编辑模式，它在主模式的基础上提供一些额外的功能。参见Section 20.2 [Minor Modes], page 194。

Some features are listed together with the minor modes whenever they are turned on, even though they are not really minor modes. ‘Narrow’ means that the buffer being displayed has editing restricted to only a portion of its text (see Section 11.5 [Narrowing], page 71). ‘Def’ means that a keyboard macro is currently being defined (see Chapter 14 [Keyboard Macros], page 110).

某些功能启用的时候会随着minor mode一起列出，即使它们并不真正的minor mode。‘Narrow’表示显示的buffer被严格限制为只能编辑部分文本。（参见Section 11.5 [Narrowing], page 71）。Def’表示当前正在定义一个键盘宏（keyboard macro）（参见Chapter 14 [Keyboard Macros], page 110）。

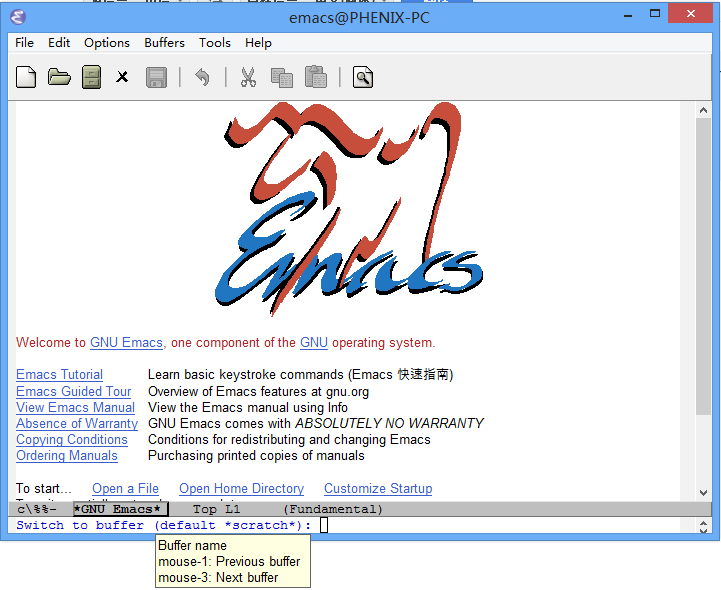
In addition, if Emacs is inside a recursive editing level, square brackets (‘[...]’) appear around the parentheses that surround the modes. If Emacs is in one recursive editing level within another, double square brackets appear, and square brackets (‘[...]’) appear around the parentheses that surround the modes. If Emacs is in one recursive editing level within another, double square brackets appear, and so on. Since recursive editing levels affect Emacs globally, such square brackets appear in the mode line of every window.so on. Since recursive editing levels affect Emacs globally, such square brackets appear in the mode line of every window. See Section 31.9 [Recursive Edit], page 394.

此外，如果Emacs处于递归编辑层，方括号(‘[...]’)出现在包围modes的括号的周围。如果一个递归编辑层次Emacs在另一个已经是递归层次里面，会出现两个方括号，等等。由于递归编辑层次全局影响Emacs，这样的方括号会出现在每一个window的mode line中。参见Section 31.9 [Recursive Edit], page 394。

You can change the appearance of the mode line as well as the format of its contents. See Section 11.18 [Optional Mode Line], page 81. In addition, the mode line is mouse-sensitive; clicking on different parts of the mode line performs various commands. See Section 18.5 [Mode Line Mouse], page 160.

可以改变mode line的外观和它的内容格式。参见Section 11.18 [Optional Mode Line], page 81。另外，mode line对鼠标是敏感的，点击mode line不同部分执行不同的命令。参见Section 18.5 [Mode Line Mouse], page 160。

译者注：可以将鼠标悬浮在mode line的相关元素上对应的查看，参考下图



**图1-8** Emacs-mouseon-modeline

## The Menu Bar（菜单栏）

Each Emacs frame normally has a menu bar at the top which you can use to perform common operations. There’s no need to list them here, as you can more easily see them yourself.

每个Emacs frame通常在顶部都有一个菜单栏（menu bar），可以用它来执行常用操作这里没有必要一一列举，它们显而易见。

On a graphical display, you can use the mouse to choose a command from the menu bar. An arrow on the right edge of a menu item means it leads to a subsidiary menu, or submenu. A ‘...’ at the end of a menu item means that the command will prompt you for further input before it actually does anything.

图形显示中，可以用鼠标从菜单栏中选择命令。菜单项右边缘的箭头意味着它会有导向一个子菜单。一个菜单项末尾的省略号意味着在它实际工作之前会提示你进一步输入。

Some of the commands in the menu bar have ordinary key bindings as well; if so, a key binding is shown in parentheses after the item itself. To view the full command name and documentation for a menu item, type C-h k, and then select the menu bar with the mouse in the usual way (see Section 7.1 [Key Help], page 39).

菜单栏的一些命令有普通的键绑定；如果是这样，一个键绑定会显示在菜单项后面的括号中。通过键入C-h k，来查看菜单项完整的命令名称和文档，按通常方法通过鼠标来选择菜单栏（参见Section 7.1 [Key Help], page 39）。

Instead of using the mouse, you can also invoke the first menu bar item by pressing F10 (to run the command menu-bar-open). You can then navigate the menus with the arrow keys. To activate a selected menu item, press RET; to cancel menu navigation, press ESC.

不使用鼠标，也可以通过按下F10来调用第一个菜单栏项目（也就是执行命令menu-bar-open）。之后通过方向键来导航菜单。按回车键激活菜单项；ESC键取消菜单导航。

On a text terminal, you can use the menu bar by typing M-‘ or F10 (these run the command tmm-menubar). This lets you select a menu item with the keyboard. A provisional choice appears in the echo area. You can use the up and down arrow keys to move through the menu to different items, and then you can type RET to select the item. Each menu item is also designated by a letter or digit (usually the initial of some word in the item’s name). This letter or digit is separated from the item name by ‘==>’. You can type the item’s letter or digit to select the item.

在文本终端，通过键入M-‘ 或F10 (这些运行命令tmm-menubar)来使用菜单栏。这可以让你通过键盘来选择一个菜单项。一个临时的选择会出现在echo area中。你可以使用上下方向键在菜单中移动到不同的菜单项，然后按下回车键选中他们。每个菜单项设计为一个字母或数字（通常该菜单项名字的最初字母）。‘==>’分开了该字母或数字和菜单项名字。你可以键入这些菜单项的字母或数字来选中菜单项。

# ****Characters, Keys and Commands****（****字符集, 键和命令）****

This chapter explains the character sets used by Emacs for input commands, and the fundamental concepts of keys and commands, whereby Emacs interprets your keyboard and mouse input.

本章介绍Emacs输入命令所使用的字符集，按键和命令的基本概念，Emacs据此来解释键盘和鼠标输入。

## Kinds of User Input（不同的用户输入）

GNU Emacs is primarily designed for use with the keyboard. While it is possible to use the mouse to issue editing commands through the menu bar and tool bar, that is not as efficient as using the keyboard. Therefore, this manual mainly documents how to edit with the keyboard.

GNU Emacs主要针对使用键盘进行设计。虽然有可能使用鼠标通过菜单栏和工具栏来处理编辑命令，但是没有通过使用键盘效率高。因而，因而该手册主要记录如何使用键盘进行编辑。

Keyboard input into Emacs is based on a heavily-extended version of ASCII. Simple characters, like ‘a’, ‘B’, ‘3’, ‘=’, and the space character (denoted as SPC), are entered by typing the corresponding key. Control characters, such as RET, TAB, DEL, ESC, F1, Home, and left, are also entered this way, as are certain characters found on non-English keyboards (see Chapter 19 [International], page 171).

到Emacs的键盘输入是基于ASCII的高度扩展版本。简单的字符，如‘a’, ‘B’, ‘3’, ‘=’,和空白符（记为SPC）,都可以通过相应的按键进行输入。控制字符，如RET, TAB, DEL, ESC, F1, Home, 和 left也可以用这种方法进行输入，非英语键盘上的字符也可这样处理。（参见Chapter 19 [International], page 171）。

Emacs also recognizes control characters that are entered using modifier keys. Two commonly-used modifier keys are Control (usually labeled Ctrl), and Meta (usually labeled Alt) 1. For example, Control-a is entered by holding down the Ctrl key while pressing a; we will refer to this as C-a for short. Similarly Meta-a, or M-a for short, is entered by holding down the Alt key and pressing a. Modifier keys can also be applied to non-alphanumerical characters, e.g., C-F1 or M-left.

Emacs同样可以识别通过修饰键输入的控制字符。两种常用的修饰键是Control（通常标示为Ctrl），和Meta（通常标示为Alt）1。例如，按住Ctrl键的同时按下a可以输入Control-a；我们将会把这它简称为C-a。同样，Meta-a, 或 M-a可以通过按住Alt键同时按下a。修饰键同样可用于非字母数字字符，例如C-F1 或M-left。

1 We refer to Alt as Meta for historical reasons。

由于历史原因我们将Alt引用为Meta。

You can also type Meta characters using two-character sequences starting with ESC. Thus, you can enter M-a by typing ESC a. You can enter C-M-a by typing ESC C-a. Unlike Meta, ESC is entered as a separate character. You don’t hold down ESC while typing the next character; instead, press ESC and release it, then enter the next character. This feature is useful on certain text terminals where the Meta key does not function reliably.

也可以通过使用以ESC开始的双字符序列来输入Meta字符。因此可以通过键入ESC a来输入M-a。可以通过键入ESC-C-a来输入C-M-a。不同于Meta，ESC作为一个单独的字符输入。在输入下一个字符的时候不要按住ESC；相反，应该按下ESC然后释放它，然后输入下一个字符。这个功能在那些Meta键并不可用的文本中是有用的。

On graphical displays, the window manager might block some keyboard inputs, including M-TAB, M-SPC, C-M-d and C-M-l. If you have this problem, you can either customize your window manager to not block those keys, or “rebind” the affected Emacs commands (see Chapter 33 [Customization], page 404).

在图形终端，窗口管理器可能会组织一些键盘输入，包括M-TAB, M-SPC, C-M-d and C-M-l。如果有这样的问题，可以修改窗口管理器来不要阻止这些按键，或者”重新绑定“那些受影响的Emacs命令（参见Chapter 33 [Customization], page 404）。

Simple characters and control characters, as well as certain non-keyboard inputs such as mouse clicks, are collectively referred to as input events. For details about how Emacs internally handles input events, see Section “Input Events” in The Emacs Lisp Reference Manual.

简单的字符和控制字符，以及类似鼠标点击这样特定的非键盘输入，被统称为输入事件。Emacs如何交互的处理输入事件的细节，参见Section “Input Events” in The Emacs Lisp Reference Manual。

## Keys（按键）

Some Emacs commands are invoked by just one input event; for example, C-f moves forward one character in the buffer. Other commands take two or more input events to invoke, such as C-x C-f and C-x 4 C-f.

一些Emacs的调用只需要一次输入事件；例如，C-f在buffer中向前移动一个字符。另外一些命令则需要两次或更多输入时间才能调用。如C-x C-f和C-x 4 C-f。

A key sequence, or key for short, is a sequence of one or more input events that is meaningful as a unit. If a key sequence invokes a command, we call it a complete key ; for example, C-f, C-x C-f and C-x 4 C-f are all complete keys. If a key sequence isn’t long enough to invoke a command, we call it a prefix key ; from the preceding example, we see that C-x and C-x 4 are prefix keys. Every key sequence is either a complete key or a prefix key.

键序列，简称为键，是一个或者多个输入事件所组成的序列，它们作为一个单元是有意义的。如果一个键序列能够调用一个命令，我们称之为完全键；例如C-f, C-x C-f 和C-x 4 C-f 都是完全键。如果一个键序列不足以达到调用一个命令的长度，称它为前缀键；前面的例子中，C-x 和 C-x 4 就是前缀键。每个键序列要么完全键要么是前缀键。

A prefix key combines with the following input event to make a longer key sequence. For example, C-x is a prefix key, so typing C-x alone does not invoke a command; instead, Emacs waits for further input (if you pause for longer than a second, it echoes the C-x key to prompt for that input; see Section 1.2 [Echo Area], page 7). C-x combines with the next input event to make a two-event key sequence, which could itself be a prefix key (such as C-x 4), or a complete key (such as C-x C-f). There is no limit to the length of key sequences, but in practice they are seldom longer than three or four input events.

前缀键和后面跟随的输入时间来构成一个更长的键序列。例如C-x是一个前缀键，所以单独键入C-x不会调用一个命令；相反，Emacs会等待进一步的输入（如果暂停超过一秒，会回显C-x提示输入；参见Section 1.2 [Echo Area] ，page 7。C-x与下一个输入事件组合构成一个双事件键序列，可能是一个前缀键（比如C-x 4），也可能是一个完成键（比如C-x C-f）。键序列的长度没有限制，但是实际中很少超过三个或四个输入事件。

You can’t add input events onto a complete key. For example, because C-f is a complete key, the two-event sequence C-f C-k is two key sequences, not one.

不能向一个完全键中添加输入事件。例如，应为C-f是一个完全键，C-f C-k就是两个键序列，而不是一个。

By default, the prefix keys in Emacs are C-c, C-h, C-x, C-x RET, C-x @, C-x a, C-x n, C-x r, C-x v, C-x 4, C-x 5, C-x 6, ESC, M-g, and M-o. (F1 and F2 are aliases for C-h and C-x 6.) This list is not cast in stone; if you customize Emacs, you can make new prefix keys. You could even eliminate some of the standard ones, though this is not recommended for most users; for example, if you remove the prefix definition of C-x 4, then C-x 4 C-f becomes an invalid key sequence. See Section 33.3 [Key Bindings], page 420.

默认情况下，Emacs中的前缀键是C-c, C-h, C-x, C-x RET, C-x @, C-x a, C-x n, C-x r, C-x v, C-x 4, C-x 5, C-x 6, ESC, M-g, and M-o。（F1 和F2是C-h和C-x 6的别名）。这个列表不是一成不变的；如果你对Emacs进行配置，你可以使用新的前缀键。你甚至可以删除一些标准的按键，尽管并不推荐大多数用户这么做；例如，如果你一出预先定义的C-x 4，那么C-x 4 C-f就会变成一个无效的键序列。参见Section 33.3 [Key Bindings], page 420。

Typing the help character (C-h or F1) after a prefix key displays a list of the commands starting with that prefix. The sole exception to this rule is ESC: ESC C-h is equivalent to C-M-h, which does something else entirely. You can, however, use F1 to display a list of commands starting with ESC. E

在前缀键后键入帮助字符（C-h or F1）会显示以该前缀键开头的命令列表。此规则唯一的例外是ESC，SC C-h等同于C-M-h，这完全是另外一回事。然而，可以使用F1来显示以ESC开始的命令列表。

## Keys and Commands（键和命令）

This manual is full of passages that tell you what particular keys do. But Emacs does not assign meanings to keys directly. Instead, Emacs assigns meanings to named commands, and then gives keys their meanings by binding them to commands.

该手册是完整的段落，告诉你特定的按键可以做什么。但是Emacs不直接向键指定特定的含义。相反，Emacs指定特定的含义到命名的命令，然后通过将键绑定到命令来给键分配含意。

Every command has a name chosen by a programmer. The name is usually made of a few English words separated by dashes; for example, next-line or forward-word. Internally, each command is a special type of Lisp function, and the actions associated with the command are performed by running the function. See Section “What Is a Function” in The Emacs Lisp Reference Manual.

每个命令都会被程序员选定一个名字。名字通常是有破折号分割的一些英语单词；例如next-line和forward-word。在内部，每个命令都是一种特殊类型的Lisp函数，通过运行该函数来完成于命令相关联的动作。参见Section “What Is a Function” in The Emacs Lisp Reference Manual。

The bindings between keys and commands are recorded in tables called keymaps. See Section 33.3.1 [Keymaps], page 421.

键和命令的绑定记录在叫做键映射（keymaps）的表中。参见Section 33.3.1 [Keymaps], page 421。

When we say that “C-n moves down vertically one line” we are glossing over a subtle distinction that is irrelevant in ordinary use, but vital for Emacs customization. The command next-line does a vertical move downward. C-n has this effect because it is bound to next-line. If you rebind C-n to the command forward-word, C-n will move forward one word instead.

当我们说“C-n纵向向下移动一行”时，我们正在掩盖一个和普通使用无关的细微差别，但是这对于Emacs定制来说是至关重要的。next-line命令竖直向下移动。C-n有这样的效果是因为它绑定到到next-line。如果将C-n重新绑定到forward-word，C-n反而会向前移动一个单词。

In this manual, we will often speak of keys like C-n as commands, even though strictly speaking the key is bound to a command. Usually we state the name of the command which really does the work in parentheses after mentioning the key that runs it. For example, we will say that “The command C-n (next-line) moves point vertically down”, meaning that the command next-line moves vertically down, and the key C-n is normally bound to it.

本手册中我们将会经常将类似C-n的按键称作命令，即使严格的来说应该是绑定到命令的按键。通常我们提及命令的名字的时候，会将它放在运行该命令的按键后面的括号中。例如，我们会这样说，“命令C-n（next-line）垂直移动向下移动point”，意味着命令next-line垂直向下移动，并且该命令通常绑定到C-n。

Since we are discussing customization, we should tell you about variables. Often the description of a command will say, “To change this, set the variable mumble-foo.” A variable is a name used to store a value. Most of the variables documented in this manual are meant for customization: some command or other part of Emacs examines the variable and behaves differently according to the value that you set. You can ignore the information about variables until you are interested in customizing them. Then read the basic information on variables (see Section 33.2 [Variables], page 412) and the information about specific variables will make sense.

由于我们讨论到定制，就应该说说变量相关的东西。命令的描述往往会说，“为了改变这种情况，设置变量mumble-foo”。变量时一个用来存储值得名字。该手册中说明的大多数变量都是针对定制的：Emacs的一些命令或者其他部分检查变量并且根据你设置的不同值来表现出不同的行为。在你没有对定制变量感兴趣之前你可以忽略他们。阅读有关变量的基本信息（参见）和特定的变量的信息才会有意义。

# ****Entering and Exiting Emacs(进入和退出Emacs)****

**This chapter explains how to enter Emacs, and how to exit it.**

**本章介绍如何进入和退出Emacs。**

## **Entering Emacs（进入Emacs）**

**The usual way to invoke Emacs is with the shell command Emacs. From a terminal window running in the X Window System, you can run Emacs in the background with Emacs &; this way, Emacs won’t tie up the terminal window, so you can use it to run other shell commands.**

**调用Emacs的常用方法是使用shell命令Emacs。在X window系统中运行的终端窗口中，可用使用Emacs &在后台运行Emacs；这种方法，Emacs不会占用终端窗口，所以你可以使用它来运行其他的shell命令。**

**When Emacs starts up, the initial frame displays a special buffer named ‘\*GNU Emacs\*’. This startup screen contains information about Emacs and links to common tasks that are useful for beginning users. For instance, activating the ‘Emacs Tutorial’ link opens the Emacs tutorial; this does the same thing as the command C-h t (help-with-tutorial). To activate a link, either move point onto it and type RET, or click on it with mouse-1 (the left mouse button).**

**Emacs启动之后，最初的frame显示一个叫做“GNU Emacs”的特殊buffer。这个启动画面包含关于Emacs的信息和一些对初级用户有用的任务的链接。例如，激活“Emacs 教程”的链接打开Emacs教程；这和命令C-h t（help-with-tutorial）做同样的事情。将point移动到链接上，按下回车键就可以激活它，也可以使用鼠标左键点击。**

**Using a command line argument, you can tell Emacs to visit one or more files as soon as it starts up. For example, Emacs foo.txt starts Emacs with a buffer displaying the contents of the file ‘foo.txt’. This feature exists mainly for compatibility with other editors, which are designed to be launched from the shell for short editing sessions. If you call Emacs this way, the initial frame is split into two windows—one showing the specified file, and the other showing the startup screen. See Chapter 17 [Windows], page 151.**

**使用命令行参数，可以告知Emacs启动之后尽快来访问一个或多个文件。例如，Emacs foo.txt使Emacs在启动后显示文件“foo.txt”的内容。存在这个功能主要是为了兼容那些从shell启动进行短暂编辑的编辑器。如果这样调用Emacs，初始frame会分割为两个window，一个显示制定的文件，另一个显示启动画面。**

**Generally, it is unnecessary and wasteful to start Emacs afresh each time you want to edit a file. The recommended way to use Emacs is to start it just once, just after you log in, and do all your editing in the same Emacs session. See Chapter 15 [Files], page 118, for information on visiting more than one file. If you use Emacs this way, the Emacs session accumulates valuable context, such as the kill ring, registers, undo history, and mark ring data, which together make editing more convenient. These features are described later in the manual.**

**一般来说，每次编辑文件的时候重新启动Emacs是不必要和浪费的。推荐做法是在登录后启动Emacs，在同样的Emacs过程中处理所有的编辑。参见Chapter 15 [Files], page 118，了解访问多个文件。如果这样使用Emacs，Emacs会积累有价值的上下文，比如剪切环（kill ring），寄存器（registers），撤销历史记录（undo history）和标记环数据（mark ring data），这些都会使编辑更加方便。这些功能会在手册后面进行描述。**

**To edit a file from another program while Emacs is running, you can use the Emacsclient helper program to open a file in the existing Emacs session. See Section 31.4 [Emacs Server], page 384.**

**当Emacs已经在运行了，如果要从另一个程序编辑一个文件，可以使用Emacsclient帮助程序在一个已经存在的Emacs会话中打开文件。参见Section 31.4 [Emacs Server], page 384。**

**Emacs accepts other command line arguments that tell it to load certain Lisp files, where to put the initial frame, and so forth. See Appendix C [Emacs Invocation], page 469. If the variable inhibit-startup-screen is non-nil, Emacs does not display the startup screen. In that case, if one or more files were specified on the command line, Emacs simply displays those files; otherwise, it displays a buffer named \*scratch\*, which can be used to evaluate Emacs Lisp expressions interactively. See Section 24.10 [Lisp Interaction], page 272. You can set the variable inhibit-startup-screen using the Customize facility (see Section 33.1 [Easy Customization], page 404), or by editing your initialization file (see Section 33.4 [Init File], page 429).1**

**Emacs接受其他的命令行参数来告诉它去加载特定的lisp文件，把初始frame放在何处等等。参见Appendix C [Emacs Invocation], page 469。如果变量inhibit-startup-screen是non-nil，Emacs就不会显示启动画面。那种情况下，如果命令行上指明一个或多个文件，Emacs会简单的显示这些文件；否则，他会显示一个名叫\*scratch\*的buffer，在其中可以直接交互的计算Emacs Lisp表达式。参见Section 24.10 [Lisp Interaction], page 272。可以使用定制功能来设置变量inhibit-startup-screen（参见Section 33.1 [Easy Customization], page 404），也可编辑配置文件（参见Section 33.4 [Init File], page 429）1。**

**1 Setting inhibit-startup-screen in site-start.el doesn’t work, because the startup screen is set up before reading site-start.el. See Section 33.4 [Init File], page 429, for information about sitestart.el.**

**1 在site-start.el中设置inhibit-startup-screen不会生效，因为启动画面会在读取site-start.el前成立。参见Section 33.4 [Init File], page 429，了解关于sitestart.el的信息。**

**You can also force Emacs to display a file or directory at startup by setting the variable initial-buffer-choice to a non-nil value. (In that case, even if you specify one or more files on the command line, Emacs opens but does not display them.) The value of initial-buffer-choice should be the name of the desired file or directory.**

**可以通过设置变量initial-buffer-choice来强制Emacs启动的时候显示一个文件或目录。（在这种情况下，即使你在命令行上制定了一个或多个文件，Emacs只会打开但是不显示它们）。initial-buffer-choice的值应该是所需的文件或者目录的名字。**

## **Exiting Emacs（退出Emacs）**

**C-x C-c Kill Emacs (save-buffers-kill-terminal).**

**C-x C-c 关闭Emacs（save-buffers-kill-terminal）。**

**C-z On a text terminal, suspend Emacs; on a graphical display, “minimize” the selected frame (suspend-Emacs).**

**C-z On 文本终端中挂起Emacs；图形显示中，“最小化”选中的frame(suspend-Emacs)。**

***Killing* Emacs means terminating the Emacs program. To do this, type C-x C-c (savebuffers-kill-terminal). A two-character key sequence is used to make it harder to type by accident. If there are any modified file-visiting buffers when you type C-x C-c, Emacs first offers to save these buffers. If you do not save them all, it asks for confirmation again, since the unsaved changes will be lost. Emacs also asks for confirmation if any subprocesses are still running, since killing Emacs will also kill the subprocesses (see Section 31.3 [Shell], page 374).**

**Killing Emacs意味着终止Emacs程序。键入C-x C-c (savebuffers-kill-terminal)可以做到这个。使用双字符键序列用来避免按键失误。如果键入C-x C-c的时候正在访问的buffer中有修改的文件，Emacs首先会保存这些buffer，如果没有全部保存它们，Emacs会再次进行确认，因为未保存的修改回丢失。如果有挂起的子进程在运行，Emacs也会要求确认，因为killing Emacs也会杀死这些子进程（参见Section 31.3 [Shell], page 374）。**

**C-x C-c behaves specially if you are using Emacs as a server. If you type it from a “client frame”, it closes the client connection. See Section 31.4 [Emacs Server], page 384.**

**如果你将Emacs作为一个服务器，C-x C-c会做出不一样的行为。如果从一个“client frame”键入，它关闭客户端连接。参见Section 31.4 [Emacs Server], page 384。**

**Emacs can, optionally, record certain session information when you kill it, such as the files you were visiting at the time. This information is then available the next time you start Emacs. See Section 31.8 [Saving Emacs Sessions], page 393. If the value of the variable confirm-kill-Emacs is non-nil, C-x C-c assumes that its value is a predicate function, and calls that function. If the result of the function call is non-nil, the session is killed, otherwise Emacs continues to run. One convenient function to use as the value of confirm-kill-Emacs is the function yes-or-no-p. The default value of confirm-kill-Emacs is nil.**

**Emacs能够在关闭时选择性的记录特定的会话信息，比如此时正在访问的文件。这些信息在下次启动Emacs的时候是可用的。参见Section 31.8 [Saving Emacs Sessions], page 393。如果变量confirm-kill-Emacs的值是non-nil，C-x C-c假设它的值是一个谓词函数，然后调用该谓词函数。如果谓词函数的返回值是non-nil，杀死会话，否则Emacs继续运行。一个方便使用confirm-kill-Emacs的值的函数就是yes-or-no-p。confirm-kill-Emacs默认值是nil。**

**To kill Emacs without being prompted about saving, type M-x kill-Emacs.**

**不提示关于保存的kill Emacs，输入M-x kill-Emacs。**

**C-z runs the command suspend-frame. On a graphical display, this command minimizes (or iconifies ) the selected Emacs frame, hiding it in a way that lets you bring it back later (exactly how this hiding occurs depends on the window system). On a text terminal, the C-z command suspends Emacs, stopping the program temporarily and returning control to the parent process (usually a shell); in most shells, you can resume Emacs after suspending it with the shell command %Emacs.**

**C-z会执行命suspend-frame。图形显示中，该命令会最小化（或）选中的Emacs frame，以一种方式藏起来，自后还可以将它带回来（具体来说如何隐藏依赖于窗口系统）。在文本终端，C-z命令挂起Emacs，暂停程序并把控制权交还父进程（通常是shell）；在大多数的shells中，挂起Emacs后可以使用命令%Emacs进行恢复。**

**Text terminals usually listen for certain special characters whose meaning is to kill or suspend the program you are running. This terminal feature is turned off while you are in Emacs. The meanings of C-z and C-x C-c as keys in Emacs were inspired by the use of C-z and C-c on several operating systems as the characters for stopping or killing a program, but that is their only relationship with the operating system. You can customize these keys to run any commands of your choice (see Section 33.3.1 [Keymaps], page 421).**

**文本终端通常监听那些用来杀死或者挂起你正在运行的程序的字符。这种终端功能在Emacs是关闭的。Emacs中按键C-z 和C-x C-c功能来自于几种操作系统中C-z 和 C-x C-c作为停止和杀死一个程序的字符，但这是它们和操作系统唯一的关系。可以根据自己的选择来定义这些按键来运行任何功能（参见Section 33.3.1 [Keymaps], page 421）。**

# ****Basic Editing Commands （基本编辑命令）****

## **Continuation Lines（续行）**

Sometimes, a line of text in the buffer—a logical line—is too long to fit in the window, and Emacs displays it as two or more screen lines. This is called line wrapping or continuation, and the long logical line is called a continued line. On a graphical display, Emacs indicates line wrapping with small bent arrows in the left and right window fringes. On a text terminal, Emacs indicates line wrapping by displaying a ‘\’ character at the right margin.

有时候，buffer中文本的行（逻辑行：logical line）对于window来说太长了，Emacs就会将它显示为两个或者更多的屏幕行（screen lines）。这叫做自动换行或者续行，长的逻辑航被叫做

Most commands that act on lines act on logical lines, not screen lines. For instance, C-k kills a logical line. As described earlier, C-n (next-line) and C-p (previous-line) are special exceptions: they move point down and up, respectively, by one screen line (see Section 4.2 [Moving Point], page 17).

Emacs can optionally truncate long logical lines instead of continuing them. This means that every logical line occupies a single screen line; if it is longer than the width of the window, the rest of the line is not displayed. On a graphical display, a truncated line is indicated by a small straight arrow in the right fringe; on a text terminal, it is indicated by a ‘$’ character in the right margin. See Section 11.21 [Line Truncation], page 84.

By default, continued lines are wrapped at the right window edge. Since the wrapping may occur in the middle of a word, continued lines can be difficult to read. The usual solution is to break your lines before they get too long, by inserting newlines. If you prefer, you can make Emacs insert a newline automatically when a line gets too long, by using Auto Fill mode. See Section 22.5 [Filling], page 206.

Sometimes, you may need to edit files containing many long logical lines, and it may not be practical to break them all up by adding newlines. In that case, you can use Visual Line mode, which enables word wrapping : instead of wrapping long lines exactly at the right window edge, Emacs wraps them at the word boundaries (i.e., space or tab characters) nearest to the right window edge. Visual Line mode also redefines editing commands such as C-a, C-n, and C-k to operate on screen lines rather than logical lines. See Section 11.22 [Visual Line Mode], page 85.

# ****The Minibuffer（迷你缓冲区）****

# ****Running Commands by Name（通过名字执行程序）****

**Every Emacs command has a name that you can use to run it. For convenience, many commands also have key bindings. You can run those commands by typing the keys, or run them by name. Most Emacs commands have no key bindings, so the only way to run them is by name. (See Section 33.3 [Key Bindings], page 420, for how to set up key bindings.)**

**By convention, a command name consists of one or more words, separated by hyphens; for example, auto-fill-mode or manual-entry. Command names mostly use complete English words to make them easier to remember.**

**To run a command by name, start with M-x, type the command name, then terminate it with RET. M-x uses the minibuffer to read the command name. The string ‘M-x’ appears at the beginning of the minibuffer as a prompt to remind you to enter a command name to be run. RET exits the minibuffer and runs the command. See Chapter 5 [Minibuffer], page 26, for more information on the minibuffer.**

**You can use completion to enter the command name. For example, to invoke the command forward-char, you can type**

**M-x forward-char RET**

**or**

**M-x forw TAB c RET**

**Note that forward-char is the same command that you invoke with the key C-f. The existence of a key binding does not stop you from running the command by name.**

**To cancel the M-x and not run a command, type C-g instead of entering the command name. This takes you back to command level.**

**To pass a numeric argument to the command you are invoking with M-x, specify the numeric argument before M-x. The argument value appears in the prompt while the command name is being read, and finally M-x passes the argument to that command.**

**When the command you run with M-x has a key binding, Emacs mentions this in the echo area after running the command. For example, if you type M-x forward-word, the message says that you can run the same command by typing M-f. You can turn off these messages by setting the variable suggest-key-bindings to nil.**

**In this manual, when we speak of running a command by name, we often omit the RET that terminates the name. Thus we might say M-x auto-fill-mode rather than M-x auto-fill-mode RET. We mention the RET only for emphasis, such as when the command is followed by arguments.**

**M-x works by running the command execute-extended-command, which is responsible for reading the name of another command and invoking it.**

# ****Help****

# ****The Mark and the Region**** （标记和区域）

Many Emacs commands operate on an arbitrary contiguous part of the current buffer. To specify the text for such a command to operate on, you set the mark at one end of it, and move point to the other end. The text between point and the mark is called the region. The region always extends between point and the mark, no matter which one comes earlier in the text; each time you move point, the region changes.

很多Emacs命令操作于当前buffer上任意连续的部分。为了指定命令操作的文本，需要在文本的一端设置标记（mark），然后将位点（point）移动到另一端。标记（mark）和位点（point）之间的文本称作区域（region）。区域（region）总是在标记（mark）和位点（point）之间延伸，不管哪个先出现在文本中；每次移动位点（point），区域（region）都会改变。

Setting the mark at a position in the text also activates it. When the mark is active, we say also that the region is active; Emacs indicates its extent by highlighting the text within it, using the region face (see Section 33.1.5 [Face Customization], page 408).

在文本的位置设置标记（mark）的同时会激活标记。如果标记（mark）被激活，区域（region）同样会激活；Emacs会使用高亮区域（region）内的文本来指示区域的范围，使用region face（参见Section 33.1.5 [Face Customization], page 408）。

After certain non-motion commands, including any command that changes the text in the buffer, Emacs automatically deactivates the mark; this turns off the highlighting. You can also explicitly deactivate the mark at any time, by typing C-g (see Section 34.1 [Quitting], page 435).

执行一些特定的非移动命令后，包括任何改变buffer中文本的命令，Emacs会自动使标记（mark）失效；这样也会关闭高亮。可以在任何时候键入C-g来使标记（mark）失效（参见Section 34.1 [Quitting], page 435）。

The above default behavior is known as Transient Mark mode. Disabling Transient Mark mode switches Emacs to an alternative behavior, in which the region is usually not highlighted. See Section 8.7 [Disabled Transient Mark], page 50.

上面的默认行为作为Transient Mark mode被人所知。关闭Transient Mark mode将Emacs切换到一个备选的行为，这样区域（region）通常不会高亮。参见Section 8.7 [Disabled Transient Mark], page 50。

Setting the mark in one buffer has no effect on the marks in other buffers. When you return to a buffer with an active mark, the mark is at the same place as before. When multiple windows show the same buffer, they can have different values of point, and thus different regions, but they all share one common mark position. See Chapter 17 [Windows], page 151. Ordinarily, only the selected window highlights its region; however, if the variable highlight-nonselected-windows is non-nil, each window highlights its own region.

在一个buffer中设置标记（mark）对其他缓冲区（buffer）中的标记（mark）没有影响。当回到一个有激活标记（mark）的缓冲区（buffer）的时候，标记（mark）还停在原处。如果同一个buffer中显示在多个window中，他们会有各自不同的point值，因而会有各自不同的region，但是都有一个共同的标记（mark）位置。参见Chapter 17 [Windows], page 151。通常，只有选中的window高亮其region，如果变量highlight-nonselected-windows是non-nil，每个window都回高亮各自的region。

## Setting the Mark（设置标记）

Here are some commands for setting the mark:

这里是一些用来设置mark的命令。

|  |  |
| --- | --- |
| C-SPC | Set the mark at point, and activate it (set-mark-command).  在point处设置mark，并激活它（set-mark-command） |
| C-@ | The same.（同上） |
| C-x C-x | Set the mark at point, and activate it; then move point where the mark used to be (exchange-point-and-mark).  在point处设置mark，并激活它；然后将point移动到上一个mark处（exchange-point-and-mark） |
| Drag-Mouse-1 | Set point and the mark around the text you drag across.  在拖过的文本周围设置point和标记。 |
| Mouse-3 | Set the mark at point, then move point to where you click (mouse-save-then-kill).  在point处设置mark，然后将point移动到点击处（mouse-save-then-kill）。 |
| ‘Shifted cursor motion keys’ | Set the mark at point if the mark is inactive, then move point. See Section 8.6 [Shift Selection], page 49.  如果mark失效，就在point处设置mark，然后移动point。参见See Section 8.6 [Shift Selection], page 49。 |

The most common way to set the mark is with C-SPC (set-mark-command) 1. This sets the mark where point is, and activates it. You can then move point away, leaving the mark behind.

设置mark最常用的方式是使用C-SPC(set-mark-command)1.它会在point之处设置mark并激活它。之后可以不管mark，将point移走了。

1 There is no C-SPC character in ASCII; usually, typing C-SPC on a text terminal gives the character C-@. This key is also bound to set-mark-command, so unless you are unlucky enough to have a text terminal that behaves differently, you might as well think of C-@ as C-SPC.

1 在ASCII中没有C-SPC字符；童话藏在文本终端中键入C-SPC给出的字符是C-@。该键也被绑定到set-mark-command，所以除非倒霉透顶得使用了一个行为不同的文本终端，可以把C-@当作C-SPC。

For example, suppose you wish to convert part of the buffer to upper case. To accomplish this, go to one end of the desired text, type C-SPC, and move point until the desired portion of text is highlighted. Now type C-x C-u (upcase-region). This converts the text in the region to upper case, and then deactivates the mark.

例如，假设想把buffer的部分转变大写。为了做到这一点，在所需文本的一端按下C-SPC，然后移动point直到另一端直到所需文本都被高亮。现在按下C-x C-u (upcase-region)。这样就会把region中的文本都变为大写，并使mark失效。

Whenever the mark is active, you can deactivate it by typing C-g (see Section 34.1 [Quitting], page 435). Most commands that operate on the region also automatically deactivate the mark, like C-x C-u in the above example.

每当mark激活之后，都可以使用C-g来使其失效（参见Section 34.1 [Quitting], page 435）。大多数操作于region的命令都回自动使mark失效，例如上面例子中的C-x C-u。

Instead of setting the mark in order to operate on a region, you can also use it to “remember” a position in the buffer (by typing C-SPC C-SPC), and later jump back there (by typing C-u C-SPC). See Section 8.4 [Mark Ring], page 48, for details.

除了为了操作region而设置mark，也可以使用它来“记住”buffer中的一个位置。（通过键入C-SPC C-SPC），，之后可以跳回那里（通过键入C-u C-SPC）。详见Section 8.4 [Mark Ring], page 48。

The command C-x C-x (exchange-point-and-mark) exchanges the positions of point and the mark. C-x C-x is useful when you are satisfied with the position of point but want to move the other end of the region (where the mark is). Using C-x C-x a second time, if necessary, puts the mark at the new position with point back at its original position. Normally, if the mark is inactive, this command first reactivates the mark wherever it was last set, to ensure that the region is left highlighted. However, if you call it with a prefix argument, it leaves the mark inactive and the region unhighlighted; you can use this to jump to the mark in a manner similar to C-u C-SPC.

命令C-x C-x (exchange-point-and-mark)交换mark和point的位置。当对point的位置满意，但是想移动到region的另一端（mark所在之处）的时候，C-x C-x是有用的。 如果必要的话，第二次使用C-x C-x会将mark设置在新的位置，而point跳回原来的位置。通常情况下，如果mark是失效的，这个命令首先会在它上次设置的地方激活它，来确保region是从左开始高亮的。然而，如果使用它的时候伴随一个前缀参数，它会使标记失效，region不高亮；可以这样使用来跳转到mark，类似C-u C-SPC的行为

You can also set the mark with the mouse. If you press the left mouse button (down-mouse-1) and drag the mouse across a range of text, this sets the mark where you first pressed the mouse button and puts point where you release it. Alternatively, clicking the right mouse button (mouse-3) sets the mark at point and then moves point to where you clicked. See Section 18.1 [Mouse Commands], page 157, for a more detailed description of these mouse commands.

也可以使用鼠标来设置mark。如果按下鼠标左键(down-mouse-1)，然后划过一个范围的文本，这将会在按下鼠标的地方设置mark，在释放鼠标的地方设置point。另外，点击鼠标右键(mouse-3)会在point所在之处设置mark，然后将point移动到下一次点击的地方。参见See Section 18.1 [Mouse Commands], page 157，了解这些鼠标命令的详细信息。

Finally, you can set the mark by holding down the shift key while typing certain cursor motion commands (such as S-right, S-C-f, S-C-n, etc.). This is called shift-selection. It sets the mark at point before moving point, but only if there is no active mark set via shift-selection. The mark set by mouse commands and by shift-selection behaves slightly differently from the usual mark: any subsequent unshifted cursor motion command deactivates it automatically. For details, See Section 8.6 [Shift Selection], page 49.

最后，在键入特定cursor移动命令（比如S-right, S-C-f, S-C-n等等）的时候按住shift键也可以设置mark。这称作移位选择（shift-selection）。但是只有在没有激活的mark，移位选择才会在移动point之前在point处设置mark。鼠标命令和移位选择设置的mark与平时的mark略有不同：任何后续的非移动的cursor移动命令都会自动使其失效。详见Section 8.6 [Shift Selection], page 49。

Many commands that insert text, such as C-y (yank), set the mark at the other end of the inserted text, without activating it. This lets you easily return to that position (see Section 8.4 [Mark Ring], page 48). You can tell that a command does this when it shows ‘Mark set’ in the echo area.

插入文本的很多命令，比如C-y (yank)，会在插入文本的另一端设置标记，但不会激活它。这使得能够容易的返回到该位置（参见）。当命令在echo area显示“Mark set”可以通过命令这么做。

Under X, every time the active region changes, Emacs saves the text in the region to the primary selection. This lets you insert that text into other X applications with mouse-2 clicks. See Section 9.3.2 [Primary Selection], page 57.

在X系统下，每次激活的region改变，Emacs都回将region中的文本保存为主要选择。这使得可以在其他的X 应用中点击鼠标中键来插入该文本。

## Commands to Mark Textual Objects（标记文本对象的命令）

Here are commands for placing point and the mark around a textual object such as a word, list, paragraph or page:

有一些命令可以在文本对象，比如a word, list, paragraph or page，周围来放置point或mark。

|  |  |
| --- | --- |
| M-@ | Set mark after end of next word (mark-word). This does not move point.  在下一个单词（mark-word）的结尾设置mark。不会移动point。 |
| C-M-@ | Set mark after end of following balanced expression (mark-sexp). This does not move point.  在接下来的平衡表达式（mark-sexp）结尾设置mark。不会移动mark。 |
| M-h | Move point to the beginning of the current paragraph, and set mark at the end (mark-paragraph).  移动point到当前段落的开始，并在段落结尾设置标记（mark-paragraph）。 |
| C-M-h | Move point to the beginning of the current defun, and set mark at the end (mark-defun).  移动point到当前函数的开始，并在函数末尾设置mark（(mark-defun)。 |
| C-x C-p | Move point to the beginning of the current page, and set mark at the end (mark-page).  移动point到当前page的开始，并在结尾设置mark（mark-page）。 |
| C-x h | Move point to the beginning of the buffer, and set mark at the end (mark-whole-buffer).  移动point到当前buffer的开始，并在结尾设置mark（mark-whole-buffer）。 |

M-@ (mark-word) sets the mark at the end of the next word (see Section 22.1 [Words], page 202, for information about words). Repeated invocations of this command extend the region by advancing the mark one word at a time. As an exception, if the mark is active and located before point, M-@ moves the mark backwards from its current position one word at a time.

M-@ (mark-word)在下一个单词的结尾设置mark（参见Section 22.1 [Words], page 202，了解关于word更多信息）。重复调用这个命令会将mark每次向后推进一个单词，进而扩展region。例外的是，如果mark是激活的并未与point之前，M-@每次都回将mark从当前位置开始向后移动一个单词。

This command also accepts a numeric argument n, which tells it to advance the mark by n words. A negative argument moves the mark back by n words.

该命令也会接受一个数字参数n，来指明每次将mark向前移动多少个单词。负数参数将会将mark向后移动n个单词。

Similarly, C-M-@ (mark-sexp) puts the mark at the end of the next balanced expression (see Section 23.4.1 [Expressions], page 240). Repeated invocations extend the region to subsequent expressions, while positive or negative numeric arguments move the mark forward or backward by the specified number of expressions.

类似的，C-M-@ (mark-sexp)将会把mark放在下一个平衡表达式（参见Section 23.4.1 [Expressions], page 240）的结尾。重复调用就会把region扩展到后续的表达式当中，如果伴随一个正数参数就像前推进，负数参数向后推进。

The other commands in the above list set both point and mark, so as to delimit an object in the buffer. M-h (mark-paragraph) marks paragraphs (see Section 22.3 [Paragraphs], page 204), C-M-h (mark-defun) marks top-level definitions (see Section 23.2.2 [Moving by Defuns], page 235), and C-x C-p (mark-page) marks pages (see Section 22.4 [Pages], page 205). Repeated invocations again play the same role, extending the region to consecutive objects; similarly, numeric arguments specify how many objects to move the mark by.

上面列表中其他命令会同时设置point和mark，从而划定缓冲区中的对象。M-h (mark-paragraph)标记段落（参见Section 22.3 [Paragraphs], page 204）。C-M-h (mark-defun)标记顶层的定义（参见Section 23.2.2 [Moving by Defuns], page 235），标记页（参见Section 22.4 [Pages], page 205）。再次重复调用会起到相同的作用，扩展region到后续的对象；同样，数字参数将会制定多个对象用来移动mark。

C-x h (mark-whole-buffer) sets up the entire buffer as the region, by putting point at the beginning and the mark at the end.

C-x h (mark-whole-buffer)通过将point放在开头，mark放在结尾来将整个buffer作为region。

## Operating on the Region（region上的操作）

Once you have a region, here are some of the ways you can operate on it:

有一个region之后，可以在上面进行如下操作：

* Kill it with C-w (see Chapter 9 [Killing], page 51).
* 剪切
* Copy it to the kill ring with M-w (see Section 9.2 [Yanking], page 53).
* 复制
* Convert case with C-x C-l or C-x C-u (see Section 22.6 [Case], page 210).
* 转换大小写
* Undo changes within it using C-u C-/ (see Section 13.1 [Undo], page 105).
* 撤销操作
* Replace text within it using M-% (see Section 12.10.4 [Query Replace], page 101).
* 替换
* Indent it with C-x TAB or C-M-\ (see Chapter 21 [Indentation], page 199).
* 缩进
* Fill it as text with M-x fill-region (see Section 22.5 [Filling], page 206).
* 用文本填充
* Check the spelling of words within it with M-$ (see Section 13.4 [Spelling], page 107).
* 改变单词的拼写。
* Evaluate it as Lisp code with M-x eval-region (see Section 24.9 [Lisp Eval], page 270).
* 计算lisp code的值。
* Save it in a register with C-x r s (see Chapter 10 [Registers], page 62).
* 在register中保存。
* Save it in a buffer or a file (see Section 9.4 [Accumulating Text], page 58).
* 在buffer或文件中保存

Some commands have a default behavior when the mark is inactive, but operate on the region if the mark is active. For example, M-$ (ispell-word) normally checks the spelling of the word at point, but it checks the text in the region if the mark is active (see Section 13.4 [Spelling], page 107). Normally, such commands use their default behavior if the region is empty (i.e., if mark and point are at the same position). If you want them to operate on the empty region, change the variable use-empty-active-region to t.

一些命令在mark失效时的默认行为会在mark激活时作用到region之上。例如，M-$ (ispell-word)通常是检查point处的单词拼写，但是如果mark激活，它会检查region中的文本（参见Section 13.4 [Spelling], page 107）。通常，如果region为空（例如，如果mark和point处于同一位置）的话，这样的命令会有各自默认的行为。如果想让它们在空region上操作，请将变量use-empty-active-region的值变为t。

As described in Section 4.3 [Erasing], page 19, the DEL (backward-delete-char) and delete (delete-forward-char) commands also act this way. If the mark is active, they delete the text in the region. (As an exception, if you supply a numeric argument n, where n is not one, these commands delete n characters regardless of whether the mark is active). If you change the variable delete-active-region to nil, then these commands don’t act differently when the mark is active. If you change the value to kill, these commands kill the region instead of deleting it (see Chapter 9 [Killing], page 51).

正如Section 4.3 [Erasing], page 19所描述的，DEL (backward-delete-char)和delete (delete-forward-char)命令也会有这样的行为。如果mark激活，他们会删除region中的文本。（例外的是，如果提供一个数字参数n，如果你不是1，这些命令不管mark是否被激活都会删除n个字符）。如果将变量delete-active-region设置为nil，当mark激活的时候，这些命令会采取同样的动作。如果将该值改变为kill，这些命令会剪切region而不是删除它（参见see Chapter 9 [Killing], page 51）。

Other commands always operate on the region, and have no default behavior. Such commands usually have the word region in their names, like C-w (kill-region) and C-x C-u (upcase-region). If the mark is inactive, they operate on the “inactive region”— that is, on the text between point and the position at which the mark was last set (see Section 8.4 [Mark Ring], page 48). To disable this behavior, change the variable mark-even-if-inactive to nil. Then these commands will instead signal an error if the mark is inactive.

另一些命令总是操作在region上，但 没有默认的行为。这些命令的名字中通常有region这个单词，比如C-w (kill-region) 和C-x C-u (upcase-region)。如果mark是失效的，它们会在“失效region”上操作，也就是point和上次mark位置之间的区域（参见see Section 8.4 [Mark Ring], page 48）。要禁用此行为，将变量mark-even-if-inactive变为nil。之后如果mark是失效的，这些命令就会发出错误信号。

By default, text insertion occurs normally even if the mark is active—for example, typing a inserts the character ‘a’, then deactivates the mark. If you enable Delete Selection mode, a minor mode, then inserting text while the mark is active causes the text in the region to be deleted first. To toggle Delete Selection mode on or off, type M-x delete-selection-mode.

默认情况下，即使mark是激活的，文本插入也会正常进行---例如，键入a就会插入字符a，然后使mark失效。如果启用了Delete Selection mode，一个minor mode，当mark激活的时候插入文本将会区域中的文本首先被删除。开启或关闭Delete Selection mode，键入M-x delete-selection-mode。

## The Mark Ring（标记环）

Each buffer remembers previous locations of the mark, in the mark ring. Commands that set the mark also push the old mark onto this ring. One of the uses of the mark ring is to remember spots that you may want to go back to.

每个buffer都会在标记环（mark ring）中记住mark之前的位置。设置mark的命令也会将就的mark放到该环中。Mark ring的一个用处就是记住你可能想要退回的点。

|  |  |
| --- | --- |
| C-SPC C-SPC | Set the mark, pushing it onto the mark ring, without activating it.  设置标记，把它放到标记环中，不激活它 |
| C-u C-SPC | Move point to where the mark was, and restore the mark from the ring of former marks.  将point移动到mark曾经所在位置，从标记环中加载先前的mark。 |

The command C-SPC C-SPC is handy when you want to use the mark to remember a position to which you may wish to return. It pushes the current point onto the mark ring, without activating the mark (which would cause Emacs to highlight the region). This is actually two consecutive invocations of C-SPC (set-mark-command); the first C-SPC sets the mark, and the second C-SPC deactivates it. (When Transient Mark mode is off, C-SPC CSPC instead activates Transient Mark mode temporarily; see Section 8.7 [Disabled Transient Mark], page 50.)

当想使用mark来记住一个位置，以便将来返回的时候，命令C-SPC C-SPC是方便的。它将当前的point放到mark ring当中，但是不会激活mark（激活mark将会导致Emacs高亮region）。这实际上是调用了C-SPC (set-mark-command)两次；第一个C-SPC 设置mark，第二个C-SPC 使mark失效。（当关闭Transient Mark mode时，C-SPC CSPC会反而会临时激活Transient Mark mode；参见Section 8.7 [Disabled Transient Mark], page 50）。

To return to a marked position, use set-mark-command with a prefix argument: C-u C-SPC. This moves point to where the mark was, and deactivates the mark if it was active. Each subsequent C-u C-SPC jumps to a prior position stored in the mark ring. The positions you move through in this way are not lost; they go to the end of the ring.

使用带有前缀参数C-u C-SPC的set-mark-command命令可以返回一个标记的位置。这会将point移动到mark的位置，并使mark失效。每个后续的C-u C-SPC都会跳到mark ring中事先保存的位置。通过这种方式移动的位置不会丢失；他们都会添加到ring的结尾。

If you set set-mark-command-repeat-pop to non-nil, then immediately after you type C-u C-SPC, you can type C-SPC instead of C-u C-SPC to cycle through the mark ring. By default, set-mark-command-repeat-pop is nil.

如果设置et-mark-command-repeat-pop为non-nil，然后立即键入C-u C-SPC，可以用C-SPC来代替C-u C-SPC来循环mark ring。默认情况下，set-mark-command-repeat-pop是nil。

Each buffer has its own mark ring. All editing commands use the current buffer’s mark ring. In particular, C-u C-SPC always stays in the same buffer.

每个buffer都有自己的mark ring。所有的编辑命令使用当前buffer的mark ring。特别是，C-u C-SPC总是呆在相同的buffer中。

The variable mark-ring-max specifies the maximum number of entries to keep in the mark ring. This defaults to 16 entries. If that many entries exist and another one is pushed, the earliest one in the list is discarded. Repeating C-u C-SPC cycles through the positions currently in the ring.

变量mark-ring-max指出了mark ring中所能保存的最大条目数。默认是16个条目。如果存在很多条目了，又进来一个条目，列表中最早的那个条目就会被丢弃。重复执行C-u C-SPC会循环经过ring中保存的位置。

If you want to move back to the same place over and over, the mark ring may not be convenient enough. If so, you can record the position in a register for later retrieval (see Section 10.1 [Saving Positions in Registers], page 62).

如果想不停的回道同一个位置，mark ring可能不太方便。这样的话，你可以将位置记录在寄存器（register）当中以后后面检索（参见Section 10.1 [Saving Positions in Registers], page 62）。

## The Global Mark Ring（全局标记环）

In addition to the ordinary mark ring that belongs to each buffer, Emacs has a single global mark ring. Each time you set a mark, this is recorded in the global mark ring in addition to the current buffer’s own mark ring, if you have switched buffers since the previous mark setting. Hence, the global mark ring records a sequence of buffers that you have been in, and, for each buffer, a place where you set the mark. The length of the global mark ring is controlled by global-mark-ring-max, and is 16 by default.

除了属于每个buffer的普通mark ring，Emacs还单独有一个全局的mark ring。没设置一个mark，除了会添加到当前buffer自己的mark ring中，还会在全局mark ring中做记录，如果你在上次mark设置后切换了buffer的话。因此，全局mark ring记录了所处的buffer的序列，以及每个buffer中设置mark的位置。全局mark ring的长度由global-mark-ring-max设置，默认是16。

The command C-x C-SPC (pop-global-mark) jumps to the buffer and position of the latest entry in the global ring. It also rotates the ring, so that successive uses of C-x C-SPC take you to earlier buffers and mark positions.

命令C-x C-SPC (pop-global-mark)跳转到全局ring中上一个记录的buffer的位置。该命令同样会遍历该环，连续的使用C-x C-SPC会带回到先前的buffer和mark位置。

## Shift Selection（移动选择）

If you hold down the shift key while typing a cursor motion command, this sets the mark before moving point, so that the region extends from the original position of point to its new position. This feature is referred to as shift-selection. It is similar to the way text is selected in other editors.

当你键入一个光标移动命令的时候按住shift键的话，就会在移动point之前设置mark，这样就会在原先point的位置和新的point之间形成一个region。这个功能叫做shift-selection。这和其他编辑器中选中文本的方式相同。

The mark set via shift-selection behaves a little differently from what we have described above. Firstly, in addition to the usual ways of deactivating the mark (such as changing the buffer text or typing C-g), the mark is deactivated by any unshifted cursor motion command. Secondly, any subsequent shifted cursor motion command avoids setting the mark anew. Therefore, a series of shifted cursor motion commands will continuously adjust the region.

通过shift-selection设置的mark与上面我们谈论的略有不同。首先，除了通常的停用mark的方式（比如改变buffer或者键入C-g），任何非移动的光标运动命令都会使mark失效。其次，任何后续的移动的光标运动命令都不会重新设置mark。因而，一系列的移动的光标动作命令将不断调整region。

Shift-selection only works if the shifted cursor motion key is not already bound to a separate command (see Chapter 33 [Customization], page 404). For example, if you bind S-C-f to another command, typing S-C-f runs that command instead of performing a shift-selected version of C-f (forward-char).

只有在移动的光标动作命令没有绑定到一个单独的命令时（参见Chapter 33 [Customization], page 404）Shift-selection才会生效。例如，将S-C-f当定到另外一个命令，按下S-C-f会运行那个命令而不是执行C-f (forward-char)的shift-selected版本。

A mark set via mouse commands behaves the same as a mark set via shift-selection (see Section 8.1 [Setting Mark], page 45). For example, if you specify a region by dragging the mouse, you can continue to extend the region using shifted cursor motion commands. In either case, any unshifted cursor motion command deactivates the mark.

通过鼠标命令设置的mark和通过shift-selection设置的mark行为有相同之处（参见Section 8.1 [Setting Mark], page 45）。例如，如果通过拖动鼠标制定了一个region，还可以通过移动的光标动作命令来继续扩展region。这两种情况下，任何非移动的光标动作命令都会停用mark。

To turn off shift-selection, set shift-select-mode to nil. Doing so does not disable setting the mark via mouse commands.

将shift-select-mode设置为nil可以关闭shift-selection。这样做不会禁用到通过鼠标命令设置光标。

## Disabling Transient Mark Mode（关闭Transient Mark Mode）

The default behavior of the mark and region, in which setting the mark activates it and highlights the region, is called Transient Mark mode. This is a minor mode that is enabled by default. It can be toggled with M-x transient-mark-mode, or with the ‘Active Region Highlighting’ menu item in the ‘Options’ menu. Turning it off switches Emacs to an alternative mode of operation:

激活mark和高亮region后，Mark和region的默认行为被称作Transient Mark mode。这是一个默认minor mode。可以通过M-x transient-mark-mode，或者option菜单中的Active Region Highlighting菜单项进行切换。将其关闭将会切换Emacs到备选的mode中。

* Setting the mark, with commands like C-SPC or C-x C-x, does not highlight the region. Therefore, you can’t tell by looking where the mark is located; you have to remember. The usual solution to this problem is to set the mark and then use it soon, before you forget where it is. You can also check where the mark is by using C-x C-x, which exchanges the positions of the point and the mark (see Section 8.1 [Setting Mark], page 45).
* 通过类似C-SPC or C-x C-x的命令设置mark不会高亮region。因而，mark的位置不能通过查找来找到，必须记录下来。这个问题常用的解决方法是设置标记并且在你忘记它在什么地方之前使用它。也可以通过C-x C-x检查mark的位置，这个命令将会交换point和mark（参见Section 8.1 [Setting Mark], page 45）。
* Some commands, which ordinarily act on the region when the mark is active, no longer do so. For example, normally M-% (query-replace) performs replacements within the region, if the mark is active. When Transient Mark mode is off, it always operates from point to the end of the buffer. Commands that act this way are identified in their own documentation.
* 某些当mark激活时作用在region上的命令不再这么做了。例如，通常M-% (query-replace) 如果标记激活的话，会在region内执行替换。如果关闭Transient Mark mode，它总是在point和buffer的末尾之间操作。这么做的命令和它们的文档描述一致。

While Transient Mark mode is off, you can activate it temporarily using C-SPC C-SPC or C-u C-x C-x.

当Transient Mark mode关闭时，你可以使用C-SPC C-SPC或C-u C-x C-x临时激活它。

|  |  |
| --- | --- |
| C-SPC C-SPC | Set the mark at point (like plain C-SPC) and enable Transient Mark mode just once, until the mark is deactivated. (This is not really a separate command;you are using the C-SPC command twice.) |
| C-u C-x C-x | Exchange point and mark, activate the mark and enable Transient Mark mode temporarily, until the mark is next deactivated. (This is the C-x C-x command, exchange-point-and-mark, with a prefix argument.) |

These commands set or activate the mark, and enable Transient Mark mode only until the mark is deactivated. One reason you may want to use them is that some commands operate on the entire buffer instead of the region when Transient Mark mode is off. Enabling Transient Mark mode momentarily gives you a way to use these commands on the region.

这些命令设置或者激活mark，并启用Transient Mark mode直到mark失效。你想使用它们的原因之一是当Transient Mark mode关闭的时候一些命令作用在整个buffer而不是region之上。临时启用Transient Mark mode使得你可以在region上使用这些命令。

When you specify a region with the mouse (see Section 8.1 [Setting Mark], page 45), or with shift-selection (see Section 8.6 [Shift Selection], page 49), this likewise activates Transient Mark mode temporarily and highlights the region.

当使用鼠标（参见Section 8.1 [Setting Mark], page 45）或shift-selection（参见Section 8.6 [Shift Selection], page 49）指定一个region，这同样临时启用Transient Mark mode并高亮region。

# ****Killing and Moving Text****

# ****Registers****

# ****Controlling the Display****

## Visual Line Mode

Another alternative to ordinary line continuation is to use word wrap. Here, each long logical line is divided into two or more screen lines, like in ordinary line continuation. However, Emacs attempts to wrap the line at word boundaries near the right window edge. This makes the text easier to read, as wrapping does not occur in the middle of words.

Word wrap is enabled by Visual Line mode, an optional minor mode. To turn on Visual Line mode in the current buffer, type M-x visual-line-mode; repeating this command turns it off. You can also turn on Visual Line mode using the menu bar: in the Options menu, select the ‘Line Wrapping in this Buffer’ submenu, followed by the ‘Word Wrap (Visual Line Mode)’ menu item. While Visual Line mode is enabled, the mode-line shows the string ‘wrap’ in the mode display. The command M-x global-visual-line-mode toggles Visual Line mode in all buffers.

In Visual Line mode, some editing commands work on screen lines instead of logical lines: C-a (beginning-of-visual-line) moves to the beginning of the screen line, C-e (end-of-visual-line) moves to the end of the screen line, and C-k (kill-visual-line) kills text to the end of the screen line.

To move by logical lines, use the commands M-x next-logical-line and M-x previous-logical-line. These move point to the next logical line and the previous logical line respectively, regardless of whether Visual Line mode is enabled. If you use these commands frequently, it may be convenient to assign key bindings to them. See Section 33.3.6 [Init Rebinding], page 424.

By default, word-wrapped lines do not display fringe indicators. Visual Line mode is often used to edit files that contain many long logical lines, so having a fringe indicator for each wrapped line would be visually distracting. You can change this by customizing the variable visual-line-fringe-indicators.

# ****Searching and Replacement****

# ****Commands for Fixing Typos****

# ****Keyboard Macros****

# ****File Handling****

# ****Using Multiple Buffers****

# ****Multiple Windows****

# ****Frames and Graphical Displays****

## Mouse Avoidance

On graphical terminals, the mouse pointer may obscure the text in the Emacs frame. Emacs provides two methods to avoid this problem.

Firstly, Emacs hides the mouse pointer each time you type a self-inserting character, if the pointer lies inside an Emacs frame; moving the mouse pointer makes it visible again. To disable this feature, set the variable make-pointer-invisible to nil.

Secondly, you can use Mouse Avoidance mode, a minor mode, to keep the mouse pointer away from point. To use Mouse Avoidance mode, customize the variable mouse-avoidance-mode. You can set this to various values to move the mouse in several ways:

|  |  |
| --- | --- |
| banish | Move the pointer to a corner of the frame on any key-press. You can customize the variable mouse-avoidance-banish-position to specify where the pointer goes when it is banished. |
| exile | Banish the pointer only if the cursor gets too close, and allow it to return once the cursor is out of the way. |
| jump | If the cursor gets too close to the pointer, displace the pointer by a random distance and direction. |
| animate | As jump, but shows steps along the way for illusion of motion. |
| cat-and-mouse | The same as animate. |
| proteus | As animate, but changes the shape of the mouse pointer too. |

You can also use the command M-x mouse-avoidance-mode to enable the mode. Whenever Mouse Avoidance mode moves the mouse, it also raises the frame.

# ****International Character Set Support**** （国际字符集支持）

Emacs supports a wide variety of international character sets, including European and Vietnamese variants of the Latin alphabet, as well as Cyrillic, Devanagari (for Hindi and Marathi), Ethiopic, Greek, Han (for Chinese and Japanese), Hangul (for Korean), Hebrew, IPA, Kannada, Lao, Malayalam, Tamil, Thai, Tibetan, and Vietnamese scripts. Emacs also supports various encodings of these characters that are used by other internationalized software, such as word processors and mailers.

Emacs allows editing text with international characters by supporting all the related activities:

You can visit files with non-ASCII characters, save non-ASCII text, and pass non-ASCII text between Emacs and programs it invokes (such as compilers, spell-checkers, and mailers). Setting your language environment (see Section 19.3 [Language Environments], page 174) takes care of setting up the coding systems and other options for a specific language or culture. Alternatively, you can specify how Emacs should encode or decode text for each command; see Section 19.10 [Text Coding], page 183.

You can display non-ASCII characters encoded by the various scripts. This works by using appropriate fonts on graphics displays (see Section 19.15 [Defining Fontsets], page 187), and by sending special codes to text displays (see Section 19.13 [Terminal Coding], page 185). If some characters are displayed incorrectly, refer to Section 19.17 [Undisplayable Characters], page 189, which describes possible problems and explains how to solve them.

Characters from scripts whose natural ordering of text is from right to left are reordered for display (see Section 19.20 [Bidirectional Editing], page 191). These scripts include Arabic, Hebrew, Syriac, Thaana, and a few others.

You can insert non-ASCII characters or search for them. To do that, you can specify an input method (see Section 19.5 [Select Input Method], page 177) suitable for your language, or use the default input method set up when you chose your language environment. If your keyboard can produce non-ASCII characters, you can select an appropriate keyboard coding system (see Section 19.13 [Terminal Coding], page 185), and Emacs will accept those characters. Latin-1 characters can also be input by using the C-x 8 prefix, see Section 19.18 [Unibyte Mode], page 190.

With the X Window System, your locale should be set to an appropriate value to make sure Emacs interprets keyboard input correctly; see Section 19.3 [Language Environments], page 174.

The rest of this chapter describes these issues in detail.

## Introduction to International Character Sets

The users of international character sets and scripts have established many more-or-less standard coding systems for storing files. These coding systems are typically multibyte, meaning that sequences of two or more bytes are used to represent individual non-ASCII characters.

Internally, Emacs uses its own multibyte character encoding, which is a superset of the Unicode standard. This internal encoding allows characters from almost every known script to be intermixed in a single buffer or string. Emacs translates between the multibyte character encoding and various other coding systems when reading and writing files, and when exchanging data with subprocesses.

The command C-h h (view-hello-file) displays the file etc/HELLO, which illustrates various scripts by showing how to say “hello” in many languages. If some characters can’t be displayed on your terminal, they appear as ‘?’ or as hollow boxes (see Section 19.17 [Undisplayable Characters], page 189).

Keyboards, even in the countries where these character sets are used, generally don’t have keys for all the characters in them. You can insert characters that your keyboard does not support, using C-q (quoted-insert) or C-x 8 RET (insert-char). See Section 4.1 [Inserting Text], page 16. Emacs also supports various input methods, typically one for each script or language, which make it easier to type characters in the script. See Section 19.4 [Input Methods], page 175.

The prefix key C-x RET is used for commands that pertain to multibyte characters, coding systems, and input methods.

The command C-x = (what-cursor-position) shows information about the character

at point. In addition to the character position, which was described in Section 4.9 [Position Info], page 22, this command displays how the character is encoded. For instance, it displays the following line in the echo area for the character ‘c’:

Char: c (99, #o143, #x63) point=28062 of 36168 (78%) column=53

The four values after ‘Char:’ describe the character that follows point, first by showing it and then by giving its character code in decimal, octal and hex. For a non-ASCII multibyte character, these are followed by ‘file’ and the character’s representation, in hex, in the buffer’s coding system, if that coding system encodes the character safely and with a single byte (see Section 19.6 [Coding Systems], page 178). If the character’s encoding is longer than one byte, Emacs shows ‘file ...’.

As a special case, if the character lies in the range 128 (0200 octal) through 159 (0237 octal), it stands for a “raw” byte that does not correspond to any specific displayable character. Such a “character” lies within the eight-bit-control character set, and is displayed as an escaped octal character code. In this case, C-x = shows ‘part of display ...’ instead of ‘file’.

With a prefix argument (C-u C-x =), this command displays a detailed description of the character in a window:

The character set name, and the codes that identify the character within that character set; ASCII characters are identified as belonging to the ascii character set.

The character’s syntax and categories.

The character’s encodings, both internally in the buffer, and externally if you were to save the file.

What keys to type to input the character in the current input method (if it supports the character).

If you are running Emacs on a graphical display, the font name and glyph code for the character. If you are running Emacs on a text terminal, the code(s) sent to the terminal.

The character’s text properties (see Section “Text Properties” in the Emacs Lisp Reference Manual), including any non-default faces used to display the character, and any overlays containing it (see Section “Overlays” in the same manual ).

Here’s an example showing the Latin-1 character A with grave accent, in a buffer whose coding system is utf-8-unix:

position: 1 of 1 (0%), column: 0

character:`A (displayed as ` A) (codepoint 192, #o300, #xc0)

preferred charset: unicode (Unicode (ISO10646))

code point in charset: 0xC0

syntax: w which means: word

category: .:Base, L:Left-to-right (strong),

j:Japanese, l:Latin, v:Viet

buffer code: #xC3 #x80

file code: not encodable by coding system undecided-unix

display: by this font (glyph code)

xft:-unknown-DejaVu Sans Mono-normal-normalnormal-\*-13-\*-\*-\*-m-0-iso10646-1 (#x82)

Character code properties: customize what to show

name: LATIN CAPITAL LETTER A WITH GRAVE

old-name: LATIN CAPITAL LETTER A GRAVE

general-category: Lu (Letter, Uppercase)

decomposition: (65 768) (’A’ ’‘’)

## Disabling Multibyte Characters

By default, Emacs starts in multibyte mode: it stores the contents of buffers and strings using an internal encoding that represents non-ASCII characters using multi-byte sequences. Multibyte mode allows you to use all the supported languages and scripts without limitations.

Under very special circumstances, you may want to disable multibyte character support, for a specific buffer. When multibyte characters are disabled in a buffer, we call that unibyte mode. In unibyte mode, each character in the buffer has a character code ranging from 0 through 255 (0377 octal); 0 through 127 (0177 octal) represent ASCII characters, and 128 (0200 octal) through 255 (0377 octal) represent non-ASCII characters.

To edit a particular file in unibyte representation, visit it using find-file-literally. See Section 15.2 [Visiting], page 119. You can convert a multibyte buffer to unibyte by saving it to a file, killing the buffer, and visiting the file again with find-file-literally. Alternatively, you can use C-x RET c (universal-coding-system-argument) and specify ‘raw-text’ as the coding system with which to visit or save a file. See Section 19.10 [Text Coding], page 183. Unlike find-file-literally, finding a file as ‘raw-text’ doesn’t disable format conversion, uncompression, or auto mode selection.

Emacs normally loads Lisp files as multibyte. This includes the Emacs initialization file, .emacs, and the initialization files of packages such as Gnus. However, you can specify unibyte loading for a particular Lisp file, by adding an entry ‘coding: raw-text’ in a file local variables section. See Section 19.8 [Specify Coding], page 182. Then that file is always loaded as unibyte text. You can also load a Lisp file as unibyte, on any one occasion, by typing C-x RET c raw-text RET immediately before loading it.

The buffer-local variable enable-multibyte-characters is non-nil in multibyte buffers, and nil in unibyte ones. The mode line also indicates whether a buffer is multibyte Chapter 19: International Character Set Support 174 or not. See Section 1.3 [Mode Line], page 8. With a graphical display, in a multibyte buffer, the portion of the mode line that indicates the character set has a tooltip that (amongst other things) says that the buffer is multibyte. In a unibyte buffer, the character set indicator is absent. Thus, in a unibyte buffer (when using a graphical display) there is normally nothing before the indication of the visited file’s end-of-line convention (colon, backslash, etc.), unless you are using an input method.

You can turn off multibyte support in a specific buffer by invoking the command toggleenable-multibyte-characters in that buffer.

## Language Environments

All supported character sets are supported in Emacs buffers whenever multibyte characters are enabled; there is no need to select a particular language in order to display its characters. However, it is important to select a language environment in order to set various defaults. Roughly speaking, the language environment represents a choice of preferred script rather than a choice of language.

The language environment controls which coding systems to recognize when reading text (see Section 19.7 [Recognize Coding], page 180). This applies to files, incoming mail, and any other text you read into Emacs. It may also specify the default coding system to use when you create a file. Each language environment also specifies a default input method.

To select a language environment, customize current-language-environment or use the command M-x set-language-environment. It makes no difference which buffer is current when you use this command, because the effects apply globally to the Emacs session. The supported language environments (see the variable language-info-alist) include:

ASCII, Belarusian, Bengali, Brazilian Portuguese, Bulgarian, Cham, ChineseBIG5, Chinese-CNS, Chinese-EUC-TW, Chinese-GB, Chinese-GBK, ChineseGB18030, Croatian, Cyrillic-ALT, Cyrillic-ISO, Cyrillic-KOI8, Czech, Devanagari, Dutch, English, Esperanto, Ethiopic, French, Georgian, German, Greek, Gujarati, Hebrew, IPA, Italian, Japanese, Kannada, Khmer, Korean, Lao, Latin-1, Latin-2, Latin-3, Latin-4, Latin-5, Latin-6, Latin-7, Latin-8 (Celtic), Latin-9 (updated Latin-1 with the Euro sign), Latvian, Lithuanian, Malayalam, Oriya, Polish, Punjabi, Romanian, Russian, Sinhala, Slovak, Slovenian, Spanish, Swedish, TaiViet, Tajik, Tamil, Telugu, Thai, Tibetan, Turkish, UTF-8 (for a setup which prefers Unicode characters and files encoded in UTF-8), Ukrainian, Vietnamese, Welsh, and Windows-1255 (for a setup which prefers Cyrillic characters and files encoded in Windows-1255).

To display the script(s) used by your language environment on a graphical display, you need to have suitable fonts. See Section 19.14 [Fontsets], page 186, for more details about setting up your fonts.

Some operating systems let you specify the character-set locale you are using by setting the locale environment variables LC\_ALL, LC\_CTYPE, or LANG. (If more than one of these is set, the first one that is nonempty specifies your locale for this purpose.) During startup, Emacs looks up your character-set locale’s name in the system locale alias table, matches its canonical name against entries in the value of the variables locale-charsetlanguage-names and locale-language-names (the former overrides the latter), and selects the corresponding language environment if a match is found. It also adjusts the display table and terminal coding system, the locale coding system, the preferred coding system as needed for the locale, and—last but not least—the way Emacs decodes non-ASCII characters sent by your keyboard.

If you modify the LC\_ALL, LC\_CTYPE, or LANG environment variables while running Emacs (by using M-x setenv), you may want to invoke the set-locale-environment function afterwards to readjust the language environment from the new locale.

The set-locale-environment function normally uses the preferred coding system established by the language environment to decode system messages. But if your locale matches an entry in the variable locale-preferred-coding-systems, Emacs uses the corresponding coding system instead. For example, if the locale ‘ja\_JP.PCK’ matches japanese-shift-jis in locale-preferred-coding-systems, Emacs uses that encoding even though it might normally use japanese-iso-8bit.

You can override the language environment chosen at startup with explicit use of the command set-language-environment, or with customization of current-languageenvironment in your init file.

To display information about the effects of a certain language environment lang-env, use the command C-h L lang-env RET (describe-language-environment). This tells you which languages this language environment is useful for, and lists the character sets, coding systems, and input methods that go with it. It also shows some sample text to illustrate scripts used in this language environment. If you give an empty input for lang-env, this command describes the chosen language environment.

You can customize any language environment with the normal hook set-language-environment-hook. The command set-language-environment runs that hook after setting up the new language environment. The hook functions can test for a specific language environment by checking the variable current-language-environment. This hook is where you should put non-default settings for specific language environments, such as coding systems for keyboard input and terminal output, the default input method, etc.

Before it starts to set up the new language environment, set-language-environment first runs the hook exit-language-environment-hook. This hook is useful for undoing customizations that were made with set-language-environment-hook. For instance, if you set up a special key binding in a specific language environment using set-languageenvironment-hook, you should set up exit-language-environment-hook to restore the normal binding for that key.

## Input Methods

An input method is a kind of character conversion designed specifically for interactive input. In Emacs, typically each language has its own input method; sometimes several languages that use the same characters can share one input method. A few languages support several input methods.

The simplest kind of input method works by mapping ASCII letters into another alphabet; this allows you to use one other alphabet instead of ASCII. The Greek and Russian input methods work this way.

A more powerful technique is composition: converting sequences of characters into one letter. Many European input methods use composition to produce a single non- ASCII letter from a sequence that consists of a letter followed by accent characters (or vice versa). For example, some methods convert the sequence o ^ into a single accented letter. These input methods have no special commands of their own; all they do is compose sequences of printing characters.

The input methods for syllabic scripts typically use mapping followed by composition. The input methods for Thai and Korean work this way. First, letters are mapped into symbols for particular sounds or tone marks; then, sequences of these that make up a whole syllable are mapped into one syllable sign.

Chinese and Japanese require more complex methods. In Chinese input methods, first you enter the phonetic spelling of a Chinese word (in input method chinese-py, among others), or a sequence of portions of the character (input methods chinese-4corner and chinese-sw, and others). One input sequence typically corresponds to many possible Chinese characters. You select the one you mean using keys such as C-f, C-b, C-n, C-p (or the arrow keys), and digits, which have special meanings in this situation.

The possible characters are conceptually arranged in several rows, with each row holding up to 10 alternatives. Normally, Emacs displays just one row at a time, in the echo area; (i/j) appears at the beginning, to indicate that this is the i th row out of a total of j rows.

Type C-n or C-p to display the next row or the previous row.

Type C-f and C-b to move forward and backward among the alternatives in the current row. As you do this, Emacs highlights the current alternative with a special color; type C-SPC to select the current alternative and use it as input. The alternatives in the row are also numbered; the number appears before the alternative. Typing a number selects the associated alternative of the current row and uses it as input.

TAB in these Chinese input methods displays a buffer showing all the possible characters at once; then clicking Mouse-2 on one of them selects that alternative. The keys C-f, C-b, C-n, C-p, and digits continue to work as usual, but they do the highlighting in the buffer showing the possible characters, rather than in the echo area.

In Japanese input methods, first you input a whole word using phonetic spelling; then, after the word is in the buffer, Emacs converts it into one or more characters using a large dictionary. One phonetic spelling corresponds to a number of different Japanese words; to select one of them, use C-n and C-p to cycle through the alternatives.

Sometimes it is useful to cut off input method processing so that the characters you have just entered will not combine with subsequent characters. For example, in input method latin-1-postfix, the sequence o ^ combines to form an ‘o’ with an accent. What if you want to enter them as separate characters?

One way is to type the accent twice; this is a special feature for entering the separate letter and accent. For example, o ^ ^ gives you the two characters ‘o^’. Another way is to

type another letter after the o—something that won’t combine with that—and immediately delete it. For example, you could type o o DEL ^ to get separate ‘o’ and ‘^’. Another method, more general but not quite as easy to type, is to use C-\ C-\ between two characters to stop them from combining. This is the command C-\ (toggle-inputmethod) used twice.

C-\ C-\ is especially useful inside an incremental search, because it stops waiting for more characters to combine, and starts searching for what you have already entered.

To find out how to input the character after point using the current input method, type C-u C-x =. See Section 4.9 [Position Info], page 22.

The variables input-method-highlight-flag and input-method-verbose-flag control how input methods explain what is happening. If input-method-highlight-flag is non-nil, the partial sequence is highlighted in the buffer (for most input methods—some disable this feature). If input-method-verbose-flag is non-nil, the list of possible characters to type next is displayed in the echo area (but not when you are in the minibuffer).

Another facility for typing characters not on your keyboard is by using C-x 8 RET (insert-char) to insert a single character based on its Unicode name or code-point; see Section 4.1 [Inserting Text], page 16.

## Selecting an Input Method

|  |  |
| --- | --- |
| C-\ | Enable or disable use of the selected input method (toggle-input-method). |
| C-x RET C-\ method RET | Select a new input method for the current buffer (set-input-method) |
| C-h I method RET  C-h C-\ method RET | Describe the input method method (describe-input-method). By default, it describes the current input method (if any). This description should give you the full details of how to use any particular input method. |
| M-x list-input-methods | Display a list of all the supported input methods. |

To choose an input method for the current buffer, use C-x RET C-\ (set-input-method). This command reads the input method name from the minibuffer; the name normally starts with the language environment that it is meant to be used with. The variable currentinput-method records which input method is selected.

Input methods use various sequences of ASCII characters to stand for non-ASCII characters. Sometimes it is useful to turn off the input method temporarily. To do this, type C-\ (toggle-input-method). To reenable the input method, type C-\ again.

If you type C-\ and you have not yet selected an input method, it prompts you to specify one. This has the same effect as using C-x RET C-\ to specify an input method.

When invoked with a numeric argument, as in C-u C-\, toggle-input-method always prompts you for an input method, suggesting the most recently selected one as the default.

Selecting a language environment specifies a default input method for use in various buffers. When you have a default input method, you can select it in the current buffer by typing C-\. The variable default-input-method specifies the default input method (nil means there is none).

In some language environments, which support several different input methods, you might want to use an input method different from the default chosen by set-languageenvironment. You can instruct Emacs to select a different default input method for a certain language environment, if you wish, by using set-language-environment-hook (see Section 19.3 [Language Environments], page 174). For example:

(defun my-chinese-setup ()

"Set up my private Chinese environment."

(if (equal current-language-environment "Chinese-GB")

(setq default-input-method "chinese-tonepy")))

(add-hook ’set-language-environment-hook ’my-chinese-setup)

This sets the default input method to be chinese-tonepy whenever you choose a ChineseGB language environment. You can instruct Emacs to activate a certain input method automatically. For example:

(add-hook ’text-mode-hook

(lambda () (set-input-method "german-prefix")))

This automatically activates the input method “german-prefix” in Text mode.

Some input methods for alphabetic scripts work by (in effect) remapping the keyboard to emulate various keyboard layouts commonly used for those scripts. How to do this remapping properly depends on your actual keyboard layout. To specify which layout your keyboard has, use the command M-x quail-set-keyboard-layout.

You can use the command M-x quail-show-key to show what key (or key sequence) to type in order to input the character following point, using the selected keyboard layout. The command C-u C-x = also shows that information, in addition to other information about the character.

M-x list-input-methods displays a list of all the supported input methods. The list gives information about each input method, including the string that stands for it in the mode line.

## Coding Systems

Users of various languages have established many more-or-less standard coding systems for representing them. Emacs does not use these coding systems internally; instead, it converts from various coding systems to its own system when reading data, and converts the internal coding system to other coding systems when writing data. Conversion is possible in reading or writing files, in sending or receiving from the terminal, and in exchanging data with subprocesses.

Emacs assigns a name to each coding system. Most coding systems are used for one language, and the name of the coding system starts with the language name. Some coding systems are used for several languages; their names usually start with ‘iso’. There are also special coding systems, such as no-conversion, raw-text, and emacs-internal.

A special class of coding systems, collectively known as codepages, is designed to support text encoded by MS-Windows and MS-DOS software. The names of these coding systems are cpnnnn, where nnnn is a 3- or 4-digit number of the codepage. You can use these encodings just like any other coding system; for example, to visit a file encoded in codepage 850, type C-x RET c cp850 RET C-x C-f filename RET.

In addition to converting various representations of non-ASCII characters, a coding system can perform end-of-line conversion. Emacs handles three different conventions for how to separate lines in a file: newline (“unix”), carriage-return linefeed (“dos”), and just carriage-return (“mac”).

|  |  |
| --- | --- |
| C-h C coding RET | Describe coding system coding (describe-coding-system). |
| C-h C RET | Describe the coding systems currently in use. |
| M-x list-coding-systems | Display a list of all the supported coding systems. |

The command C-h C (describe-coding-system) displays information about particular coding systems, including the end-of-line conversion specified by those coding systems. You can specify a coding system name as the argument; alternatively, with an empty argument, it describes the coding systems currently selected for various purposes, both in the current buffer and as the defaults, and the priority list for recognizing coding systems (see Section 19.7 [Recognize Coding], page 180).

To display a list of all the supported coding systems, type M-x list-coding-systems. The list gives information about each coding system, including the letter that stands for it in the mode line (see Section 1.3 [Mode Line], page 8).

Each of the coding systems that appear in this list—except for no-conversion, which means no conversion of any kind—specifies how and whether to convert printing characters, but leaves the choice of end-of-line conversion to be decided based on the contents of each file. For example, if the file appears to use the sequence carriage-return linefeed to separate lines, DOS end-of-line conversion will be used.

Each of the listed coding systems has three variants, which specify exactly what to do for end-of-line conversion:

|  |  |
| --- | --- |
| ...-unix | Don’t do any end-of-line conversion; assume the file uses newline to separate lines. (This is the convention normally used on Unix and GNU systems, and Mac OS X.) |
| ...-dos | Assume the file uses carriage-return linefeed to separate lines, and do the appropriate conversion. (This is the convention normally used on Microsoft systems.1) |
| ...-mac | Assume the file uses carriage-return to separate lines, and do the appropriate conversion. (This was the convention used on the Macintosh system prior to OS X.) |

1 It is also specified for MIME ‘text/\*’ bodies and in other network transport contexts. It is different from the SGML reference syntax record-start/record-end format, which Emacs doesn’t support directly.

These variant coding systems are omitted from the list-coding-systems display for brevity, since they are entirely predictable. For example, the coding system iso-latin-1 has variants iso-latin-1-unix, iso-latin-1-dos and iso-latin-1-mac.

The coding systems unix, dos, and mac are aliases for undecided-unix, undecideddos, and undecided-mac, respectively. These coding systems specify only the end-of-line conversion, and leave the character code conversion to be deduced from the text itself.

The coding system raw-text is good for a file which is mainly ASCII text, but may contain byte values above 127 that are not meant to encode non-ASCII characters. With rawtext, Emacs copies those byte values unchanged, and sets enable-multibyte-characters to nil in the current buffer so that they will be interpreted properly. raw-text handles end-of-line conversion in the usual way, based on the data encountered, and has the usual three variants to specify the kind of end-of-line conversion to use.

In contrast, the coding system no-conversion specifies no character code conversion at all—none for non-ASCII byte values and none for end of line. This is useful for reading or writing binary files, tar files, and other files that must be examined verbatim. It, too, sets enable-multibyte-characters to nil.

The easiest way to edit a file with no conversion of any kind is with the M-x find-file-literally command. This uses no-conversion, and also suppresses other Emacs features that might convert the file contents before you see them. See Section 15.2 [Visiting], page 119.

The coding system emacs-internal (or utf-8-emacs, which is equivalent) means that the file contains non-ASCII characters stored with the internal Emacs encoding. This coding system handles end-of-line conversion based on the data encountered, and has the usual three variants to specify the kind of end-of-line conversion.

## Recognizing Coding Systems

Whenever Emacs reads a given piece of text, it tries to recognize which coding system to use. This applies to files being read, output from subprocesses, text from X selections, etc. Emacs can select the right coding system automatically most of the time—once you have specified your preferences.

Some coding systems can be recognized or distinguished by which byte sequences appear in the data. However, there are coding systems that cannot be distinguished, not even potentially. For example, there is no way to distinguish between Latin-1 and Latin-2; they use the same byte values with different meanings.

Emacs handles this situation by means of a priority list of coding systems. Whenever Emacs reads a file, if you do not specify the coding system to use, Emacs checks the data against each coding system, starting with the first in priority and working down the list, until it finds a coding system that fits the data. Then it converts the file contents assuming that they are represented in this coding system.

The priority list of coding systems depends on the selected language environment (see Section 19.3 [Language Environments], page 174). For example, if you use French, you probably want Emacs to prefer Latin-1 to Latin-2; if you use Czech, you probably want Latin-2 to be preferred. This is one of the reasons to specify a language environment.

However, you can alter the coding system priority list in detail with the command M-x prefer-coding-system. This command reads the name of a coding system from the minibuffer, and adds it to the front of the priority list, so that it is preferred to all others. If you use this command several times, each use adds one element to the front of the priority list.

If you use a coding system that specifies the end-of-line conversion type, such as iso-8859-1-dos, what this means is that Emacs should attempt to recognize iso-8859-1 with priority, and should use DOS end-of-line conversion when it does recognize iso-8859-1.

Sometimes a file name indicates which coding system to use for the file. The variable file-coding-system-alist specifies this correspondence. There is a special function modify-coding-system-alist for adding elements to this list. For example, to read and write all ‘.txt’ files using the coding system chinese-iso-8bit, you can execute this Lisp expression:

(modify-coding-system-alist ’file "\\.txt\\’" ’chinese-iso-8bit)

The first argument should be file, the second argument should be a regular expression that determines which files this applies to, and the third argument says which coding system to use for these files.

Emacs recognizes which kind of end-of-line conversion to use based on the contents of the file: if it sees only carriage-returns, or only carriage-return linefeed sequences, then it chooses the end-of-line conversion accordingly. You can inhibit the automatic use of endof-line conversion by setting the variable inhibit-eol-conversion to non-nil. If you do that, DOS-style files will be displayed with the ‘^M’ characters visible in the buffer; some people prefer this to the more subtle ‘(DOS)’ end-of-line type indication near the left edge of the mode line (see Section 1.3 [Mode Line], page 8).

By default, the automatic detection of coding system is sensitive to escape sequences. If Emacs sees a sequence of characters that begin with an escape character, and the sequence is valid as an ISO-2022 code, that tells Emacs to use one of the ISO-2022 encodings to decode the file.

However, there may be cases that you want to read escape sequences in a file as is. In such a case, you can set the variable inhibit-iso-escape-detection to non-nil. Then the code detection ignores any escape sequences, and never uses an ISO-2022 encoding. The result is that all escape sequences become visible in the buffer.

The default value of inhibit-iso-escape-detection is nil. We recommend that you not change it permanently, only for one specific operation. That’s because some Emacs Lisp source files in the Emacs distribution contain non-ASCII characters encoded in the coding system iso-2022-7bit, and they won’t be decoded correctly when you visit those files if you suppress the escape sequence detection.

The variables auto-coding-alist and auto-coding-regexp-alist are the strongest way to specify the coding system for certain patterns of file names, or for files containing certain patterns, respectively. These variables even override ‘-\*-coding:-\*-’ tags in the file itself (see Section 19.8 [Specify Coding], page 182). For example, Emacs uses auto-codingalist for tar and archive files, to prevent it from being confused by a ‘-\*-coding:-\*-’ tag in a member of the archive and thinking it applies to the archive file as a whole.

Another way to specify a coding system is with the variable auto-coding-functions. For example, one of the builtin auto-coding-functions detects the encoding for XML files. Unlike the previous two, this variable does not override any ‘-\*-coding:-\*-’ tag.

When you get new mail in Rmail, each message is translated automatically from the coding system it is written in, as if it were a separate file. This uses the priority list of coding systems that you have specified. If a MIME message specifies a character set, Rmail obeys that specification. For reading and saving Rmail files themselves, Emacs uses the coding system specified by the variable rmail-file-coding-system. The default value is nil, which means that Rmail files are not translated (they are read and written in the Emacs internal character code).

## Specifying a File’s Coding System

If Emacs recognizes the encoding of a file incorrectly, you can reread the file using the correct coding system with C-x RET r (revert-buffer-with-coding-system). This command prompts for the coding system to use. To see what coding system Emacs actually used to decode the file, look at the coding system mnemonic letter near the left edge of the mode line (see Section 1.3 [Mode Line], page 8), or type C-h C (describe-coding-system).

You can specify the coding system for a particular file in the file itself, using the ‘-\*-...-\*-’ construct at the beginning, or a local variables list at the end (see Section 33.2.4 [File Variables], page 416). You do this by defining a value for the “variable” named coding. Emacs does not really have a variable coding; instead of setting a variable, this uses the specified coding system for the file. For example, ‘-\*-mode: C; coding: latin-1;-\*-’ specifies use of the Latin-1 coding system, as well as C mode. When you specify the coding explicitly in the file, that overrides file-coding-system-alist.

## Choosing Coding Systems for Output

Once Emacs has chosen a coding system for a buffer, it stores that coding system in bufferfile-coding-system. That makes it the default for operations that write from this buffer into a file, such as save-buffer and write-region. You can specify a different coding system for further file output from the buffer using set-buffer-file-coding-system (see Section 19.10 [Text Coding], page 183).

You can insert any character Emacs supports into any Emacs buffer, but most coding systems can only handle a subset of these characters. Therefore, it’s possible that the characters you insert cannot be encoded with the coding system that will be used to save the buffer. For example, you could visit a text file in Polish, encoded in iso-8859-2, and add some Russian words to it. When you save that buffer, Emacs cannot use the current value of buffer-file-coding-system, because the characters you added cannot be encoded by that coding system.

When that happens, Emacs tries the most-preferred coding system (set by M-x prefer-coding-system or M-x set-language-environment). If that coding system can safely encode all of the characters in the buffer, Emacs uses it, and stores its value in buffer-file-coding-system. Otherwise, Emacs displays a list of coding systems suitable for encoding the buffer’s contents, and asks you to choose one of those coding systems.

If you insert the unsuitable characters in a mail message, Emacs behaves a bit differently. It additionally checks whether the most-preferred coding system is recommended for use in MIME messages; if not, it informs you of this fact and prompts you for another coding system. This is so you won’t inadvertently send a message encoded in a way that your recipient’s mail software will have difficulty decoding. (You can still use an unsuitable coding system if you enter its name at the prompt.)

When you send a mail message (see Chapter 29 [Sending Mail], page 343), Emacs has four different ways to determine the coding system to use for encoding the message text. It tries the buffer’s own value of buffer-file-coding-system, if that is non-nil. Otherwise, it uses the value of sendmail-coding-system, if that is non-nil. The third way is to use the default coding system for new files, which is controlled by your choice of language environment, if that is non-nil. If all of these three values are nil, Emacs encodes outgoing mail using the Latin-1 coding system.

## Specifying a Coding System for File Text

In cases where Emacs does not automatically choose the right coding system for a file’s contents, you can use these commands to specify one:

|  |  |
| --- | --- |
| C-x RET f coding RET | Use coding system coding to save or revisit the file in the current buffer (setbuffer-file-coding-system). |
| C-x RET c coding RET | Specify coding system coding for the immediately following command (universal-coding-system-argument). |
| C-x RET r coding RET | Revisit the current file using the coding system coding (revert-buffer-withcoding-system). |
| M-x recode-region RET right RET wrong RET | Convert a region that was decoded using coding system wrong, decoding it using coding system right instead. |

The command C-x RET f (set-buffer-file-coding-system) sets the file coding system for the current buffer (i.e., the coding system to use when saving or reverting the file). You specify which coding system using the minibuffer. You can also invoke this command by clicking with Mouse-3 on the coding system indicator in the mode line (see Section 1.3 [Mode Line], page 8).

If you specify a coding system that cannot handle all the characters in the buffer, Emacs will warn you about the troublesome characters, and ask you to choose another coding system, when you try to save the buffer (see Section 19.9 [Output Coding], page 182).

You can also use this command to specify the end-of-line conversion (see Section 19.6 [Coding Systems], page 178) for encoding the current buffer. For example, C-x RET f dos RET will cause Emacs to save the current buffer’s text with DOS-style carriage-return linefeed line endings.

Another way to specify the coding system for a file is when you visit the file. First use the command C-x RET c (universal-coding-system-argument); this command uses the minibuffer to read a coding system name. After you exit the minibuffer, the specified coding system is used for the immediately following command.

So if the immediately following command is C-x C-f, for example, it reads the file using that coding system (and records the coding system for when you later save the file). Or if the immediately following command is C-x C-w, it writes the file using that coding system.When you specify the coding system for saving in this way, instead of with C-x RET f, there is no warning if the buffer contains characters that the coding system cannot handle.

Other file commands affected by a specified coding system include C-x i and C-x C-v, as well as the other-window variants of C-x C-f. C-x RET c also affects commands that start subprocesses, including M-x shell (see Section 31.3 [Shell], page 374). If the immediately following command does not use the coding system, then C-x RET c ultimately has no effect.

An easy way to visit a file with no conversion is with the M-x find-file-literally command. See Section 15.2 [Visiting], page 119.

The default value of the variable buffer-file-coding-system specifies the choice of

coding system to use when you create a new file. It applies when you find a new file,

and when you create a buffer and then save it in a file. Selecting a language environment

typically sets this variable to a good choice of default coding system for that language

environment.

If you visit a file with a wrong coding system, you can correct this with C-x RET r

(revert-buffer-with-coding-system). This visits the current file again, using a coding

system you specify.

If a piece of text has already been inserted into a buffer using the wrong coding system,

you can redo the decoding of it using M-x recode-region. This prompts you for the proper

coding system, then for the wrong coding system that was actually used, and does the

conversion. It first encodes the region using the wrong coding system, then decodes it again

using the proper coding system.

## Coding Systems for Interprocess Communication

This section explains how to specify coding systems for use in communication with other processes.

|  |  |
| --- | --- |
| C-x RET x coding RET | Use coding system coding for transferring selections to and from other graphical  applications (set-selection-coding-system). |
| C-x RET X coding RET | Use coding system coding for transferring one selection—the next one—to or from another graphical application (set-next-selection-coding-system). |
| C-x RET p input-coding RET output-coding RET | Use coding systems input-coding and output-coding for subprocess input and output in the current buffer (set-buffer-process-coding-system). |

The command C-x RET x (set-selection-coding-system) specifies the coding system for sending selected text to other windowing applications, and for receiving the text of selections made in other applications. This command applies to all subsequent selections, until you override it by using the command again. The command C-x RET X (set-nextselection-coding-system) specifies the coding system for the next selection made in Emacs or read by Emacs.

The variable x-select-request-type specifies the data type to request from the X Window System for receiving text selections from other applications. If the value is nil (the default), Emacs tries UTF8\_STRING and COMPOUND\_TEXT, in this order, and uses various heuristics to choose the more appropriate of the two results; if none of these succeed, Emacs falls back on STRING. If the value of x-select-request-type is one of the symbols COMPOUND\_TEXT, UTF8\_STRING, STRING, or TEXT, Emacs uses only that request type. If the value is a list of some of these symbols, Emacs tries only the request types in the list, in order, until one of them succeeds, or until the list is exhausted.

The command C-x RET p (set-buffer-process-coding-system) specifies the coding system for input and output to a subprocess. This command applies to the current buffer; normally, each subprocess has its own buffer, and thus you can use this command to specify translation to and from a particular subprocess by giving the command in the corresponding buffer.

You can also use C-x RET c (universal-coding-system-argument) just before the command that runs or starts a subprocess, to specify the coding system for communicating with that subprocess. See Section 19.10 [Text Coding], page 183.

The default for translation of process input and output depends on the current language environment.

The variable locale-coding-system specifies a coding system to use when encoding and decoding system strings such as system error messages and format-time-string formats and time stamps. That coding system is also used for decoding non-ASCII keyboard input on the X Window System. You should choose a coding system that is compatible with the underlying system’s text representation, which is normally specified by one of the environment variables LC\_ALL, LC\_CTYPE, and LANG. (The first one, in the order specified above, whose value is nonempty is the one that determines the text representation.)

## Coding Systems for File Names

C-x RET F coding RET

Use coding system coding for encoding and decoding file names (set-filename-coding-system).The command C-x RET F (set-file-name-coding-system) specifies a coding system to use for encoding file names. It has no effect on reading and writing the contents of files.

In fact, all this command does is set the value of the variable file-name-codingsystem. If you set the variable to a coding system name (as a Lisp symbol or a string), Emacs encodes file names using that coding system for all file operations. This makes it possible to use non-ASCII characters in file names—or, at least, those non-ASCII characters that the specified coding system can encode.

If file-name-coding-system is nil, Emacs uses a default coding system determined by the selected language environment, and stored in the default-file-name-coding-system variable. In the default language environment, non-ASCII characters in file names are not encoded specially; they appear in the file system using the internal Emacs representation.

**Warning**: if you change file-name-coding-system (or the language environment) in the middle of an Emacs session, problems can result if you have already visited files whose names were encoded using the earlier coding system and cannot be encoded (or are encoded differently) under the new coding system. If you try to save one of these buffers under the visited file name, saving may use the wrong file name, or it may encounter an error. If such a problem happens, use C-x C-w to specify a new file name for that buffer.

If a mistake occurs when encoding a file name, use the command M-x recode-file-name to change the file name’s coding system. This prompts for an existing file name, its old coding system, and the coding system to which you wish to convert.

## Coding Systems for Terminal I/O

C-x RET t coding RET

Use coding system coding for terminal output (set-terminal-codingsystem).

C-x RET k coding RET

Use coding system coding for keyboard input (set-keyboard-coding-system). The command C-x RET t (set-terminal-coding-system) specifies the coding system for terminal output. If you specify a character code for terminal output, all characters output to the terminal are translated into that coding system.

This feature is useful for certain character-only terminals built to support specific languages or character sets—for example, European terminals that support one of the ISO Latin character sets. You need to specify the terminal coding system when using multibyte text, so that Emacs knows which characters the terminal can actually handle.

By default, output to the terminal is not translated at all, unless Emacs can deduce the proper coding system from your terminal type or your locale specification (see Section 19.3 [Language Environments], page 174).

The command C-x RET k (set-keyboard-coding-system), or the variable keyboardcoding-system, specifies the coding system for keyboard input. Character-code translation of keyboard input is useful for terminals with keys that send non-ASCII graphic characters— for example, some terminals designed for ISO Latin-1 or subsets of it.

By default, keyboard input is translated based on your system locale setting. If your

terminal does not really support the encoding implied by your locale (for example, if you find it inserts a non-ASCII character if you type M-i), you will need to set keyboardcoding-system to nil to turn off encoding. You can do this by putting

(set-keyboard-coding-system nil)

in your init file.

There is a similarity between using a coding system translation for keyboard input, and using an input method: both define sequences of keyboard input that translate into single characters. However, input methods are designed to be convenient for interactive use by humans, and the sequences that are translated are typically sequences of ASCII printing characters. Coding systems typically translate sequences of non-graphic characters.

## Fontsets（字体集）

A font typically defines shapes for a single alphabet or script. Therefore, displaying the entire range of scripts that Emacs supports requires a collection of many fonts. In Emacs, such a collection is called a fontset. A fontset is defined by a list of font specifications, each assigned to handle a range of character codes, and may fall back on another fontset for characters that are not covered by the fonts it specifies.

字体通常定义了一个单独的字母或脚本的形状。因而，显示Emacs所支持的全部脚本需要很多字体的集合。Emacs中，这样一个集合叫做字体集。一个由字体规范构成的列表定义了字体集，每个字体规范处理一个范围的字符代码，如果它制定的字体没有包含这个范围内的字符编码，可能会会退到另一个字符集。

Each fontset has a name, like a font. However, while fonts are stored in the system

and the available font names are defined by the system, fontsets are defined within Emacs itself. Once you have defined a fontset, you can use it within Emacs by specifying its name, anywhere that you could use a single font. Of course, Emacs fontsets can use only the fonts that the system supports. If some characters appear on the screen as empty boxes or hex codes, this means that the fontset in use for them has no font for those characters. In this case, or if the characters are shown, but not as well as you would like, you may need to install extra fonts. Your operating system may have optional fonts that you can install; or you can install the GNU Intlfonts package, which includes fonts for most supported scripts.2

每个字符集有一个名字，比如字体。然而，虽然系统存储了字体并定义了可用的字体名字，但是Emacs内部自行定义了字体集。一旦定义了一个字体集，在任何可以使用一个单独字体的地方通过制定它的名字来使用这个字体集。当然，Emacs字体集只能使用系统支持的字体。如果屏幕上的一些字符显示为空盒子或者16进制代码，这意味着使用的字体集中没有这些字符的字体。这种情况下，即使显示字符，也不会是你喜欢的那种，你需要安装额外的字体。操作系统应该含有能够安装的可选字体；或者可以安装GNU Intlfonts package，它包括了大多数被支持的脚本的字体。2

2 *If you run Emacs on X, you may need to inform the X server about the location of the newly installed fonts with commands such as:*

*如果在X上运行Emacs，需要用以下命令通知X server关于新安装的字体的位置：*

*xset fp+ /usr/local/share/emacs/fonts*

*xset fp rehash*

Emacs creates three fontsets automatically: the standard fontset, the startup fontset and the default fontset. The default fontset is most likely to have fonts for a wide variety of non-ASCII characters, and is the default fallback for the other two fontsets, and if you set a default font rather than fontset. However, it does not specify font family names, so results can be somewhat random if you use it directly. You can specify use of a particular fontset by starting Emacs with the ‘-fn’ option. For example,

emacs -fn fontset-standard

You can also specify a fontset with the ‘Font’ resource (see Appendix D [X Resources], page 484).

Emacs自动创建了三种字体集：标准字体集，启动字体集和默认字体集。默认字体集最有可能包含支持各种各样非ASCII字符的字体，并且是另外两种字体集的回退，如果你设置了一个默认字体而不是字体集的话。然而，它没有指明字体系列的名字，所以直接使用它的话结果可能是随机的。在启动Emacs的时候可以用选项-fn来制定使用一个特殊的字体集。例如，

emacs -fn fontset-standard

也可以使用“字体”资源来制定一个字体集（参见Appendix D [X Resources], page 484）。

If no fontset is specified for use, then Emacs uses an ASCII font, with ‘fontset-default’ as a fallback for characters the font does not cover. The standard fontset is only used if explicitly requested, despite its name.

如果没有制定使用的字体集，Emacs会使用一个ASCII字体，‘fontset-default’作为ASCII字体没有覆盖时候的回滚。标准字符集只有在明确要求的时候才会使用，尽管它的名字可能不让人这么认为。

A fontset does not necessarily specify a font for every character code. If a fontset specifies no font for a certain character, or if it specifies a font that does not exist on your system, then it cannot display that character properly. It will display that character as a hex code or thin space or an empty box instead. (See Section 11.19 [glyphless characters], page 83, for details.)

字体集没必要为每个字符代码指定一个字体。如果字体集没有为某一字符制定字体，或者制定了一个在你的系统上不存在的字体，这类字符就不会正确的显示出来，他将会将这类字符显示为16进制代码或者窄空格或者一个空的盒子。

## Defining fontsets（定义字符集）

When running on X, Emacs creates a standard fontset automatically according to the value of standard-fontset-spec. This fontset’s name is

当在X上运行是，Emacs能通过standard-fontset-spec的值自动创建标准字体集。该字体集的名字是：

-\*-fixed-medium-r-normal-\*-16-\*-\*-\*-\*-\*-fontset-standard

or just ‘fontset-standard’ for short.

或者简写为‘fontset-standard’。

On GNUstep and Mac OS X, the standard fontset is created using the value of ns-standard-fontset-spec, and on MS Windows it is created using the value of w32-standard-fontset-spec.

GNUstep and Mac OS X中，使用ns-standard-fontset-spec创建标准字体集，在MS Windows中，使用w32-standard-fontset-spec来创建标准字体集。

Bold, italic, and bold-italic variants of the standard fontset are created automatically. Their names have ‘bold’ instead of ‘medium’, or ‘i’ instead of ‘r’, or both.

标准字体集的Bold, italic, and bold-italic会自动简历。它们的名字中使用‘bold’ 代替 ‘medium’, or ‘i’ 代替‘r’, 或者两者都有。

Emacs generates a fontset automatically, based on any default ASCII font that you specify with the ‘Font’ resource or the ‘-fn’ argument, or the default font that Emacs found when it started. This is the startup fontset and its name is fontset-startup. It does this by replacing the charset registry field with ‘fontset’, and replacing charset encoding field with ‘startup’, then using the resulting string to specify a fontset.

Emacs会根据你制定的默认的ASCII字体自动产生一个字体集，该ASCII字体是你通过font资源或-fn参数指定的，或者是当Emacs启动时候的默认字体。这是启动字体，它的名字是fontset-startup。他通过替换字符集注册表字段为fontset来做到这个。替换字符集编码字段为startup，然后通过生成字符串来制定一个字体集。

For instance, if you start Emacs with a font of this form,

例如，如果使用这种形式的字体启动Emacs

emacs -fn "\*courier-medium-r-normal--14-140-\*-iso8859-1"

Emacs generates the following fontset and uses it for the initial X window frame:

Emacs会产生下面的字符集，并使用它来初始化X window frame。

-\*-courier-medium-r-normal-\*-14-140-\*-\*-\*-\*-fontset-startup

The startup fontset will use the font that you specify, or a variant with a different registry and encoding, for all the characters that are supported by that font, and fallback on ‘fontset-default’ for other characters.

对于改字体支持的所有的字符，启动字体集将会使用你指定的字体，或者是一个使用不同注册字段和编码字段的变种。对于其他的字符则会回滚到”fontset-default“。

With the X resource ‘Emacs.Font’, you can specify a fontset name just like an actual font name. But be careful not to specify a fontset name in a wildcard resource like ‘Emacs\*Font’—that wildcard specification matches various other resources, such as for menus, and menus cannot handle fontsets. See Appendix D [X Resources], page 484. You can specify additional fontsets using X resources named ‘Fontset-n’, where n is an integer starting from 0. The resource value should have this form:

使用X资源“Emacs.Font”，你可以就想实际名字一样制定一个字符集名字。但要小心，不要指定一个字符集名字 以通配符资源像“Emacs\*Font”---通配符指定匹配各种其他资源，比如菜单，但是菜单不能处理字体集。参见Appendix D [X Resources], page 484。你可以制定额外的字体集使用X资源名为“Fontset-n”，n是一个从0开始的正数。资源的值应该是下面这种形式。

fontpattern, [charset:font]. . .

fontpattern should have the form of a standard X font name (see the previous fontset-startup example), except for the last two fields. They should have the form ‘fontset-alias’.

The fontset has two names, one long and one short. The long name is fontpattern. The short name is ‘fontset-alias’. You can refer to the fontset by either name.

The construct ‘charset:font’ specifies which font to use (in this fontset) for one particular character set. Here, charset is the name of a character set, and font is the font to use for that character set. You can use this construct any number of times in defining one fontset.

For the other character sets, Emacs chooses a font based on fontpattern. It replaces ‘fontset-alias’ with values that describe the character set. For the ASCII character font, ‘fontset-alias’ is replaced with ‘ISO8859-1’.

In addition, when several consecutive fields are wildcards, Emacs collapses them into a single wildcard. This is to prevent use of auto-scaled fonts. Fonts made by scaling larger fonts are not usable for editing, and scaling a smaller font is not also useful, because it is better to use the smaller font in its own size, which is what Emacs does.

Thus if fontpattern is this,

-\*-fixed-medium-r-normal-\*-24-\*-\*-\*-\*-\*-fontset-24

the font specification for ASCII characters would be this:

-\*-fixed-medium-r-normal-\*-24-\*-ISO8859-1

and the font specification for Chinese GB2312 characters would be this:

-\*-fixed-medium-r-normal-\*-24-\*-gb2312\*-\*

You may not have any Chinese font matching the above font specification. Most X distributions include only Chinese fonts that have ‘song ti’ or ‘fangsong ti’ in the family field. In such a case, ‘Fontset-n’ can be specified as:

Emacs.Fontset-0: -\*-fixed-medium-r-normal-\*-24-\*-\*-\*-\*-\*-fontset-24,\

chinese-gb2312:-\*-\*-medium-r-normal-\*-24-\*-gb2312\*-\*

Then, the font specifications for all but Chinese GB2312 characters have ‘fixed’ in the family field, and the font specification for Chinese GB2312 characters has a wild card ‘\*’ in the family field.

The function that processes the fontset resource value to create the fontset is called create-fontset-from-fontset-spec. You can also call this function explicitly to create a fontset. See Section 18.8 [Fonts], page 162, for more information about font naming.

## Modifying Fontsets（修改字符集）

Fontsets do not always have to be created from scratch. If only minor changes are required it may be easier to modify an existing fontset. Modifying ‘fontset-default’ will also affect other fontsets that use it as a fallback, so can be an effective way of fixing problems with the fonts that Emacs chooses for a particular script.

Fontsets can be modified using the function set-fontset-font, specifying a character, a charset, a script, or a range of characters to modify the font for, and a font specification for the font to be used. Some examples are:

;; Use Liberation Mono for latin-3 charset.

(set-fontset-font "fontset-default" ’iso-8859-3

"Liberation Mono")

;; Prefer a big5 font for han characters

(set-fontset-font "fontset-default"

’han (font-spec :registry "big5")

nil ’prepend)

;; Use DejaVu Sans Mono as a fallback in fontset-startup

;; before resorting to fontset-default.

(set-fontset-font "fontset-startup" nil "DejaVu Sans Mono"

nil ’append)

;; Use MyPrivateFont for the Unicode private use area.

(set-fontset-font "fontset-default" ’(#xe000 . #xf8ff)

"MyPrivateFont")

## Undisplayable Characters（不可显示的字符）

There may be some non-ASCII characters that your terminal cannot display. Most text terminals support just a single character set (use the variable default-terminal-codingsystem to tell Emacs which one, Section 19.13 [Terminal Coding], page 185); characters that can’t be encoded in that coding system are displayed as ‘?’ by default.

Graphical displays can display a broader range of characters, but you may not have fonts installed for all of them; characters that have no font appear as a hollow box.

If you use Latin-1 characters but your terminal can’t display Latin-1, you can arrange to display mnemonic ASCII sequences instead, e.g., ‘"o’ for o-umlaut. Load the library iso-ascii to do this.

If your terminal can display Latin-1, you can display characters from other European character sets using a mixture of equivalent Latin-1 characters and ASCII mnemonics. Customize the variable latin1-display to enable this. The mnemonic ASCII sequences mostly correspond to those of the prefix input methods.

## Unibyte Editing Mode

The ISO 8859 Latin-n character sets define character codes in the range 0240 to 0377 octal (160 to 255 decimal) to handle the accented letters and punctuation needed by various European languages (and some non-European ones). Note that Emacs considers bytes with codes in this range as raw bytes, not as characters, even in a unibyte buffer, i.e., if you disable multibyte characters. However, Emacs can still handle these character codes as if they belonged to one of the single-byte character sets at a time. To specify which of these codes to use, invoke M-x set-language-environment and specify a suitable language environment such as ‘Latin-n’.

For more information about unibyte operation, see Section 19.2 [Disabling Multibyte], page 173.

Emacs can also display bytes in the range 160 to 255 as readable characters, provided the terminal or font in use supports them. This works automatically. On a graphical display, Emacs can also display single-byte characters through fontsets, in effect by displaying the equivalent multibyte characters according to the current language environment. To request this, set the variable unibyte-display-via-language-environment to a non-nil value. Note that setting this only affects how these bytes are displayed, but does not change the fundamental fact that Emacs treats them as raw bytes, not as characters.

If your terminal does not support display of the Latin-1 character set, Emacs can display these characters as ASCII sequences which at least give you a clear idea of what the characters are. To do this, load the library iso-ascii. Similar libraries for other Latin-n character sets could be implemented, but have not been so far.

Normally non-ISO-8859 characters (decimal codes between 128 and 159 inclusive) are displayed as octal escapes. You can change this for non-standard “extended” versions of ISO-8859 character sets by using the function standard-display-8bit in the disp-table library.

There are two ways to input single-byte non-ASCII characters:

You can use an input method for the selected language environment. See Section 19.4 [Input Methods], page 175. When you use an input method in a unibyte buffer, the non-ASCII character you specify with it is converted to unibyte.

If your keyboard can generate character codes 128 (decimal) and up, representing nonASCII characters, you can type those character codes directly. On a graphical display, you should not need to do anything special to use these keys; they should simply work. On a text terminal, you should use the command M-x set-keyboard-coding-system or customize the variable keyboard-coding-system to specify which coding system your keyboard uses (see Section 19.13 [Terminal Coding], page 185). Enabling this feature will probably require you to use ESC to type Meta characters; however, on a console terminal or in xterm, you can arrange for Meta to be converted to ESC and still be able type 8-bit characters present directly on the keyboard or using Compose or AltGr keys. See Section 2.1 [User Input], page 11.

For Latin-1 only, you can use the key C-x 8 as a “compose character” prefix for entry of non-ASCII Latin-1 printing characters. C-x 8 is good for insertion (in the minibuffer as well as other buffers), for searching, and in any other context where a key sequence is allowed.

C-x 8 works by loading the iso-transl library. Once that library is loaded, the ALT

modifier key, if the keyboard has one, serves the same purpose as C-x 8: use ALT

together with an accent character to modify the following letter. In addition, if the

keyboard has keys for the Latin-1 “dead accent characters”, they too are defined to

compose with the following character, once iso-transl is loaded.

Use C-x 8 C-h to list all the available C-x 8 translations.

## Charsets

In Emacs, charset is short for “character set”. Emacs supports most popular charsets (such as ascii, iso-8859-1, cp1250, big5, and unicode), in addition to some charsets of its own (such as emacs, unicode-bmp, and eight-bit). All supported characters belong to one or more charsets.

Emacs normally “does the right thing” with respect to charsets, so that you don’t have to worry about them. However, it is sometimes helpful to know some of the underlying details about charsets.

One example is font selection (see Section 18.8 [Fonts], page 162). Each language environment (see Section 19.3 [Language Environments], page 174) defines a “priority list” for the various charsets. When searching for a font, Emacs initially attempts to find one that can display the highest-priority charsets. For instance, in the Japanese language environment, the charset japanese-jisx0208 has the highest priority, so Emacs tries to use a font whose registry property is ‘JISX0208.1983-0’.

There are two commands that can be used to obtain information about charsets. The command M-x list-charset-chars prompts for a charset name, and displays all the characters in that character set. The command M-x describe-character-set prompts for a charset name, and displays information about that charset, including its internal representation within Emacs.

M-x list-character-sets displays a list of all supported charsets. The list gives the names of charsets and additional information to identity each charset; see the International Register of Coded Character Sets for more details. In this list, charsets are divided into two categories: normal charsets are listed first, followed by supplementary charsets. A supplementary charset is one that is used to define another charset (as a parent or a subset), or to provide backward-compatibility for older Emacs versions.

To find out which charset a character in the buffer belongs to, put point before it and type C-u C-x = (see Section 19.1 [International Chars], page 171).

## Bidirectional Editing

Emacs supports editing text written in scripts, such as Arabic and Hebrew, whose natural ordering of horizontal text for display is from right to left. However, digits and Latin text embedded in these scripts are still displayed left to right. It is also not uncommon to have small portions of text in Arabic or Hebrew embedded in an otherwise Latin document; e.g., as comments and strings in a program source file. For these reasons, text that uses these scripts is actually bidirectional: a mixture of runs of left-to-right and right-to-left characters.

This section describes the facilities and options provided by Emacs for editing bidirectional text.

Emacs stores right-to-left and bidirectional text in the so-called logical (or reading ) order: the buffer or string position of the first character you read precedes that of the next character. Reordering of bidirectional text into the visual order happens at display time. As result, character positions no longer increase monotonically with their positions on display. Emacs implements the Unicode Bidirectional Algorithm described in the Unicode Standard Annex #9, for reordering of bidirectional text for display.

The buffer-local variable bidi-display-reordering controls whether text in the buffer is reordered for display. If its value is non-nil, Emacs reorders characters that have rightto-left directionality when they are displayed. The default value is t.

Each paragraph of bidirectional text can have its own base direction, either right-toleft or left-to-right. (Paragraph boundaries are empty lines, i.e., lines consisting entirely of whitespace characters.) Text in left-to-right paragraphs begins on the screen at the left margin of the window and is truncated or continued when it reaches the right margin. By contrast, text in right-to-left paragraphs is displayed starting at the right margin and is continued or truncated at the left margin.

Emacs determines the base direction of each paragraph dynamically, based on the text at the beginning of the paragraph. However, sometimes a buffer may need to force a certain base direction for its paragraphs. The variable bidi-paragraph-direction, if non-nil, disables the dynamic determination of the base direction, and instead forces all paragraphs in the buffer to have the direction specified by its buffer-local value. The value can be either right-to-left or left-to-right. Any other value is interpreted as nil.

Alternatively, you can control the base direction of a paragraph by inserting special formatting characters in front of the paragraph. The special character RIGHT-TO-LEFT MARK, or rlm, forces the right-to-left direction on the following paragraph, while LEFTTO-RIGHT MARK, or lrm forces the left-to-right direction. (You can use C-x 8 RET to insert these characters.) In a GUI session, the lrm and rlm characters display as very thin blank characters; on text terminals they display as blanks.

Because characters are reordered for display, Emacs commands that operate in the logical order or on stretches of buffer positions may produce unusual effects. For example, C-f and C-b commands move point in the logical order, so the cursor will sometimes jump when point traverses reordered bidirectional text. Similarly, a highlighted region covering a contiguous range of character positions may look discontinuous if the region spans reordered text. This is normal and similar to the behavior of other programs that support bidirectional text.

# ****Major and Minor Modes**** （主模式和辅助模式）

# ****Indentation**** （缩进）

Indentation refers to inserting or adjusting whitespace characters (space and/or tab characters) at the beginning of a line of text. This chapter documents indentation commands and options which are common to Text mode and related modes, as well as programming language modes. See Section 23.3 [Program Indent], page 237, for additional documentation about indenting in programming modes.

缩进是指在文本的开头的一行中插入或调整空白字符（空格和/或制表符）。本章说明缩进命令和选项，常见于Text mode和相关modes，以及编程语言modes。参见Section 23.3 [Program Indent], page 237，来了解关于编程语言modes中关于缩进的额外说明。

The simplest way to perform indentation is the TAB key. In most major modes, this runs the command indent-for-tab-command. (In C and related modes, TAB runs the command c-indent-line-or-region, which behaves similarly).

执行缩进最简单的方法是TAB键。在大多数的major modes中，它会执行命令indent-for-tab-command。（在C和相关modes中，TAB执行命令c-indent-line-or-region，其行为类似）。

|  |  |
| --- | --- |
| TAB | Insert whitespace, or indent the current line, in a mode-appropriate way (indent-for-tab-command). If the region is active, indent all the lines within it.  以模式恰当的方式(indent-for-tab-command)插入空白符，或缩进当前行。如果区域是激活的，用它缩进所有的行。 |

The exact behavior of TAB depends on the major mode. In Text mode and related major modes, TAB normally inserts some combination of space and tab characters to advance point to the next tab stop (see Section 21.2 [Tab Stops], page 200). For this purpose, the position of the first non-whitespace character on the preceding line is treated as an additional tab stop, so you can use TAB to “align” point with the preceding line. If the region is active (see Section 8.3 [Using Region], page 47), TAB acts specially: it indents each line in the region so that its first non-whitespace character is aligned with the preceding line.

TAB键的确切行为取决于major mode。在Text mdoe和相关的mode中，TAB通常会插入一些空格符和制表符的组合来将point推进到下一个tab stop（参见Section 21.2 [Tab Stops], page 200）。为了这个目的，上一行中第一个非空白字符的位置被视为一个额外的tab stop，所以你可以使用TAB来与上一行进行“对齐”。如果区域是活跃的（参见Section 21.2 [Tab Stops], page 200），TAB会特别表现：它缩进该区域的每一行，那样它的第一个非空白字符和上一行对齐。

In programming modes, TAB indents the current line of code in a way that makes sense given the code in the preceding lines. If the region is active, all the lines in the region are indented this way. If point was initially within the current line’s indentation, it is repositioned to the first non-whitespace character on the line.

在编程模式中，TAB以这样一种方式缩进当前代码行，上一行中给出的代码。如果区域是活跃的，区域中所有行以这种方式缩进。如果point被使用当前行的缩进初始化，它重定位在行上的第一个非空白字符。

If you just want to insert a tab character in the buffer, type C-q TAB (see Section 4.1 [Inserting Text], page 16).

如果只是想插入一个制表符，键入C-q TAB（参见Section 4.1 [Inserting Text], page 16）。

## Indentation Commands（缩进命令）

Apart from the TAB (indent-for-tab-command) command, Emacs provides a variety of commands to perform indentation in other ways.

除了TAB（indent-for-tab-command）命令，Emacs还提供了很多命令来以其他方式执行缩进。

|  |  |
| --- | --- |
| C-j | Perform RET followed by TAB (newline-and-indent).  执行先回车然后缩进（newline-and-indent）。 |
| C-M-o | Split the current line at point (split-line). The text on the line after point becomes a new line, indented to the same column where point is located. This command first moves point forward over any spaces and tabs. Afterward, point is positioned before the inserted newline.  在point上分割当前行（split-line）。行上point之后的文本将会成为一个新行，和point所在的列进行对其。该命令首先向前移动，越过所有的空格和制表符。随后，point放置在插入的新行前面。 |
| M-m | Move (forward or back) to the first non-whitespace character on the current line (back-to-indentation). If there are no non-whitespace characters on the line, move to the end of the line.  移动（向前或向后）移动到当前行第一个非空白字符（back-to-indentation）。如果行上没有非空白字符，就移动到行末尾。 |
| M-i | Indent whitespace at point, up to the next tab stop (tab-to-tab-stop). See Section 21.2 [Tab Stops], page 200.  在point处缩进空白字符，直到下一个tab stop（tab-to-tab-stop）。参见Section 21.2 [Tab Stops], page 200。 |
| M-x indent-relative | Insert whitespace at point, until point is aligned with the first non-whitespace character on the previous line (actually, the last non-blank line). If point is already farther right than that, run tab-to-tab-stop instead—unless called with a numeric argument, in which case do nothing.  在point处插入空白字符，直到point与前一行第一个非空白字符对齐为止（实际上是最后一个非空行）。如果point过于靠右了，反而执行tab-to-tab-stop除非通过一个数字参数调用，这种情况下什么都不做。 |
| M-^ | Merge the previous and the current line (delete-indentation). This “joins” the two lines cleanly, by replacing any indentation at the front of the current line, together with the line boundary, with a single space. As a special case (useful for Lisp code), the single space is omitted if the characters to be joined are consecutive opening and closing parentheses, or if the junction follows another newline. If there is a fill prefix, M-^ deletes the fill prefix if it appears after the newline that is deleted. See Section 22.5.3 [Fill Prefix], page 208.  合并前一行和当前行（delete-indentation）。这样通过将当前行前面的缩进和行边界替换为单一的空格符，来将两行干净的连接在一起。特殊情况下（对Lisp mode有用），如果被连接的连续的开或闭括号，或者连接后面跟着的是另一个换行符，单一的空格符会被删除。如果有一个填充前缀，如果它出现在删除的换行符后面，M-^会删除填充前缀。参见Section 22.5.3 [Fill Prefix], page 208。 |
| C-M-\ | Indent all the lines in the region, as though you had typed TAB at the beginning of each line (indent-region). If a numeric argument is supplied, indent every line in the region to that column number.  缩进region内所有的行，就想你已经在每行的前面都按下TAB键一样。如果提供一个数字参数缩进region内的所有行那么多列。 |
| C-x TAB | Shift each line in the region by a fixed distance, to the right or left (indent-rigidly). The distance to move is determined by the numeric argument (positive to move rightward, negative to move leftward). This command can be used to remove all indentation from the lines in the region, by invoking it with a large negative argument, e.g., C-u -1000 C-x TAB.  向右或向左缩进region内的每一行一个固定的距离(indent-rigidly)。移动的距离取决于数字参数（正数向右，负数向左）。该命令在调用时指定一个大的负数，能够用来移除region内所有行的缩进。例如，C-u -1000 C-x TAB。 |

## Tab Stops（）

Emacs defines certain column numbers to be tab stops. These are used as stopping points by TAB when inserting whitespace in Text mode and related modes (see Chapter 21 [Indentation], page 199), and by commands like M-i (see Section 21.1 [Indentation Commands], page 199). By default, tab stops are located every 8 columns. These positions are stored in the variable tab-stop-list, whose value is a list of column numbers in increasing order.

Emacs定义了特定的列数目作为tab stops。当在Text mode和相关modes中插入空白符（参见Chapter 21 [Indentation], page 199），TAB键或者像M-i的命令 (see Section 21.1 [Indentation Commands], page 199)使用这些tab stops用来做停止的points。默认情况下，tab stops被设定为每8列。这些位置存储在变量tab-stop-list中，该变量的值是一个递增的列数目的列表。

Instead of customizing the variable tab-stop-list directly, a convenient way to view and set tab stops is via the command M-x edit-tab-stops. This switches to a buffer containing a description of the tab stop settings, which looks like this:

不要直接定制tab-stop-list变量，通过命令M-x edit-tab-stops可以方便的查看和设置tab stops。他回切换到一个描述tab stop设置的buffer，看起来像这样：

: : : : : :

0 1 2 3 4

01234567890123456789012345678901234567890123456780123456789

To install changes, type C-c C-c

如果安装更改，键入C-c C-c

The first line contains a colon at each tab stop. The numbers on the next two lines are present just to indicate where the colons are.

第一行在每个tab stop处包含一个冒号。接下来两行上出现的数字只是待变每个冒号的位置。

You can edit this buffer to specify different tab stops by placing colons on the desired columns. The buffer uses Overwrite mode (see Section 20.2 [Minor Modes], page 194). When you are done, type C-c C-c to make the new tab stops take effect. Normally, the new tab stop settings apply to all buffers. However, if you have made the tab-stop-list variable local to the buffer where you called M-x edit-tab-stops (see Section 33.2.3 [Locals], page 415), then the new tab stop settings apply only to that buffer. To save the tab stop settings for future Emacs sessions, use the Customize interface to save the value of tab-stop-list (see Section 33.1 [Easy Customization], page 404).

你可以编辑这个buffer来制定的不同的tab stops，只需要将冒号放在所需要的列上。该buffer使用了Overwrite mode (参见Section 20.2 [Minor Modes], page 194)。当你做完以后，按下C-c C-c来使新的tab stops生效。一般，新的tab stop设置会应用到所有的buffer。然而，如果将tab-stop-list变量设置为只对你调用M-x edit-tab-stops的buffer本地化（参见Section 33.2.3 [Locals], page 415），新的tab stop设置就会只应用在该buffer。为将来的Emacs 会话保存tab stop设置，使用定制界面来保存tab-stop-list的值（参见Section 33.1 [Easy Customization], page 404）。

Note that the tab stops discussed in this section have nothing to do with how tab characters are displayed in the buffer. Tab characters are always displayed as empty spaces extending to the next display tab stop. See Section 11.19 [Text Display], page 83.

请注意本节所讨论的tab stops和buffer中tab字符如何显示没有任何关系。Tab 字符总是显示为空白直到下一个tab stop。参见Section 11.19 [Text Display], page 83。

## Tabs vs. Spaces（制表符VS.空格符）

Normally, indentation commands insert (or remove) an optimal mix of space characters and tab characters to align to the desired column. Tab characters are displayed as a stretch of empty space extending to the next display tab stop. By default, there is one display tab stop every tab-width columns (the default is 8). See Section 11.19 [Text Display], page 83.

通常情况下，缩进命令插入（或移除）空格符和制表符的最佳组合已达到对齐到所需列的目的。制表符直到下一个tab stop都会显示为一片的空格符。默认情况下，每tab-width个列显示为一个tab top。参见Section 11.19 [Text Display], page 83。

If you prefer, all indentation can be made from spaces only. To request this, set the buffer-local variable indent-tabs-mode to nil. See Section 33.2.3 [Locals], page 415, for information about setting buffer-local variables. Note, however, that C-q TAB always inserts a tab character, regardless of the value of indent-tabs-mode.

如你所愿，只需要空格符就能完成所有的缩进。要这样做，设置buffer-local变量indent-tabs-mode为nil。参见See Section 33.2.3 [Locals], page 415，来获得关于设置buffer-local变量的信息。请注意，然而，C-q TAB总是插入一个制表符，无视indent-tabs-mode值。

One reason to set indent-tabs-mode to nil is that not all editors display tab characters in the same way. Emacs users, too, may have different customized values of tab-width. By using spaces only, you can make sure that your file always looks the same. If you only care about how it looks within Emacs, another way to tackle this problem is to set the tab-width variable in a file-local variable (see Section 33.2.4 [File Variables], page 416).

将indent-tabs-mode设置为nil的原因之一是并不是所有的编辑器都以相同的方式来显示制表符。Emacs用户同样对tab-width有各种各样的定制。只是用空格符，可以确保文件总是看起来相同。如果只是关心文件如何在Emacs中显示，解决这个问题的另一种方法是在file-local变量中设置tab-width（参见Section 33.2.4 [File Variables], page 416）。

There are also commands to convert tabs to spaces or vice versa, always preserving the columns of all non-whitespace text. M-x tabify scans the region for sequences of spaces, and converts sequences of at least two spaces to tabs if that can be done without changing indentation. M-x untabify changes all tabs in the region to appropriate numbers of spaces.

有些命令可以将制表符转变为空格符，反之亦然，它们总是保存非空白字符文本的列数。M-x tabify扫描region来获得空格符的序列，并将至少两个空格符转换为一个制表符，如果这么做不会影响缩进的话。M-x untabify将region内的所有的制表符转换为恰当数量的空格符。

## Convenience Features for Indentation（方便缩进的函数）

The variable tab-always-indent tweaks the behavior of the TAB (indent-for-tab-command) command. The default value, t, gives the behavior described in Chapter 21 [Indentation], page 199. If you change the value to the symbol complete, then TAB first tries to indent the current line, and if the line was already indented, it tries to complete the text at point (see Section 23.8 [Symbol Completion], page 248). If the value is nil, then TAB indents the current line only if point is at the left margin or in the line’s indentation; otherwise, it inserts a tab character.

变量tab-always-indent会对TAB（indent-for-tab-command）的行为进行微调。该变量默认值是t，Chapter 21 [Indentation], page 199给出了它对应的行为。如果将它的值变为符号补全，然后TAB首先尝试缩进该行，然后，如果该行已经缩进，他就会尝试补全该point处的文本（参见Section 23.8 [Symbol Completion], page 248）。如果该值是nil，只有在point处于该行左边缘的时候才TAB键才会缩进该行；否则，就插入一个制表符。

Electric Indent mode is a global minor mode that automatically indents the line after every RET you type. To toggle this minor mode, type M-x electric-indent-mode.

Electric Indent mode是一个全局minor mode，可以在按下回车键之后自动缩进每一行。输入M-x electric-indent-mode可以切换到这个minor mode。

# ****Commands for Human Languages****

# ****Editing Programs**** （编辑程序）

This chapter describes Emacs features for facilitating editing programs. Some of the things these features can do are:

* Find or move over top-level definitions (see Section 23.2 [Defuns], page 235).
* Apply the usual indentation conventions of the language (see Section 23.3 [Program Indent], page 237).
* Balance parentheses (see Section 23.4 [Parentheses], page 240).
* Insert, kill or align comments (see Section 23.5 [Comments], page 243).
* Highlight program syntax (see Section 11.12 [Font Lock], page 76).

## Major Modes for Programming Languages

Emacs has specialized major modes (see Section 20.1 [Major Modes], page 193) for many programming languages. A programming language mode typically specifies the syntax of expressions, the customary rules for indentation, how to do syntax highlighting for the language, and how to find the beginning or end of a function definition. It often has features for compiling and debugging programs as well. The major mode for each language is named after the language; for instance, the major mode for the C programming language is c-mode.

Emacs has programming language modes for Lisp, Scheme, the Scheme-based DSSSL expression language, Ada, ASM, AWK, C, C++, Delphi, Fortran, Icon, IDL (CORBA), IDLWAVE, Java, Javascript, Metafont (TEX’s companion for font creation), Modula2, Objective-C, Octave, Pascal, Perl, Pike, PostScript, Prolog, Python, Ruby, Simula, Tcl, and VHDL. An alternative mode for Perl is called CPerl mode. Modes are also available for the scripting languages of the common GNU and Unix shells, VMS DCL, and MSDOS/MS-Windows ‘BAT’ files, and for makefiles, DNS master files, and various sorts of configuration files.

Ideally, Emacs should have a major mode for each programming language that you might want to edit. If it doesn’t have a mode for your favorite language, the mode might be implemented in a package not distributed with Emacs (see Chapter 32 [Packages], page 400); or you can contribute one.

In most programming languages, indentation should vary from line to line to illustrate the structure of the program. Therefore, in most programming language modes, typing TAB updates the indentation of the current line (see Section 23.3 [Program Indent], page 237). Furthermore, DEL is usually bound to backward-delete-char-untabify, which deletes backward treating each tab as if it were the equivalent number of spaces, so that you can delete one column of indentation without worrying whether the whitespace consists of spaces or tabs.

Entering a programming language mode runs the custom Lisp functions specified in the hook variable prog-mode-hook, followed by those specified in the mode’s own mode hook (see Section 20.1 [Major Modes], page 193). For instance, entering C mode runs the hooks prog-mode-hook and c-mode-hook. See Section 33.2.2 [Hooks], page 414, for information about hooks.

The Emacs distribution contains Info manuals for the major modes for Ada, C/C++/Objective C/Java/Corba IDL/Pike/AWK, and IDLWAVE. For Fortran mode, see Section “Fortran” in Specialized Emacs Features.

## Top-Level Definitions, or Defuns

In Emacs, a major definition at the top level in the buffer, such as a function, is called a defun. The name comes from Lisp, but in Emacs we use it for all languages.

### Left Margin Convention

Many programming-language modes assume by default that any opening delimiter found at the left margin is the start of a top-level definition, or defun. Therefore, don’t put an opening delimiter at the left margin unless it should have that significance. For instance, never put an open-parenthesis at the left margin in a Lisp file unless it is the start of a top-level list.

很多编程语言模式默认假设

The convention speeds up many Emacs operations, which would otherwise have to scan back to the beginning of the buffer to analyze the syntax of the code.

If you don’t follow this convention, not only will you have trouble when you explicitly use the commands for motion by defuns; other features that use them will also give you trouble. This includes the indentation commands (see Section 23.3 [Program Indent], page 237) and Font Lock mode (see Section 11.12 [Font Lock], page 76).

The most likely problem case is when you want an opening delimiter at the start of a line inside a string. To avoid trouble, put an escape character (‘\’, in C and Emacs Lisp, ‘/’ in some other Lisp dialects) before the opening delimiter. This will not affect the contents of the string, but will prevent that opening delimiter from starting a defun. Here’s an example:

(insert "Foo:

\(bar)

")

To help you catch violations of this convention, Font Lock mode highlights confusing opening delimiters (those that ought to be quoted) in bold red.

If you need to override this convention, you can do so by setting the variable openparen-in-column-0-is-defun-start. If this user option is set to t (the default), opening parentheses or braces at column zero always start defuns. When it is nil, defuns are found by searching for parens or braces at the outermost level.

Usually, you should leave this option at its default value of t. If your buffer contains parentheses or braces in column zero which don’t start defuns, and it is somehow impractical to remove these parentheses or braces, it might be helpful to set the option to nil. Be aware that this might make scrolling and display in large buffers quite sluggish. Furthermore, the parentheses and braces must be correctly matched throughout the buffer for it to work properly.

### Moving by Defuns

These commands move point or set up the region based on top-level major definitions, also called defuns.

|  |  |
| --- | --- |
| C-M-a | Move to beginning of current or preceding defun (beginning-of-defun). |
| C-M-e | Move to end of current or following defun (end-of-defun). |
| C-M-h | Put region around whole current or following defun (mark-defun). |

The commands to move to the beginning and end of the current defun are C-M-a (beginning-of-defun) and C-M-e (end-of-defun). If you repeat one of these commands, or use a positive numeric argument, each repetition moves to the next defun in the direction of motion.

C-M-a with a negative argument--n moves forward n times to the next beginning of a defun. This is not exactly the same place that C-M-e with argument n would move to; the end of this defun is not usually exactly the same place as the beginning of the following defun. (Whitespace, comments, and perhaps declarations can separate them.) Likewise, C-M-e with a negative argument moves back to an end of a defun, which is not quite the same as C-M-a with a positive argument.

To operate on the current defun, use C-M-h (mark-defun), which sets the mark at the end of the current defun and puts point at its beginning. See Section 8.2 [Marking Objects], page 46. This is the easiest way to get ready to kill the defun in order to move it to a different place in the file. If you use the command while point is between defuns, it uses the following defun. If you use the command while the mark is already active, it sets the mark but does not move point; furthermore, each successive use of C-M-h extends the end of the region to include one more defun.

In C mode, C-M-h runs the function c-mark-function, which is almost the same as mark-defun; the difference is that it backs up over the argument declarations, function name and returned data type so that the entire C function is inside the region. This is an example of how major modes adjust the standard key bindings so that they do their standard jobs in a way better fitting a particular language. Other major modes may replace any or all of these key bindings for that purpose.

### Imenu

The Imenu facility offers a way to find the major definitions in a file by name. It is also useful in text formatter major modes, where it treats each chapter, section, etc., as a definition. (See Section 25.3 [Tags], page 291, for a more powerful feature that handles multiple files together.)

If you type M-x imenu, it reads the name of a definition using the minibuffer, then moves point to that definition. You can use completion to specify the name; the command always displays the whole list of valid names.

Alternatively, you can bind the command imenu to a mouse click. Then it displays mouse menus for you to select a definition name. You can also add the buffer’s index to the menu bar by calling imenu-add-menubar-index. If you want to have this menu bar item available for all buffers in a certain major mode, you can do this by adding imenu-addmenubar-index to its mode hook. But if you have done that, you will have to wait a little while each time you visit a file in that mode, while Emacs finds all the definitions in that buffer.

When you change the contents of a buffer, if you add or delete definitions, you can update the buffer’s index based on the new contents by invoking the ‘\*Rescan\*’ item in the menu. Rescanning happens automatically if you set imenu-auto-rescan to a non-nil value. There is no need to rescan because of small changes in the text.

You can customize the way the menus are sorted by setting the variable imenu-sortfunction. By default, names are ordered as they occur in the buffer; if you want alphabetic sorting, use the symbol imenu--sort-by-name as the value. You can also define your own comparison function by writing Lisp code.

Imenu provides the information to guide Which Function mode (see below). The Speedbar can also use it (see Section 18.9 [Speedbar], page 165).

### Which Function Mode

Which Function mode is a global minor mode (see Section 20.2 [Minor Modes], page 194) which displays the current function name in the mode line, updating it as you move around in a buffer.

To either enable or disable Which Function mode, use the command M-x which-function-mode. Which Function mode is a global minor mode. By default, it takes effect in all major modes major modes that know how to support it (i.e., all the major modes that support Imenu). You can restrict it to a specific list of major modes by changing the value of the variable which-func-modes from t (which means to support all available major modes) to a list of major mode names.

## Indentation for Programs

The best way to keep a program properly indented is to use Emacs to reindent it as you change it. Emacs has commands to indent either a single line, a specified number of lines, or all of the lines inside a single parenthetical grouping.

保持程序正确缩进的最好方法就是通过Emacs重新缩进来改变它。Emacs有一些命令能够缩进单独的一行或者几行，或者一个括号分组之内的所有行。

See Chapter 21 [Indentation], page 199, for general information about indentation. This section describes indentation features specific to programming language modes.

参见Chapter 21 [Indentation], page 199，了解有关缩进的概要信息。本节描述和编程语言模式相关的特定缩进。

Emacs also provides a Lisp pretty-printer in the pp package, which reformats Lisp objects with nice-looking indentation.

Emacs还在pp包中提供了一个lisp 漂亮的打印机，可以使用好看的缩进来重新格式化lisp对象。

### Basic Program Indentation Commands

|  |  |
| --- | --- |
| TAB | Adjust indentation of current line (indent-for-tab-command). |
| C-j | Insert a newline, then adjust indentation of following line (newline-and-indent) |

The basic indentation command is TAB (indent-for-tab-command), which was documented in Chapter 21 [Indentation], page 199. In programming language modes, TAB indents the current line, based on the indentation and syntactic content of the preceding lines; if the region is active, TAB indents each line within the region, not just the current line.

基本的缩进命令式TAB（indent-for-tab-command），在Chapter 21 [Indentation], page 199有说明。在编程语言模式中，TAB缩进当前航，基于前一行的格式或者语法内容；如果区域是活跃的，TAB缩进区域内的所有行，而不只是当前行。

The command C-j (newline-and-indent), which was documented in Section 21.1 [Indentation Commands], page 199, does the same as RET followed by TAB: it inserts a new line, then adjusts the line’s indentation.

命令C-j（newline-and-indent），在Section 21.1 [Indentation Commands], page 199描述，相当于回车之后按TAB键：它插入一个新行，然后调整新行的缩进。

When indenting a line that starts within a parenthetical grouping, Emacs usually places the start of the line under the preceding line within the group, or under the text after the parenthesis. If you manually give one of these lines a nonstandard indentation (e.g., for aesthetic purposes), the lines below will follow it.

当缩进括号分组内的

The indentation commands for most programming language modes assume that a openparenthesis, open-brace or other opening delimiter at the left margin is the start of a function. If the code you are editing violates this assumption—even if the delimiters occur in strings or comments—you must set open-paren-in-column-0-is-defun-start to nil for indentation to work properly. See Section 23.2.1 [Left Margin Paren], page 235.

### Indenting Several Lines

Sometimes, you may want to reindent several lines of code at a time. One way to do this is to use the mark; when the mark is active and the region is non-empty, TAB indents every line in the region. Alternatively, the command C-M-\ (indent-region) indents every line in the region, whether or not the mark is active (see Section 21.1 [Indentation Commands], page 199).

In addition, Emacs provides the following commands for indenting large chunks of code:

|  |  |
| --- | --- |
| C-M-q | Reindent all the lines within one parenthetical grouping. |
| C-u TAB | Shift an entire parenthetical grouping rigidly sideways so that its first line is properly indented. |
| M-x indent-code-rigidly | Shift all the lines in the region rigidly sideways, but do not alter lines that start inside comments and strings. |

To reindent the contents of a single parenthetical grouping, position point before the beginning of the grouping and type C-M-q. This changes the relative indentation within the grouping, without affecting its overall indentation (i.e., the indentation of the line where the grouping starts). The function that C-M-q runs depends on the major mode; it is indentpp-sexp in Lisp mode, c-indent-exp in C mode, etc. To correct the overall indentation as well, type TAB first.

If you like the relative indentation within a grouping but not the indentation of its first line, move point to that first line and type C-u TAB. In Lisp, C, and some other major modes, TAB with a numeric argument reindents the current line as usual, then reindents by the same amount all the lines in the parenthetical grouping starting on the current line. It is clever, though, and does not alter lines that start inside strings. Neither does it alter C preprocessor lines when in C mode, but it does reindent any continuation lines that may be attached to them.

The command M-x indent-code-rigidly rigidly shifts all the lines in the region sideways, like indent-rigidly does (see Section 21.1 [Indentation Commands], page 199). It doesn’t alter the indentation of lines that start inside a string, unless the region also starts inside that string. The prefix arg specifies the number of columns to indent.

### Customizing Lisp Indentation

The indentation pattern for a Lisp expression can depend on the function called by the expression. For each Lisp function, you can choose among several predefined patterns of indentation, or define an arbitrary one with a Lisp program.

The standard pattern of indentation is as follows: the second line of the expression is indented under the first argument, if that is on the same line as the beginning of the expression; otherwise, the second line is indented underneath the function name. Each following line is indented under the previous line whose nesting depth is the same.

If the variable lisp-indent-offset is non-nil, it overrides the usual indentation pattern for the second line of an expression, so that such lines are always indented lisp-indent-offset more columns than the containing list.

Certain functions override the standard pattern. Functions whose names start with def treat the second lines as the start of a body, by indenting the second line lisp-body-indent additional columns beyond the open-parenthesis that starts the expression.

You can override the standard pattern in various ways for individual functions, according to the lisp-indent-function property of the function name. This is normally done for macro definitions, using the declare construct. See Section “Defining Macros” in the Emacs Lisp Reference Manual.

### Commands for C Indentation

Here are special features for indentation in C mode and related modes:

|  |  |
| --- | --- |
| C-c C-q | Reindent the current top-level function definition or aggregate type declaration (c-indent-defun). |
| C-M-q | Reindent each line in the balanced expression that follows point (c-indent-exp). A prefix argument inhibits warning messages about invalid syntax. |
| TAB | Reindent the current line, and/or in some cases insert a tab character (cindent-command).  If c-tab-always-indent is t, this command always reindents the current line and does nothing else. This is the default.  If that variable is nil, this command reindents the current line only if point is at the left margin or in the line’s indentation; otherwise, it inserts a tab (or the equivalent number of spaces, if indent-tabs-mode is nil).  Any other value (not nil or t) means always reindent the line, and also insert a tab if within a comment or a string. |

To reindent the whole current buffer, type C-x h C-M-\. This first selects the whole buffer as the region, then reindents that region.

To reindent the current block, use C-M-u C-M-q. This moves to the front of the block and then reindents it all.

### Customizing C Indentation

C mode and related modes use a flexible mechanism for customizing indentation. C mode indents a source line in two steps: first it classifies the line syntactically according to its contents and context; second, it determines the indentation offset associated by your selected style with the syntactic construct and adds this onto the indentation of the anchor statement.

C-c . RET style RET

Select a predefined style style (c-set-style).

A style is a named collection of customizations that can be used in C mode and the related modes. Section “Styles” in The CC Mode Manual, for a complete description.

Emacs comes with several predefined styles, including gnu, k&r, bsd, stroustrup, linux, python, java, whitesmith, ellemtel, and awk. Some of these styles are primarily intended for one language, but any of them can be used with any of the languages supported by these modes. To find out what a style looks like, select it and reindent some code, e.g., by typing

C-M-q at the start of a function definition.

To choose a style for the current buffer, use the command C-c .. Specify a style name as an argument (case is not significant). This command affects the current buffer only, and it affects only future invocations of the indentation commands; it does not reindent the code already in the buffer. To reindent the whole buffer in the new style, you can type C-x h C-M-\.

You can also set the variable c-default-style to specify the default style for various major modes. Its value should be either the style’s name (a string) or an alist, in which each element specifies one major mode and which indentation style to use for it. For example,

(setq c-default-style

’((java-mode . "java")

(awk-mode . "awk")

(other . "gnu")))

specifies explicit choices for Java and AWK modes, and the default ‘gnu’ style for the other C-like modes. (These settings are actually the defaults.) This variable takes effect when you select one of the C-like major modes; thus, if you specify a new default style for Java mode, you can make it take effect in an existing Java mode buffer by typing M-x java-mode there.

The gnu style specifies the formatting recommended by the GNU Project for C; it is the default, so as to encourage use of our recommended style.

See Section “Indentation Engine Basics” in the CC Mode Manual , and Section “Customizing Indentation” in the CC Mode Manual , for more information on customizing indentation for C and related modes, including how to override parts of an existing style and how to define your own styles.

As an alternative to specifying a style, you can tell Emacs to guess a style by typing M-x c-guess in a sample code buffer. You can then apply the guessed style to other buffers with M-x c-guess-install. See Section “Guessing the Style” in the CC Mode Manual, for details.

## Commands for Editing with Parentheses

This section describes the commands and features that take advantage of the parenthesis structure in a program, or help you keep it balanced.

When talking about these facilities, the term “parenthesis” also includes braces, brackets, or whatever delimiters are defined to match in pairs. The major mode controls which delimiters are significant, through the syntax table (see Section “Syntax Tables” in The Emacs Lisp Reference Manual). In Lisp, only parentheses count; in C, these commands apply to braces and brackets too.

You can use M-x check-parens to find any unbalanced parentheses and unbalanced string quotes in the buffer.

### Expressions with Balanced Parentheses

Each programming language mode has its own definition of a balanced expression. Balanced expressions typically include individual symbols, numbers, and string constants, as well as pieces of code enclosed in a matching pair of delimiters. The following commands deal with balanced expressions (in Emacs, such expressions are referred to internally as sexps 1 ).

1The word “sexp” is used to refer to an expression in Lisp.

|  |  |
| --- | --- |
| C-M-f | Move forward over a balanced expression (forward-sexp). |
| C-M-b | Move backward over a balanced expression (backward-sexp). |
| C-M-k | Kill balanced expression forward (kill-sexp). |
| C-M-t | Transpose expressions (transpose-sexps). |
| C-M-@  C-M-SPC | Put mark after following expression (mark-sexp). |

To move forward over a balanced expression, use C-M-f (forward-sexp). If the first significant character after point is an opening delimiter (e.g., ‘(’, ‘[’ or ‘{’ in C), this command moves past the matching closing delimiter. If the character begins a symbol, string, or number, the command moves over that.

The command C-M-b (backward-sexp) moves backward over a balanced expression—like C-M-f, but in the reverse direction. If the expression is preceded by any prefix characters (single-quote, backquote and comma, in Lisp), the command moves back over them as well.

C-M-f or C-M-b with an argument repeats that operation the specified number of times; with a negative argument means to move in the opposite direction. In most modes, these two commands move across comments as if they were whitespace. Note that their keys, C-M-f and C-M-b, are analogous to C-f and C-b, which move by characters (see Section 4.2 [Moving Point], page 17), and M-f and M-b, which move by words (see Section 22.1 [Words], page 202).

To kill a whole balanced expression, type C-M-k (kill-sexp). This kills the text that C-M-f would move over.

C-M-t (transpose-sexps) switches the positions of the previous balanced expression and the next one. It is analogous to the C-t command, which transposes characters (see Section 13.2 [Transpose], page 106). An argument to C-M-t serves as a repeat count, moving the previous expression over that many following ones. A negative argument moves the previous balanced expression backwards across those before it. An argument of zero, rather than doing nothing, transposes the balanced expressions ending at or after point and the mark.

To operate on balanced expressions with a command which acts on the region, type C-M-SPC (mark-sexp). This sets the mark where C-M-f would move to. While the mark is active, each successive call to this command extends the region by shifting the mark by one expression. Positive or negative numeric arguments move the mark forward or backward by the specified number of expressions. The alias C-M-@ is equivalent to C-M-SPC. See Section 8.2 [Marking Objects], page 46, for more information about this and related commands.

In languages that use infix operators, such as C, it is not possible to recognize all balanced expressions because there can be multiple possibilities at a given position. For example, C mode does not treat ‘foo + bar’ as a single expression, even though it is one C expression; instead, it recognizes ‘foo’ as one expression and ‘bar’ as another, with the ‘+’ as punctuation between them. However, C mode recognizes ‘(foo + bar)’ as a single expression, because of the parentheses.

### Moving in the Parenthesis Structure

The following commands move over groupings delimited by parentheses (or whatever else serves as delimiters in the language you are working with). They ignore strings and comments, including any parentheses within them, and also ignore parentheses that are “quoted” with an escape character. These commands are mainly intended for editing programs, but can be useful for editing any text containing parentheses. They are referred to internally as “list” commands because in Lisp these groupings are lists.

These commands assume that the starting point is not inside a string or a comment. If you invoke them from inside a string or comment, the results are unreliable.

|  |  |
| --- | --- |
| C-M-n | Move forward over a parenthetical group (forward-list). |
| C-M-p | Move backward over a parenthetical group (backward-list). |
| C-M-u | Move up in parenthesis structure (backward-up-list). |
| C-M-d | Move down in parenthesis structure (down-list). |

The “list” commands C-M-n (forward-list) and C-M-p (backward-list) move forward or backward over one (or n) parenthetical groupings.

C-M-n and C-M-p try to stay at the same level in the parenthesis structure. To move up one (or n) levels, use C-M-u (backward-up-list). C-M-u moves backward up past one unmatched opening delimiter. A positive argument serves as a repeat count; a negative argument reverses the direction of motion, so that the command moves forward and up one or more levels.

To move down in the parenthesis structure, use C-M-d (down-list). In Lisp mode, where ‘(’ is the only opening delimiter, this is nearly the same as searching for a ‘ (’. An argument specifies the number of levels to go down.

### Matching Parentheses

Emacs has a number of parenthesis matching features, which make it easy to see how and whether parentheses (or other delimiters) match up.

Whenever you type a self-inserting character that is a closing delimiter, the cursor moves momentarily to the location of the matching opening delimiter, provided that is on the screen. If it is not on the screen, Emacs displays some of the text near it in the echo area. Either way, you can tell which grouping you are closing off. If the opening delimiter and closing delimiter are mismatched—such as in ‘[x)’—a warning message is displayed in the echo area.

Three variables control the display of matching parentheses:

* blink-matching-paren turns the feature on or off: nil disables it, but the default is t to enable it.
* blink-matching-delay says how many seconds to leave the cursor on the matching opening delimiter, before bringing it back to the real location of point. This may be an integer or floating-point number; the default is 1.
* blink-matching-paren-distance specifies how many characters back to search to find the matching opening delimiter. If the match is not found in that distance, Emacs stops scanning and nothing is displayed. The default is 102400.

Show Paren mode, a global minor mode, provides a more powerful kind of automatic matching. Whenever point is before an opening delimiter or after a closing delimiter, both that delimiter and its opposite delimiter are highlighted. To toggle Show Paren mode, type M-x show-paren-mode.

Electric Pair mode, a global minor mode, provides a way to easily insert matching delimiters. Whenever you insert an opening delimiter, the matching closing delimiter is automatically inserted as well, leaving point between the two. To toggle Electric Pair mode, type M-x electric-pair-mode.

## Manipulating Comments

Because comments are such an important part of programming, Emacs provides special commands for editing and inserting comments. It can also do spell checking on comments with Flyspell Prog mode (see Section 13.4 [Spelling], page 107).

Some major modes have special rules for indenting different kinds of comments. For example, in Lisp code, comments starting with two semicolons are indented as if they were lines of code, while those starting with three semicolons are supposed to be aligned to the left margin and are often used for sectioning purposes. Emacs understand these conventions; for instance, typing TAB on a comment line will indent the comment to the appropriate position.

;; This function is just an example.

;;; Here either two or three semicolons are appropriate.

(defun foo (x)

;;; And now, the first part of the function:

;; The following line adds one.

(1+ x)) ; This line adds one.

### Comment Commands

The following commands operate on comments:

|  |  |
| --- | --- |
| M-; | Insert or realign comment on current line; if the region is active, comment or uncomment the region instead (comment-dwim). |
| C-u M-; | Kill comment on current line (comment-kill). |
| C-x ; | Set comment column (comment-set-column). |
| C-M-j  M-j | Like RET followed by inserting and aligning a comment (comment-indent-newline). See Section 23.5.2 [Multi-Line Comments], page 245. |
| M-x comment-region  C-c C-c (in C-like modes) | Add comment delimiters to all the lines in the region. |

The command to create or align a comment is M-; (comment-dwim). The word “dwim” is an acronym for “Do What I Mean”; it indicates that this command can be used for many different jobs relating to comments, depending on the situation where you use it.

When a region is active (see Chapter 8 [Mark], page 45), M-; either adds comment delimiters to the region, or removes them. If every line in the region is already a comment, it “uncomments” each of those lines by removing their comment delimiters. Otherwise, it adds comment delimiters to enclose the text in the region.

If you supply a prefix argument to M-; when a region is active, that specifies the number of comment delimiters to add or delete. A positive argument n adds n delimiters, while a negative argument -n removes n delimiters.

If the region is not active, and there is no existing comment on the current line, M-; adds a new comment to the current line. If the line is blank (i.e., empty or containing only whitespace characters), the comment is indented to the same position where TAB would indent to (see Section 23.3.1 [Basic Indent], page 237). If the line is non-blank, the comment is placed after the last non-whitespace character on the line; normally, Emacs tries putting it at the column specified by the variable comment-column (see Section 23.5.3 [Options for Comments], page 245), but if the line already extends past that column, it puts the comment at some suitable position, usually separated from the non-comment text by at least one space. In each case, Emacs places point after the comment’s starting delimiter, so that you can start typing the comment text right away.

You can also use M-; to align an existing comment. If a line already contains the comment-start string, M-; realigns it to the conventional alignment and moves point after the comment’s starting delimiter. As an exception, comments starting in column 0 are not moved. Even when an existing comment is properly aligned, M-; is still useful for moving directly to the start of the comment text.

C-u M-; (comment-dwim with a prefix argument) kills any comment on the current line, along with the whitespace before it. Since the comment is saved to the kill ring, you can reinsert it on another line by moving to the end of that line, doing C-y, and then M-; to realign the comment. You can achieve the same effect as C-u M-; by typing M-x comment-kill (comment-dwim actually calls comment-kill as a subroutine when it is given a prefix argument).

The command M-x comment-region is equivalent to calling M-; on an active region, except that it always acts on the region, even if the mark is inactive. In C mode and related modes, this command is bound to C-c C-c. The command M-x uncomment-region uncomments each line in the region; a numeric prefix argument specifies the number of comment delimiters to remove (negative arguments specify the number of comment to delimiters to add).

For C-like modes, you can configure the exact effect of M-; by setting the variables cindent-comment-alist and c-indent-comments-syntactically-p. For example, on a line ending in a closing brace, M-; puts the comment one space after the brace rather than at comment-column. For full details see Section “Comment Commands” in The CC Mode Manual.

# ****Compiling and Testing Programs**** （编译和运行程序）

# ****Maintaining Large Programs**** （维护大型程序）

# ****Abbrevs****

# ****Dired, the Directory Editor**** （Dired，目录编辑器）

# ****The Calendar and the Diary****

# ****Sending Mail****

# ****Reading Mail with Rmail****

# ****Miscellaneous Commands****

# ****Emacs Lisp Packages（Emacs lisp包）****

**Emacs includes a facility that lets you easily download and install packages that implement additional features. Each package is a separate Emacs Lisp program, sometimes including other components such as an Info manual.**

**Emacs包含了一个功能，可以让你轻松的下载并安装软件包，这样可以实现额外的功能。每个软件包都是一个单独的Emacs Lisp程序，有时候还包括其他部分，比如信息手册。**

**M-x list-packages brings up a buffer named \*Packages\* with a list of all packages. You can install or uninstall packages via this buffer. See Section 32.1 [Package Menu], page 400.The command C-h P (describe-package) prompts for the name of a package, and displays a help buffer describing the attributes of the package and the features that it implements.**

**M-x list-packages产生一个包含有软件包列表的名叫\*Packages\*的buffer。你可以通过这个buffer来安装或卸载软件包。参见Section 32.1 [Package Menu], page 400。命令C-h P (describe-package)会提示软件包的名字，并显示help buffer来描述软件包的属性和它实现的功能。**

**By default, Emacs downloads packages from a package archive maintained by the Emacs developers and hosted by the GNU project. Optionally, you can also download packages from archives maintained by third parties. See Section 32.2 [Package Installation], page 401.**

**默认情况下，Emacs从包存当中下载软件包，该存档由Emacs开发者维护，GNU 项目主持。或者，你也可以从第三方维护的存档中下载软件包。参见Section 32.2 [Package Installation], page 401。**

**For information about turning an Emacs Lisp program into an installable package, See Section “Packaging” in The Emacs Lisp Reference Manual. For information about finding third-party packages and other Emacs Lisp extensions, See Section “Packages that do not come with Emacs” in GNU Emacs FAQ.**

**关于如何将一个Emacs Lisp程序转变为一个可安装的软件包的信息，参见Section “Packaging” in The Emacs Lisp Reference Manual。关于第三方软件包的信息和其他Emacs Lisp扩展，参见Section “Packages that do not come with Emacs” in GNU Emacs FAQ。**

## **The Package Menu Buffer（软件包菜单缓冲区）**

**The command M-x list-packages brings up the package menu. This is a buffer listing all the packages that Emacs knows about, one on each line, with the following information:**

**命令M-x list-packages将会产生一个软件包菜单。该buffer列出了Emacs知道的所有的软件包，每行一个，包含以下信息：**

* **The package name (e.g., ‘auctex’).**
* **软件包名字（例如，‘auctex’）。**
* **The package’s version number (e.g., ‘11.86’).**
* **软件包的版本号（例如，‘11.86’）。**
* **The package’s status—normally one of ‘available’ (can be downloaded from the package archive), ‘installed’, or ‘built-in’ (included in Emacs by default). The status can also be ‘new’. This is equivalent to ‘available’, except that it means the package became newly available on the package archive after your last invocation of M-x list-packages. In other instances, a package may have the status ‘held’, ‘disabled’, or ‘obsolete’. See Section 32.2 [Package Installation], page 401.**
* **软件包的状态-通常是‘可用（available）’ (能够从包存档中下载), ‘已安装（installed）’, 或 ‘内建（built-in）’ (默认包含在Emacs中)当中的一个。状态也可能是“新建（new）”。它等于“可用（available）“，但它意味着在你上次调用M-x list-packages之后它在包存档中变为新的可用。在其他情况下，一个包可能有状态为”持有（held）“，‘禁用（disabled）’, 或‘过时（obsolete）’。参见Section 32.2 [Package Installation], page 401。**
* **A short description of the package.**
* **软件包的简短描述。**

**The list-packages command accesses the network, to retrieve the list of available packages from the package archive server. If the network is unavailable, it falls back on the most recently retrieved list.**

**list-packages命令访问网络，来从包存档服务器来检索可用的软件包列表。如果网络不可用，它将会返回最近检索到的列表。**

**The following commands are available in the package menu:**

**下面的命令在软件包菜单中可用：**

|  |  |
| --- | --- |
| **h** | **Print a short message summarizing how to use the package menu (package-menu-quick-help).**  **打印一条简短消息来总结如何使用软件包菜单(package-menu-quick-help)。** |
| **?**  **RET** | **Display a help buffer for the package on the current line (package-menu-describe-package), similar to the help window displayed by the C-h P command (see Chapter 32 [Packages], page 400).**  **为当前行上的软件包显示一个帮助buffer（package-menu-describe-package），类似C-h P命令显示的帮助window。（参见Chapter 32 [Packages], page 400）。** |
| **i** | **Mark the package on the current line for installation (package-menu-mark-install). If the package status is ‘available’, this adds an ‘I’ character to the start of the line; typing x (see below) will download and install the package.**  **将当前行上的软件包标记为待安装（package-menu-mark-install）。如果软件包状态是”可用（available）“，这将会在该行起始处添加”I”字符；按下 x（见下文）将会下载并安装该软件包。** |
| **d** | **Mark the package on the current line for deletion (package-menu-mark-delete). If the package status is ‘installed’, this adds a ‘D’ character to the start of the line; typing x (see below) will delete the package. See Section 32.3 [Package Files], page 402, for information about what package deletion entails.**  **将当前行上的软件包标记为待删除（package-menu-mark-delete）。如果软件包的状态是“安装（installed）“，这将会在当前行的开始出添加”D”字符，按下x（见下文）将会删除该软件包。参见，获取关删除软件包需要了解的信息。** |
| **u** | **Remove any installation or deletion mark previously added to the current line by an i or d command.**  **移除任何之前通过i或d命令添加到当前行的安装或删除标记。** |
| **U** | **Mark all package with a newer available version for “upgrading” (packagemenu-mark-upgrades). This places an installation mark on the new available versions, and a deletion mark on the old installed versions.**  **将全部由新的可用版本的包做标记来进行“升级（upgrading）”（packagemenu-mark-upgrades）。这将会在新的可用版本的设置安装标记，在旧的已安装版本设置删除标记。** |
| **x** | **Download and install all packages marked with i, and their dependencies; also, delete all packages marked with d (package-menu-execute). This also removes the marks.**  **下载并安装所有通过i进行标记的软件包，以及他们的依赖，删除所有通过d标记的软件包（package-menu-execute）。这同时会移除那些标记。** |
| **r** | **Refresh the package list (package-menu-refresh). This fetches the list of available packages from the package archive again, and recomputes the package list.**  **刷新包列表（package-menu-refresh）。这将会再次从包存档上获取可用软件包的列表，并重新计算包列表。** |

**For example, you can install a package by typing i on the line listing that package, followed by x.**

**例如，你可以通过在显示软件包的行上按下i x来安装。**

## **Package Installation（软件包安装）**

**Packages are most conveniently installed using the package menu (see Section 32.1 [Package Menu], page 400), but you can also use the command M-x package-install. This prompts for the name of a package with the ‘available’ status, then downloads and installs it.**

**通过使用软件包菜单是安装软件包最方便的方法（参见Section 32.1 [Package Menu], page 400），但是也可以使用命令M-x package-install。这会提示处于“可用（available）”状态的软件包的名字，然后下载并安装它。**

**A package may require certain other packages to be installed, because it relies on functionality provided by them. When Emacs installs such a package, it also automatically downloads and installs any required package that is not already installed. (If a required package is somehow unavailable, Emacs signals an error and stops installation.) A package’s requirements list is shown in its help buffer.**

**一个软件包可能会要求一样要安装其他的软件包，因为它依赖于他们所提供的功能。当Emacs安装这样的软件包的时候，它会自动下载并安装任何没有安装但是所需的包。（如果一个所需的包有些时候不可用，Emacs会显示一个错误并停止安装。）一个软件包的要求清淡会显示在它的帮助buffer中。**

**By default, packages are downloaded from a single package archive maintained by the Emacs developers. This is controlled by the variable package-archives, whose value is a list of package archives known to Emacs. Each list element must have the form (*id . location*), where id is the name of a package archive and location is the HTTP address or directory name of the package archive. You can alter this list if you wish to use third party package archives—but do so at your own risk, and use only third parties that you think you can trust!**

**默认情况下，软件包会从一个Emacs开发者维护的软件包存档进行下载。这是由变量package-archives进行控制的，它的值是一个Emacs知道的包存档的列表。每个列表元素必须是这种形式（id.location），id是软件包存档的名字，location是HTTP地址或者包存档的目录。可以按照你的意愿来修改这个列表来使用第三方的软件包存档，但是这样会增加你的风险，只使用你认为相信的第三方。**

**Once a package is downloaded and installed, it is loaded into the current Emacs session. Loading a package is not quite the same as loading a Lisp library (see Section 24.8 [Lisp Libraries], page 269); its effect varies from package to package. Most packages just make some new commands available, while others have more wide-ranging effects on the Emacs session. For such information, consult the package’s help buffer.**

**一旦软件包下载并安装，它就会加载到当前的Emacs会话中。加载软件包和加载lisp lisbrary（参见Section 24.8 [Lisp Libraries], page 269）不太一样；包和包所做的影响不一样。大多数的软件包只是使一些新的命令可用，但是另外一些会广泛的影响到Emacs的会话。对于这样的信息，查阅包的帮助缓冲区。**

**By default, Emacs also automatically loads all installed packages in subsequent Emacs sessions. This happens at startup, after processing the init file (see Section 33.4 [Init File], page 429). As an exception, Emacs does not load packages at startup if invoked with the ‘-q’ or ‘--no-init-file’ options (see Section C.2 [Initial Options], page 470).**

**默认情况下，Emacs会在后续的Emacs会话中加载所有已安装的软件包。这发生在启动时，处理init文件（参见Section 33.4 [Init File], page 429）之后。例外的是，如果启动时调用‘-q’ 或‘--no-init-file’就就不会加载软件包（参见Section C.2 [Initial Options], page 470）。**

**To disable automatic package loading, change the variable package-enable-at-startup to nil.**

**将变量package-enable-at-startup设置为nil可以关闭自动加载软件包。**

**The reason automatic package loading occurs after loading the init file is that user options only receive their customized values after loading the init file, including user options which affect the packaging system. In some circumstances, you may want to load packages explicitly in your init file (usually because some other code in your init file depends on a package). In that case, your init file should call the function package-initialize. It is up to you to ensure that relevant user options, such as package-load-list (see below), are set up prior to the package-initialize call. You should also set package-enable-at-startup to nil, to avoid loading the packages again after processing the init file. Alternatively, you may choose to completely inhibit package loading at startup, and invoke the command M-x package-initialize to load your packages manually.**

**在加载init文件后自动加载软件包的原因是：在加载init文件之后用户选项才会收到它们的定制值，包括哪些会影响到软件包系统的用户选项。在某些情况下，可能想在你的init文件中明确的加载一些软件包(通常是因为你init 文件中的一些代码依赖这些软件包)。在这种情况下，init文件应该调用package-initialize函数。这使你能够确定相关的用户选项，比如package-load-list（见下文），你可以将其设置为在软件包初始化前调用。也应该将package-enable-at-startup设置为nil，来避免在加载init文件的时候再次加载那个软件包。另外，你可以选择禁止在启动的时候加载软件包，通过调用命令M-x package-initialize手动加载软件包。**

**For finer control over package loading, you can use the variable package-load-list. Its value should be a list. A list element of the form (name version) tells Emacs to load version version of the package named name. Here, version should be a version string (corresponding to a specific version of the package), or t (which means to load any installed version), or nil (which means no version; this “disables” the package, preventing it from being loaded). A list element can also be the symbol all, which means to load the latest installed version of any package not named by the other list elements. The default value is just ’(all).**

**使用变量package-load-list可以更精细的控制软件包的加载。它的值是一个列表。（name version）形式的列表元素告诉Emacs去加载版本为version，名字为name的软件包。此处，version应该是一个版本字符串（对应软件包版本），或者是t（这意味着加载任何已安装的版本），或者nil（这意味着没有版本；这会禁用软件包，组织它被加载）。一个列表原色也可以全部是符号，这意味着加载任何不被其他列表元素命令的软件包的最新的已安装版本。**

**For example, if you set package-load-list to ’((muse "3.20") all), then Emacs only loads version 3.20 of the ‘muse’ package, plus any installed version of packages other than ‘muse’. Any other version of ‘muse’ that happens to be installed will be ignored. The ‘muse’ package will be listed in the package menu with the ‘held’ status.**

**例如，如果将package-load-list设置为’((muse "3.20") all)，Emac只加载3.20 版本的‘muse’软件包，外加任何版本的非muse软件包。任何其他版本的muse都会被忽略安装。Muse软件包将会在软件包菜单中列为“持有（held）”状态。**

## **Package Files and Directory Layout（包文件和目录布局）**

**Each package is downloaded from the package archive in the form of a single package file—either an Emacs Lisp source file, or a tar file containing multiple Emacs Lisp source and other files. Package files are automatically retrieved, processed, and disposed of by the Emacs commands that install packages. Normally, you will not need to deal directly with them, unless you are making a package (see Section “Packaging” in The Emacs Lisp Reference Manual). Should you ever need to install a package directly from a package file, use the command M-x package-install-file.**

**每个包文件都以单一封装文件的形式从包归档下载下来的—要么是一个Emacs Lisp源文件，要么包含多个Emacs Lisp源文件和其他文件的打包文件。Emacs安装软件包的命令会自动检索、加工和处理包文件。正常情况下，你不需要直接和他们打交道，除非是你在制作一个软件包（参见）。如果你需要从一个包文件直接安装软件包，应该使用命令M-x package-install-file。**

**Once installed, the contents of a package are placed in a subdirectory of ~/.emacs.d/elpa/ (you can change the name of that directory by changing the variable package-user-dir). The package subdirectory is named name-version, where name is the package name and version is its version string.**

**一旦安装，软件包的内容就会放在~/.emacs.d/elpa/的子目录下（你可以通过改变变量package-user-dir来改变那个目录的名字）。软件包子目录命名为name-version，name是软件包的名字，version是它的版本字符串。**

**In addition to package-user-dir, Emacs looks for installed packages in the directories listed in package-directory-list. These directories are meant for system administrators to make Emacs packages available system-wide; Emacs itself never installs packages there. The package subdirectories for package-directory-list are laid out in the same way as in package-user-dir.**

**除了ackage-user-dir，Emacs还会从package-directory-list所列的目录中差早安装的软件包。这些软件包对于系统管理员是有意义的，因为是Emacs软件包在系统范围内可用；Emacs本身不会在那些地方安装软件包。package-directory-list软件包子目录会以同样方式出现在package-user-dir中。**

# ****Customization****

# ****Dealing with Common Problems****

# ****Appendix A GNU GENERAL PUBLIC LICENSE****

# ****Appendix B GNU Free Documentation License****

# ****Appendix C Command Line Arguments for Emacs****

**Emacs supports command line arguments to request various actions when invoking Emacs. These are for compatibility with other editors and for sophisticated activities. We don’t recommend using them for ordinary editing (See Section 31.4 [Emacs Server], page 384, for a way to access an existing Emacs job from the command line).**

**Arguments starting with ‘-’ are options, and so is ‘+linenum’. All other arguments specify files to visit. Emacs visits the specified files while it starts up. The last file specified on the command line becomes the current buffer; the other files are also visited in other buffers. As with most programs, the special argument ‘--’ says that all subsequent arguments are file names, not options, even if they start with ‘-’.**

**Emacs command options can specify many things, such as the size and position of the X window Emacs uses, its colors, and so on. A few options support advanced usage, such as running Lisp functions on files in batch mode. The sections of this chapter describe the available options, arranged according to their purpose.**

**There are two ways of writing options: the short forms that start with a single ‘-’, and the long forms that start with ‘--’. For example, ‘-d’ is a short form and ‘--display’ is the corresponding long form.**

**The long forms with ‘--’ are easier to remember, but longer to type. However, you don’t have to spell out the whole option name; any unambiguous abbreviation is enough. When a long option takes an argument, you can use either a space or an equal sign to separate the option name and the argument. Thus, you can write either ‘--display sugar-bombs:0.0’ or ‘--display=sugar-bombs:0.0’. We recommend an equal sign because it makes the relationship clearer, and the tables below always show an equal sign.**

**Most options specify how to initialize Emacs, or set parameters for the Emacs session. We call them initial options. A few options specify things to do, such as loading libraries or calling Lisp functions. These are called action options. These and file names together are called action arguments. The action arguments are stored as a list of strings in the variable command-line-args. (Actually, when Emacs starts up, command-line-args contains all the arguments passed from the command line; during initialization, the initial arguments are removed from this list when they are processed, leaving only the action arguments.)**

## **C.1 Action Arguments**

**Here is a table of action arguments:**

## **C.2 Initial Options**

**The initial options specify parameters for the Emacs session. This section describes the more general initial options; some other options specifically related to the X Window System appear in the following sections.**

## **C.3 Command Argument Example**

**Here is an example of using Emacs with arguments and options. It assumes you have a Lisp program file called hack-c.el which, when loaded, performs some useful operation on the current buffer, expected to be a C program.**

**emacs --batch foo.c -l hack-c -f save-buffer >& log**

**This says to visit foo.c, load hack-c.el (which makes changes in the visited file), save foo.c (note that save-buffer is the function that C-x C-s is bound to), and then exit back to the shell (because of ‘--batch’). ‘--batch’ also guarantees there will be no problem redirecting output to log, because Emacs will not assume that it has a display terminal to work with.**

## **C.4 Environment Variables**

**The environment is a feature of the operating system; it consists of a collection of variables with names and values. Each variable is called an environment variable ; environment variable names are case-sensitive, and it is conventional to use upper case letters only. The values are all text strings.**

### **C.4.1 General Variables**

**Here is an alphabetical list of environment variables that have special meanings in Emacs. Most of these variables are also used by some other programs. Emacs does not require any of these environment variables to be set, but it uses their values if they are set.**

### **C.4.2 Miscellaneous Variables**

**These variables are used only on particular configurations.**

### **C.4.3 The MS-Windows System Registry**

**On MS-Windows, the installation program addpm.exe adds values for emacs\_dir, EMACSLOADPATH, EMACSDATA, EMACSPATH, EMACSDOC, SHELL and TERM to the HKEY\_LOCAL\_MACHINE section of the system registry, under /Software/GNU/Emacs.** **It does this because there is no standard place to set environment variables across different versions of Windows. Running addpm.exe is no longer strictly necessary in recent versions of Emacs, but if you are upgrading from an older version, running addpm.exe ensures that you do not have older registry entries from a previous installation, which may not be compatible with the latest version of Emacs.**

**在MS-Windows中，安装程序addpm.exe将在系统注册表的HKEY\_LOCAL\_MACHINE添加以下变量：emacs\_dir, EMACSLOADPATH, EMACSDATA, EMACSPATH, EMACSDOC, SHELL and TERM。这样做是因为在不同的windows版本没有一个标准的地方来设置环境变量。在最近版本的Emacs中运行addpm.exe并不是必须的，但是如果是从一个旧版本升级而来的，运行addpm.exe确保了注册表中不会残留以前安装版本的信息，那些信息可能与当前版本不匹配。**

**When Emacs starts, as well as checking the environment, it also checks the System Registry for those variables and for HOME, LANG and PRELOAD\_WINSOCK. To determine the value of those variables, Emacs goes through the following procedure.**

**当Emacs启动的时候，正如会检查环境变量一样，他也会检查系统注册表中的HOME, LANG and PRELOAD\_WINSOCK，来确定这些变量的信息，Emacs经历以下过程。**

**First, the environment is checked. If the variable is not found there, Emacs looks for registry keys by that name under /Software/GNU/Emacs; first in the HKEY\_CURRENT\_USER section of the registry, and if not found there, in the HKEY\_LOCAL\_MACHINE section. Finally, if Emacs still cannot determine the values, compiled-in defaults are used.**

**首先检查环境变量。如果没有找到变量，Emacs就会在注册表中的/Software/GNU/Emacs名字下面查找，首先检查HKEY\_CURRENT\_USER节，如果这里没有找到，就去找HKEY\_LOCAL\_MACHINE，最后如果Emacs不能确定变量的值，就使用编译的值。**

**In addition to the environment variables above, you can also add many of the settings which on X belong in the** **.Xdefaults file (see Appendix D [X Resources], page 484) to the /Software/GNU/Emacs registry key.**

**除了上面说到的环境变量，在X上还可以添加很多到.Xdefaults文件中**

### **C.5 Specifying the Display Name**

**The environment variable DISPLAY tells all X clients, including Emacs, where to display their windows. Its value is set by default in ordinary circumstances, when you start an X server and run jobs locally. You can specify the display yourself; one reason to do this is if you want to log into another system and run Emacs there, and have the window displayed at your local terminal.**

# ****Appendix D X Options and Resources****

# ****Appendix E Emacs 23 Antinews****

# ****Appendix F Emacs and Mac OS / GNUstep****

# ****Appendix G Emacs and Microsoft Windows/MSDOS****

**This section describes peculiarities of using Emacs on** **Microsoft Windows. Some of these peculiarities are also relevant to Microsoft’s older MS-DOS “operating system” (also known as “MS-DOG”). However, Emacs features that are relevant only to MS-DOS are described in a separate manual (****see Section “MS-DOS” in Specialized Emacs Features).**

**本节介绍在Microsoft Windows上使用Emacs的一些特性。其中一些特性与微软旧的MS-DOS操作系统（亦称“MS-DOG”）相关。然而，Emacs的功能，仅与MS-DOS相关的Emacs特性描述在一个单独的手册（see Section “MS-DOS” in Specialized Emacs Features）。**

**The behavior of Emacs on MS-Windows is reasonably similar to what is documented in the rest of the manual, including support for long file names, multiple frames, scroll bars, mouse menus, and subprocesses. However, a few special considerations apply, and they are described here.**

## ****G.1 How to Start Emacs on MS-Windows****

**There are several ways of starting Emacs on MS-Windows:**

**1. From the desktop shortcut icon: either double-click the left mouse button on the icon, or click once, then press RET. The desktop shortcut should specify as its “Target” (in the “Properties” of the shortcut) the full absolute file name of runemacs.exe, not of emacs.exe. This is because runemacs.exe hides the console window that would have been created if the target of the shortcut were emacs.exe (which is a console program, as far as Windows is concerned). If you use this method, Emacs starts in the directory specified by the shortcut. To control where that is, right-click on the shortcut, select “Properties”, and in the “Shortcut” tab modify the “Start in” field to your liking.**

**2. From the Command Prompt window, by typing emacs RET at the prompt. The Command Prompt window where you did that will not be available for invoking other commands until Emacs exits. In this case, Emacs will start in the current directory of the Windows shell.**

**3. From the Command Prompt window, by typing runemacs RET at the prompt. The Command Prompt window where you did that will be immediately available for invoking other commands. In this case, Emacs will start in the current directory of the Windows shell.**

**4. Via emacsclient.exe or emacsclientw.exe, which allow you to invoke Emacs from other programs, and to reuse a running Emacs process for serving editing jobs required by other programs. See Section 31.4 [Emacs Server], page 384. The difference between emacsclient.exe and emacsclientw.exe is that the former is a console program, while the latter is a Windows GUI program. Both programs wait for Emacs to signal that the editing job is finished, before they exit and return control to the program that invoked them. Which one of them to use in each case depends on the expectations of the program that needs editing services. If that program is itself a console (text-mode) program, you should use emacsclient.exe, so that any of its messages and prompts appear in the same command window as those of the invoking program. By contrast, if the invoking program is a GUI program, you will be better off using emacsclientw.exe, because emacsclient.exe will pop up a command window if it is invoked from a GUI program. A notable situation where you would want emacsclientw.exe is when you right-click on a file in the Windows Explorer and select “Open With” from the pop-up menu. Use the ‘--alternate-editor=’ or ‘-a’ options if Emacs might not be running (or not running as a server) when emacsclient is invoked—that will always give you an editor. When invoked via emacsclient, Emacs will start in the current directory of the program that invoked emacsclient.**

**Note that, due to limitations of MS-Windows, Emacs cannot have both GUI and textmode frames in the same session. It also cannot open text-mode frames on more than a single Command Prompt window, because each Windows program can have only one console at any given time. For these reasons, if you invoke emacsclient with the -c option, and the Emacs server runs in a text-mode session, Emacs will always create a new text-mode frame in the same Command Prompt window where it was started; a GUI frame will be created only if the server runs in a GUI session. Similarly, if you invoke emacsclient with the -t option, Emacs will create a GUI frame if the server runs in a GUI session, or a text-mode frame when the session runs in text mode in a Command Prompt window. See Section 31.4.2 [emacsclient Options], page 385.**

# ****The GNU Manifesto****

# ****Glossary****

# ****Key (Character) Index****

# ****Command and Function Index****

# ****Variable Index****

# ****Concept Index****